Instituto Tecnológico y de Estudios Superiores de Monterrey

“Campus Querétaro"

Bionic filtering

Programming languages final project

Author:

A01704446 Viviana Elizabeth Dueñas Chávez

Professor:

Benjamín Valdés Aguirre

May 27, 2021

1 Context

1.1 Introduction

Retinal diseases are the most common cause of childhood blindness worldwide. The retina is the tissue placed on the back of the eye, in charge of photoreception and light processing for visual recognition. The eye has photoreceptor cells capable of detecting qualities of color and light intensities. This is the information that the retina senses and transform to signals for vision purposes.

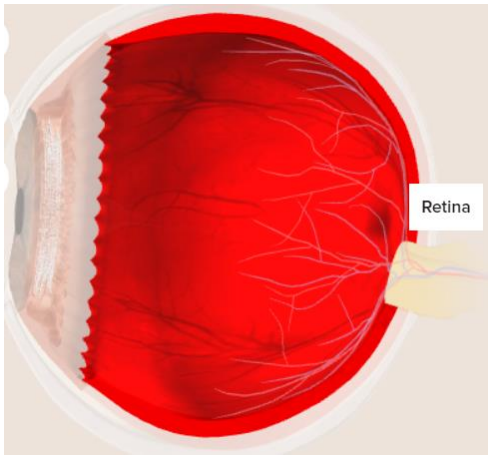


Figure 1.- Eye structure highlighting the retina layer. Retrieved from:

https://www.healthline.com/human-body-maps/retina#1

The causes of blindness changes per wealthiness on each country, but the common denominator in middle, like in Latin America, and most wealthy countries are diseases due to retina damages. The leading causes of blindness in those regions are retina diseases.

Nowadays, the bionic eye, an implant capable of restoring sight, is a fact in the ophthalmological field, letting patients completely blind, due to a damaged the retina, a common cause in old people, to perceive the surroundings in a grayscale of 576-pixel range. The most recent method places the vision-restoring sensor (camera) on top of the patient's retina and power it with an external laser, as follows in Figure 1:

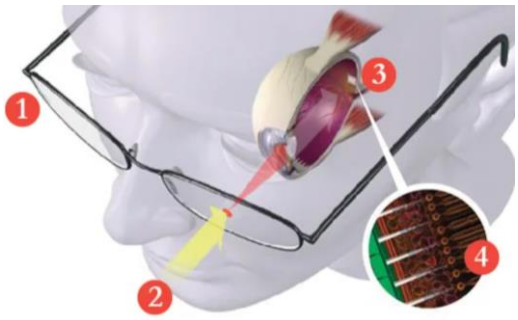


Figure 2.- Bionic implant and its external laser-powering demonstration. Retrieved from:

https://www.popsci.com/technology/article/2012-06/bio-retina-implant-could-give-sight-blind-laser-power/

Each of the shades of the grayscale range is being calculated from an image processor that receives the light information given by the photoreceptors on the chip, that replaces the missing photoreceptor cells sensing and processing operations, caused by a retina disease.

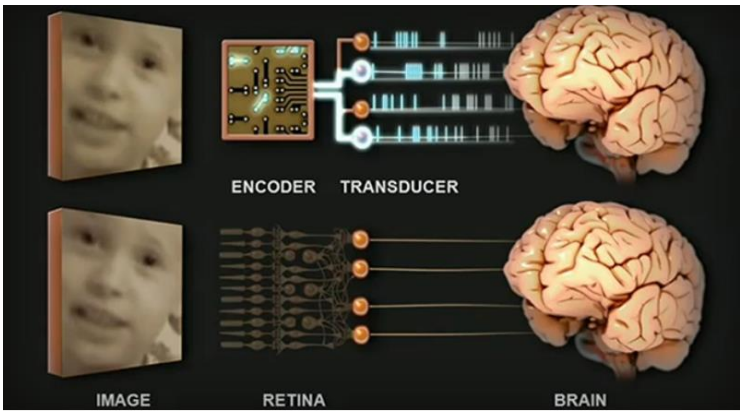


Figure 3.- Bionic implant sensors/transducer comparison with the retina. Retrieved from:

https://www.extremetech.com/extreme/110031-a-bionic-prosthetic-eye-that-speaks-the-language-of-your-brain

Due to this problem and new technology, capable of giving a grayscale vision to the blind, this project approaches the alternative of using filtering for improving the object recognition in the whole image perception. More specifically, the project navigates the possibility of applying edge detection to a video capture, which resembles the video-sensing chip (camera), installed as the bionic eye or the replacement of the damage retina, information.

1.2 Image processing

A digital image can be saw as spaces of memory in which certain information, about a shade or a color, depending on the format, is stored. On the other hand, a digital video will be a representation of a sequence of digital images in binary format. Which basically means, an amount of images in a given period. For this Project, an image will be seen as a 2-dimensional matrix, stored as long line, that contains the information discussed before plus its header that can tell us sizing and format information.

For image processing, there is a technique called “Kernel convolution” which let the applying of filters like blurring and edge detection on digital images. This kernel is matrix smaller than the original image. This structure is passed over the whole image, which is transformed based on the number within the kernel used. For example, if we use a 3x3 convolution kernel, filled with just the number 1, the result obtained would be the application of a blurring filter. The overall result of doing this is a weighted average, since each of the numbers in the kernel are being multiplied by each pixel of the original image, being summed, and divided by the total number of elements in the kernel.

After applying a convolutional kernel to a digital image, the output needs to be placed on a different image for avoiding messing with the original data. The resulting convoluted image is a bit smaller than the original, so the input image needs to have a little “padding” on the borders to maintain the sizing defined in the header.

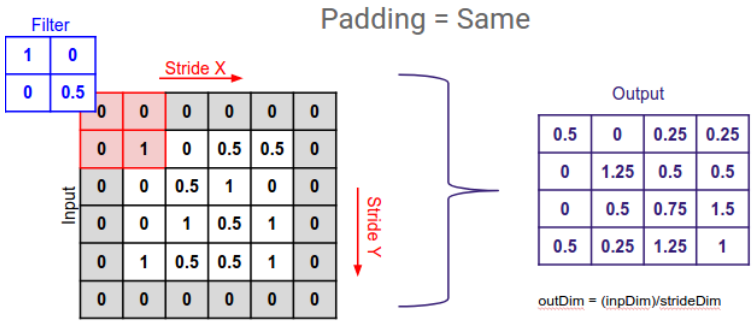


Figure 4.- Convolutional kernel (left), padding technique (middle) and the resulting convoluted image (right).

Retrieved from: https://deepai.org/machine-learning-glossary-and-terms/padding

The “padding” is for the edges of the image, that are usually lost after the convolution step, and can be done by just a zero filling (shown in Figure 4 and 5), a duplication of the borders pixels or by passing the kernel through it with zeros instead of the missing pixels due the border's nature, ignoring the data is undetermined and trying to get the calculations on the border pixel as well.

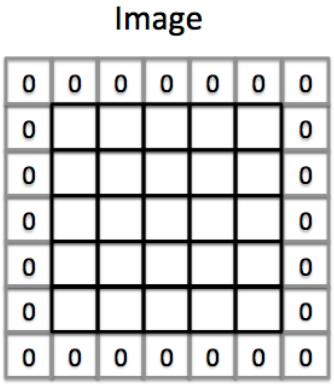


Figure 5.- Zero filling method for padding the image. Retrieved from:

https://medium.com/machine-learning-algorithms/what-is-padding-in-convolutional-neural-network-c120077469cc

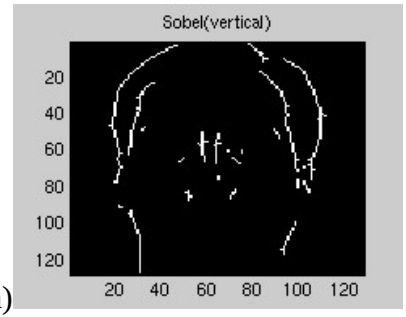
1.3 Edge detection

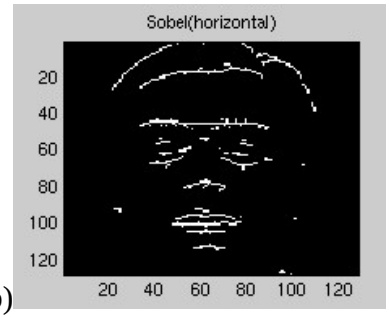
In general, the edge detection is a tool that helps us identify the sharpe changes of intensities in an image:

Figure 6.- Contrast of intensity levels.

Original Lena's image (Left) retrieved from: Rick Swenson's robotic vision material.

The Sobel filter is a way of getting the edges on an image. There can be horizontal and vertical edges:

a) 

b) 

Figures 7.- a) shows the vertical edges using one of the Sobel gradients, and the figure b) shows the horizontal ones.

For calculating the x-gradient, figure 8a, which calculates the derivative of the image to highlighting the vertical changes, we need to use a Sobel operator, that basically separates one region to the other vertically. For the y-gradient, figure 8b, which calculates the same derivative but for highlighting the horizontal edges, we need to use another symmetric Sobel operator, which, as shown before, separates the regions to check for a sharpe change of intensities horizontally:

a) Gx:

-1	0	+1
-2	0	+2
-1	0	+1

b) Gy:

+1	+2	+1
0	0	0
-1	-2	-1

Figures 8.- a) shows the x gradient, and the figure b) shows the y gradient.

This "Sobel operators" are the convolutional kernels applied to each pixel of the image. One kernel for each of the two, as a 2d-dimensioned image, perpendicular orientation. When there is no change the opposite signs in the kernel make the resulting value 0. The gradients are calculated separately from the same input image, and then mixed for the final absolute gradient calculation per pixel, as shown on figures 8a and b, which is the

resulting value of applying the Sobel's filter. Therefore, the resulting sign of the x and y gradients does not affect the result at all.

a) $|G| = \sqrt{Gx^2 + Gy^2}$ b) $|G| = |Gx| + |Gy|$

Figures 9.- a) shows the absolute magnitude value for each orientation, and the figure b) shows the approximation of that magnitude. Retrieved from: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

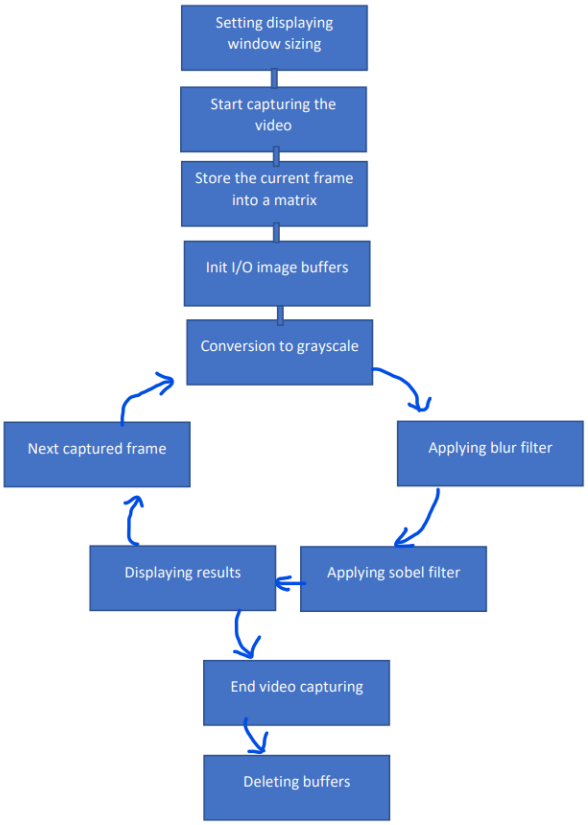
Solution

For this problem, inspired by the bionic eye technology, my proposed solution was to implement a program capable of applying Sobel filter into real time video capturing. That means, a lot of calculations needed to be done at a time. For just one image, we need to apply 2 kernel convolutions, which means twice the calculations per pixel, and for video capturing, per image.

Due to the nature of image processing and the digital image itself, being taken as big matrix, the parallelizing method is the best approach. That is why I chose to implement this in CUDA, using the GPU cores, to do all the calculations at once, with the help of multiple threads that will process each calculation at the same time. The corresponding CPU performance is going to be checked as we are testing the program.

1.1 Algorithm (Overall flow diagram)

To understand how this project works, the following diagram represents the overall flow of the program:



Now, I will breakdown to you how the algorithm works:

1) **Setting displaying window sizing:** This was done by using the function capture.set() as capture.set(CV_CAP_PROP_FRAME_WIDTH,640) for a 640 pixels' resolution for the width and capture.set(CV_CAP_PROP_FRAME_HEIGHT,480) for a 480 pixels' resolution for the height.
2) **Start capturing the video:** As stated before, the video capturing is done through OpenCV functions, in this case: *VideoCapture capture(0).*
3) **Store the current frame into the matrix:** With the help of OpenCV *Mat* class. This an n-dimensional dense array class, capable of storing matrices, grayscale, or color images. For this application, it is okay to use a single channel 8-bit array, due to the grayscale nature of the input image used. That is specified by the constant *CV_8U*.
4) **Init I/O image buffers**: Here I'll be using more Mat containers for storing the future results, one per data being studied. For a better presentation of the image processing part, I'm going to do 5 instantiations of Mat: Original (grayscale conversion storage), Blurr (Noise removal), Edge (Final result), gradX (Gradient X demonstration) and gradY (Gradient Y demonstration).
5) **Conversion to grayscale:** For the RGB to grayscale conversion, I'll be using the "cvtColor()" function with the constant RGB2GRAY. The method they use to do it is to be applying a factor per channel: 0.299 to the red channel, 0.587 to the green one and 0.114 to the blue.
6) **Applying blur filter:** Here, I did the function *boxFilter_host()* which calls everything from the CPU and sets all the GPU pointers. This uses the GPU kernel called *boxFilter_device()* which does a weighted average to each pixel. This is basically like using a 3 by 3 convolutional kernel filled by ones: Everything is going to be multiplied by the ones and then sum and divided by the total amount of elements in the kernel (Image processing methodology stated before).
7) **Applying Sobel filter:** Here, I did the function *sobelFilter_host()* which calls everything from the CPU and sets all the GPU pointers. This uses the GPU kernel called *sobelFilter_device()* which applied all we saw before edge detection using convolutional kernels.
8) **Displaying results:** For displaying the image results, I used the function *cvShowImage(),* which displays the resulting matrix stored in the Mat class called *edge* on a window with the original's sizing.
9) **End video capturing:** If ESC is press, there is a condition that breaks the while(1) loop, so the video capturing and all the processing end. Otherwise, the while continuous with the next frame/image of the video capturing an we go back to the step 5.
10) **Deleting buffers:** This the final action to free all the allocated data stored by all Mat's instantiations. Sidenote, all the memory allocated on GPU and CPU for the filters application is done at the end of each host/device blurr/sobel methods.

1.2 Solution architecture (Tools, steps)

This project uses OpenCV for the video capturing and the result displaying, aside for the in between RGB to grayscale conversions. Also, the CUDA framework is used due to the

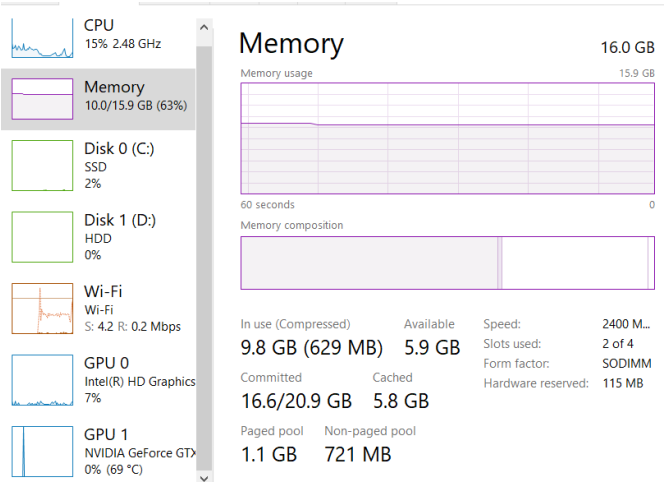
nature of parallelizing all the pixel calculations and to create the resulting images: All the gradients, the blur and edge detection application's resulting image.

Before applying the sobel filter, there is being applied the box filter for a touch of blurring to erase some noise from the original image. The result of the first filter is passed to the second method that gets the edge detection final computations. The final performance is going to be tested too. It is expected a low processing done by CPU, due to the multi-threading.

Results

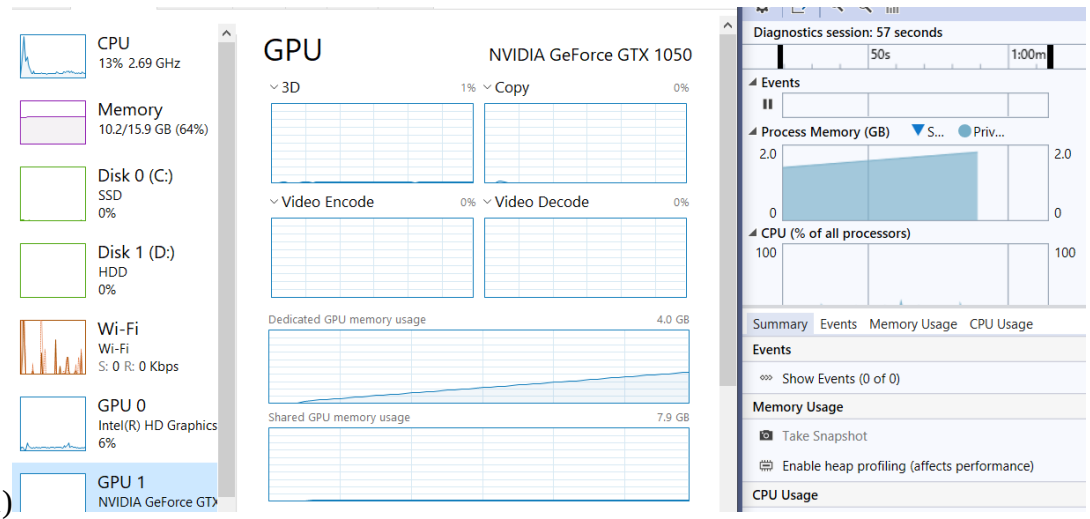
The solution was tried several times, after 57 seconds of running it, to see how the performance in the CPU was been doing and if it was a stable behavior:

Before testing:

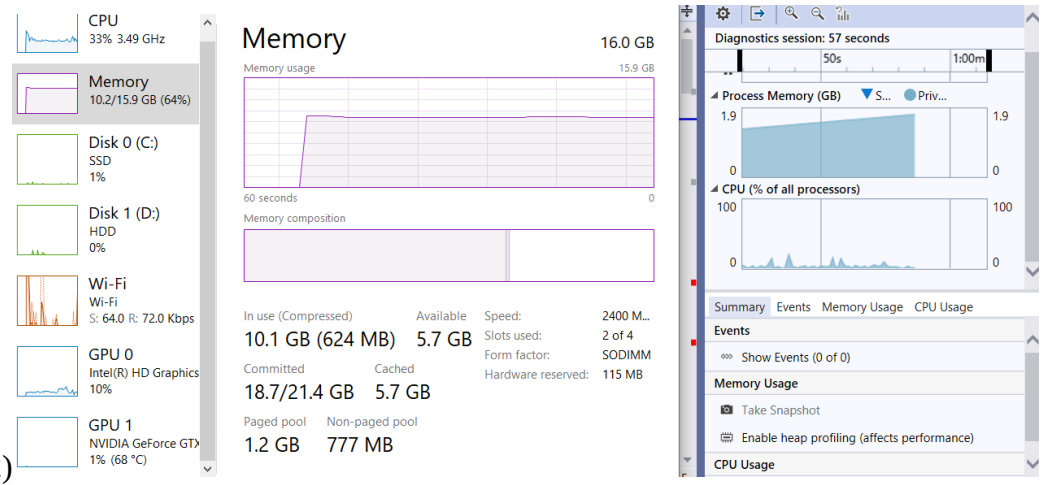


While testing:

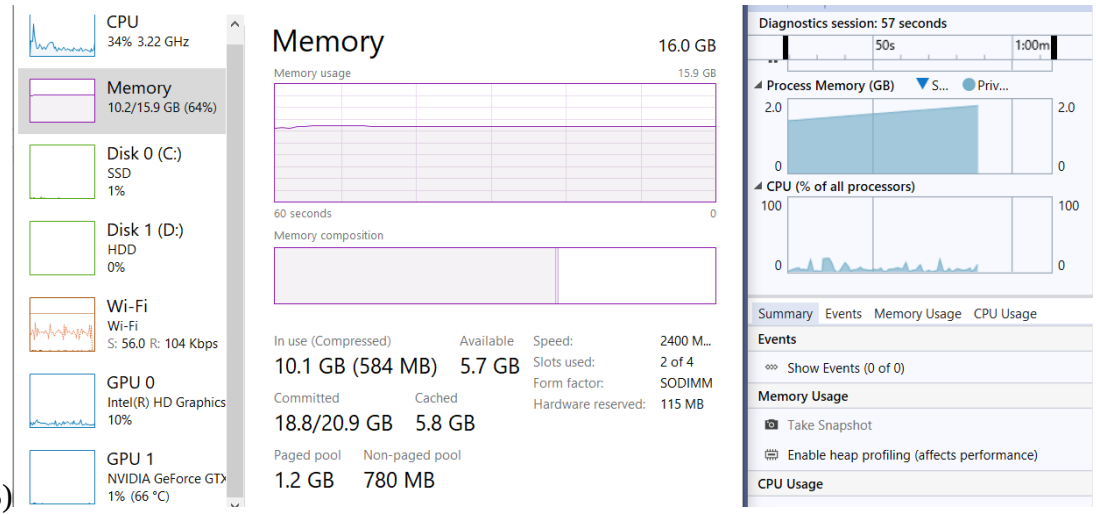
1)



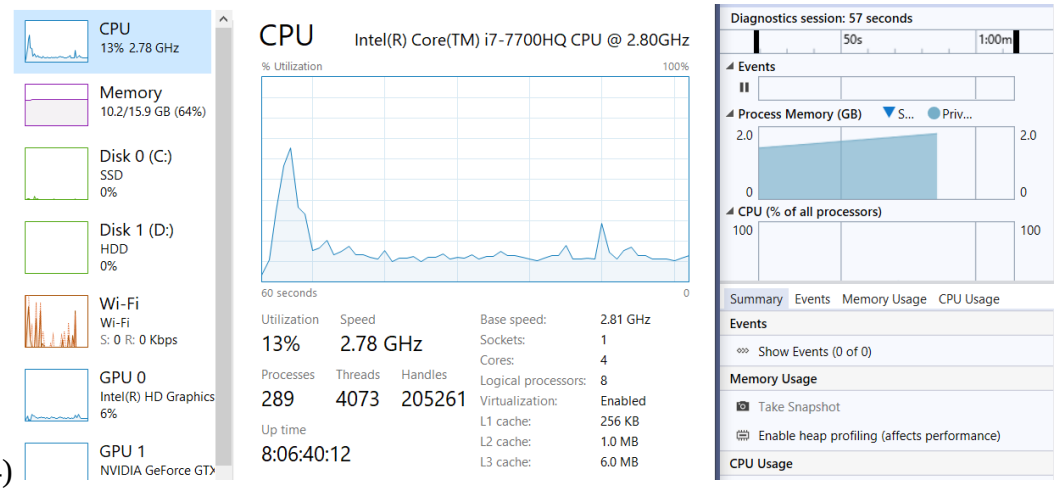
2)



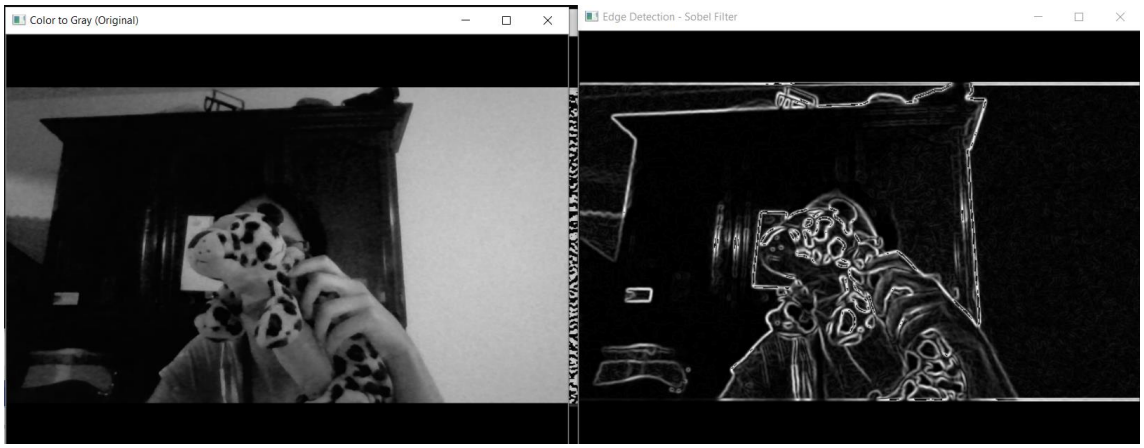
3)



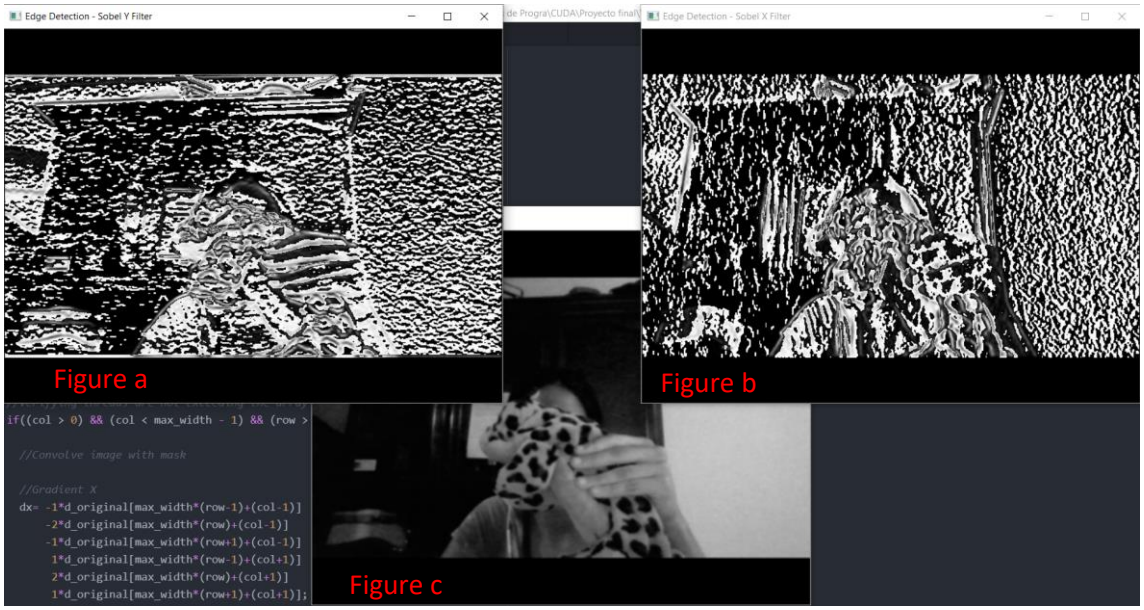
4)



Here is the filtering final result of the edge detection through Sobel's filtering convolutional kernel versus the original image in grayscale(left), for being able to compare visually the accurateness:



This how the separate gradients look like, one denotes more the horizontal edges (Figure a) and the other the vertical (Figure b) ones. The comparison is being going with the original image in grayscale (Figure c), so it is possible to see the input image.



Conclusions

In conclusion, it is interesting to see how parallelism can be used as a tool for computer vision as it is useful to make a lot of calculations at once through multi-threading using the GPU. I was impressed to see how the ophthalmology field has evolved to this technology capable of restoring sigh, even though the resolution is still kind of low comparing to the non-diseased retinas. I saw an opportunity for computer vision to take over and improve as much as possible the current 576-pixel resolution artificial sight.

This project is just a very small product of what can be added to the future artificial sight for the patients to use it as much as they can. I think it would be amazing to have much accurate filters done specially for these purposes, which are improving the information given by the photoreception of sensors like the bionic eyes.

I think I was able to learn a lot more how computer vision and CUDA can perform an efficient and hopeful job together. I liked how I was able to apply multi-threading in a different way. I really saw the industry of image processing improving with a framework

like CUDA that give us the opportunity to do many simple calculations, like with convolutional kernel technique, at once, saving the CPU for the hardest ones.

Even thought I had some trouble getting the set up done for CUDA to run with OpenCV on windows, after tackling it I learnt a lot of how to debug that kind of problems and how to approach them. That is why I can gladly say that I have a better understanding now of how the environment interacts with the program in general.

Setup instructions

For being able to use this program, it is important to follow the next steps in this order:

1. Install Visual Studio 2019 from https://visualstudio.microsoft.com/es/downloads/
2. Download OpenCV, the version a used was 4.2.0, so that would be the one I recommend. Here is the link to do it: https://opencv.org/releases/
3. Now, follow the set up instructions of the this video: https://www.youtube.com/watch?v=M-VHaLHC4XI&list=PL8EVpiUkkr15pqamnFH-9NYU2kSqtq0gk&index=4&t=500s&ab_channel=CoffeeBeforeArchCoffeeBeforeArch
4. Finally, you need to install CUDA, the version I used was 11.3: https://developer.nvidia.com/cuda-downloads

   Note: You can follow this video (https://www.youtube.com/watch?v=cuCWbztXk4Y&list=PL8EVpiUkkr15pqamnFH-9NYU2kSqtq0gk&index=5&t=138s) to set it up, from the minute 1:43 or this guide: NVIDIA CUDA Installation Guide for Microsoft Windows

5. Clone the repository or download the zip from GitHub: https://github.com/ViveliDuCh/ProgLanguages
6. Once you have it on your computer, create a new project on Visual Studio with the CUDA environment. You can support on this video: https://www.youtube.com/watch?v=cuCWbztXk4Y&list=PL8EVpiUkkr15pqamnFH-9NYU2kSqtq0gk&index=5&t=138s (From the minute 4:29)
7. Add the .c, .cu and .h stored in the folder "code" of the repository cloned/downloaded before and click "Local Windows Debugger" button (Or run it).

Sidenote: This steps are for running it on windows using visual studio.

References

1) Anthony, S. (2011, December 22). *A bionic prosthetic eye that speaks the language of your brain*. ExtremeTech. https://www.extremetech.com/extreme/110031-a-bionic-prosthetic-eye-that-speaks-the-language-of-your-brain

2) Anthony, S. (2012, July 17). *The laser-powered bionic eye that gives 576-pixel grayscale vision to the blind*. ExtremeTech. https://www.extremetech.com/extreme/132918-the-laser-powered-bionic-eye-that-gives-576-pixel-grayscale-vision-to-the-blind
3) C., "YouTube," YouTube, 04-Nov-2015. [Online]. Available: https://www.youtube.com/watch?v=uihBwtPIBxM. [Accessed: 29-May-2021].
4) Computerphile. (2015, February 15). *Digital Images - Computerphile* [Video]. YouTube. https://www.youtube.com/watch?v=06OHflWNCOE&ab_channel=Computerphile
5) Cooper, A. (2021, April 26). *Bio-Retina Implant Could Give Laser-Powered Sight to the Blind*. Popular Science. https://www.popsci.com/technology/article/2012-06/bio-retina-implant-could-give-sight-blind-laser-power/
6) Healthline. (2003). *Retinal Diseases and VISION 2020*. PubMed Central (PMC). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1705858/
7) OpenCV API: https://docs.opencv.org/master/de/d25/imgproc_color_conversions.html#color_convert_rgb_gray
8) R., "Image Processing: Edge Detection," Rice Web Services, n.d. [Online]. Available: https://www.owlnet.rice.edu/%7Eelec539/Projects97/morphjrks/moredge.html. [Accessed: 29-May-2021].
9) Retinal Diseases and VISION 2020. (2003). *Community eye health*, *16*(46), 19–20. Retrieved by: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1705858/
10) The Healthline Editorial Team. (2018, January 22). *Retina*. Healthline. https://www.healthline.com/human-body-maps/retina#1
11) Threads examples – Pedro Perez: https://github.com/Manchas2k4/advanced_programming
12) Torpy, J., Glass, T., & Glass, R. (2007). *Retinopathy*. JAMA Network. https://jamanetwork.com/journals/jama/fullarticle/208559#:~:text=Retinopathy%20means%20disease%20of%20the,the%20small%20retinal%20blood%20vessels.