CUDA : Image processing

I) Prezentare :

Proiectul presupune implementarea unei aplicaţi care sa aplice diferite filtre folosind GPU pentru procesare(CUDA), acest lucru facandu-se pentru am mari viteza de procesare. Pentru partea de GUI s-a folosit python pentru a putea sa fie portabil pe toate sistemele de operare.

II) Specificarea cerintelor:

a) Cerinţe funcţionale:

Aplicatia trebuie sa efectueza prelucarea pe imaginii, aplicand operaţie selectate de către utilizator, sa ofere posibilitatea sa selecteze imaginea pe care o doreşte de prelucrat, sa ofere posibilitatea sa salveze rezultatul, sa se poată întoarce la varianta originală a imagini. Pentru a oferi un timp foarte bun de răspuns aplicaţia se va folosi de GPU, deoarece operaţiile sunt simple si ele pot sa fie efectuate in paralel, in majoritate cazurile neexistand conflicte pentru date.

Operatile pe care trebui sa le poata efectua aplicaţia

1. File:
    a. Incarcare imagini din fişier, indiferent de format(vor exit cateva formate permise, specificate de catre aplicaţie).
    b. Salvarea imagine.
    c. Întoarcerea la varianta originală.
2. Operaţii de bază:
    a. Gray Scale
    b. Red Only
    c. Green Only
    d. Blue Only
    e. Red Remove

f. Green Remove
g. Blue Remove

3. Filtre pe culoare:
    a. Gray Scale
    b. Invers
    c. Contrat +128
    d. Contrast +60
4. Filtre Spatiale:
    a. Gaussian
    b. Vignette
    c. Smooth
    d. Sharpen
    e. Mean
    f. Emboss
    g. Gaussian, Smooth, Sharpern, Mean (GSSM)
5. Edge Detection:
    a. Sober
    b. Prewitt

b) **Cerinţe nefuncţionale:**

Utilzator va trebui sa aibe un calculator pe care sa gaseste o placa grafica de la nVidia, si aiba python

Cerintele de sistem:

1. Placa grafica nVidia >= GTX 940
2. Python >= 3.4
3. CUDA ToolKit
4. Pillow >= 7.0
5. tkinter
6. cupy (pentru varianta de CUDA ToolKit).

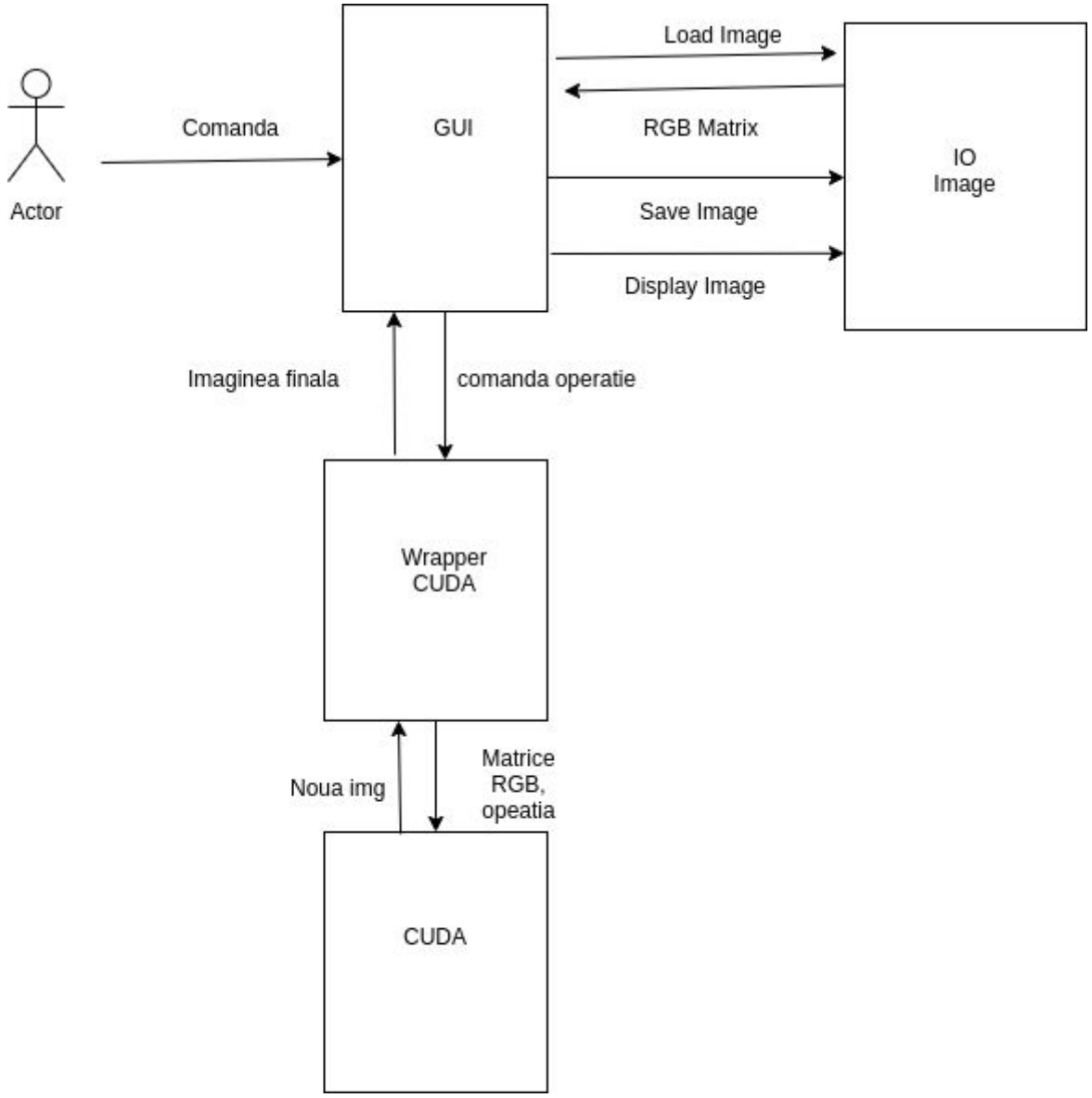
II) Arhitectura:

Principale module ale apicatie sunt:

1. Modul de GUI
2. Modul pentru încărcarea, salvare, afişarea imagini

3. Modul pentru functie de kernel care vor efectua operatile
4. Modul care sa lege codul de cuda -> python

GUI va trimite comenzile către celelalte module, astfel va fii “creierul” aplicaţie. Pentru a extrage matricea RGB din orice timp de imagine, se va folosi modul nr 2., el va mai avea ca si scop sa salvarea imagini, si de afiare a imagini curente. Pentru prelucrare pe CUDA o sa existe o serie de kernel function care vor face fiecare operaţiile specificate. O sa existe un modul care sa abstractize apelarea funcţiilor de kernel pentru prelucrare de imagine, si o sa actualize imaginea afişată.



III) Implementare:

Din motive de a facilita implementarea s-a luat decizia de a introduce un modul, class, care sa reţine variable globale necesare comunicării mai uşoare între module: global_var: GlobalVar, in rest exista cate un modul pentru fiecare modul descris mai sus. GUI : gui_app.py; CUDA: kernel_function.py; Wrapper CUDA : image_processing.py; IO iamge: io_image.py.

Pentru afişarea s-a folosit tkinker, s-a create un root, care este reţinut, si apoi s-a aduagat meniu pentru optiuni:

```
GlobalVar.window = Tk()
GlobalVar.window.title("CUDA : Image processing")
GlobalVar.window.geometry("400x500")
menu = Menu(GlobalVar.window)
```

```
menu.add_cascade(label='File', menu=file_menu)
menu.add_cascade(label='Simple Operation', menu=simple_operation)
menu.add_cascade(label='Filter', menu=filter_menu)
menu.add_cascade(label='Edge Detection', menu=edge_detection)
```

Pentru fiecare dintre aceste exista o acţiune asociată:

```
simple_operation = Menu(menu)
simple_operation.add_command(label='GrayScale', command=update_grayscale)
simple_operation.add_separator()
simple_operation.add_command(label='Red Only', command=update_red_only)
simple_operation.add_command(label='Green Only', command=update_green_only)
simple_operation.add_command(label='Blue Only', command=update_blue_only)
simple_operation.add_separator()
simple_operation.add_command(label='Inverse', command=update_inverse)
simple_operation.add_separator()
simple_operation.add_command(label='Red Remove', command=update_red_remove)
simple_operation.add_command(label='Green Remove', command=update_green_remove)
simple_operation.add_command(label='Blue Remove', command=update_blue_remove)
simple_operation.add_separator()
simple_operation.add_command(label='Contrast 128', command=contrast_p_128)
simple_operation.add_command(label='Contrast 60', command=contrast_60)
```

Pentru prelucrare de imagine s-a folosit biblioteca PIL, din care Image si ImageTk(pentru afişare), extragerea datelor s-a făcut cu funcţia:

```
def read_image(file_path):
    img = Image.open(file_path, "r")
    GlobalVar.width, GlobalVar.height = img.size
    GlobalVar.pixels = cp.array(img.getdata(), dtype=cp.int32)
    GlobalVar.original_img = deepcopy(GlobalVar.pixels)

    new_size = str(str(GlobalVar.width) + "x" + str(GlobalVar.height))
    GlobalVar.window.geometry(new_size)
    GlobalVar.window.resizable(width=True, height=True)

    if not GlobalVar.is_image:
        tk_img = ImageTk.PhotoImage(img)
        GlobalVar.panel_img = Label(GlobalVar.window, image=tk_img)
        GlobalVar.panel_img.pack(side="bottom", fill="both", expand="yes")
        GlobalVar.is_image = True
        GlobalVar.panel_img.mainloop()
    else:
        tk_img = ImageTk.PhotoImage(img)
        GlobalVar.panel_img.configure(image=tk_img)
        GlobalVar.panel_img.image = tk_img
```

Pentru a putea sa fie prelucrate de către kernel, datele au trebui sa fie convertite într-un format suportat de către cupy, acest lucru s-a făcut cu ajutorul motodei: cupy.array, care primeşte ca si input matricea de mixel returnata de catre Image.

Afişarea imagine se face cu ajutorul:

```
def display_new_image():
    # acum se face update la imagine
    new_data = cp.asnumpy(GlobalVar.pixels).reshape(GlobalVar.height * GlobalVar.width, 3)
    new_data = [tuple(x) for x in new_data]

    new_img = Image.new(mode="RGB", size=(GlobalVar.width, GlobalVar.height))
    new_img.putdata(new_data)
    tk_img = ImageTk.PhotoImage(new_img)

    GlobalVar.panel_img.configure(image=tk_img)
    GlobalVar.panel_img.image = tk_img
```

Pentru prelucrarea imagine se apelează wrapper care ştie cum sa apele modul de kernel asociat fiecărei operaţii:

Exemplu: Sober:

```
def update_sobel():
    update_image2(sobel, 'sobel')
```

```python
def update_image2(function, name):
    if not GlobalVar.is_image:
        print("Nu este nici o imagine incarcata")
        return

    grid = (GlobalVar.width // BLOCK_SIZE + 1, GlobalVar.height // BLOCK_SIZE + 1, 1)
    input_pixel = deepcopy(GlobalVar.pixels)
    function(grid, (BLOCK_SIZE, BLOCK_SIZE, 1), (input_pixel, GlobalVar.pixels, GlobalVar.width, GlobalVar.height))
    GlobalVar.name_operation = name

    display_new_image()
```

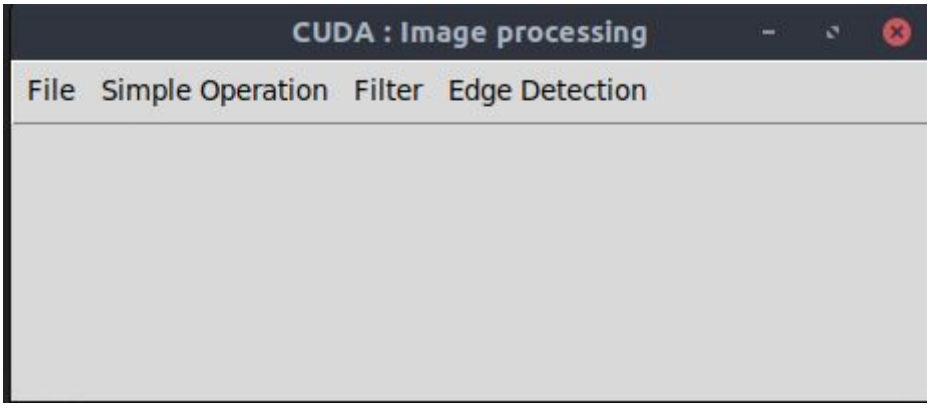
```
sobel = cp.RawKernel(r'''
                        extern "C"
                            __global__ void sobel(int3 * input, int3* output, int width, int height){
                                int row = blockIdx.y * blockDim.y + threadIdx.y;
                                int col = blockIdx.x * blockDim.x + threadIdx.x;
                                //const int Gx[3][3] = {{-1, 0, 1}, {-1, 0, 1}, {-1, 0, 1}};
                                //const int Gy[3][3] = {{-1, -1, -1}, {0, 0, 0}, {1, 1, 1}};

                                const int Gx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
                                const int Gy[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};

                                if (col < width && row < height) {
                                    float sum_x = 0;
                                    float sum_y = 0;

                                    for(int i = -1; i <= 1; i++) {
                                        for(int j = -1; j <= 1;j++) {
                                            if(col + j >= 0 && col + j < width && row + i >= 0 && row + i < height) {
                                                int idx = (row + i) * width + col + j;
                                                int gray = 0.2125 * input[idx].x + 0.7154 * input[idx].y + 0.0721 * input[idx].z;

                                                sum_x += Gx[i + 1][j + 1] * gray;
                                                sum_y += Gy[i + 1][j + 1] * gray;
                                            }
                                        }
                                    }

                                    int idx = row * width + col;
                                    float sum = sqrt(sum_x * sum_x + sum_y * sum_yscreenshot);
                                    if (sum > 255){
                                        sum = 255;
                                    }

                                    output[idx].x = sum;
                                    output[idx].y = sum;
                                    output[idx].z = sum;
                                }
                            }

                ''', 'sobel')
```

Salvare imagini se face folosind funcția:

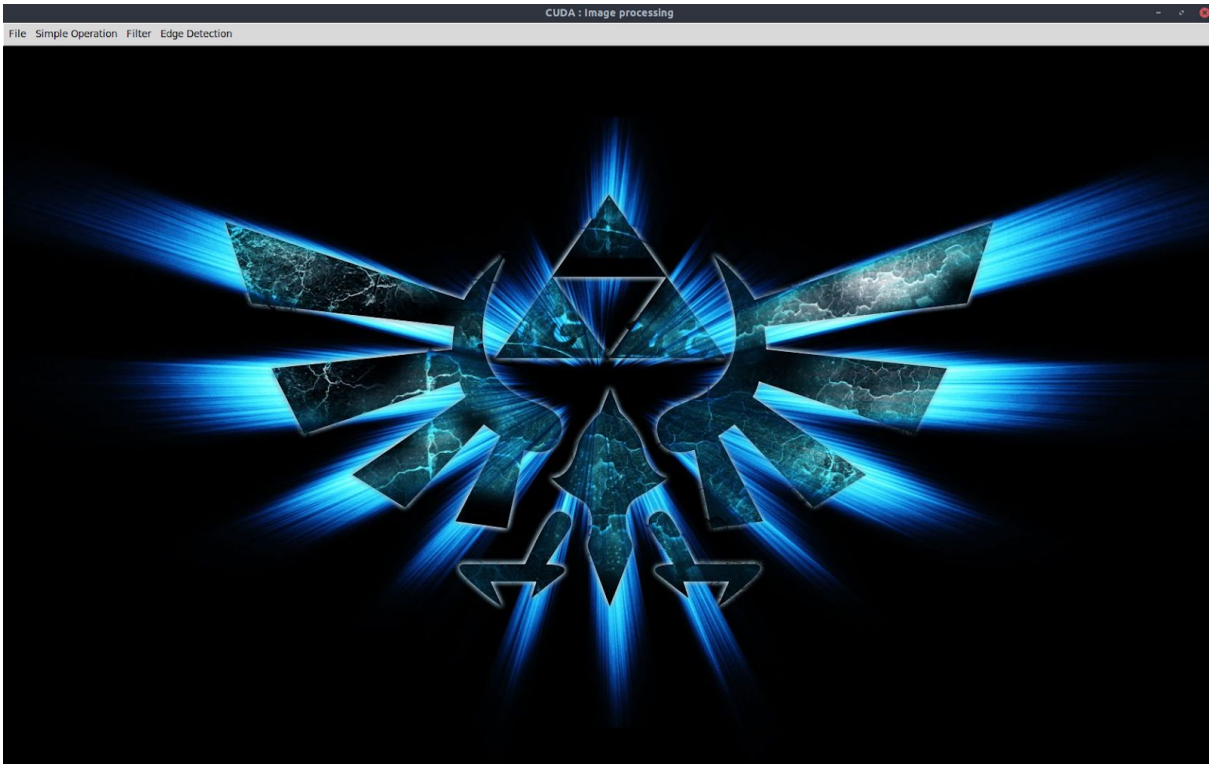
```python
def save_image(name: str):
    new_data = cp.asnumpy(GlobalVar.pixels).reshape(GlobalVar.height * GlobalVar.width, 3)
    new_data = [tuple(x) for x in new_data]

    img = Image.new(mode="RGB", size=(GlobalVar.width, GlobalVar.height))
    img.putdata(new_data)
    img.save(name)
```

IV) Rezultate:

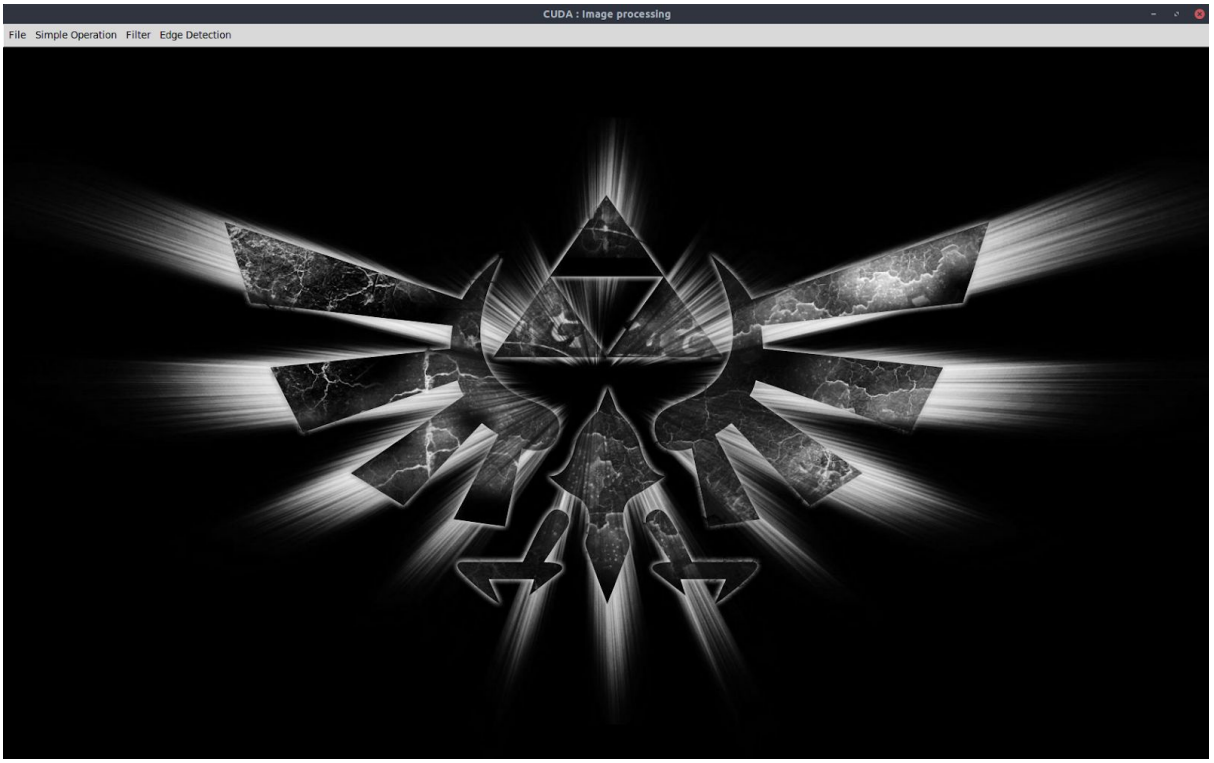
Aplicație cand nu este incarcata o imagine:



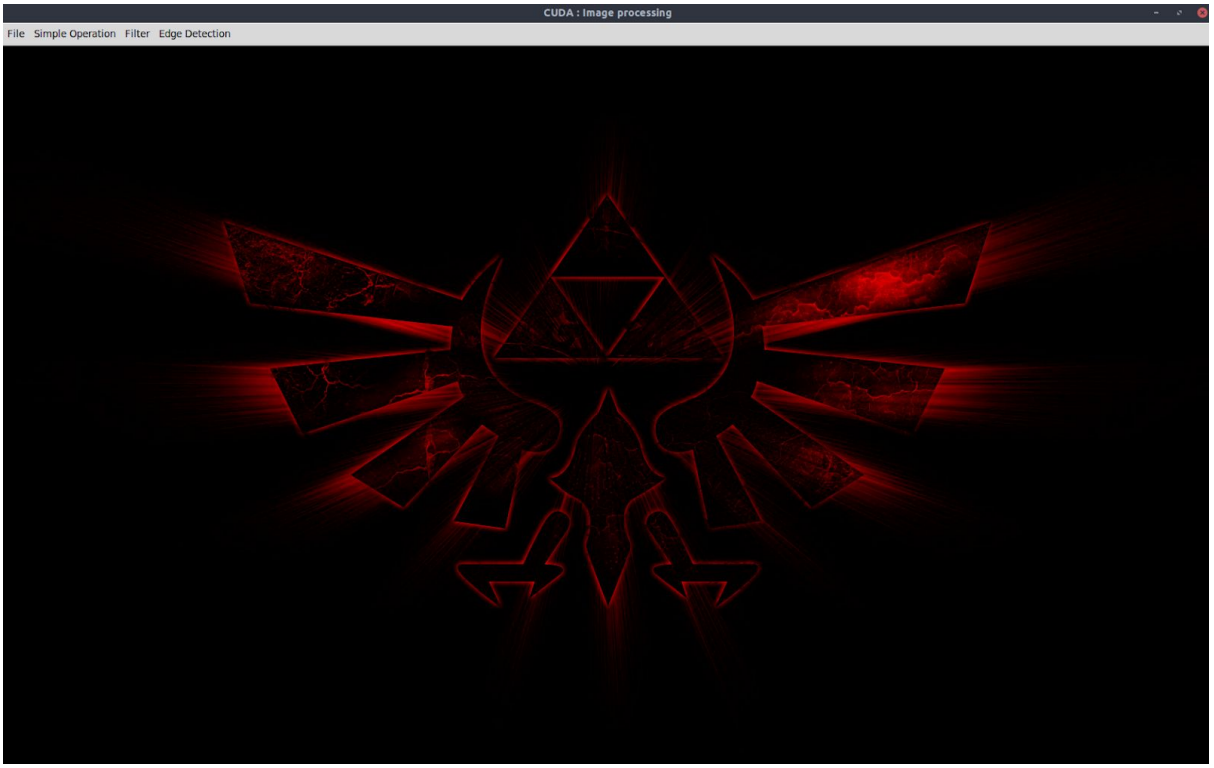
Imagine orginal:



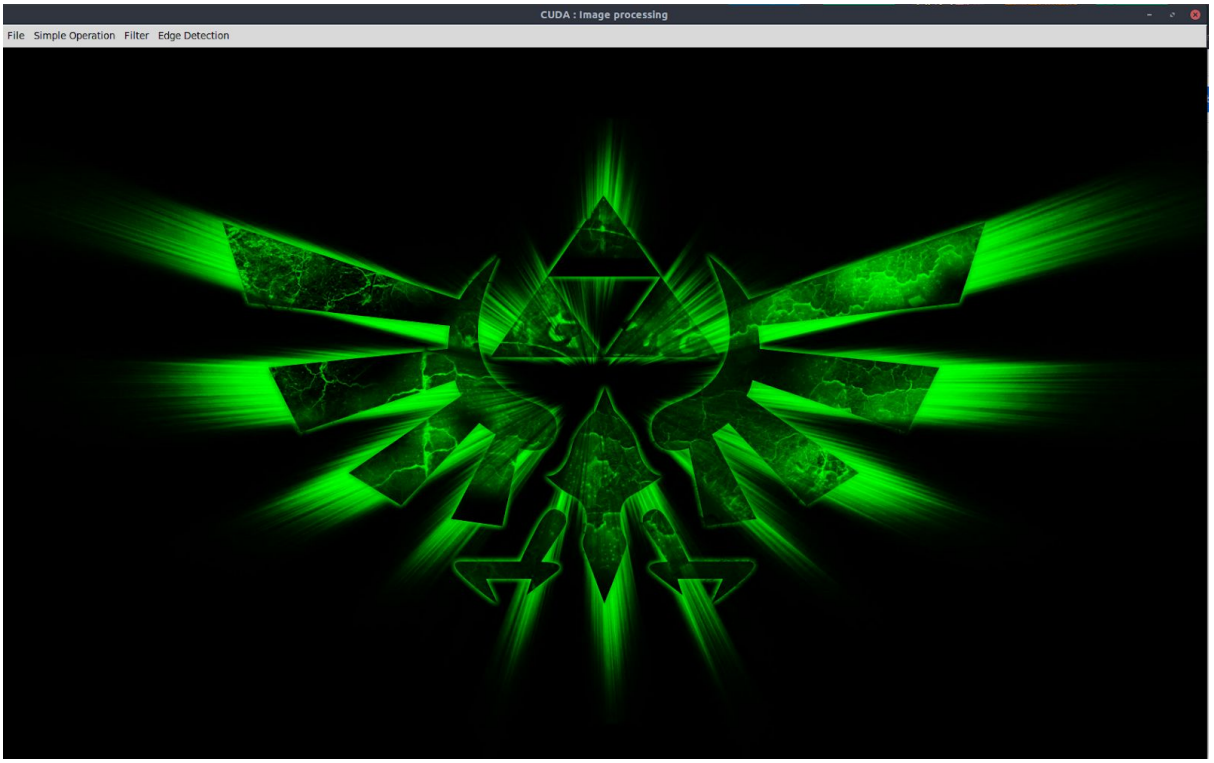
Dupa aplicarea Gray Scale:



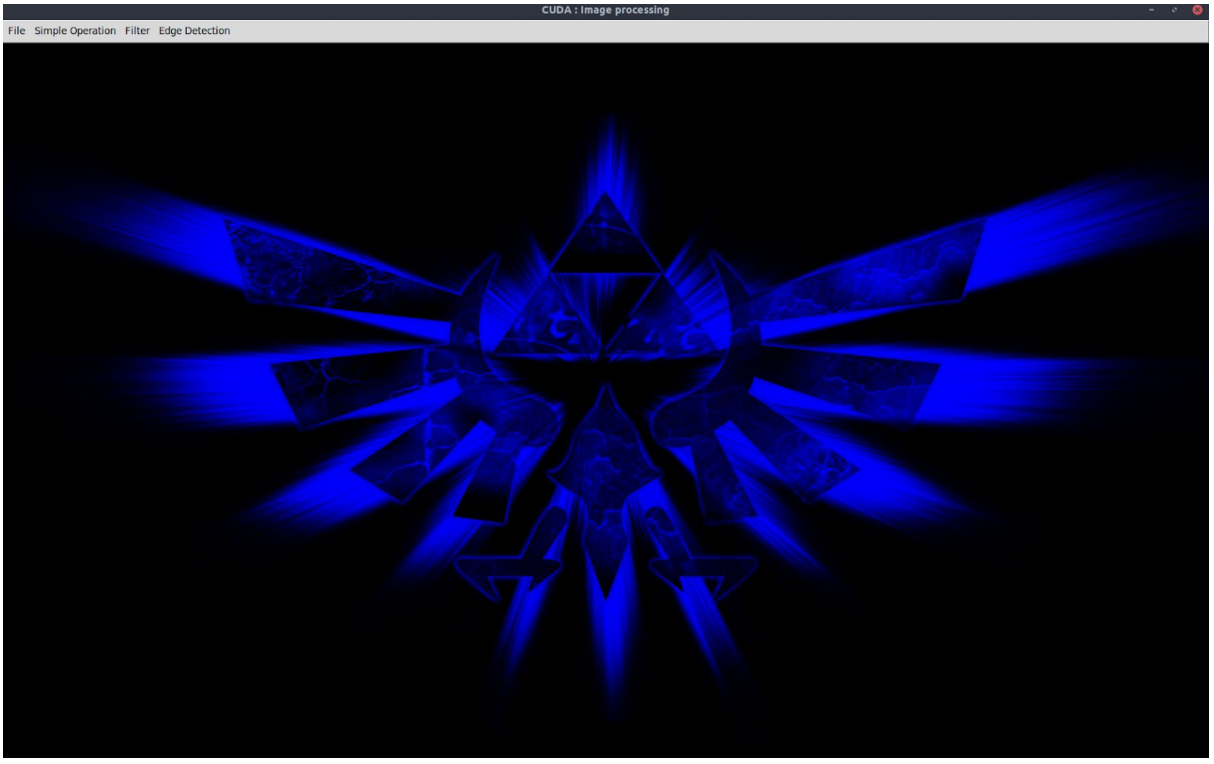
Red Only:



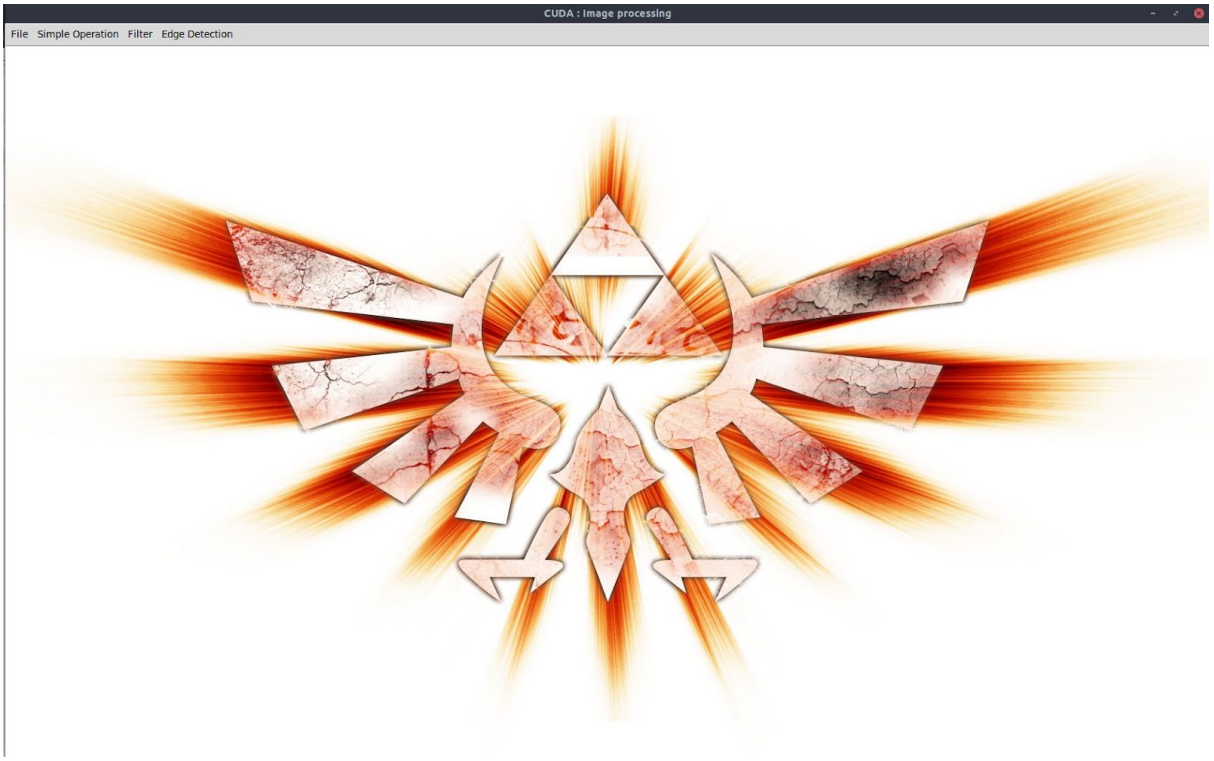
Green Only:



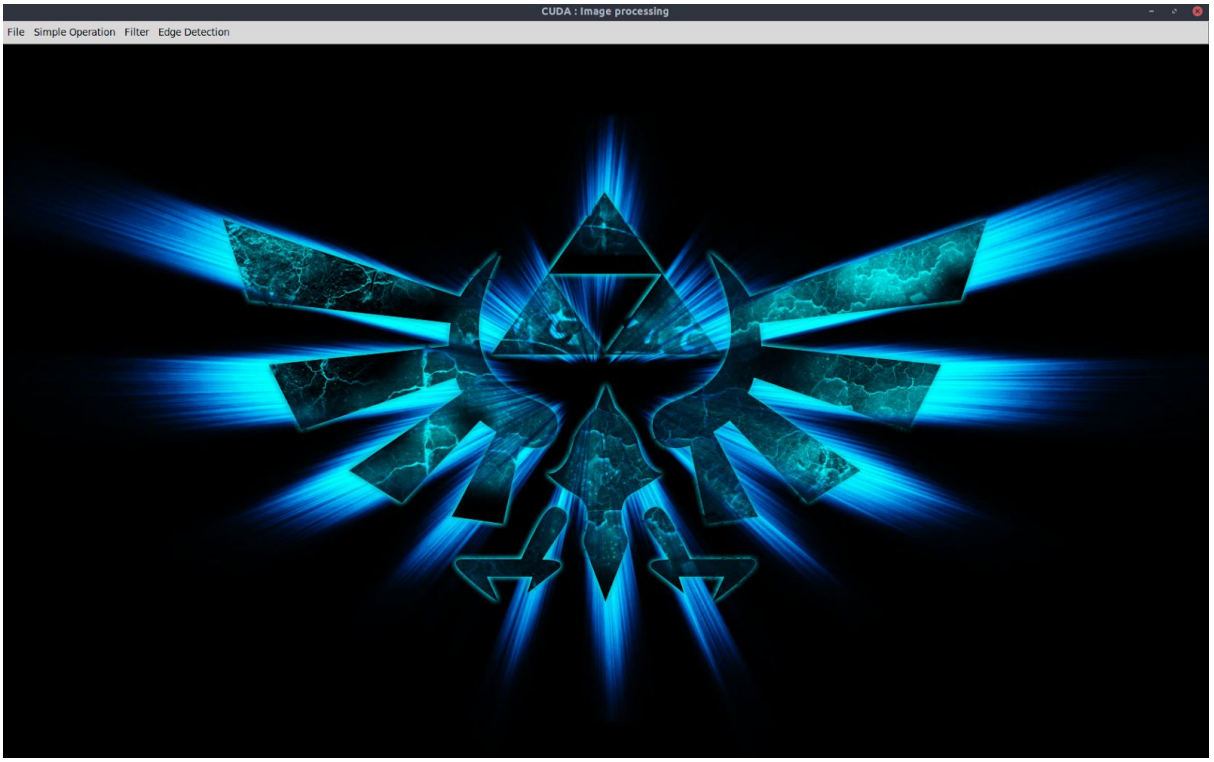
Blue Only:



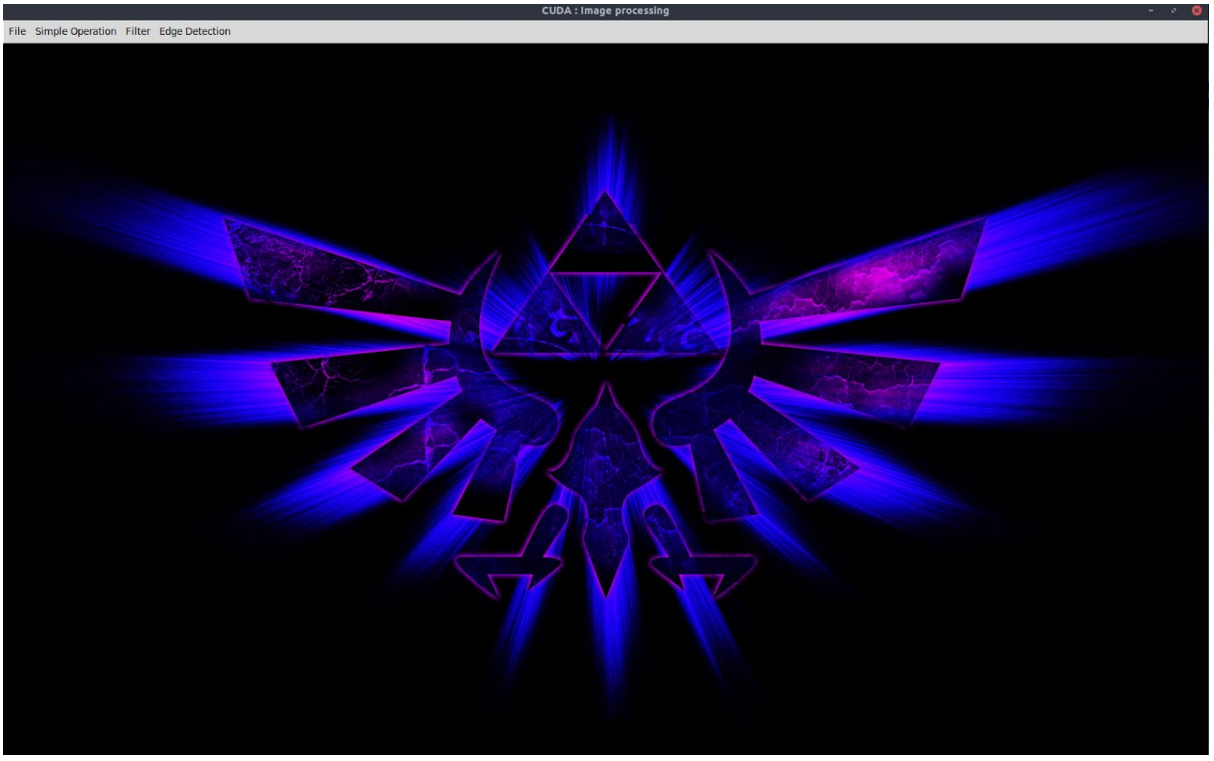
Inverse:



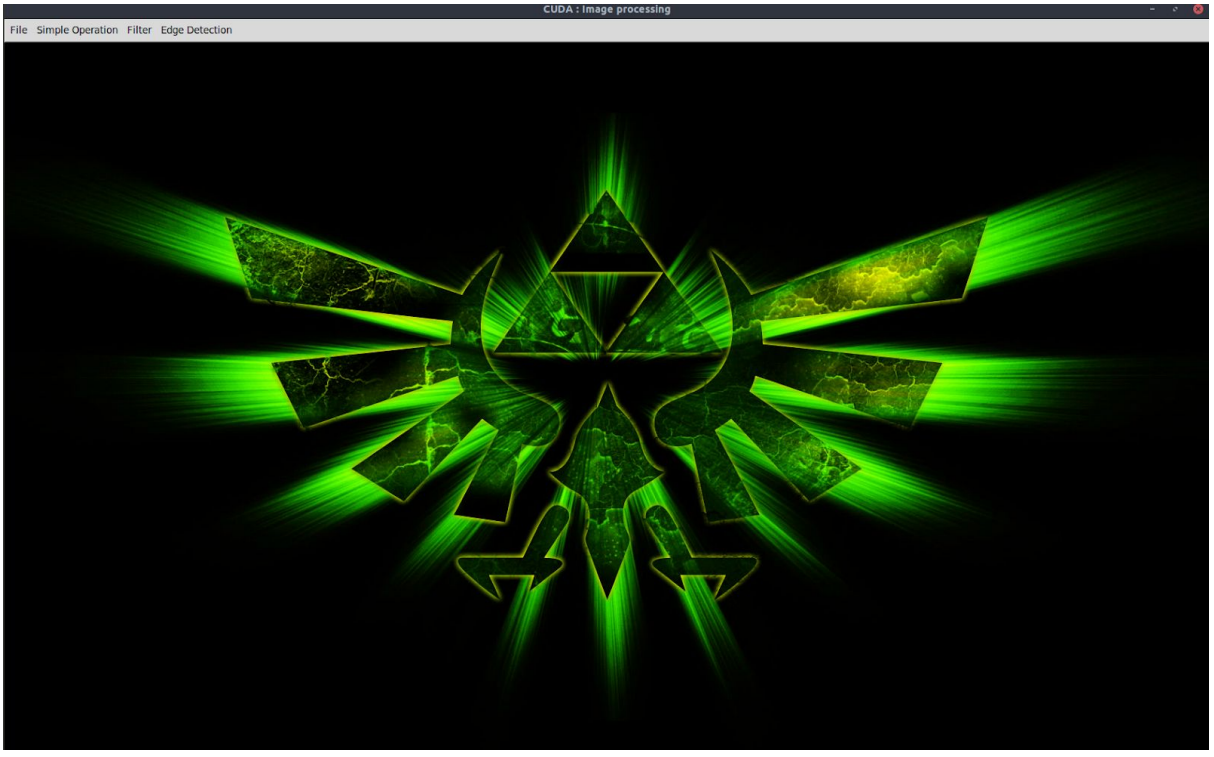
Red remove:



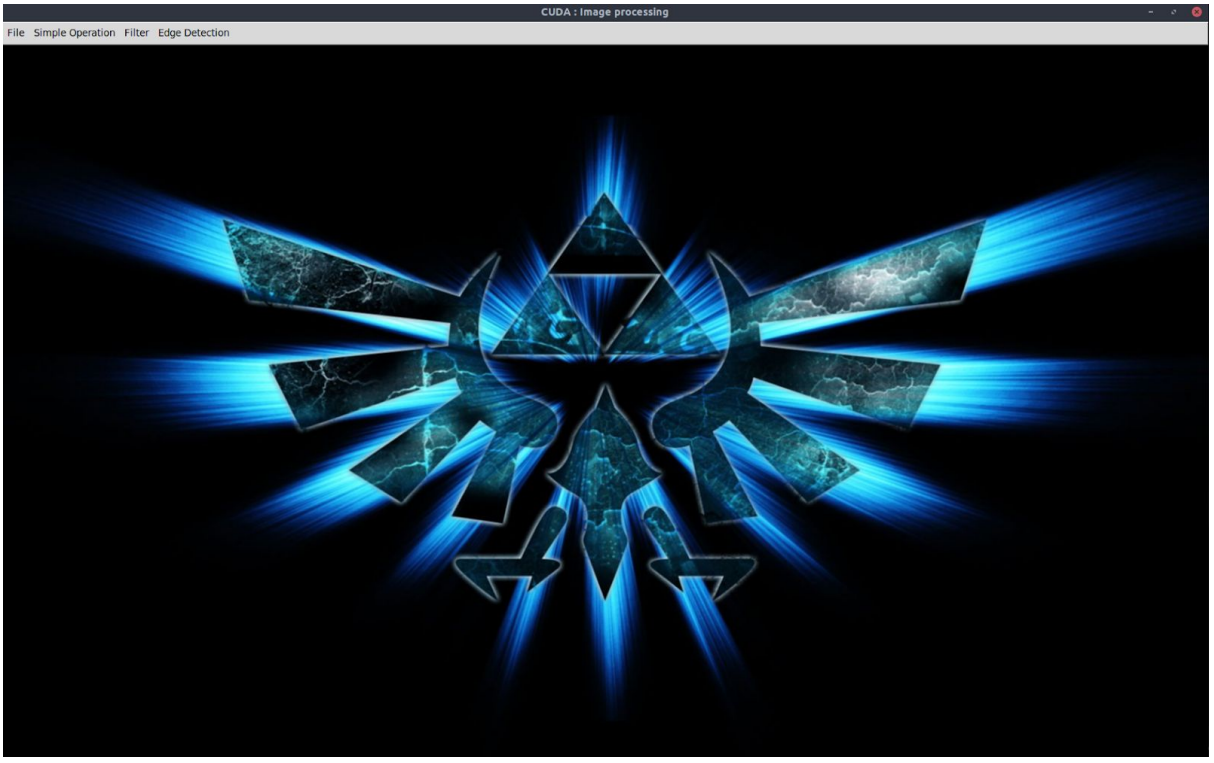
Green remove:



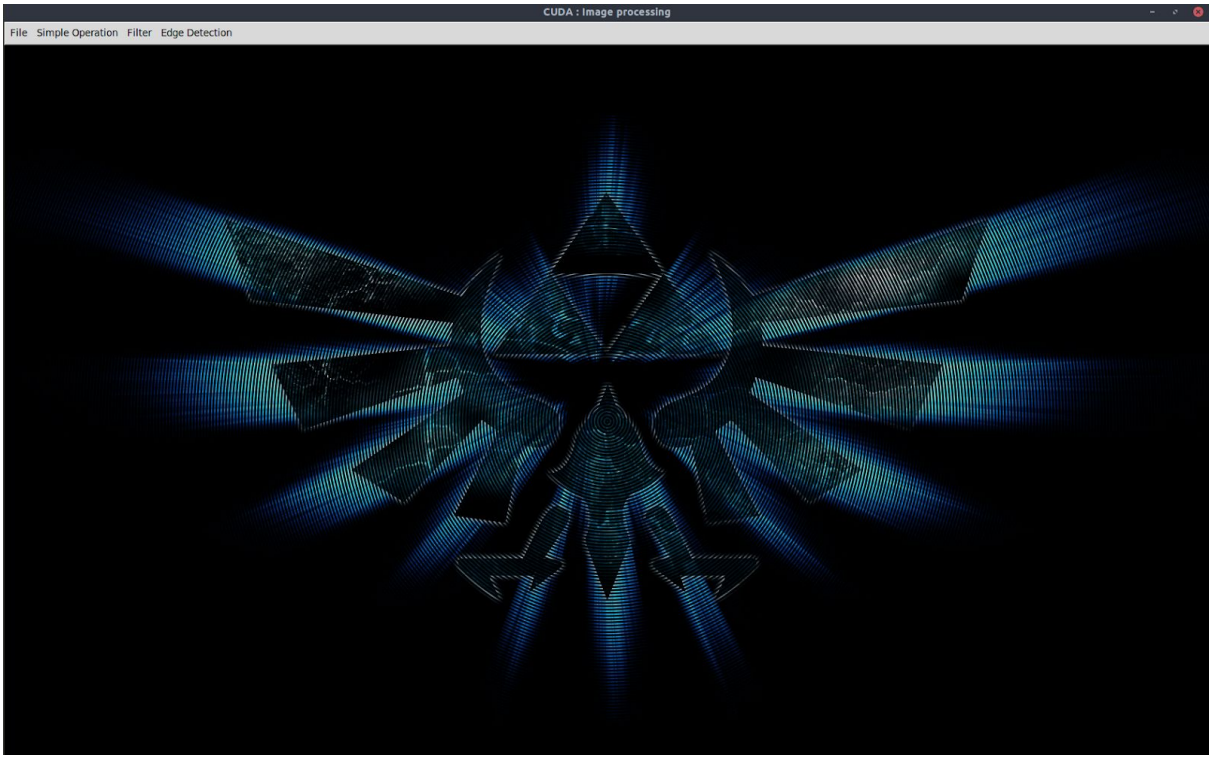
Blue remove:



Guassian:

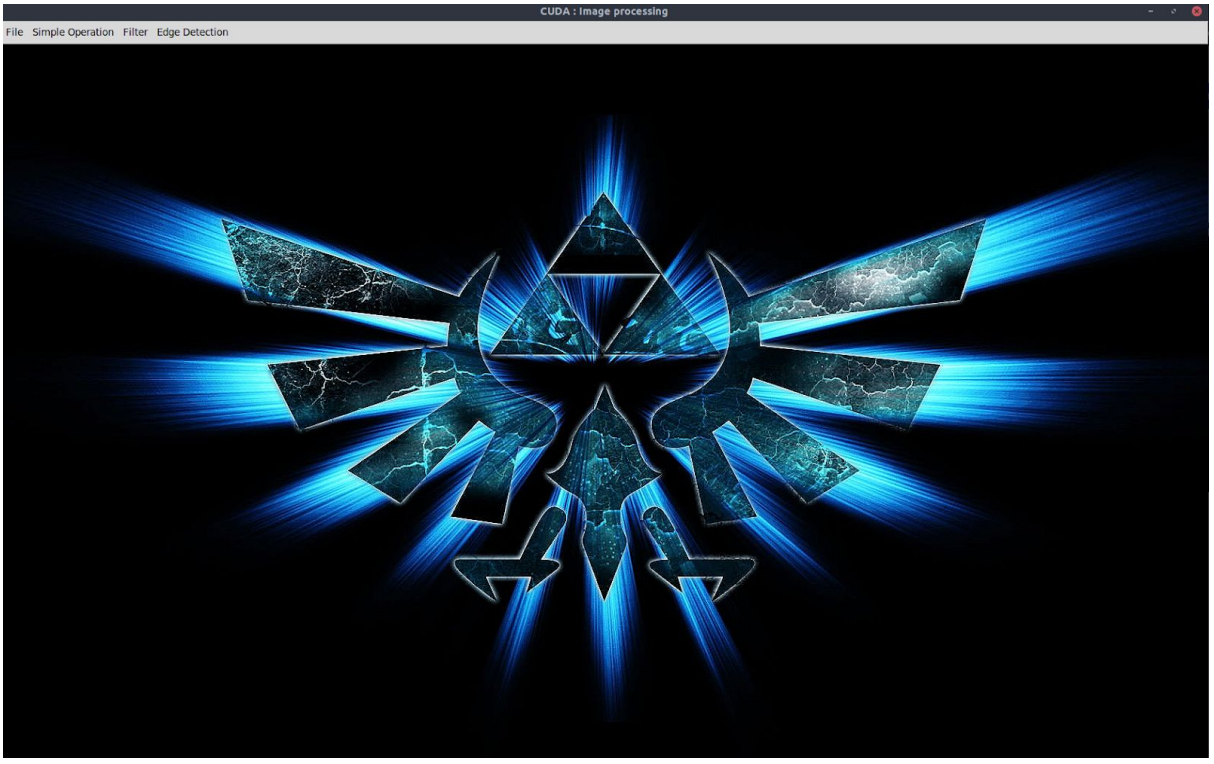


Vignette:

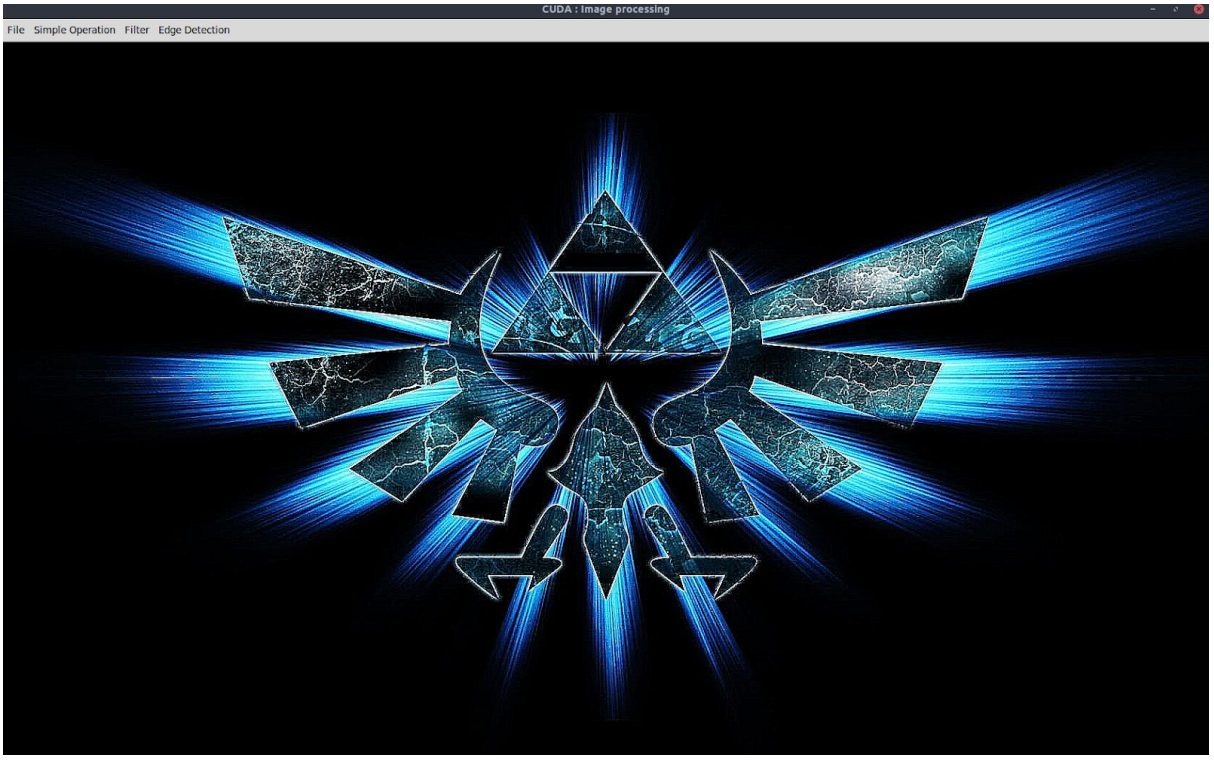


Smooth (x 10):

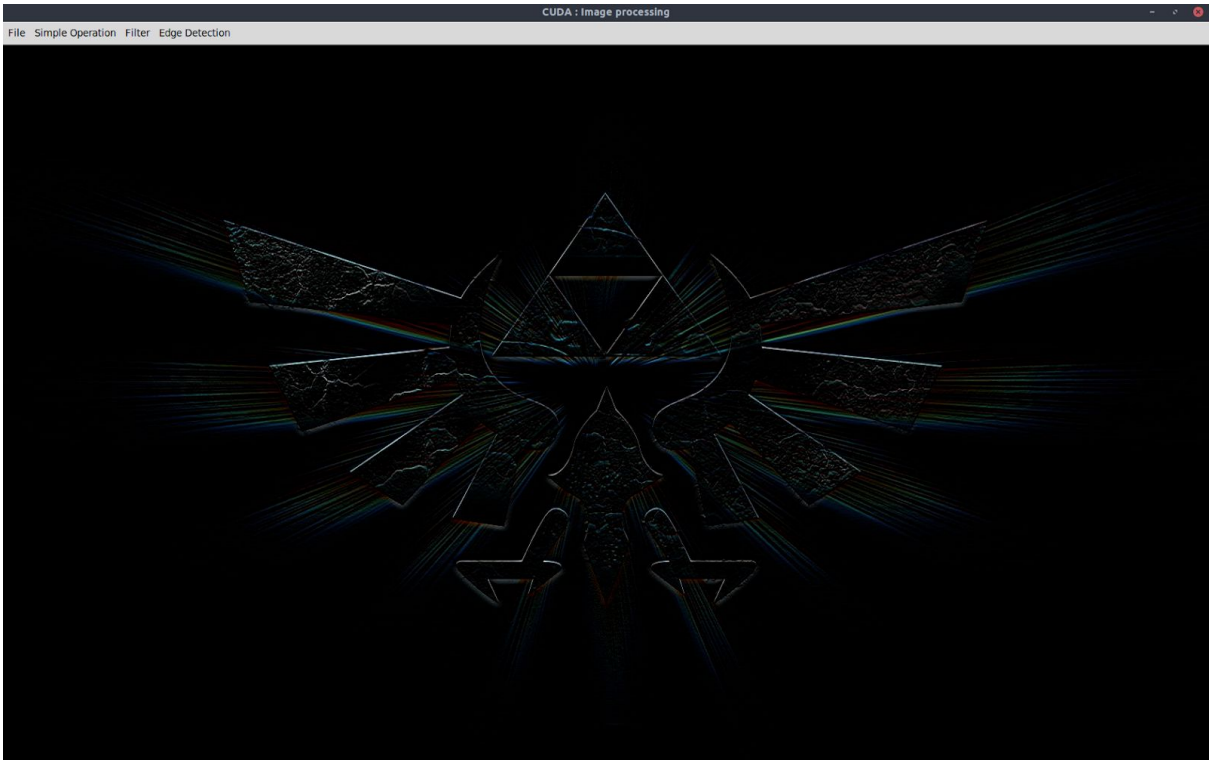
Sharpen:



Mean:



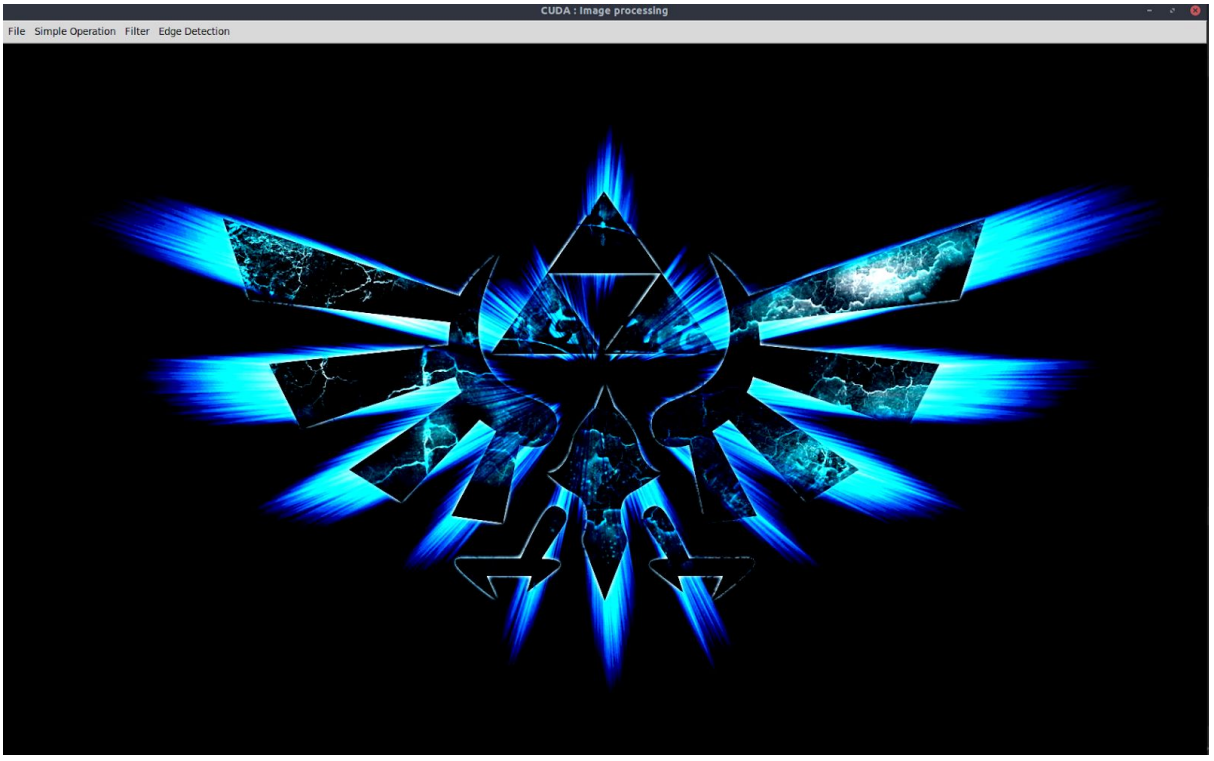
Embross:



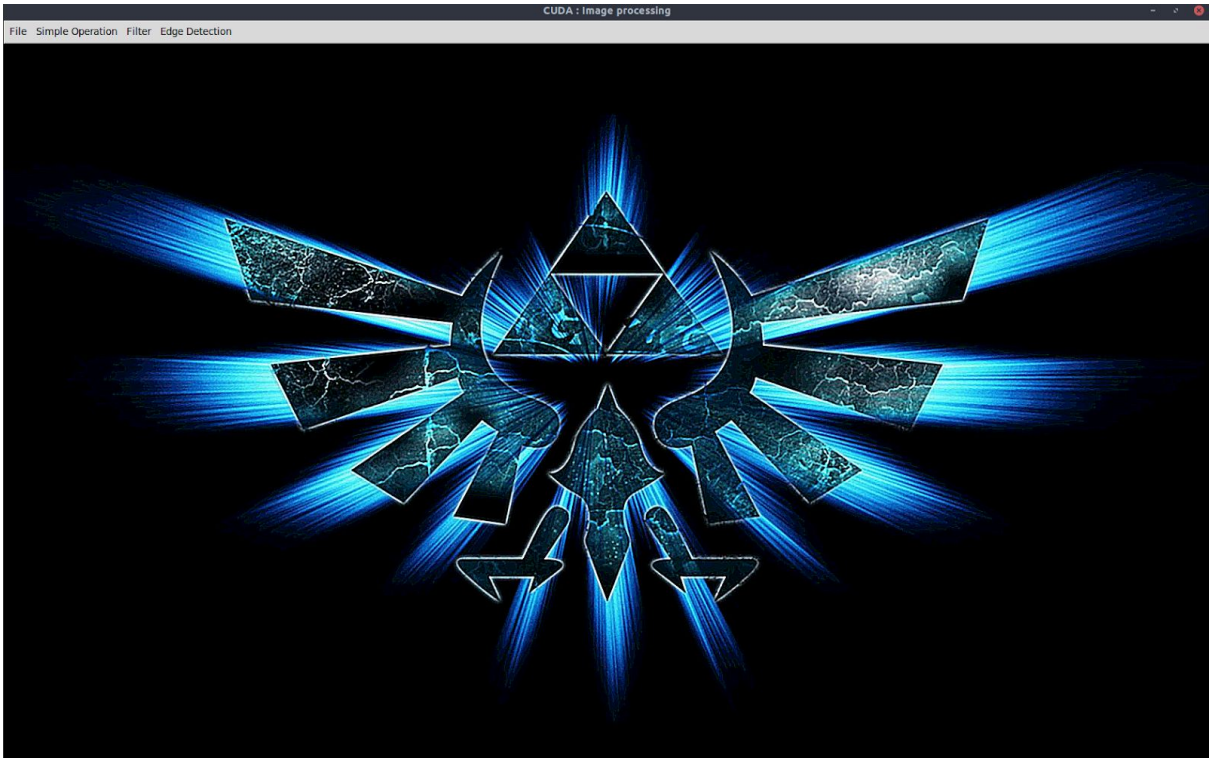
Sobel:

Prewitt:

Contrast 128:



GSSM:



Referinte:

https://www.geeksforgeeks.org/python-gui-tkinter/
https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html
https://docs-cupy.chainer.org/en/stable/index.html
https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/
https://en.wikipedia.org/wiki/Sobel_operator
https://en.wikipedia.org/wiki/Prewitt_operator
https://stackoverflow.com/questions/2976274/adjust-bitmap-image-brightness-contrast-using-c
https://github.com/MihaiAlexandru1606/APD---Image-processing-with-MPI
https://docs-cupy.chainer.org/en/v1.0.0/tutorial/kernel.html
https://github.com/cupy/cupy/
https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_rgb_to_gray.html
https://innat.github.io/innat.github.io/Image-Processing-in-Python-Part-2/