Shahil Patel-200010039
Pranav Talegaonkar-200010041

Assignment-6
CS-314:Operating Systems Laboratory

Part – 1

ConvertGrayscale & HorizontalBlur were two successive changes that were made in Part 1 of the assignment. The following subsections show how the concept was put into practise:

1.1 ConvertGrayscale

For converting an RGB image to a grayscale format, a weighted average approach was integrated. Given that red has the highest wavelengths of the three hues, green is the colour that has lower wavelengths than red while simultaneously having a more calming effect on the eyes. This means that we must reduce the contribution of red, raise the contribution of green, and place the contribution of blue hue in between these two. As a result, the revised equation is as follows:

values[p][q].colour = (colour_blue * 0.114) + (colour_red * 0.299) + (colour_green * 0.587);

where colour is replaced with red, green or blue.

where, values is the image matrix that is updated to create a new transformed Image.

According to this equation, Red has a contribution of about 29.9%, Green has a contribution 58.7% which is greater in all three colours and Blue has a contribution of only about 11.4%.

1.2 HorizontalBlur

Every pixel is essentially made to look more like those surrounding it when there is any form of blur. Each pixel in a horizontal blur can be averaged with those in a certain direction, creating the appearance of motion.

Each new pixel's value will be calculated for this programme as being equal to half of its prior value. Averaged from a predetermined number of pixels to the right is the remaining half. The blurAmount is the name for this constant amount. The amount of blurring will increase with the value's size.

This is done independently for the red, green, and blue components of each pixel.
This is expressed in the following algorithm:

- For each row of the image:
  - For each pixel in the row:
    - Set red to be half the pixels red component.
    - Set green to be half the pixels green component.
    - Set blue to be half the pixels blue component.
    - For i from 1 up to blurAmount:
      - Increment red by R×0.5blurAmount, where R is the red component of the pixel i to the right of the current pixel.
      - Increment green by G×0.5blurAmount, where G is the green component of the pixel i to the right of the current pixel.
      - Increment blue by B×0.5blurAmount, where B is the blue component of the pixel i to the right of the current pixel.
    - Save red,green,blue as the new color values for this pixel.

You must also ensure you don’t access pixels off the bounds of the image. For example, the third pixel from the right should only average itself (half weight), and the next two (at a quarter weight each).

For this program, set the **blurAmount** to ***50***.

Input:

Output:

2. Observations

Image Size	Part1 (Sequential) (microseconds)	Part2.1a (Threads - atomic operations) (microseconds)	Part2.1b (Threads – Semaphores) (microseconds)	Part 2.2 (Process – Shared memory) (microseconds)	Part 2.3 (Process – Pipe) (microseconds)
239 bytes	19	231	414	1748 2371 4363 6143	824
177 kB	17932	28024	19834	15683 30625 35191 45130	45369
10 MB	1127559	1280655	1113082	806090 1635099 2094010 2585997	2415869

3. Analysis

Since the sequential programme executes each step sequentially, it should have taken longer than other ways to complete the work at hand. In contrast, alternative systems use numerous threads and processes that operate in some kind of parallelization.

Using atomics and semaphores, the sequential technique is more expensive than parallel threading.

Due to the additional writing and reading of values to and from memory, shared memory is superior to sequential memory.

The most expensive approach is the pipeline approach because I am sending updated values from T1 to T2 in batches using the pipeline and because each pixel in my T2 requires completion of the T1 for the subsequent 50 pixels in order to process further. As a result, the total run time in this case is increased by the sending and receiving times.

On the whole, we can see that the shared memory strategy is beneficial but that it only permits shared memory within certain bounds.

If the data is huge, the pipeline takes a long time to transport and read on the other end.

Atomic locks and Semaphore locks function nearly identically (Semaphores better a little bit).