

RTDM – Final Extended Lab



Topic:

Affine Transformation and Sobel filter on 576p SD video

Presented by:

Bhaumik Bhatt & Vinit Vyas

Index

No	Topic	Page
1	Introduction	3
2	Frame enhancement algorithm design	4
3	Major Software Modules	
	3.1 Source Video	5
	3.2 FFmpeg	7
	3.3 Transformations	8
	3.3.1 Sobel Filter	8
	3.3.2 Affine Transform	9
4	Code Information	12
5	Analysis	
	5.1 No optimization + no Multithreading	14
	5.2 No optimization + Multithreading	15
	5.3 Optimization + Multithreading	17
	5.4 Only GCC Optimization + No Multithreading	19
	5.5 Overall Analysis	21
6	Output Videos Specifications	22
7	Conclusions	26
8	References	27

1. Introduction

The team members for this extended lab assignment are **Bhaumik Bhatt** and **Vinit Vyas**.

In this extended lab, we mimic a video post-production environment wherein we apply transformations to a video and implement it using two filters over a Standard definition video. One is a Sobel color filter, the other transformation is the Affine (warp + rotate). The time profile for CPU utilization and transformation program execution time is obtained. We then apply gcc compiler optimization and Intel SSE hardware acceleration in order to obtain speed-up of transformation. Thus, we observe that these highly computationally intensive processes can be offloaded to hardware for reduced CPU utilization and improved execution time for us to be able to reduce time spent in waiting.

Our objective here was to work with C/C++ code for accessing and modifying files and the Open Source Computer Vision (OpenCV) library which enables image transformations on those files by providing a bunch of APIs for different transform functions.

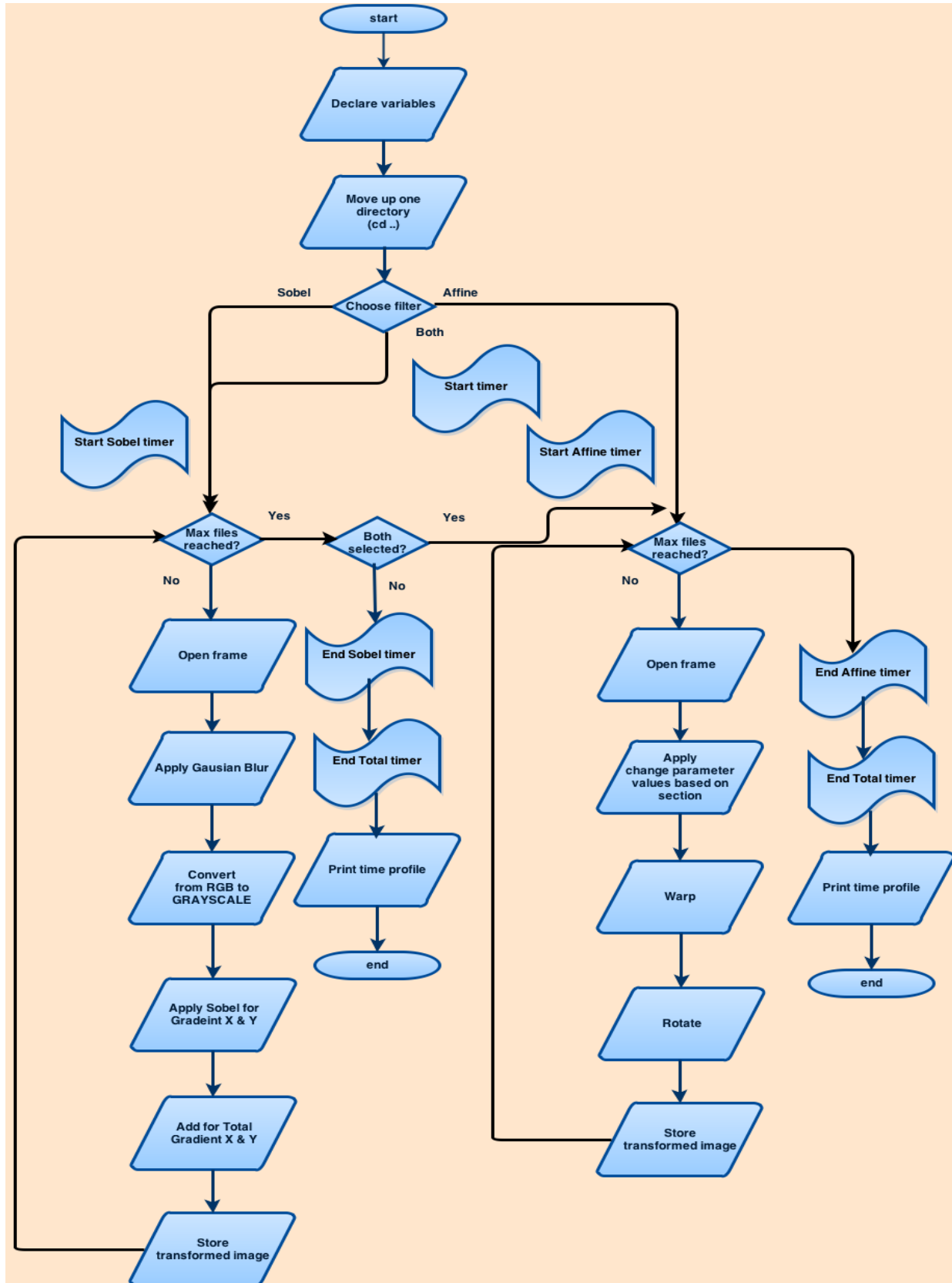
At first, we take an open source free-to-use SD (1280x576, 24 fps) video. We extract uncompressed ppm (Portable Pixel Map) images from that SD video using ffmpeg on Linux. Then, to those uncompressed stream of images, we apply the filters giving the user the option to choose the same. For example, he/she can apply Sobel filter or Affine transform (warp+rotate) alone or he/she can do both (i.e. first apply Sobel, then Affine). Later using the modified image frames, we use ffmpeg to re-create the video resulting with the effects in place. We run a time profile for the same to calculate the amount of time spent in the transformations. Those results are compared with the time profile, when we later add Intel SSE hardware accelerations and GCC compiler optimization.

We take note that even a minor color correction or inclusion of effects and animations on top of high-definition, high-quality video is very computationally intensive. This we can observe comparing the time profile for the frame transformation: CPU utilization and time taken. By taking a system-wide profile as well, we can observe results for the same.

We conclude that for high-definition video, color-correction and inclusion of effects in a lengthy movie must be aided with Graphics Processing Units and/or other hardware acceleration offloading intensive video work to other chips, keeping the CPU free. We attempted use of OpenCV CUDA GPU module for effective offloading but it would require a native Linux and NVIDIA GPU system, which was unavailable with us.

We shall take a look at details of our code, algorithm, tools, techniques and results from our extended lab.

2. Frame enhancement algorithm design



3. Major Software Modules

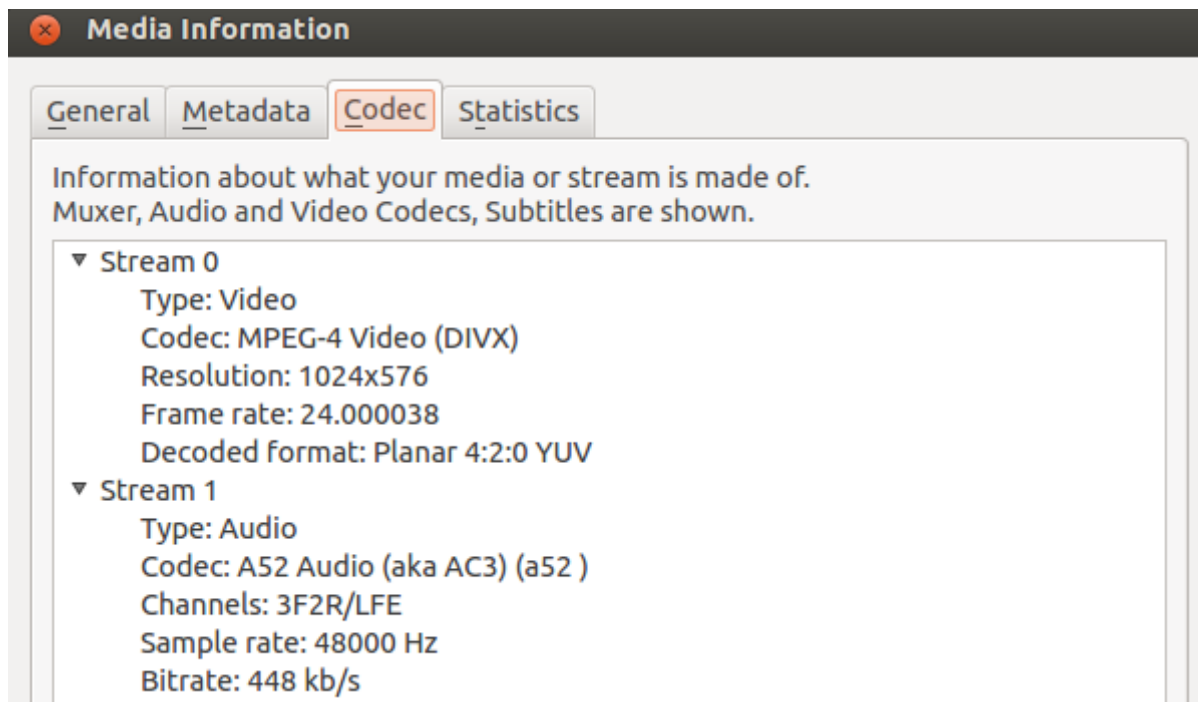
3.1 Source Video

We have used the open source 576p, standard definition mpeg2 video from 'Elephant's Dream' (<http://www.elephantsdream.org/>).

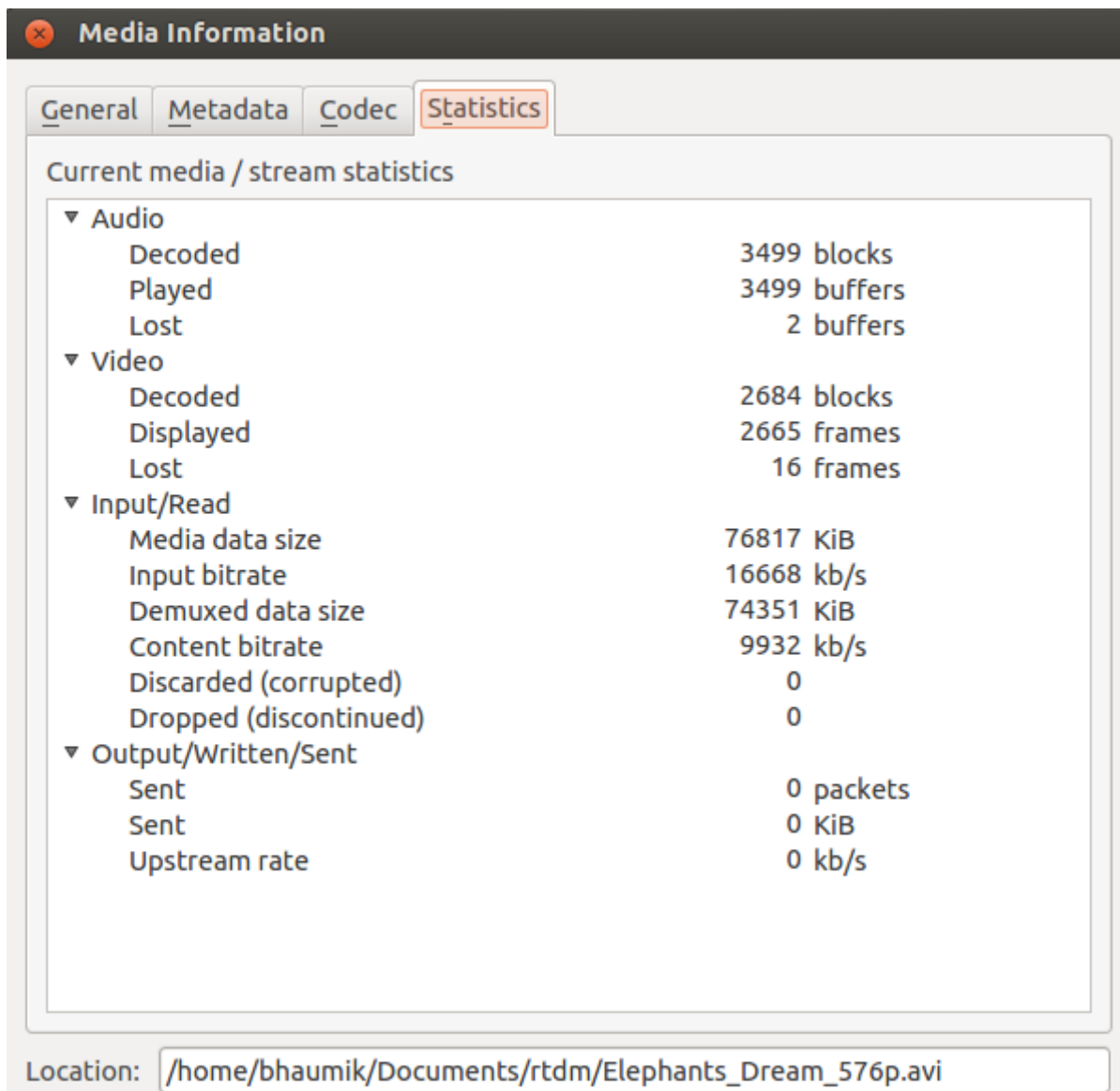
1. 'avprobe' output:

```
bhaumik@bhaumik-VirtualBox:~/Documents/rtdm$ avprobe Elephants_Dream_576p.avi
avprobe version 0.8.13-4:0.8.13-0ubuntu0.12.04.1, Copyright (c) 2007-2014 the Libav developers
built on Jul 15 2014 12:56:47 with gcc 4.6.3
Input #0, avi, from 'Elephants_Dream_576p.avi':
  Metadata:
    encoder      : AVI-Mux GUI 1.17.5, Apr  5 2006  18:41:17
    JUNK         :
  Duration: 00:10:53.79, start: 0.000000, bitrate: 5455 kb/s
    Stream #0.0: Video: mpeg4 (Simple Profile), yuv420p, 1024x576 [PAR 1:1 DAR 16:9], 24 tbr, 24 tbn,
24 tbc
    Stream #0.1: Audio: ac3, 48000 Hz, 5.1, s16, 448 kb/s
  Metadata:
    title       : ED-CM-5.1-DVD-Final2
```

2. Codec information and statistics from VLC media player:



3. VLC Media Statistics



We will use the above information for comparison with the final output video.

3.2 FFmpeg

We use FFmpeg in order to extract uncompressed PPM images from the video.

1. FFmpeg command:

```
bhaumik@bhaumik-VirtualBox:~/Documents/rtdm$ ffmpeg -t 90 -i Elephants_Dream_576p.avi "./Original_Frames/Frame_no_%05d.ppm"
```

```

bhaumik@bhaumik-VirtualBox: ~/Documents/rtdm
ffmpeg version 0.8.13-4:0.8.13-0ubuntu0.12.04.1, Copyright (c) 2000-2014 the Libav developers
  built on Jul 15 2014 12:56:47 with gcc 4.6.3
*** THIS PROGRAM IS DEPRECATED ***
This program is only provided for compatibility and will be removed in a future release. Please use avconv instead.
Input #0, avi, from 'Elephants_Dream_576p.avi':
  Metadata:
    encoder      : AVI-Mux GUI 1.17.5, Apr  5 2006  18:41:17
    JUNK         :
  Duration: 00:10:53.79, start: 0.000000, bitrate: 5455 kb/s
    Stream #0.0: Video: mpeg4 (Simple Profile), yuv420p, 1024x576 [PAR 1:1 DAR 16:9], 24 tbr, 24 tbn, 24 tbc
    Stream #0.1: Audio: ac3, 48000 Hz, 5.1, s16, 448 kb/s
  Metadata:
    title        : ED-CM-5.1-DVD-Final2
Incompatible pixel format 'yuv420p' for codec 'ppm', auto-selecting format 'rgb24'
[buffer @ 0xdc7da0] w:1024 h:576 pixfmt:yuv420p
[avsink @ 0xdc67a0] auto-inserting filter 'auto-inserted scaler 0' between the filter 'src' and the filter 'out'
[scale @ 0xdc6dc0] w:1024 h:576 fmt:yuv420p -> w:1024 h:576 fmt:rgb24 flags:0x4
Output #0, image2, to './Original_Frames/Frame_no_%05d.ppm':
  Metadata:
    JUNK         :
    encoder      : Lavf53.21.1
    Stream #0.0: Video: ppm, rgb24, 1024x576 [PAR 1:1 DAR 16:9], q=2-31, 200 kb/s, 90k tbn, 24 tbc
Stream mapping:
  Stream #0.0 -> #0.0
Press ctrl-c to stop encoding
frame=  97 fps= 21 q=0.0 size=      -0kB time=4.04 bitrate=  -0.0kbits/s

```

We place the extracted frames in the Original Frames directory and move on to image transformation code.

2. To create mpeg4 video file out of images:

```

bhaumik@bhaumik-VirtualBox: ~/Documents/rtdm
bhaumik@bhaumik-VirtualBox:~/Documents/rtdm$ ffmpeg -i ./Original_Frames/Frame_no_%05d.ppm
-vcodec mpeg4 -sameq Transformed-ElephantsDreamSD.mp4

```

3.3 Transformations

The algorithms we have used are implementing the Affine Transformation and Sobel image filters from the Open CV library.

3.3.1 Sobel filter

Sobel filter is mainly an edge-detection algorithm which creates and emphasizes edges and transitions. Sobel operator is used to detect two kinds of edges in an image which are Vertical direction edges and Horizontal direction edges. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction. More precisely, it uses intensity values only in a 3×3 region around each image point to approximate the corresponding image gradient, and it uses only integer values for the coefficients which weigh the image intensities to produce the gradient approximation.

To say the least, it computes an approximation of the gradient of an image intensity function and combines Gaussian smoothing and differentiation.

Processing steps:

1. Apply Gaussian Blur to reduce noise from source image.
2. Convert image to grayscale.
3. Calculate gradients in X and Y axes (Horizontal and Vertical directions).
4. Approximate the gradient by adding both directional gradients for X and Y axes.





Result: We get bright edges on a darker background

3.3.2 Affine transformation

Affine transformation is any transformation that can be expressed in the form of a *matrix multiplication* (linear transformation) followed by a *vector addition* (translation). As it is a relation between two images, we can take Source and Destination image reference points to apply `warpAffine` and then rotate with respect to the center of the image. An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). While an affine transformation preserves *proportions* on lines, it does not necessarily preserve angles or lengths. Hence, we can see that the resulting transformed video has frames which appear compressed or stretched but are in the same frame and do not go out of the frame. (<http://mathworld.wolfram.com/AffineTransformation.html>)

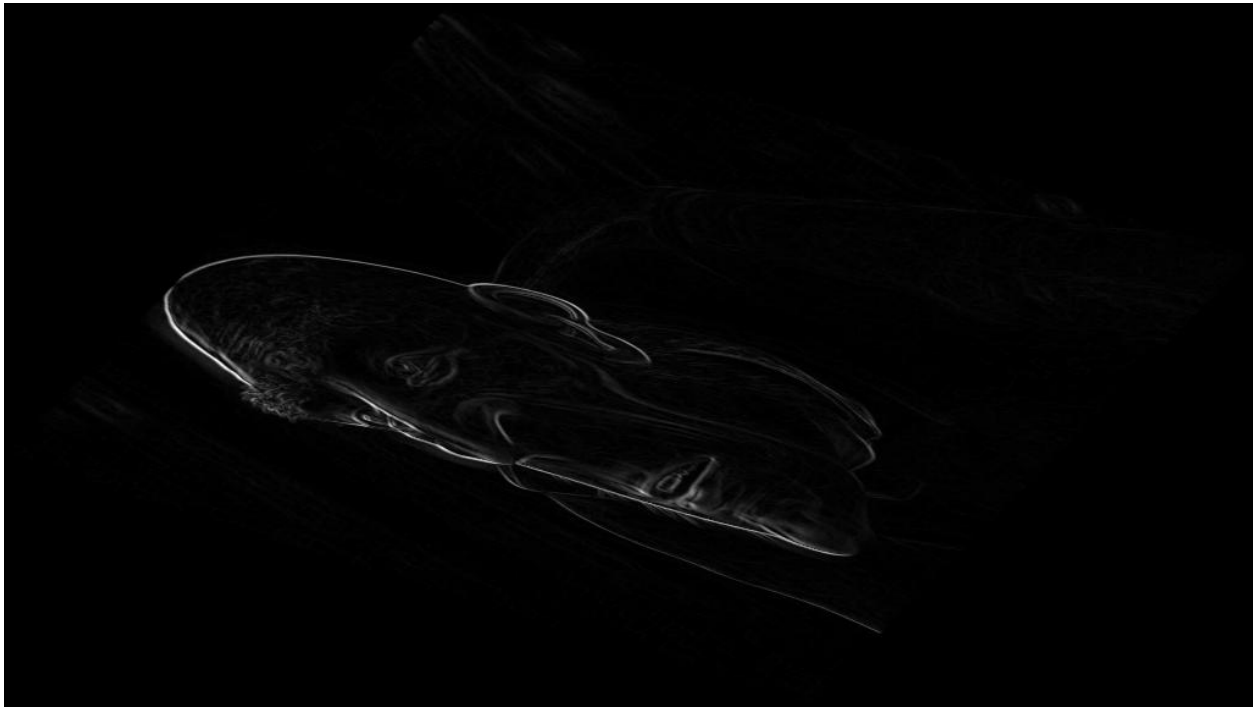
Processing steps:

1. Create blank destination image filled entirely with zeros, matching type and size.
2. Set four points from source (corners) for reference.
3. Draw destination point locations based on the requirement (we do it by section).
4. Calculate Affine Transform from the two source and destination references.
5. Apply the Affine Transform to the image buffer using the resultant matrix.
6. Rotate the image with respect to the center. We use 0.6 as the image scaling factor.
7. Apply rotation using the resultant rotation matrix.



We apply affine transformation to frames which rotate as per section and frame number.

Result:



Finally, we get images that move from the bottom left corner to the bottom right corner and do a 360 degree rotation overall for the video throughout 1 minute and 23 seconds.

4. Code information

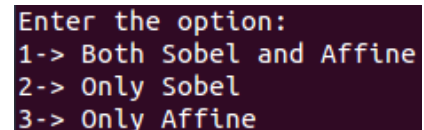
Our algorithm does not one but two image transformations.

We have only used Standard Definition video and no High Definition video because we worked on two transformations, as suggested by professor Siewert.

- ✓ The first image transformation is a Sobel filter, which detects edges and enhances them in the frames for the video.
- ✓ The second one is an Affine Transformation, which warps and rotates the image while preserving the perspective of the picture in the video.

Some features for the single threaded program are:

1. It asks the user for which transformations are to be done...
2. If the user selects option 1, the video will be transformed using both in this order: First, Sobel and then Affine.
3. In the overall flow, the program waits for 3 seconds between both transformations if they are to be done or else it skips the sleep time.
4. We start both Total timer and Selection timer i.e. Sobel or Affine timer depending on the selection.
5. We have files being opened one by one, and filters being applied to each in a 'for' loop. The loop goes on for 2000 files (83.33 seconds*24fps for a 1:23 minute video).
6. Please note that the video file we have provided is 2:30 minutes of runtime since we used 1:23 for profiling analysis.



```
Enter the option:
1-> Both Sobel and Affine
2-> Only Sobel
3-> Only Affine
```

Flow of the program:

1. Wait for user to select the option for frame transformations
2. If both filters are to be done, i.e. if 'option 1' is selected or if only Sobel is selected, apply Sobel filter:
 - a. Open source/frame file
 - b. Do Gaussian Blur to remove noise from source image
 - c. RGB to Grayscale conversion for darker background, in order to have edges visible
 - d. Apply Sobel for Gradient X (Horizontal)
 - e. Apply Sobel for Gradient Y (Vertical)
 - f. Add for total gradient
 - g. Store transformed file
 - h. Repeat until all files are done

3. Check if both filters are to be applied i.e. if option 1 is selected. If yes, sleep for 3 seconds and proceed with Affine transformation or else, print time profile for Sobel filter and Total time then exit
4. Affine transformation to original or Sobel filtered frames:
 - a. Open source/frame file
 - b. Calculate section based on number of input files for the video (in our case it is 500 files per section)
 - c. For each case (every 500 files), apply parameter change values for angle of warp/rotate as needed. 90 degree rotation occurs every 500 frames/20.75 seconds
 - d. Apply warp and save matrix. Then apply resultant matrix to destination matrix for further processing
 - e. Now, use the previously saved destination matrix to apply rotation with a 0.6 scaling factor with respect to the center of the image
 - f. Store transformed file
 - g. Repeat until all files are done
5. Calculate time profile for Affine transform and total timer. If both filters were done, subtract 'sleep time' of 3 seconds from Total time
6. Exit

5. Analysis

5.1

CASE I: No optimization + no Multithreading

We use the following parameters for compilation:

```
b50644@bluebeard:~/Desktop/Project/Source_Code$ make
g++ -O0 -g -o affine affine.c `pkg-config opencv --cflags --libs`
```

```
b50644@bluebeard:~/Desktop/Project/Source_Code$ make
g++ -O0 -g -o affine affine.c `pkg-config opencv --cflags --libs`
b50644@bluebeard:~/Desktop/Project/Source_Code$ ./affine
Enter the option:
1-> Both Sobel and Affine
2-> Only Sobel
3-> Only Affine
1

Starting Sobel Filtering!!
Applying Sobel Filtering
Sobel Filtering done!!

Sleeping for 3 seconds

Starting Affine Filtering!!
Applying Affine Filtering
Affine Transform done!!

-----
| Profiling Statistics | Minutes | Seconds |
-----
| Sobel Transform Timings | 01 | 59 |
-----
| Affine Transform Timings | 02 | 20 |
-----
| Total Transform Timings | 04 | 19 |
-----
b50644@bluebeard:~/Desktop/Project/Source_Code$
```

5.2

CASE 2: No Optimization + Multithreading

1. Timing values:

```

b50644@bluebeard:~/Desktop/Project/Source_Code$ make
g++ -O0 -g -o affine affine.c `pkg-config opencv --cflags --libs`
b50644@bluebeard:~/Desktop/Project/Source_Code$ ./affine
Enter the option:
1-> Both Sobel and Affine
2-> Only Sobel
3-> Only Affine
1

Number of Threads : 20
Number of Frames per Thread : 100

Starting Sobel Filtering!!
Applying Sobel Filtering using multithreading...
Sobel Filtering done!!

Sleeping for 3 seconds

Starting Affine Filtering!!
Applying Affine Filtering using multithreading...
Affine Transform done!!

-----
| Profiling Statistics | Minutes | Seconds |
-----
| Sobel Transform Timings | 02 | 28 |
-----
| Affine Transform Timings | 01 | 44 |
-----
| Total Transform Timings | 04 | 12 |
-----

```

2. Sysprof screenshot:

Descendants	Self	Cumulative ▲
▼ [Everything]	0.00 %	100.00 %
▼ [affine]	0.00 %	79.11 %
▶ No map [affine]	0.00 %	27.52 %
▶ cv::cvtScaleAbs16s8u(short const*, unsigned long, unsigned char co...	15.97 %	16.53 %
▶ void cv::remapBilinear<cv::FixedPtCast<int, unsigned char, 15>, cv::...	9.72 %	9.78 %
▶ _ZNK2cv21SymmRowSmallVec_8u32sclEPKhPhii.part.23	4.66 %	4.69 %
▶ icvCvt_BGR2RGB_8u_C3R(unsigned char const*, int, unsigned char*,...	3.90 %	4.42 %
▶ cv::warpAffine(cv::_InputArray const&, cv::_OutputArray const&, cv...	4.06 %	4.09 %
▶ read	0.01 %	3.03 %
▶ cv::SymmColumnSmallFilter<cv::FixedPtCastEx<int, unsigned char...	2.90 %	2.92 %
▶ cv::cvtColor(cv::_InputArray const&, cv::_OutputArray const&, int, int)	2.53 %	2.56 %
▶ cv::SymmColumnSmallFilter<cv::Cast<int, short>, cv::SymmColumn...	1.28 %	1.95 %

We can observe that Affine on pthreads is much faster in comparison to Affine on single threaded implementation. However, Sobel filter is slower on pthreads – multi-threaded implementation. We use 20 threads to work on 100 files each starting Frame_no_00000 to Frame_no_02000.

5.3

CASE 03: Optimization + Multithreading

1. Timing

```

b50644@bluebeard:~/Desktop/Project/Source_Code$ make
g++ -lpthread -pthread -O3 -mssse3 -malign-double -g -o affine affine.c
b50644@bluebeard:~/Desktop/Project/Source_Code$ ./affine
Enter the option:
1-> Both Sobel and Affine
2-> Only Sobel
3-> Only Affine
1

Number of Threads : 20
Number of Frames per Thread : 100

Starting Sobel Filtering!!
Applying Sobel Filtering using multithreading...
Sobel Filtering done!!

Sleeping for 3 seconds

Starting Affine Filtering!!
Applying Affine Filtering using multithreading...
Affine Transform done!!

-----
| Profiling Statistics | Minutes | Seconds |
-----
| Sobel Transform Timings | 02 | 22 |
-----
| Affine Transform Timings | 01 | 39 |
-----
| Total Transform Timings | 04 | 01 |
-----
b50644@bluebeard:~/Desktop/Project/Source_Code$ 

```

2. Sysprof Screenshot:

Descendants	Self	Cumulative ▲
▼ [Everything]	0.00 %	100.00 %
▼ [affine]	0.00 %	86.57 %
▸ No map [affine]	0.00 %	30.00 %
▸ cv::cvtScaleAbs16s8u(short const*, unsigned long, unsigned char co...	17.34 %	18.08 %
▸ void cv::remapBilinear<cv::FixedPtCast<int, unsigned char, 15>, cv::...	10.41 %	10.48 %
▸ _ZNK2cv21SymmRowSmallVec_8u32sclEPKhPhii.part.23	4.97 %	5.00 %
▸ icvCvt_BGR2RGB_8u_C3R(unsigned char const*, int, unsigned char*,...	3.99 %	4.72 %
▸ cv::warpAffine(cv::_InputArray const&, cv::_OutputArray const&, cv::...	4.38 %	4.41 %
▸ read	0.02 %	3.36 %
▸ cv::SymmColumnSmallFilter<cv::FixedPtCastEx<int, unsigned char...	3.14 %	3.18 %
▸ cv::cvtColor(cv::_InputArray const&, cv::_OutputArray const&, int, int)	2.74 %	2.77 %
▸ cv::SymmColumnSmallFilter<cv::Cast<int, short>, cv::SymmColumn...	1.35 %	2.74 %
▸ In file /lib/x86_64-linux-gnu/libc-2.15.so	0.51 %	0.57 %
▸ munmap	0.01 %	0.18 %
▸ In file /lib/x86_64-linux-gnu/ld-2.15.so	0.01 %	0.14 %
▸ __close	0.01 %	0.09 %
▸ mmap	0.00 %	0.09 %
▸ cv::RemapVec_8u::operator()(cv::Mat const&, void*, short const*, ...	0.04 %	0.07 %
▸ cv::FilterEngine::proceed(unsigned char const*, int, int, unsigned ch...	0.06 %	0.06 %
▸ lseek64	0.01 %	0.05 %
▸ cv::remap(cv::_InputArray const&, cv::_OutputArray const&, cv::_In...	0.04 %	0.04 %
▸ brk	0.00 %	0.04 %
▸ cv::SymmRowSmallFilter<unsigned char, int, cv::SymmRowSmallVe...	0.03 %	0.03 %
▸ In file /usr/local/lib/libopencv_imgproc.so.2.4.0	0.03 %	0.03 %
▸ cv::PxMEncoder::write(cv::Mat const&, std::vector<int, std::allocat...	0.02 %	0.03 %
▸ __fxstat	0.00 %	0.02 %
▸ _IO_un_link	0.02 %	0.02 %
▸ malloc	0.02 %	0.02 %
▸ _IO_fwrite	0.02 %	0.02 %

We can observe that the GCC compiler optimization along with SSE instruction flag `-mssse3` and POSIX pthreads, execution makes a good difference.

5.4

CASE 04: Only GCC Optimization + No Multithreading

1. Timing

```

b50644@bluebeard:~/Desktop/Project/Source_Code$ make
g++ -O3 -g -o affine affine.c `pkg-config opencv --cflags --libs`
b50644@bluebeard:~/Desktop/Project/Source_Code$ ./affine
Enter the option:
1-> Both Sobel and Affine
2-> Only Sobel
3-> Only Affine
1
Starting Sobel Filtering!!
Applying Sobel Filtering using multithreading...
Sobel Filtering done!!

Sleeping for 3 seconds

Starting Affine Filtering!!
Applying Affine Filtering using multithreading...
Affine Transform done!!

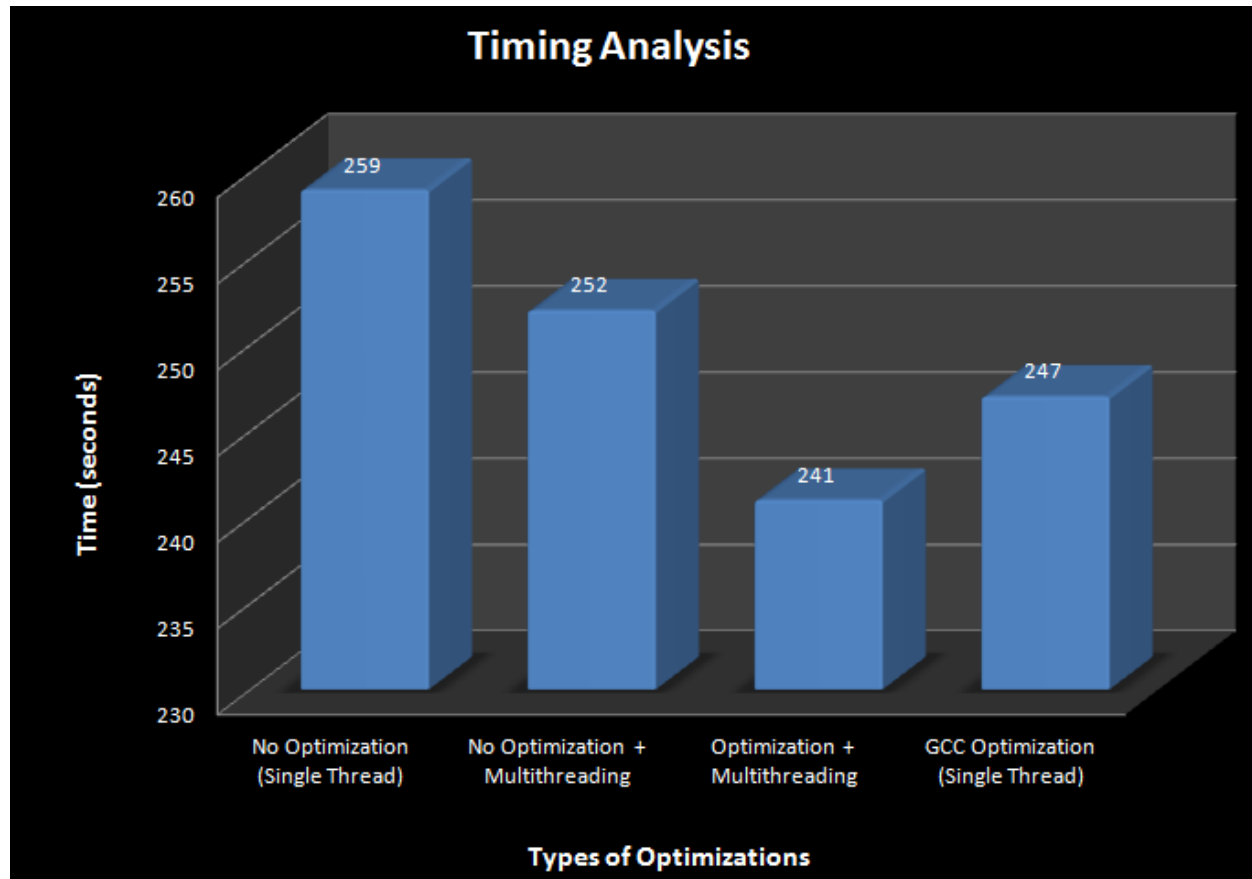
-----
|   Profiling Statistics   |   Minutes   |   Seconds   |
-----
| Sobel Transform Timings |     02     |     03     |
-----
| Affine Transform Timings |     02     |     04     |
-----
| Total Transform Timings |     04     |     07     |
-----
b50644@bluebeard:~/Desktop/Project/Source_Code$ 

```

2. Sysprof Values:

Descendants	Self	Cumulative ▲
▼ [Everything]	0.00 %	100.00 %
▼ [affine]	0.00 %	86.86 %
▶ No map [affine]	0.78 %	28.86 %
▶ cv::cvtScaleAbs16s8u(short const*, unsigned long, unsigned char co...	17.16 %	18.07 %
▶ void cv::remapBilinear<cv::FixedPtCast<int, unsigned char, 15>, cv::...	13.23 %	13.24 %
_ZNK2cv21SymmRowSmallVec_8u32sclEPKhPhii.part.23	5.07 %	5.07 %
▶ cv::warpAffine(cv::_InputArray const&, cv::_OutputArray const&, cv...	4.30 %	4.31 %
▶ icvCvt_BGR2RGB_8u_C3R(unsigned char const*, int, unsigned char*,...	3.38 %	4.07 %
▶ cv::SymmColumnSmallFilter<cv::FixedPtCastEx<int, unsigned char...	3.04 %	3.04 %
▶ cv::SymmColumnSmallFilter<cv::Cast<int, short>, cv::SymmColumn...	1.23 %	3.02 %
▶ read	0.01 %	3.00 %
cv::cvtColor(cv::_InputArray const&, cv::_OutputArray const&, int, int)	2.69 %	2.69 %
▶ In file /lib/x86_64-linux-gnu/libc-2.15.so	0.44 %	0.50 %
▶ munmap	0.00 %	0.13 %
▶ In file /lib/x86_64-linux-gnu/ld-2.15.so	0.01 %	0.13 %
▶ __close	0.00 %	0.09 %
▶ mmap	0.00 %	0.07 %
cv::remap(cv::_InputArray const&, cv::_OutputArray const&, cv::_In...	0.05 %	0.05 %
cv::FilterEngine::proceed(unsigned char const*, int, int, unsigned ch...	0.05 %	0.05 %
▶ lseek64	0.01 %	0.04 %
▶ cv::RemapVec_8u::operator()(cv::Mat const&, void*, short const*, ...	0.03 %	0.03 %
cv::SymmRowSmallFilter<unsigned char, int, cv::SymmRowSmallVe...	0.03 %	0.03 %
In file /usr/local/lib/libopencv_imgproc.so.2.4.0	0.02 %	0.02 %
▶ __fxstat	0.00 %	0.02 %
▶ _IO_fwrite	0.02 %	0.02 %
cv::PxMEncoder::write(cv::Mat const&, std::vector<int, std::allocat...	0.02 %	0.02 %
▶ write	0.00 %	0.02 %
cv::checkHardwareSupport(int)	0.02 %	0.02 %
malloc	0.01 %	0.01 %

5.5 Overall Analysis:



1. No optimization + no Multithreading:

As expected, the no optimization single threaded code was the most time consuming. Please note that we have inserted respective code for the same under the 'Archive' folder. Else, one can also verify the code using the 'Source_Code' files and modifying the NUM_THREADS to 1 and NO_OF_FRAMES_PER_THREAD to 2000. One can also modify the set_affinity function that is commented in the affine.c file.

2. No optimization + Multithreading:

The next computation involved having 20 threads and 100 frames per thread. Note that there was no optimization, meaning that during 'run'ing the code, we add '-O0' and NOT 'O3'. It showed sizeable decrease in time.

3. Optimization + Multithreading:

This part involved optimizing using '-O3 -mssse3 -malign-double'. The thread implementation was done by having 20 threads that are assigned 100 frames each. Predictably enough, this section showed maximum optimization.

4. Only GCC Optimization + No Multithreading:

The last part was implementing only GCC optimization onto a single threaded code. The time it clocked was 4 minutes and 7 seconds.

6. Output Videos Specification:

6.1 Only Sobel Video:

```
bhaumik@bhaumik-VirtualBox:~/Documents/rtdm$ avprobe Sobel.mp4
avprobe version 0.8.13-4:0.8.13-0ubuntu0.12.04.1, Copyright (c) 2007-2014 the Libav develop
ers
built on Jul 15 2014 12:56:47 with gcc 4.6.3
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'Sobel.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2mp41
  encoder          : Lavf53.21.1
Duration: 00:02:24.04, start: 0.000000, bitrate: 16676 kb/s
Stream #0.0(und): Video: mpeg4 (Simple Profile), yuv420p, 1024x576 [PAR 1:1 DAR 16:9],
16674 kb/s, 25 fps, 25 tbr, 25 tbn, 25 tbc
```

VLC Statistics:

▼ Stream 0	
Type: Video	
Codec: MPEG-4 Video (mp4v)	
Resolution: 1024x576	
Frame rate: 25	
Decoded format: Planar 4:2:0 YUV	
▼ Audio	
Decoded	0 blocks
Played	0 buffers
Lost	0 buffers
▼ Video	
Decoded	790 blocks
Displayed	749 frames
Lost	20 frames
▼ Input/Read	
Media data size	118452 KiB
Input bitrate	14197 kb/s
Demuxed data size	118396 KiB
Content bitrate	12939 kb/s
Discarded (corrupted)	0
Dropped (discontinued)	0
▼ Output/Written/Sent	
Sent	0 packets
Sent	0 KiB
Upstream rate	0 kb/s

6.2 Only Affine Video

```

bhaumik@bhaumik-VirtualBox:~/Documents/rtdm$ avprobe Sobel.mp4
avprobe version 0.8.13-4:0.8.13-0ubuntu0.12.04.1, Copyright (c) 2007-2014 the Libav develop
ers
built on Jul 15 2014 12:56:47 with gcc 4.6.3
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'Sobel.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2mp41
  encoder          : Lavf53.21.1
Duration: 00:02:24.04, start: 0.000000, bitrate: 16676 kb/s
  Stream #0.0(und): Video: mpeg4 (Simple Profile), yuv420p, 1024x576 [PAR 1:1 DAR 16:9],
16674 kb/s, 25 fps, 25 tbr, 25 tbn, 25 tbc

```

VLC Statistics:

▼ Audio	
Decoded	0 blocks
Played	0 buffers
Lost	0 buffers
▼ Video	
Decoded	1711 blocks
Displayed	1687 frames
Lost	13 frames
▼ Input/Read	
Media data size	47690 KiB
Input bitrate	2805 kb/s
Demuxed data size	47649 KiB
Content bitrate	2642 kb/s
Discarded (corrupted)	0
Dropped (discontinued)	0
▼ Output/Written/Sent	
Sent	0 packets
Sent	0 KiB
Upstream rate	0 kb/s

6.3 Sobel+Affine Video

```

bhaumik@bhaumik-VirtualBox:~/Documents/rtdm$ avprobe Sobel+Affine.mp4
avprobe version 0.8.13-4:0.8.13-0ubuntu0.12.04.1, Copyright (c) 2007-2014 the Libav develop
ers
built on Jul 15 2014 12:56:47 with gcc 4.6.3
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'Sobel+Affine.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2mp41
  encoder          : Lavf53.21.1
Duration: 00:02:24.04, start: 0.000000, bitrate: 11954 kb/s
Stream #0.0(und): Video: mpeg4 (Simple Profile), yuv420p, 1024x576 [PAR 1:1 DAR 16:9],
11952 kb/s, 25 fps, 25 tbr, 25 tbn, 25 tbc

```

VLC Statistics:

▼ Audio	
Decoded	0 blocks
Played	0 buffers
Lost	0 buffers
▼ Video	
Decoded	404 blocks
Displayed	367 frames
Lost	16 frames
▼ Input/Read	
Media data size	23962 KiB
Input bitrate	13656 kb/s
Demuxed data size	23913 KiB
Content bitrate	16198 kb/s
Discarded (corrupted)	0
Dropped (discontinued)	0
▼ Output/Written/Sent	
Sent	0 packets
Sent	0 KiB
Upstream rate	0 kb/s

Codec information supplied by VLC media player – Same across output files:

▼ Stream 0
Type: Video
Codec: MPEG-4 Video (mp4v)
Resolution: 1024x576
Frame rate: 25
Decoded format: Planar 4:2:0 YUV

This is because we used '**-vcodec mpeg4**' on FFmpeg while putting together a video file using transformed PPM images as frames.

Looking at '**bitrate**' values shows a difference in the three output files, mainly dependent on the amount of detail and CPU a transformation has utilized.

7. Conclusions

This was certainly a mini-post-production environment simulation drill for us and we have learned a lot on the aspects of CPU utilization, offloading, load-balancing, POSIX pthreads apart from designing the algorithm, working with image processing, multiple frame transforms, color filters and use of OpenCV.

It was observed that it is a very time-consuming and computationally intensive process of editing video frames and working with raw images to apply color filters, correction and effects on digital video. It comprises of all factors such as CPU, RAM and storage. Hardware acceleration is thus best employed when working with graphics, video or images for such heavy-duty purposes.

We attempted a number of things, and at first, we were stuck on 'How to get the image to rotate as per the time to get a complete 360 degree rotation by the end of the video?' Brainstorming and adding workarounds, with the trial-error process solved our issues. We added another reference point in order to make sure we are headed in the right path for the warp-rotate affine transformation. It sure helped clear our understanding.

Our attempt at changing the scheduling policy from SCHED_OTHER to SCHED_FIFO worsened the execution time and was not considered due to that. It was due to wait times for threads, to be able to execute in sequential queue mannerism.

Another attempt at using CUDA GPU namespace for the OpenCV supplied code we had written and modified, was not considered. This was due to the unavailability of a native machine with an NVIDIA GPU, as the ecee-gpu4 machine did not compile with OpenCV. Hence, usage of OpenCV GPU module was 'out of the picture' for us.

8. References

1. <http://www.elephantsdream.org/>
2. <http://docs.opencv.org/>
3. http://en.wikipedia.org/wiki/Sobel_operator
4. http://en.wikipedia.org/wiki/Affine_transformation
5. <http://mathworld.wolfram.com/AffineTransformation.html>
6. http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html
7. http://docs.opencv.org/2.4.8/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html
8. <http://opencvexamples.blogspot.com/2013/10/sobel-edge-detection.html>
9. http://www.tutorialspoint.com/java_dip/applying_sobel_operator.htm
10. <http://stackoverflow.com/questions/5172107/c-string-and-int-concatenation>

fin