1. ABSTRACT

A benchmark is the act of running a process or a computer program, a set of programs, or other operations, in order to assess the relative performance of a system. In our project “OS Benchmarking”, we utilise various python libraries (built- in and third party) to test the performance of the system. Our test focus on multiprocessing, image processing and large calculations. This benchmark is useful for measuring the performance of the system components, as well as the operating system the system is running on. With this, we can get an idea of how different operating systems perform and how well they are optimised to work with the given hardware. The benchmark program also gives the user options of which tests to run and, in some cases, the specification of the test. The results are then shown on a browser window which also show the tests run and their results.

2. INTRODUCTION

2.1 Problem Statement

There are many applications available in the internet to benchmark a system. Some of the most popular benchmarking suites are Geekbench and UserBenchMark. However, there are problems with these applications. Geekbench is a synthetic benchmark and those results do not indicate real world performance of the system. UserBenchMark is known to be biased towards particular systems featuring Intel components and the scores rely mostly on single core performance. We plan to create a case study that compares several operating systems using simple tasks, one of which also tests the capabilities of the system’s multi core performance.

2.2 Literature Survey

2.2.1. How to Build a Benchmark

This paper provides an insight into the benchmark development criteria as employed be the SPEC and TPC consortia. It provides a definition for benchmarks and rating tools, differentiating between benchmarks for competitive purposes and rating tools for research purposes, regulatory programs, or as part of a system improvement and development approach. It explains the differences between the three major types of benchmarks: specification-based, kit based, and hybrid. Finally, it describes the major quality criteria of industrial benchmarks: relevancy, repeatability, fairness, verifiability, and usability, including examples on how the criteria are ensured in standardized benchmarks.

2.2.2. Python Based Image Processing

This paper gives details on how to perform the Image processing tasks in Python programming language, so that it becomes easy for all to understand the concepts related to it. This paper also provides the use of Python Image

Library (PIL), using which we can prominently develop the Python based image processing software and can be useful for number of applications like remote sensing, agriculture, space centre, satellites, medical and health sciences, etc. Thus, it can be concluded that Python and Image processing proves to be the better combination for learning, developing and understanding the capabilities provided in it. We also now know that image processing tasks are an effective way to stress the system and compute its performance.

3. DESCRIPTION OF THE PROJECT

3.1 Brief Introduction

This project uses various third-party python libraries such as:

i. streamlit
ii. NumPy
iii. OpenCV
iv. PIL (Python Imaging Library)
v. glob

And some included python libraries like:

i. multiprocessing
ii. time
iii. math

The streamlit library is the library that allows for a graphical user interface. Running the program using streamlit lets us give the user a representation of the tests that are being run and let us build a web application to complement the program where the tests are run. This library allows developers to create web apps with little experience in front end.

The NumPy library allows for specialised calculations that would otherwise be very difficult to develop normally for a python program. Using this library, we developed tests that perform matrix multiplication on very large datasets to stress the system.

OpenCV and PIL are the two libraries we use to perform various image processing tasks. With these libraries, we perform operations like edge detection, blending and warping, erosion and dilation and Gaussian blur.

We use glob to access the resources (like images) that are present in the folder specified by the path in the program.

The library multiprocessing is required to enable the python program to operate on multiple logical cores simultaneously. This is important for testing the multitasking capabilities of the processor and the operating system. We measure the time taken for the tests to complete using the time system library. This library lets the program log the start time and the end time of the test to calculate the time taken for the respective test. Lastly, the math library allows us to develop slightly complex calculations without having to go into detail on the specifics of the operations.

3.2 Requirement Analysis

3.2.1 Software Requirements

Operating System:

i. 64-bit operating system
ii. 64-bit version of Python 3.8.5
iii. Microsoft Visual C++ Build Tools 2015 (for Windows)
iv. Up to date web browser that supports HTML5

Python libraries:

i. streamlit which includes:
   a. NumPy
   b. PIL
   c. panda
ii. glob
iii. OpenCV

3.2.2 Hardware Requirements

i. 64-bit processor
ii. 4 core CPU or higher
iii. Minimum 2GHz clock speed
iv. 4GB RAM
v. 2GB free disk space

3.3 Detailed Designs

3.3.1 System Design

The benchmark program consists of six tests. These are:

i. Blending and Warping
ii. Edge Detection
iii. Gaussian Blur
iv. Erosion and Dilation
v. Matrix Multiplication
vi. Factorial Calculation

In the blend and warp test, the program reads two preloaded images. Then the image is blended and stored as a temporary file. This file is read again in grayscale and then based on the user's input; it can perform either:

i. Vertical wave
ii. Horizontal wave
iii. Concave effect
iv. Both vertical and horizontal wave

After either of the operations have been completed, the program then separately stores the input and output files. The first image, the second image and the final image are displayed on streamlit.

The output for the test is as follows:

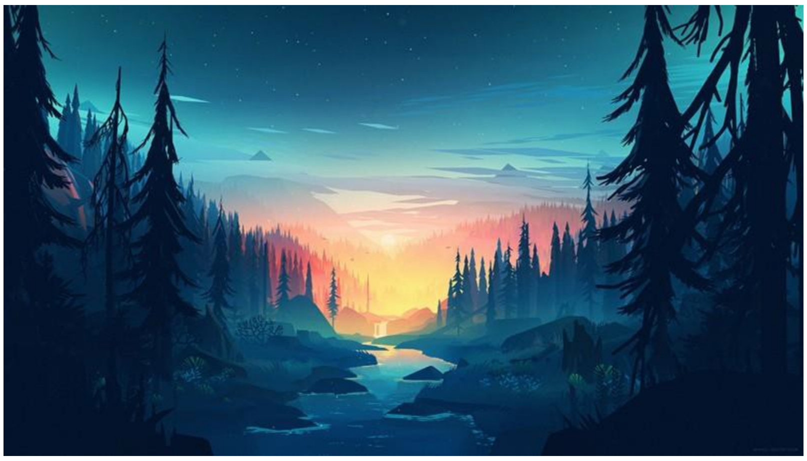


Fig 3.3.1 – Original image 1

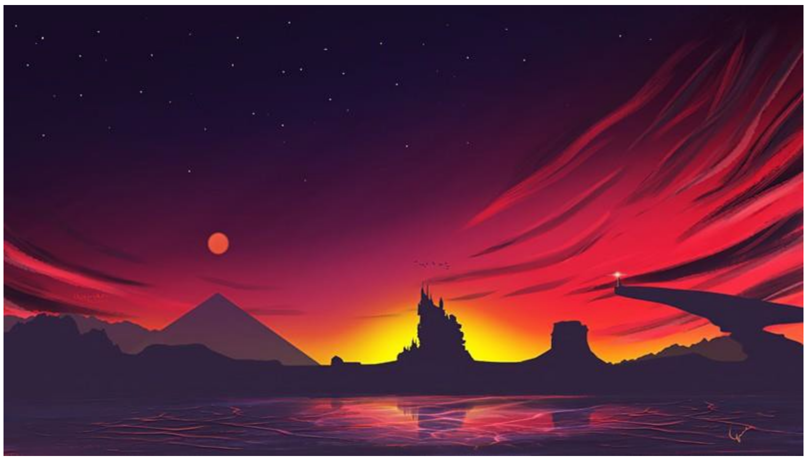


Fig 3.3.2 – Original image 2

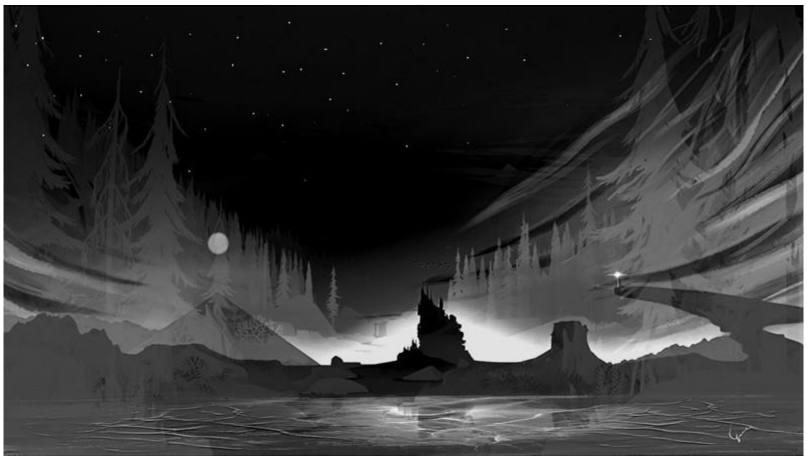


Fig 3.3.3 – After blending and converting to grayscale

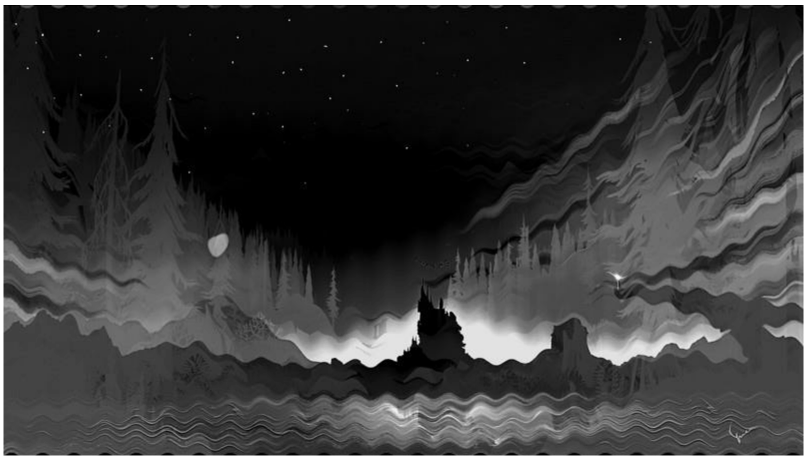


Fig 3.3.4 – After horizontal warping

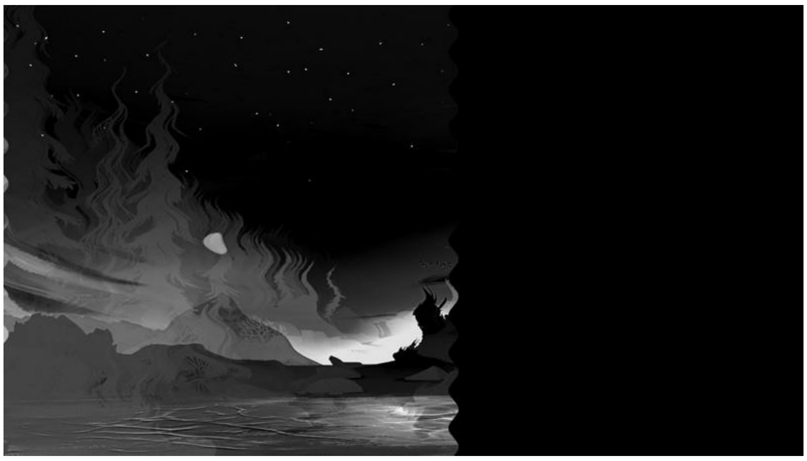


Fig 3.3.5 – After vertical warping

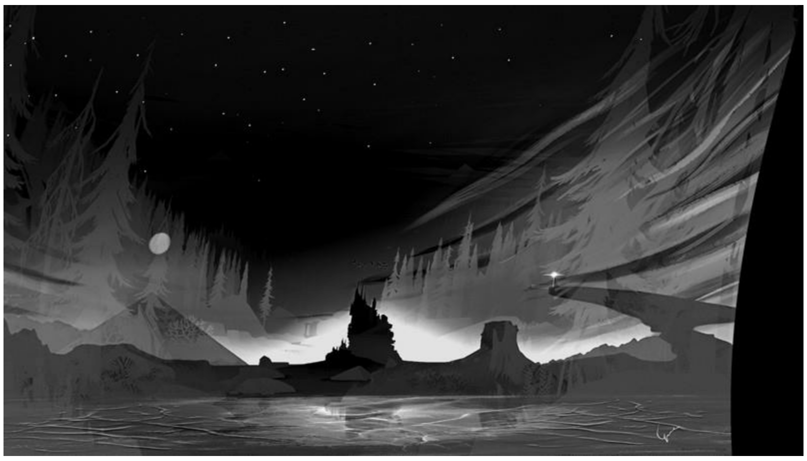


Fig 3.3.6 – After applying concave effect

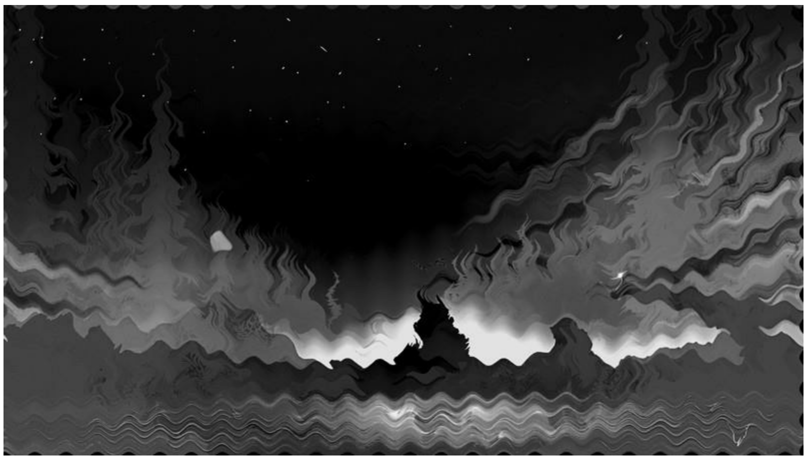


Fig 3.3.7 – After applying vertical and horizontal warping

In the edge detection test, the program loads up a set of images that are part of a separate folder. The program then uses algorithms to detect edges in the image. The time taken to run the test is measured and is displayed on streamlit.

A sample output for the test is as follows:

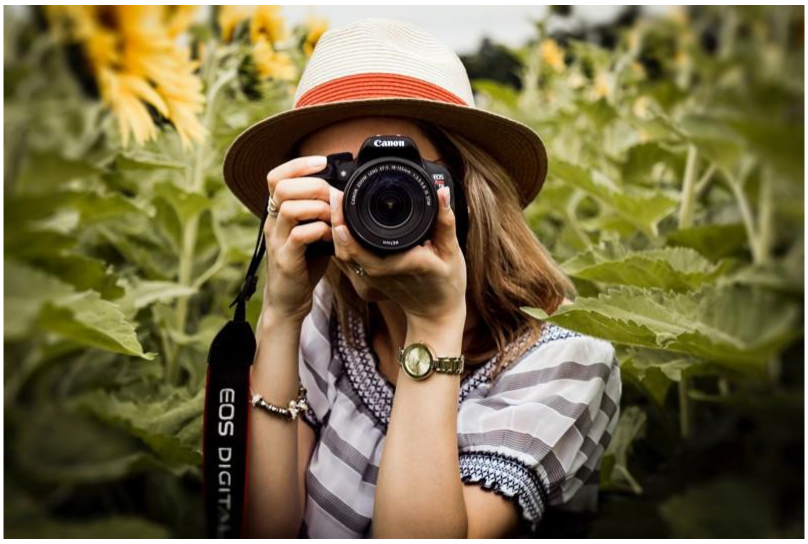


Figure 3.3.8 – Original image

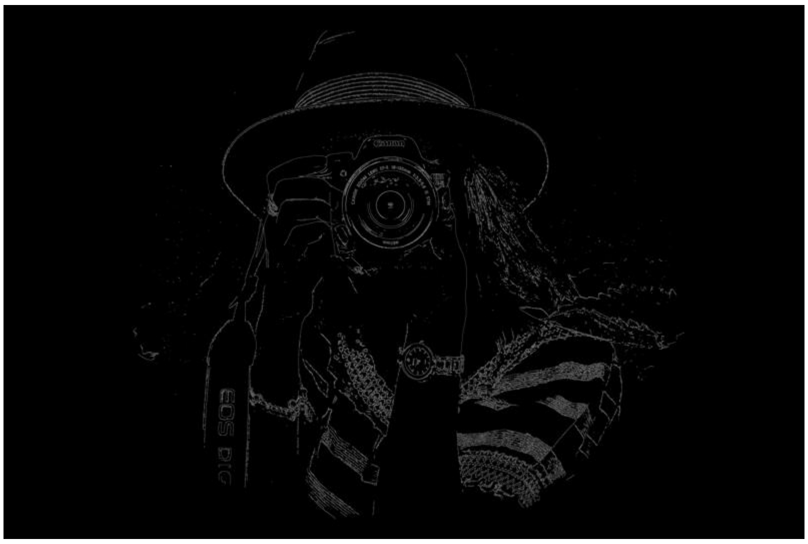


Figure 3.3.9 – After performing edge detection

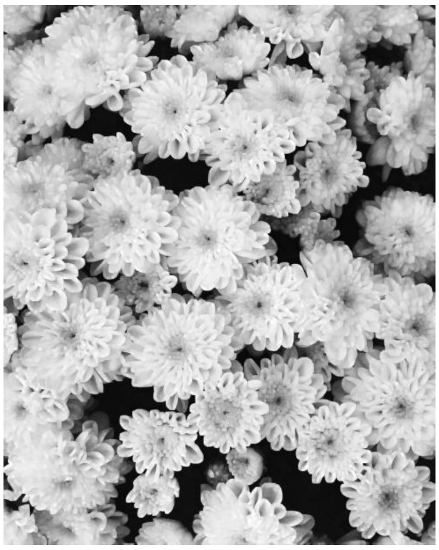


Fig 3.3.14 – Result after dilation

Next the program performs a set of mathematical tests to stress the CPU. These tests are the matrix multiplication test and the factorial test.

The multiplication test uses Python's multiprocessing library as well as NumPy to perform mathematical operations that stress all CPU cores at once to determine the multitasking capabilities of it. The time taken to run this benchmark is displayed on streamlit.

Finally, the factorial test calculates the factorial of large numbers. This test also creates large arrays and uses them to calculate the factorials. The time taken to run this benchmark is measured and displayed on streamlit.

In the case of the default (standard) benchmark, the program adds up the time taken for all the results and displays the final output. In the case of the custom benchmark, the time taken to run each test when called is displayed. Tests can be run multiple times and in any order.

3. 4 Data flow design

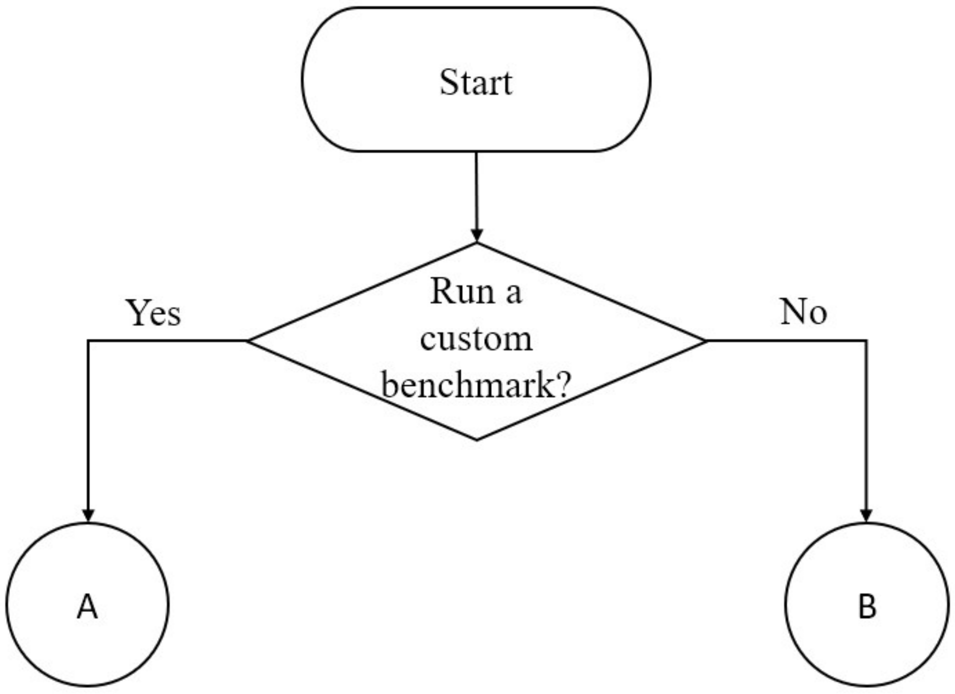


Fig 3.4.1 – First the user chooses between a default and a custom benchmark

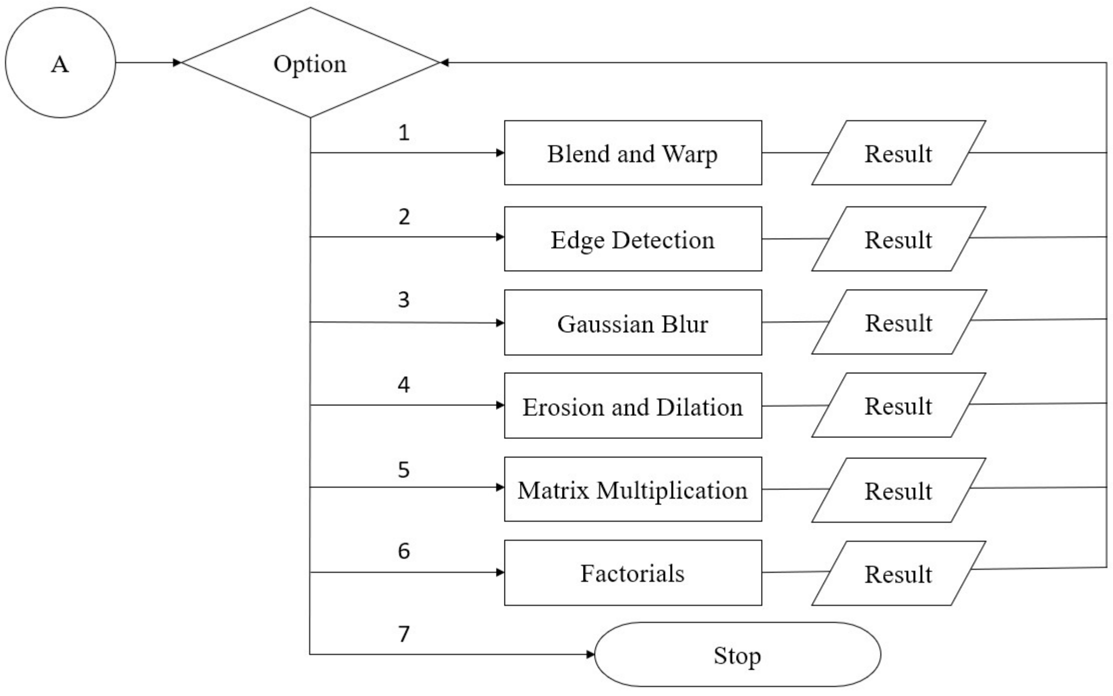


Fig 3.4.2 – Steps involved in the custom benchmark

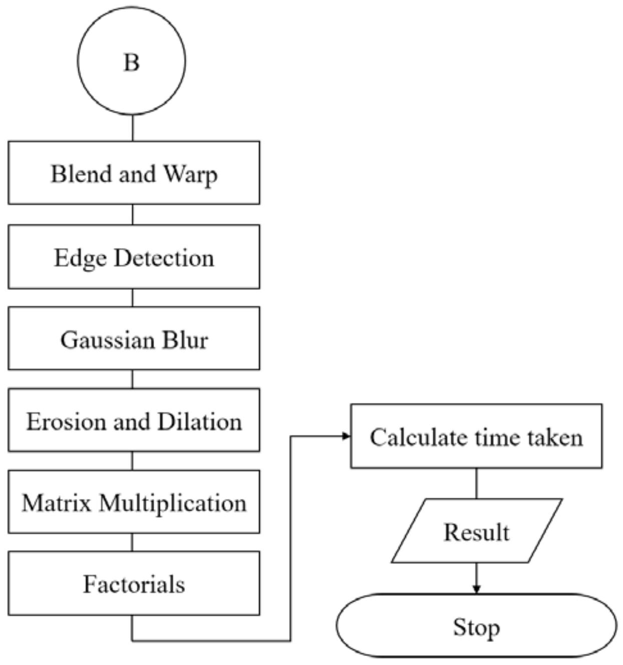


Fig 3.4.3 – Steps involved in the default benchmark

COMPLETE CODE

```
from PIL import Image
import numpy as np
import cv2
import streamlit as st
import time
import math
from multiprocessing import Process, cpu_count
import glob
import sys

def option1(option = 'd'):
    st.header("Blend & Warp Test")
    start_t1 = time.time()

    im1 = Image.open("landscape1.jpg")
    im2 = Image.open("landscape2.jpg")

    im3 = Image.blend(im1, im2, 0.5)

    end_t1 = time.time()
    im3.save("Blended.jpg")
```

```
img = cv2.imread("Blended.jpg", cv2.IMREAD_GRAYSCALE)

start_t2 = time.time()

rows, cols = img.shape

# Vertical wave

if (option == 'a'):
    img_output = np.zeros(img.shape, dtype = img.dtype)

    for a in range(5):
        for i in range(rows):
            for j in range(cols):
                offset_x = int(25.0 * math.sin(2 * 3.14 * i / 180))
                offset_y = 0
                if j+offset_x < rows:
                    img_output[i,j] = img[i,(j+offset_x)%cols]
                else:
                    img_output[i,j] = 0

# Horizontal wave

elif (option == 'b'):
    img_output = np.zeros(img.shape, dtype=img.dtype)

    for a in range(5):
        for i in range(rows):
            for j in range(cols):
                offset_x = 0
                offset_y = int(16.0 * math.sin(2 * 3.14 * j / 150))
                if i+offset_y < rows:
                    img_output[i,j] = img[(i+offset_y)%rows,j]
                else:
                    img_output[i,j] = 0

# Concave effect elif

(option == 'c'):
    img_output = np.zeros(img.shape, dtype=img.dtype)

    for a in range(5):
        for i in range(rows):
            for j in range(cols):
                offset_x = int(350.0 * math.sin(2 * 3.14 * i /
(2*cols)))
                offset_y = 0
                if j+offset_x < cols:
                    img_output[i,j] = img[i,(j+offset_x)%cols]
                else:
                        img_output[i,j] = 0

    # Both horizontal and vertical

    if (option == 'd'):
        img_output = np.zeros(img.shape, dtype=img.dtype)

        for a in range(5):
            for i in range(rows):
                for j in range(cols):
                    offset_x = int(20.0 * math.sin(2 * 3.14 * i / 150))
                    offset_y = int(20.0 * math.cos(2 * 3.14 * j / 150))

                    if i + offset_y < rows and j+offset_x < cols:
                        img_output[i,j] =
img[(i+offset_y)%rows,(j+offset_x)%cols]
                    else:
                        img_output[i,j] = 0

    end_t2 = time.time()
    time_taken = (end_t1 - start_t1) + (end_t2 - start_t2)

    cv2.imwrite('Input.jpg', img)
    cv2.imwrite('BNWOutput.jpg', img_output)

    st.image(im1, width = 800, caption = 'ORIGINAL1')
    st.image(im2, width = 800, caption = 'ORIGINAL2')
    st.image(img, width = 800, caption = 'AFTER BLEND')
    st.write("\n")
    st.image(img_output, width = 800, caption = 'AFTER BLEND & WARP')

    st.subheader("Time taken to run Blend & Warp Test is " + str("%.3f" %
time_taken) + " seconds")

    return time_taken

def option2():

    st.header("Edge Detection Test")

    start_time = time.time()

    # Read image from disk.
    for j in range(12):

        X_data = []
        files = glob.glob (r"C:\Users\Desktop\OS_Images\*.jpg")

        i = 0
        count = 0
        for myFile in files:
            image = cv2.imread (myFile)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            X_data.append (image)

            edges = cv2.Canny(image, 100, 200)
            cv2.imwrite('image-{}.jpg'.format(i), edges)

            incr_time = time.time() - start_time

            i = i + 1
            count += 1

    end_time = time.time()

    for i in range(count):
        st.image(X_data[i], width = 800)
        st.image('image-{}.jpg'.format(i), width = 800)

    time_taken = end_time - start_time

    st.subheader("Time taken to run Edge Detection Test is " + str("%.3f" % 
time_taken) + " seconds")

    return time_taken

def option3():

    st.header("Gaussian Blur Test")

    start_time = time.time()

    img = cv2.imread('flowers.jpg')

    blur = cv2.GaussianBlur(img, (7, 7), 0)
    for i in range(1800):
        blur = cv2.GaussianBlur(blur, (7, 7), 0)

    end_time = time.time()
    time_taken = end_time - start_time

    #st.markdown('<style>body{background-color: 
Black;}</style>',unsafe_allow_html=True)

    st.image(img, width = 800, caption = 'Original Image')
    st.image(blur, width = 800, caption = 'Blurred Image')
    cv2.imwrite('flowersBlurred.jpg', blur)

    st.subheader("Time taken to run Gaussian Blur Test is " + str("%.3f" % 
time_taken) + " seconds")
    return time_taken
```

```
def option4():

    st.header("Erosion & Dilation Test")

    start_time = time.time()

    img = cv2.imread('flowers.jpg', 0)

    # Taking a matrix of size 5 as the kernel
    kernel = np.ones((5,5), np.uint8)

    # The first parameter is the original image,
    # kernel is the matrix with which image is
    # convolved and third parameter is the number
    # of iterations, which will determine how much
    # you want to erode/dilate a given image.

    for i in range(1800):

        img_erosion = cv2.erode(img, kernel, iterations = 4)
        img_dilation = cv2.dilate(img, kernel, iterations = 4)

    end_time = time.time()
    time_taken = end_time - start_time

    cv2.imwrite('Erosion.jpg', img_erosion)
    cv2.imwrite('Dilation.jpg', img_dilation)

    st.image(img, width = 800, caption = "Original Image")
    st.image(img_erosion, width = 800, caption = "Eroded Image")
    st.image(img_dilation, width = 800, caption = "Dilated Image")

    st.subheader("Time taken to run Erosion & Dilation Test is " +
str("%.3f" % time_taken) + " seconds")

    return time_taken

def matrixMult():
    X = []
    result = []

    for i in range(400):
        a = []
        b = []

        for j in range(400):
            a.append(int(j+1))
            b.append(int(0))

        X.append(a)
        result.append(b)

    for i in range(len(X)):

        for j in range(len(X[0])):

            for k in range(len(X)):
                result[i][j] += X[i][k] * X[k][j]

def option5(choice = 'b', PROCESSES = cpu_count()):

    st.header("Matrix Multiplication Tests")

    start_time = time.time()
    if (choice == 'a'):
        for i in range (8):
            matrixMult()

    elif (choice == 'b'):
        start_t = time.time()
        A = list()

        if   name___== "  main  ":
            for i in range(PROCESSES):
                A.append(Process(target = matrixMult))
                A[i].start()

            for i in A:
                i.join()

            dt = time.time() - start_t

    end_time = time.time()
    time_taken = end_time - start_time

    st.subheader("Time taken to run Matrix Multiplication Test is " +
str("%.3f" % time_taken) + " seconds")

    return time_taken

def option6():

    st.header("Factorial Tests")

    start_time = time.time()
    fact = [1 for i in range(7000)]
    a = [i for i in range(7000)]

    # for i in range(7000):
    #     a.append(int(i+1))
    #     fact.append(1)
    for i in range(7000):

        for j in range(1,a[i]+1):
            fact[i] = fact[i] * j

    time_taken = time.time() - start_time

    st.subheader("Time taken to run Factorial Test is " + str("%.3f" % 
time_taken) + " seconds")

    return time_taken

if   name___== "  main  ":

    st.title("RA\u00b2 Benchmark")

    print("\nDo you want to run a custom benchmark?\n")
    print("Note that typing in 'no' will let you run a default 
benchmark\n")
    print("Typing 'yes' will let you choose the tests you want to run 
individually\n")

    run_choice = input("Enter your choice: ")

    if (run_choice == 'yes'):

        time_taken = 0

        while(1):
            print("\nMenu:\n1. Blend and warp\n2. Edge detection\n3. 
Gaussian Blur\n4. Erosion & Dilation\n5. Matrix Multiplication\n6. 
Factorial\n7. Exit")

            a = int(input("\nEnter choice: "))

            if (a == 1):
                print("\na. Vertical\nb. Horizontal\nc. Concave\nd. 
Multidirectional\n")
                opt = input("Enter choice: ")
                print("\nTime taken for blend & warp test is %.3f seconds" 
% option1(opt))

            elif (a == 2):
                print("Time taken for edge detection test is %.3f seconds" 
% option2())

            elif (a == 3):
                print("Time taken to perform Gaussian Blur test is %.3f 
seconds" % option3())
```

```
            elif (a == 4):
                print("Time taken to perform Dilation & Erosion test is 
%.3f seconds" % option4())

            elif (a == 5):
                print("\na. Single core\nb. Multi core")
                opt = input("\nEnter choice: ")

                if opt == 'b':
                    ch = input("\ni.  Default (number of logical 
cores)\nii. Custom\n\nEnter choice: ")
                    if (ch == 'i'):
                        print("\nTime taken to create and execute %d 
processes using multiprocessing is %.3f seconds" % 
(cpu_count(),option5(opt)))

                    if (ch == 'ii'):
                        thr = int(input("Enter number of processes: "))
                        print("\nTime taken to create and execute %d 
processes using multiprocessing is %.3f seconds" % (thr,option5(opt, thr)))

                else:
                    print("\nTime taken to multiply the matrices 8 times 
using a single core is %.3f seconds" % option5(opt))

            elif (a == 6):
                print("Time taken to perform factorial test is %.3f 
seconds" % option6())

            elif (a == 7):
                sys.exit(0)

    elif (run_choice == 'no'):


        run_1 = option1()
        print("\nTime taken for blend & warp test is %.3f seconds" % run_1)

        run_2 = option2()
        print("\nTime taken for edge detection test is %.3f seconds" % 
run_2)

        run_3 = option3()
        print("\nTime taken to perform Gaussian Blur test is %.3f seconds" 
% run_3)

        run_4 = option4()
        print("\nTime taken to perform Dilation & Erosion test is %.3f 
seconds" % run_4)

        run_5 = option5()
        print("\nTime taken to execute %d processes performing matrix
multiplication using multiprocessing is %.3f seconds" % (cpu_count(),
run_5))

        run_6 = option6()
        print("\nTime taken to perform factorial test is %.3f seconds" %
run_6)

        print("\nTotal time taken to run the benchmark is %.3f seconds" %
(run_1 + run_2 + run_3 + run_4 + run_5 + run_6))

        st.header("Total time taken to run the benchmark is " + str("%.3f"
% (run_1 + run_2 + run_3 + run_4 + run_5 + run_6)) + " seconds")
```

SCREENSHOTS AND OUTPUT

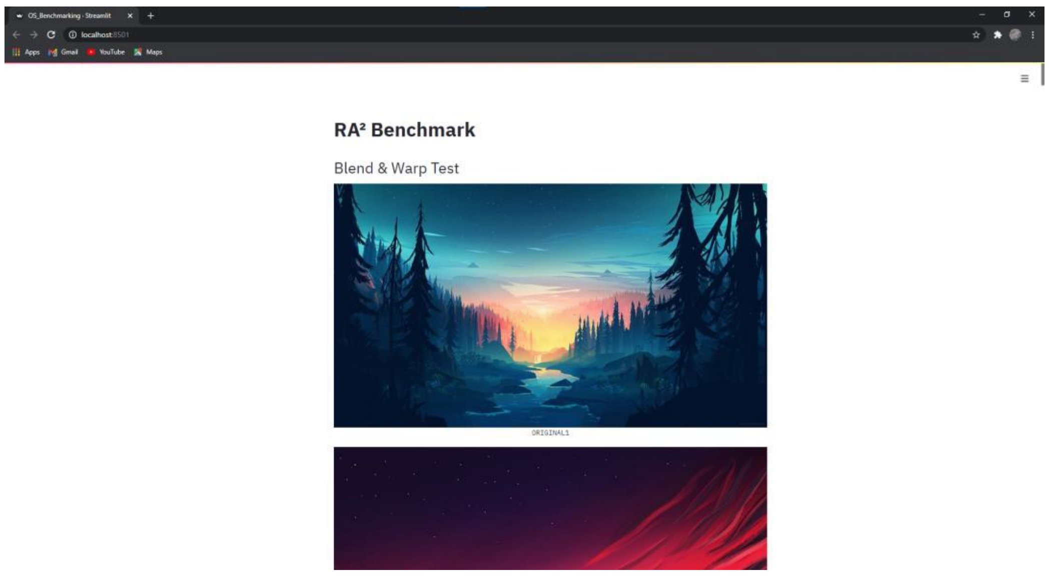
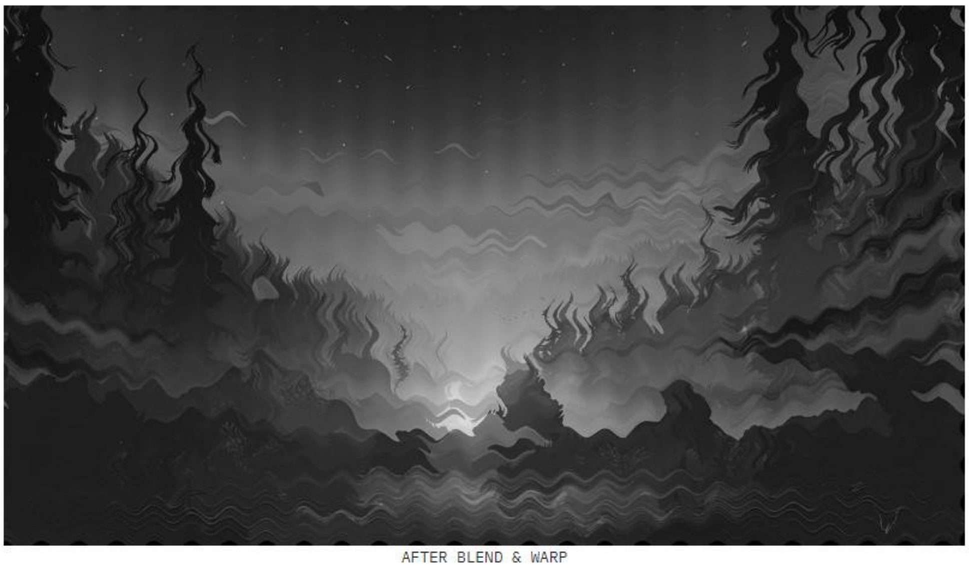
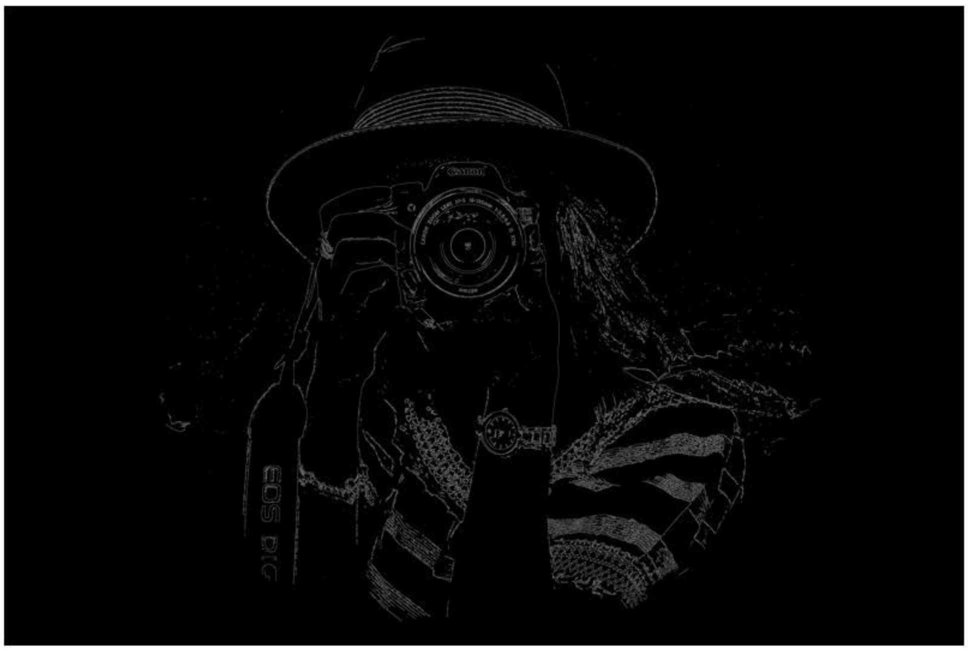


Fig 4.3.2 – The streamlit interface



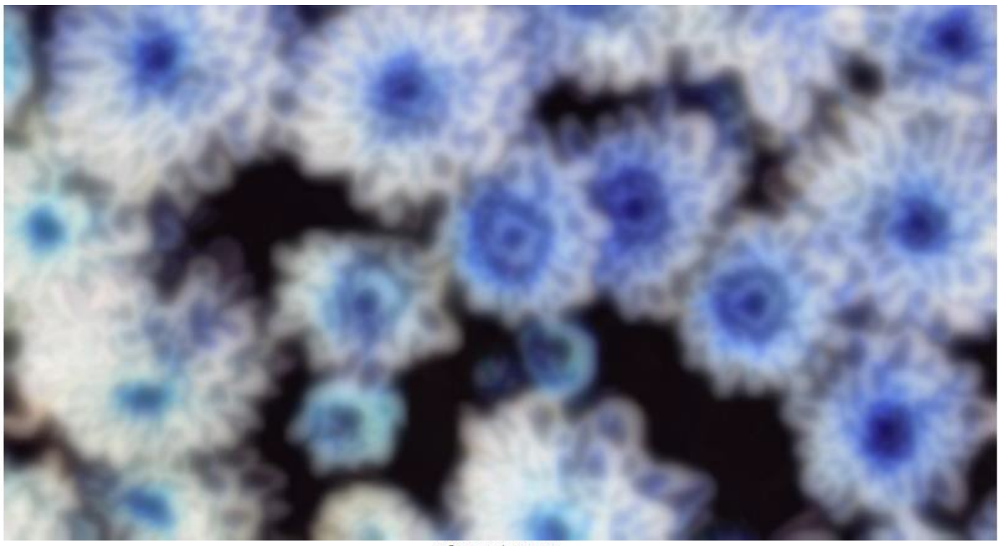
Time taken to run Blend & Warp Test is 37.536 seconds

Fig 4.3.3 – The result of the blend and warp test



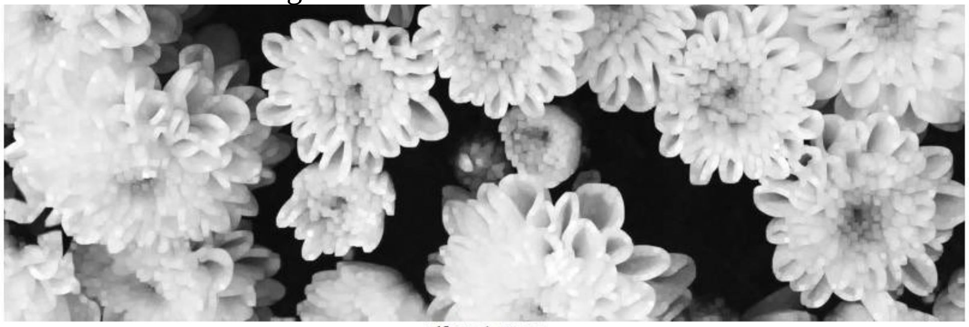
Time taken to run Edge Detection Test is 33.671 seconds

F ig 4.3.4 – The edge detection test result



Time taken to run Gaussian Blur Test is 24.062 seconds

Fig 4.3.5 – The Gaussian Blur result



Time taken to run Erosion & Dilation Test is 23.316 seconds

Matrix Multiplication Tests

Time taken to run Matrix Multiplication Test is 26.178 seconds

Factorial Tests

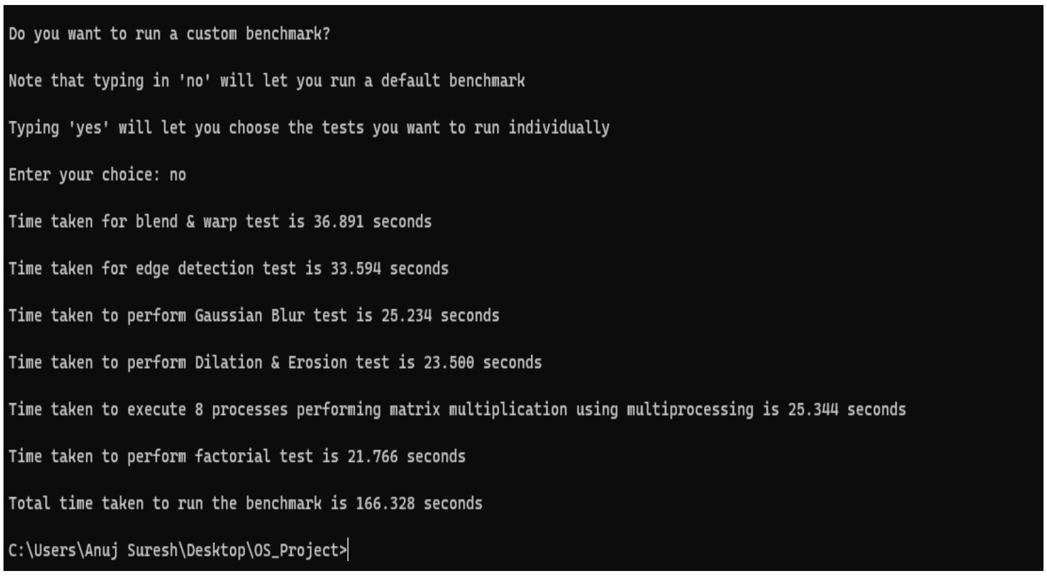
Time taken to run Factorial Test is 21.292 seconds

Total time taken to run the benchmark is 166.055 seconds

Fig 4.3.6 – The remaining test results

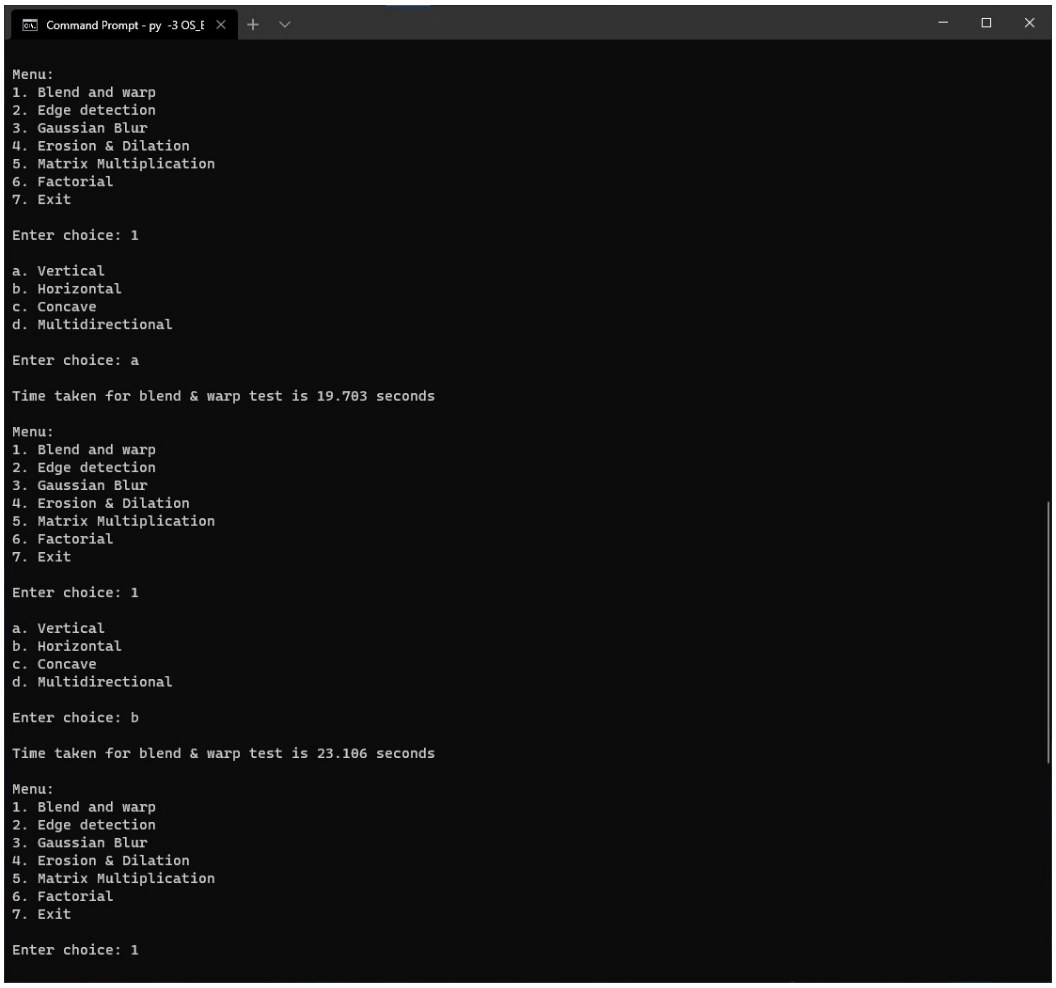
4. RESULTS

4.1 Test Screenshots



```
Do you want to run a custom benchmark?

Note that typing in 'no' will let you run a default benchmark

Typing 'yes' will let you choose the tests you want to run individually

Enter your choice: no

Time taken for blend & warp test is 36.891 seconds

Time taken for edge detection test is 33.594 seconds

Time taken to perform Gaussian Blur test is 25.234 seconds

Time taken to perform Dilation & Erosion test is 23.500 seconds

Time taken to execute 8 processes performing matrix multiplication using multiprocessing is 25.344 seconds

Time taken to perform factorial test is 21.766 seconds

Total time taken to run the benchmark is 166.328 seconds

C:\Users\Anuj Suresh\Desktop\OS_Project>
```

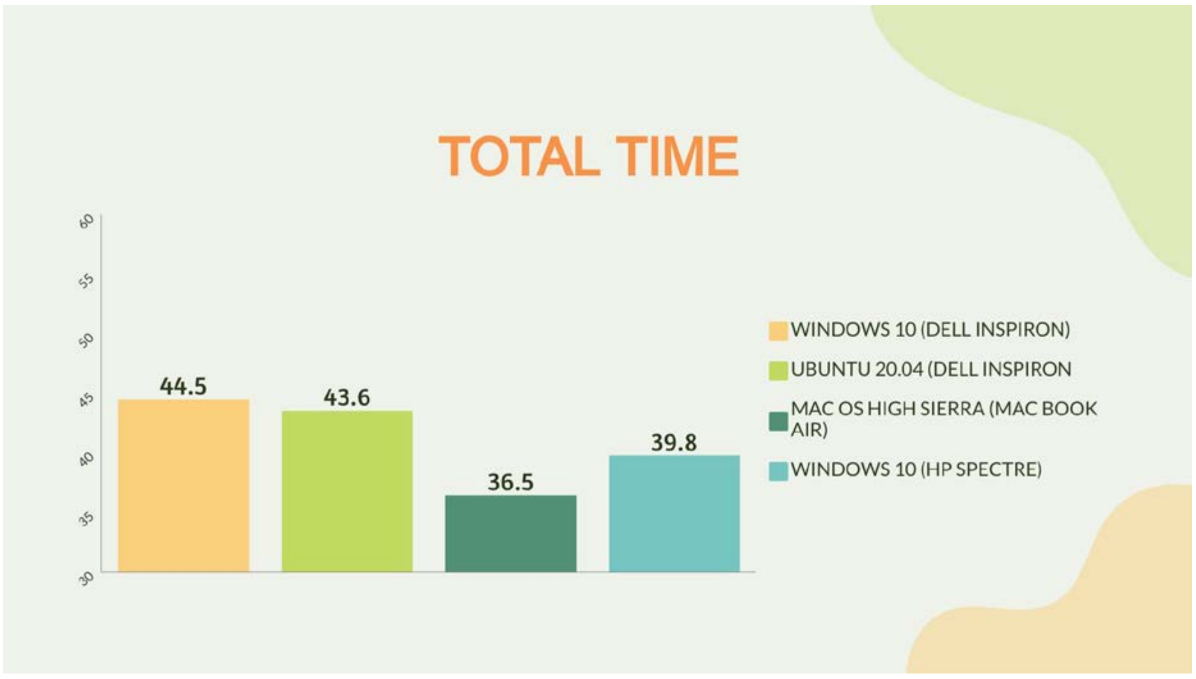
Fig 5.1.1 – Default benchmark execution

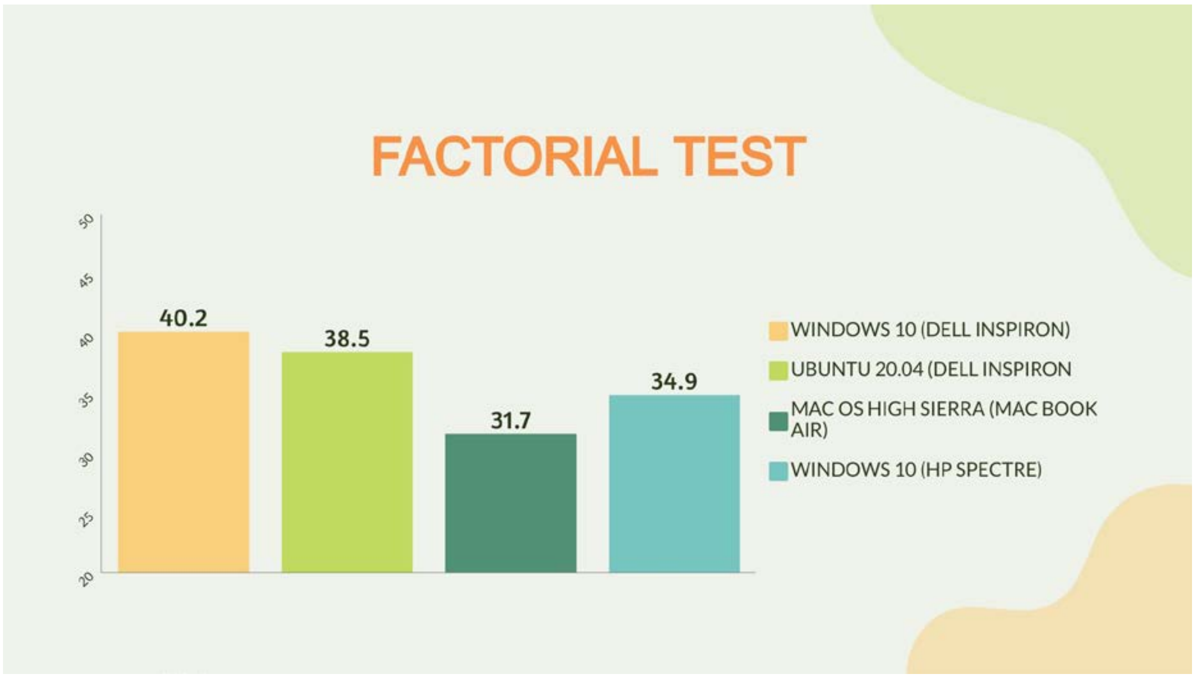


```
Menu:
1. Blend and warp
2. Edge detection
3. Gaussian Blur
4. Erosion & Dilation
5. Matrix Multiplication
6. Factorial
7. Exit

Enter choice: 1

a. Vertical
b. Horizontal
c. Concave
d. Multidirectional

Enter choice: a

Time taken for blend & warp test is 19.703 seconds

Menu:
1. Blend and warp
2. Edge detection
3. Gaussian Blur
4. Erosion & Dilation
5. Matrix Multiplication
6. Factorial
7. Exit

Enter choice: 1

a. Vertical
b. Horizontal
c. Concave
d. Multidirectional

Enter choice: b

Time taken for blend & warp test is 23.106 seconds

Menu:
1. Blend and warp
2. Edge detection
3. Gaussian Blur
4. Erosion & Dilation
5. Matrix Multiplication
6. Factorial
7. Exit

Enter choice: 1
```

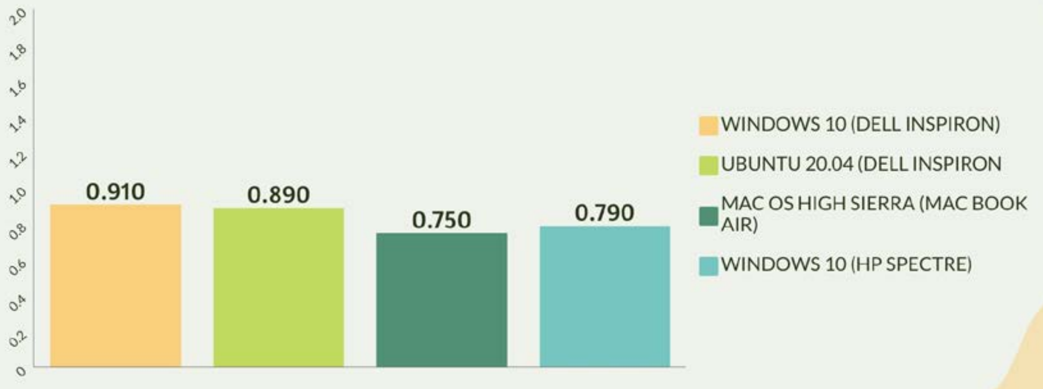
Fig 5.1.2 – Custom blend and warp execution

4.2 Graphs

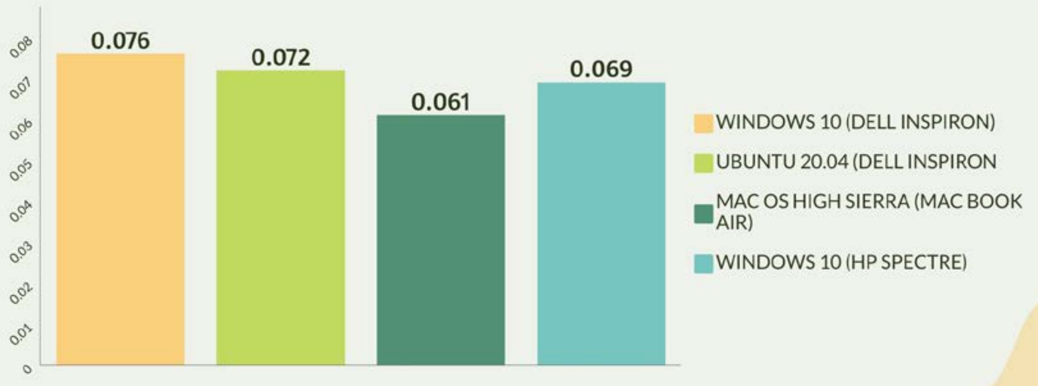


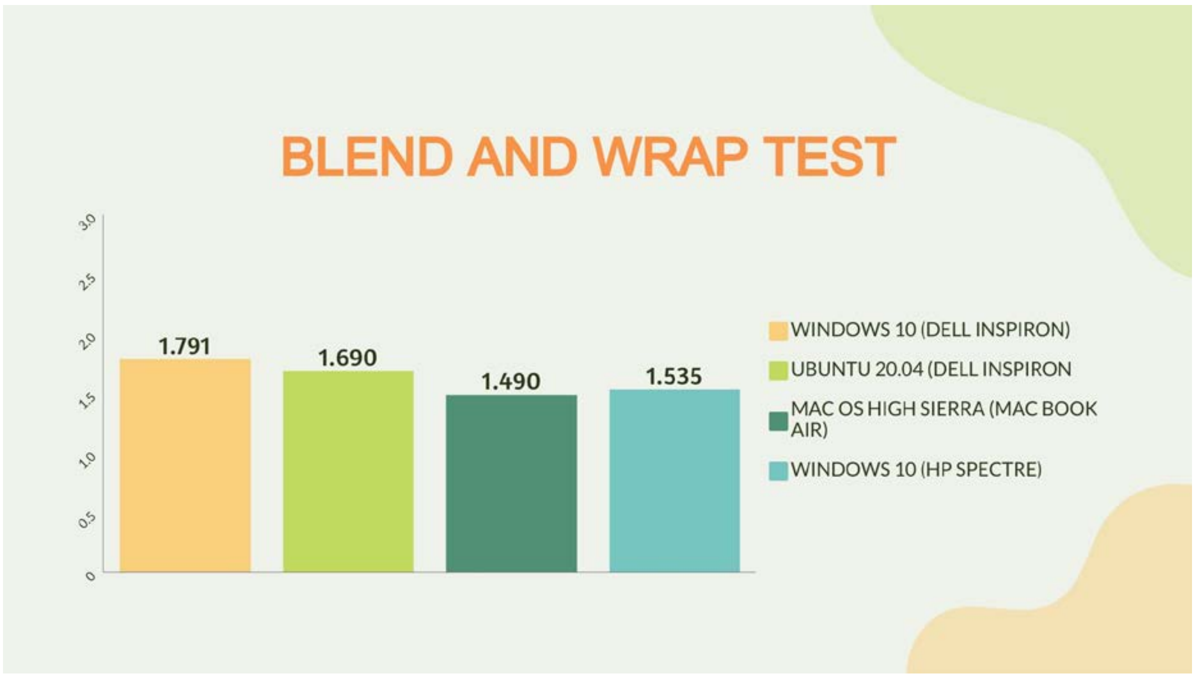


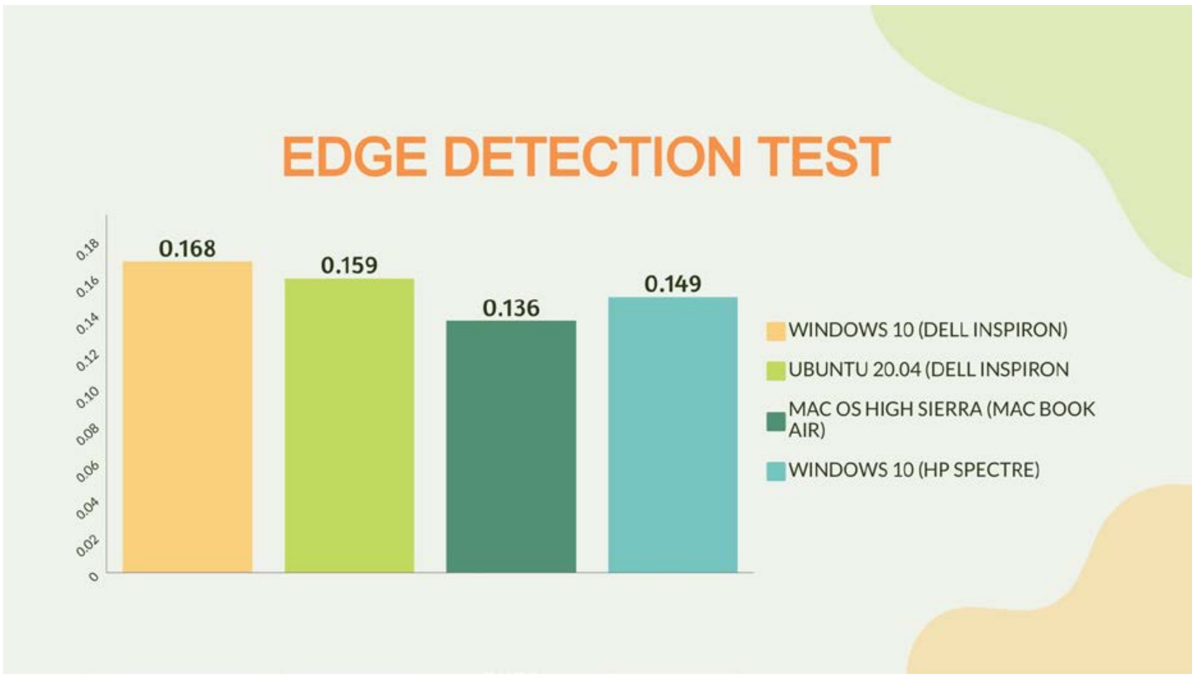
MATRIX MULTIPLICATION

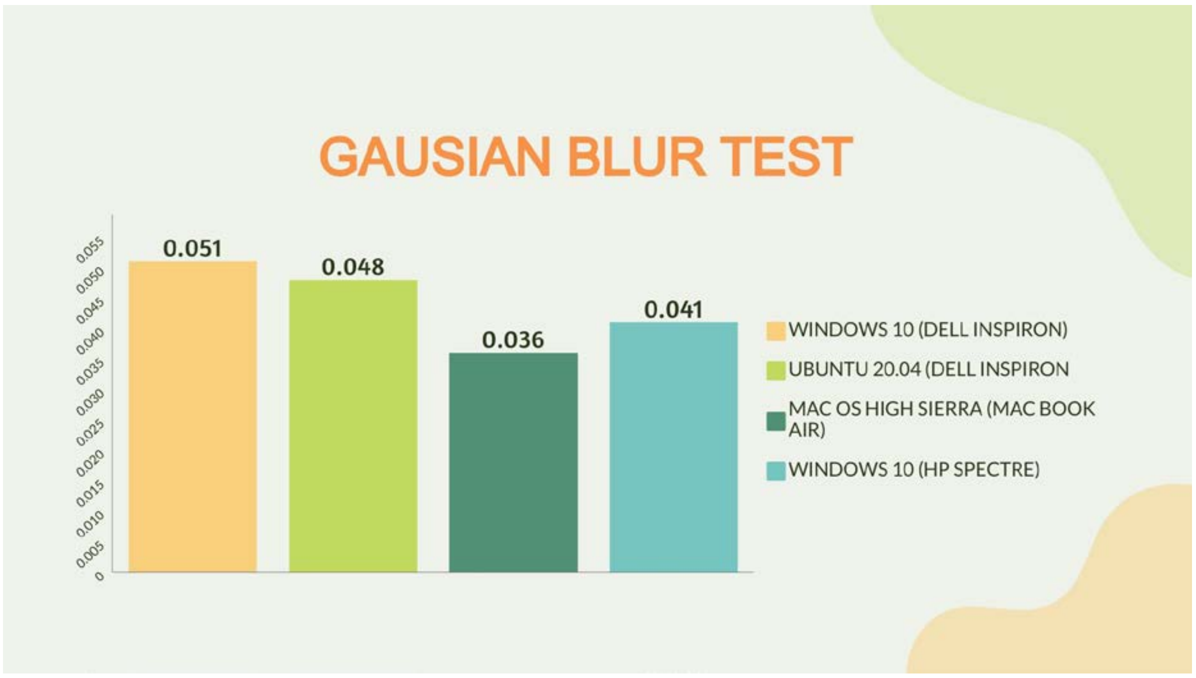


EROSION AND DILATION TEST




BLEND AND WRAP TEST
3.0
2.5
2.0
1.5
1.0
0.5
0
1.791
1.690
1.490
1.535
WINDOWS 10 (DELL INSPIRON)
UBUNTU 20.04 (DELL INSPIRON
MAC OS HIGH SIERRA (MAC BOOK AIR)
WINDOWS 10 (HP SPECTRE)


EDGE DETECTION TEST
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0
0.168
0.159
0.136
0.149
WINDOWS 10 (DELL INSPIRON)
UBUNTU 20.04 (DELL INSPIRON
MAC OS HIGH SIERRA (MAC BOOK AIR)
WINDOWS 10 (HP SPECTRE)



5. CONCLUSIONS

5.1 Limitations

The mentioned testing methods had the following limitations:

1. Our system that runs on macOS has an inferior processor compared to the rest and hence the results cannot be taken at face value. The most fair comparisons are between the same devices running both Ubuntu and Windows in different tests.
2. We have only limited devices at our disposal at this time and hence the testing may not be the most ideal.
3. The test concentrates on stressing the CPU to achieve results and not the GPU as such.

5.2 Inferences

From the following benchmark tests, we inferred that on the systems that were tested using both Windows and Ubuntu, the tests run on the Ubuntu operating system completed significantly faster than those run on the Windows operating system. On some tests, this difference was at much as 20 seconds. This gave the Ubuntu operating system around a 25% faster completion time over the Windows system. This was most apparent in the image processing tests.

6. REFERENCES

[1] https://www.researchgate.net/publication/273133047_How_to_Build_a_Benchmark
[2] https://linuxhint.com
[3] https://www.cis.upenn.edu/~milom/cis501-Fall12/lectures/04_performance.pdf
[4] https://www.geeksforgeeks.org/data-science-apps-using-streamlit/
[5] https://www.streamlit.io
[6] https://www.tutorialspoint.com/image-processing-in-python
[7] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc /py_table_of_contents_
imgproc/py_table_of_contents_imgproc.html [8]
https://www.geeksforgeeks.org/opencv-python-tutorial/