Sudarshan Singh Bisht

GPU Accelerated Image and Video Processing

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

25 November 2023

Abstract

Author:	Sudarshan Singh Bisht
Title:	GPU Accelerated Image and Video Processing
Number of Pages:	34 pages + 2 appendices
Date:	25 November 2023
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services / Medical Technology
Supervisors:	Ville Jääskeläinen, Principal Lecturer

Central Processing Unit (CPU) is called the brain of the computers and the mobile devices and designed in such a way that it performs best in executing tasks sequentially. So, is the CPU alone fast enough when it comes to perform an operation on high quality images and videos to provide the best user experience? Graphics Processing Unit (GPU) is a dedicated hardware which is mainly used for rendering graphics; however, its general-purpose nature allows it to be used in other generic tasks where high level of parallelism is required.

The objective of this thesis is to find out how the GPU plays essential role when it comes to process the images and the videos and what makes the GPU so efficient to process them compared to the CPU. A comparative study is made in this thesis to analyse why the GPU has an edge over the CPU when images and videos need to be processed fasters. This is verified with the help of some tests which were performed on a laptop computer equipped with Intel CPU and Nvidia GPU. Tests results clearly show that the GPU outperforms the CPU with 4 to 6 times faster processing rate. But the same time processing power of the GPU can not be accessed directly, the CPU's involvement is something which is a must in order to access GPU's functionality.

Keywords: Images, Videos, CPU, GPU, Processing

Contents

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
4K	Four times Full High-Definition Video (3840x2160)
5G	Fifth-generation technology standard for cellular networks
8K	Eight times Full High-Definition Video (7680x4320)
API	Application Programming Interface
B-frame	Bidirectional Predictive Frame
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DCT	Discreate Cosine Transform
DFT	Discreate Fourier Transform
DWT	Discreate Wavelet Transform
FLOPS	Floating Point Operations Per Second
Full HD	Video resolution of 1920x1080 pixels
GHz	Gigahertz, unit of CPU’s processing power
GLSL	OpenGL Shading Language
GPU	Graphics Processing Unit
GPGPU	General Purpose GPU
HEVC	High Efficiency Video Coding
I-frame	Intra Coded Frame
IPU	Image Processing Unit
JPEG	Joint Photographic Experts Group
MB	Mega Bytes
MHz	Megahertz
MM	Multimedia Framework
OpenGL	Open Graphics Layer
P-frame	Predictive Frame
PCI	Peripheral Communication Interconnect
RAM	Random Access Memory
RGB	Red Green Blue colour component
Y42B	Four-character code for a video format supported by Nvidia

YUY2	Four-character code for a video format with 4:2:2 chroma sampling

1 Introduction

Images and videos are ubiquitous, their respective use cases are found in the field of entertainment, scientific research, industrial automation, medical sector and consumer applications. The reason behind their large-scale usages is that the current era we are living in, where revolution is witnessed currently in the field of multimedia, which is mostly led by the affordable and high-speed internet data. Along with it, less expensive smart phones, tablets, laptops and desktop computers boosted the growth of the images and videos used in the consumer market. With the evolution of internet speed, image and video sharing related use cases have increased by many folds. Some of the prominent use cases are live video streaming, video calls, video on demand, applying filters on the images etc. On YouTube, which has 15 million content creators and 2.6 billion active users worldwide, over 400 hours of videos are uploaded every minute and 694000 hours of video content is watched per minute. [1] Instagram was launched in 2010, since then 50 billion images and videos are shared on it. 1074 images are uploaded on this content sharing platform every single second. [2] Facebook alone have gotten 250 billion images uploaded so far and 350 million images are uploading each day. [3] In the year 2021, 8.6 billion videos were uploaded in TikTok. [4] Globally WhatsApp users make over 55 million video calls per day, which means spending more than 340 million minutes per day on these calls. [5] All these platforms and services are somewhere leveraging on the basic concepts of image and video processing.

For image and video processing applications, an image or a video must be captured first. This task is performed by a camera module, it could be a standalone camera or a smart phone camera. Camera module contains several other components, such as lens, sensor, Image Processing Unit (IPU) etc. Whenever sun light or electric light strikes on the object, it starts emitting photons. These photons are directed via lens to the sensor, then sensor generates an electrical signal. IPU converts this electrical signal into digital data, which is usually referred as a raw image or a raw video data. Till this point the data is uncompressed, therefore the size is very large, which is practically very difficult

to handle by most of the applications. To resolve this issue, raw data must be compressed to make it suitable for application usage. And compression is one of the most important steps of image and video processing pipeline. There are many image and video compression standards have been established by company consortiums to ensure interoperability. Joint Photographic Experts Group (JPEG) and High Efficiency Video Coding (HEVC) are couple of them. The compression ratio what JPEG could offer ranges between 1.1:1 to 50:1. This compression ratio also depends on the quality factor, the lower the quality factor the better the compression is. [6] HEVC, the most proliferated video compression technology these days, lowers down the bitrate requirement of 4K video transmission to 19 Mbps at 30 frames per second rendering rate. [7] Without this compression we would need staggering 5.8 Gbps of bitrate and there is no practical network which could bear this much load. With 5G, it is apparent that the internet speed is increasing but it cannot be denied that the video content resolution has not stagnated to Full HD, in fact it is also evolving as 4K, or 8K video content delivery is becoming more and more common these days to provide users better quality videos. Therefore, compression technology will also evolve in tandem with internet speed to suffice the need of consumers.

By now it’s been very well understood that the compression makes image and video data sharables and transmissible as it reduces the size by many folds. Without compression techniques it would be difficult to imagine that YouTube, Facebook, Instagram, TikTok, WhatsApp and many other companies had ever existed as their primary business revolves around image and video-based services. But what is the best way to process images and videos on desktop computers, laptops and mobile devices? These days desktop computers, laptops and mobile devices such as mobile phones and tablets are equipped with powerful Central Processing Unit (CPU) then why there is a need of Graphics Processing Unit (GPU) in them? This thesis conducts studies that how the CPUs and the GPUs behave differently when it comes to processing images and videos. This can be demonstrated with help of test system developed during this thesis. The main goal of this thesis is to answer the question: Why GPUs are preferred over CPUs when it comes to process images and videos in real time.

This thesis has been divided in 5 sections, the introduction part briefly describes the use cases of the images and the videos; and how encoding and decoding operations help them make suitable to be used by the users. The next section talks about fundamentals of the images and the videos and various processing techniques those can be applied on them to make them viable for information extraction or transformation. After that the evolution of the CPU and the GPU is introduced along with their respective basic architecture and key differences. Then comes a comparative study where a few tests were performed to measure latencies when encoding and decoding operations were performed, first on the CPU and then on the GPU. In the end of the thesis, conclusion part covers the analysis of abovementioned tests and recommendation for someone who wants to perform image and video processing related tasks on their computer.

2 Images, Videos, and their Processing

This section introduces the basic concepts about the images and the videos, such as how they are formed and what basic characteristics they have etc. Further some explanation is provided about what kind of processing can be applied on them to transform them into suitable desired form.

2.1 Images

Image is a projection of Three-Dimensional (3D) scene onto Two-Dimensional (2D) plane. In mathematical terms, an image is a two-dimensional function, it is represented by a function, such as f(x,y), where x and y are spatial coordinates, then the function results into the intensity or colour of the image at the coordinate (x,y).

When an image is stored on a computer or on a mobile device, usually it remains in a compressed form (mostly in JPEG format, as this is most widely used format to store images digitally), but when opened with some application and displayed on the screen then they get transformed into uncompressed format, although JPEG is a lossy compression method, so there would be some degradation in the quality of the picture compared to the uncompressed form, but these degradations are so minor that a human eye can hardly detect them. For most of the image processing tasks, uncompressed images are required. So, it's better to understand first how these uncompressed images look. Basically, such images are composed of small singular units, called pixels. The word "pixel" is made by combining two words picture + element. Each pixel in the image contains Red, Green and Blue (RGB) colour components. These components are of 8 bits (one byte) in size. Optionally there might be an alpha component which decides the transparency level of the pixel. All these pixels are laid out in a form of a grid. Number of rows and columns in this grid are often referred to as height and width of the image respectively. Following figure represents an uncompressed or a raw image, which has total 64 pixels.

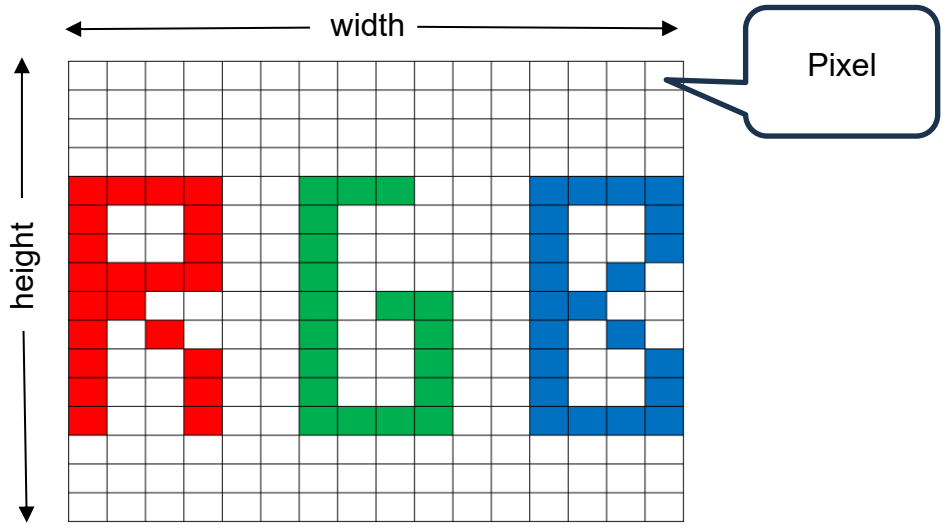


Figure 1. An uncompressed image and pixel components.

2.2 Image Processing

Image processing techniques are carried out to manipulate and analyse a given image in order to improve its quality and usability. With the help of a computer or a mobile device, image processing contributes towards handling, transmission, enhancement, and analysis of a given image. Broadly there are three types of processing that are applied to an image. [8]

2.2.1 Low-level Processing

This kind of processing focuses on image enhancement and image restoration. It talks about primitive operations such as noise reduction, edge sharpening, zooming, contrast improvement, deblurring etc. Both, Input, and output for this kind processing are images. Figure below explains how zooming effect takes place in an image. A simple bilinear interpolation method is used, where two neighbouring pixels are averaged out in order to create a new pixel. This newly formed pixel gets very well blended between two pixels as the value of it is very close to the already existing pixels.

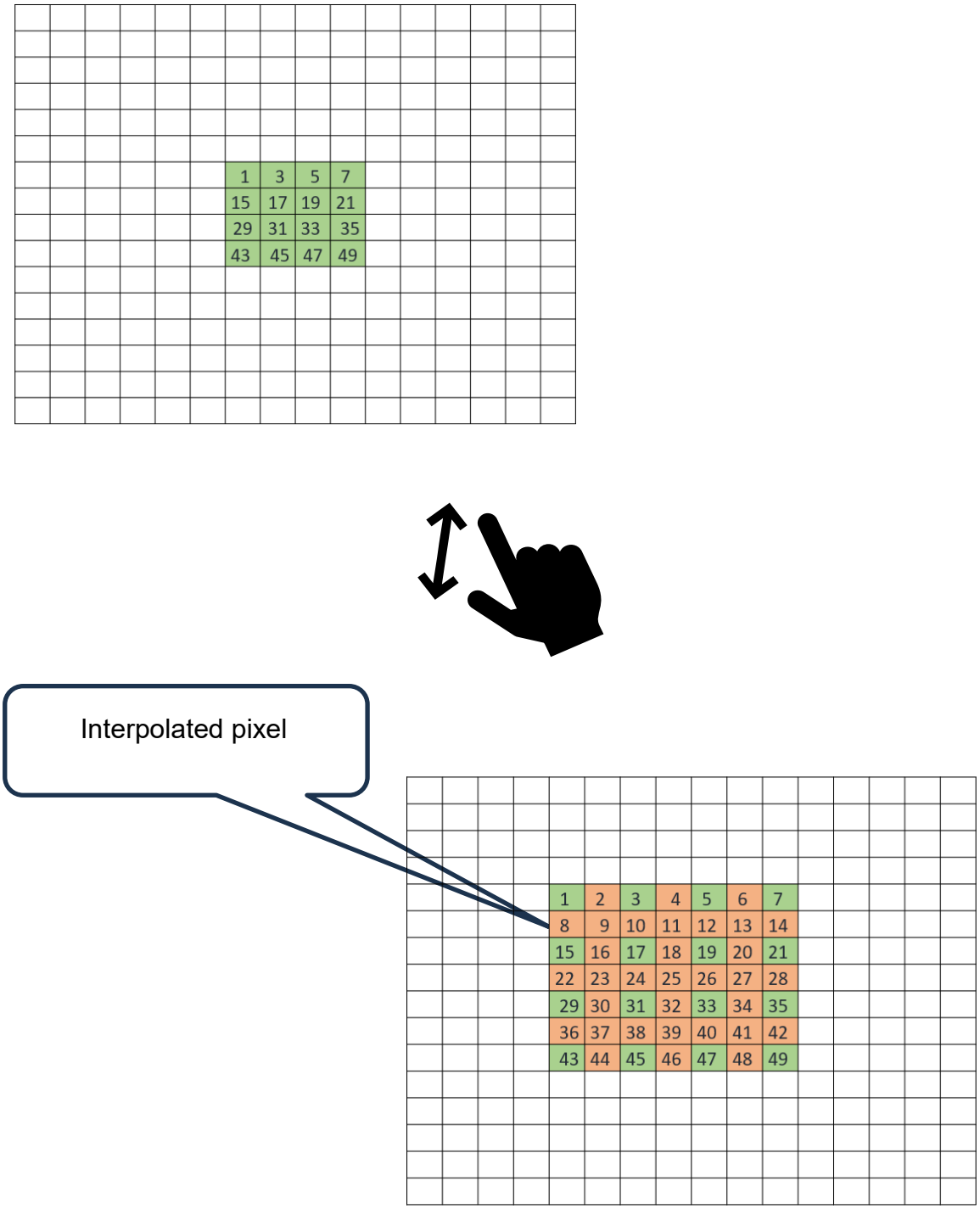


Figure 2. How zooming can be done using interpolation.

2.2.2 Intermediate-level Processing

Image transformation is done in this kind of processing, where images are converted from spatial domain to transform domain such as 2D Discrete Fourier

Transform (DFT), 2D Discreate Cosine Transform (DCT) etc. Basically, these transforms (or in simple terms mathematical formulas) convert image data into frequencies that image data contain. There are certain characteristics of the images which cannot be identified in the spatial domain, therefore transform domain is helpful to know them. Operations such as dimension reduction and feature extraction are performed under this kind of processing. Following figure shows how storage requirement gets lowered down when DCT is applied on 16x16 block of image. Low frequency components get concentrated on the top left side of the grid and higher frequency component on the right bottom side. High and mid frequency components can be ignored as it is less perceptive to human eyes. Therefore, DCT is considered one of the most important steps of image compression algorithm (JPEG) as it reduces the redundant data from the given input. [9] Although there are several additional steps involved in JPEG compression such applying weight table, normalizing and quantizing frequency components, which compliments DCT.

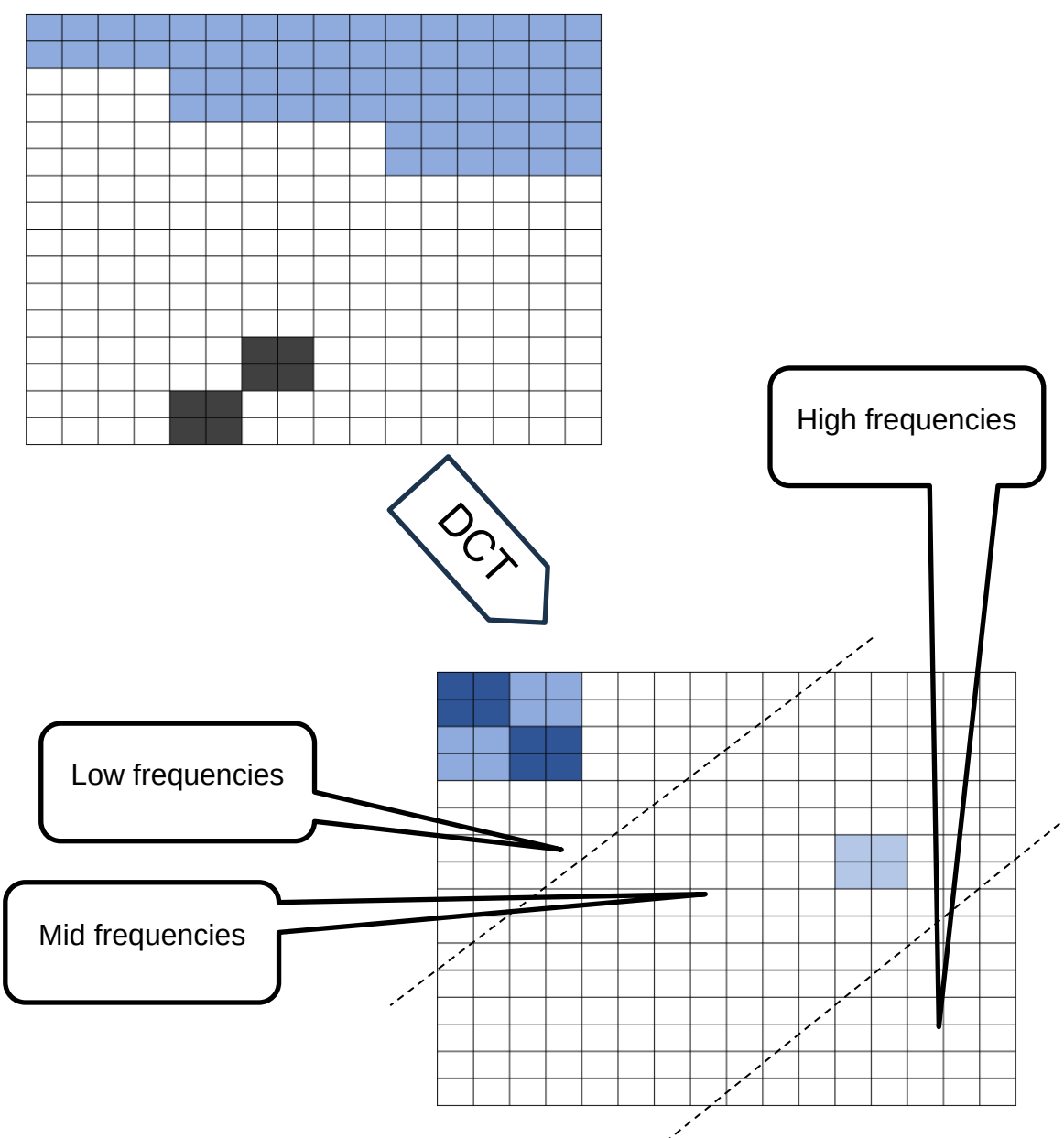


Figure 3. DCT operation transforms image from spatial domain to frequency domain.

2.2.3 High-level Processing

This is mainly used in computer vision domain where multiple objects and their relationship is determined, which ultimately provides vital information about the captured scene. Segmentation, feature extraction and classification operations make it possible to interpret the content. Segmentation process basically identifies and isolates a particular object from the scene so that it's feature can be detected precisely, then desired features are extracted from segmented objects, finally classification is done based on the quantitative evaluation of the extracted features.

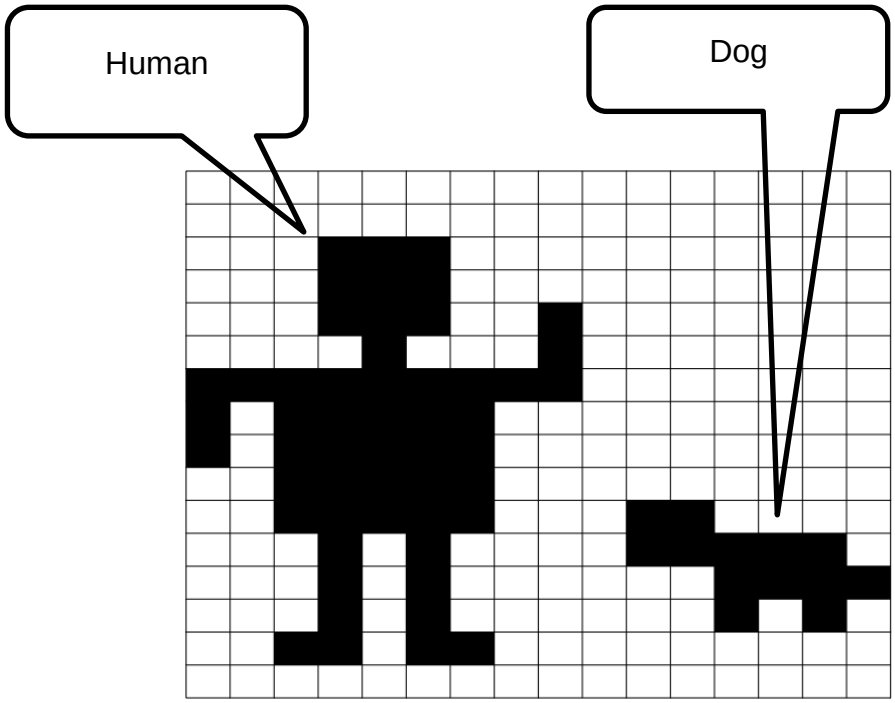


Figure 4. Using computer vision objects can be tracked and localised.

2.3 Video

Video is nothing but a collection of several images displayed in a series. These images in the video are often called frames. Camera captures each frame at certain frame rate such as 24, 30 or 60 frames per seconds, then a video player plays it at the same frame rate. The higher the frame rate, the better the visual quality looks, but for human visual system it must be at least 24 frame per second, otherwise it looks sluggish and causes discomfort to the viewer. As videos are made of several images in a sequence, they inherit certain attributes from images, for example they both are digitized spatially, but as the videos have a motion over the time, that makes them digitized temporally as well. Therefore, they can be operated in both spatial and temporal domain.

Video compression techniques take advantage of the fact that the videos are being digitalized spatially and temporally both. High Efficiency Video Coding (HEVC), also known as H.265 is the latest standard for such compression. It tries to mitigate the redundancies present in the captured scene by processing video frames in both spatial (as image compression does by using DCT) and temporal domain. Frames are categorised as Intra Coded Frames (I-frame), Predictive Frame (P-frame) and Bidirectional Predictive Frame (B-frame) in temporal domain. I-frames are coded independently, they mostly reply on spatial compression, therefore they require higher number of bits to get encoded. P-frames are encoded based on previously encoded I-frames or P-frames, so the bits required to encode them is lower than I-frames. B-frames can be thought of a frame which is interpolated from past and future frames; therefore, they tend to be very efficient in compressing the size of the video, but the same time bring computational complexity as the process to create them is resource intensive. A group of frames where first and last frames are an I-frame is called Group of Pictures (GOP).

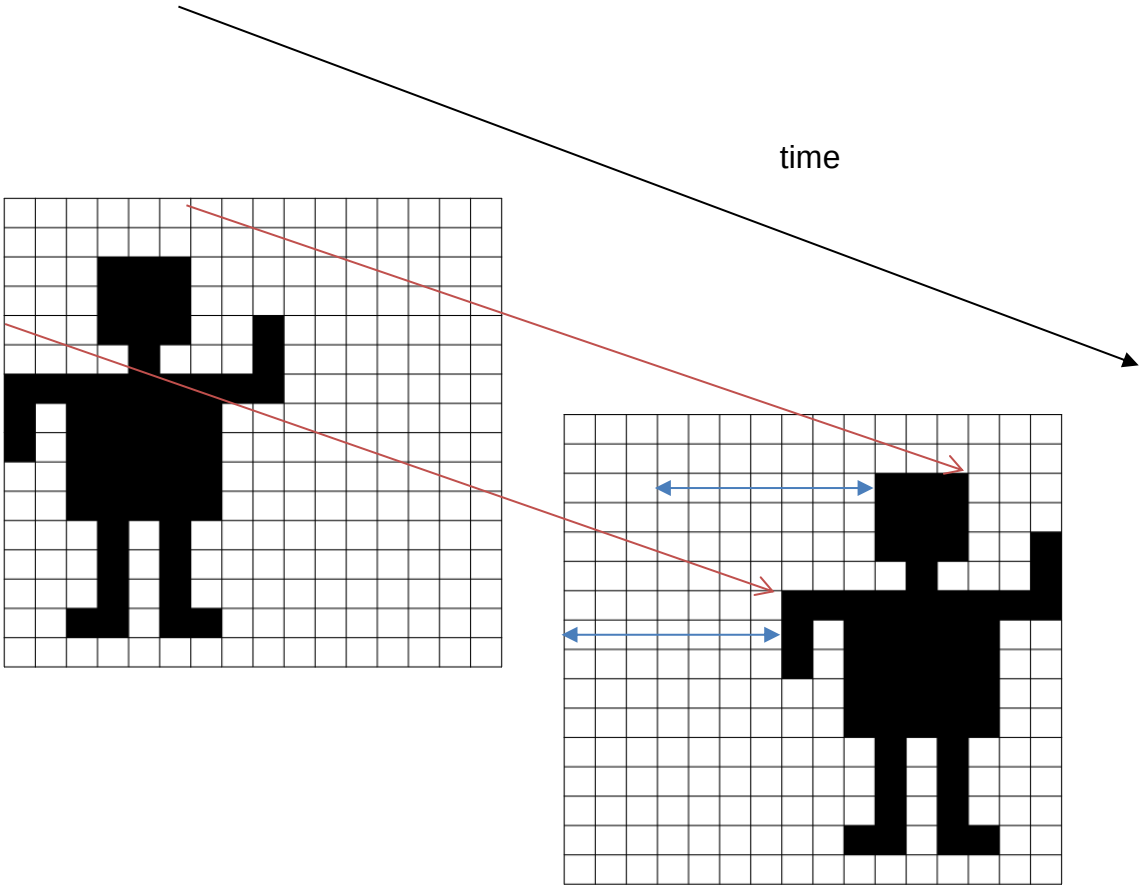


Figure 5. Video compression takes temporal difference into account.

2.4 Video Processing

Video processing methods result into extracting statistical information by applying relevant filters. Some of the basic video processing techniques are video trimming, re-sizing, cropping, brightness and contrast adjustment, fade in and fade out etc. Advanced video processing techniques takes usages of computer vision technology to perform actions such as face recognition and tracking, object detection and tracking. [10] Colour space conversion, is an example of basic video processing where each pixel is transformed into different format to reduce the video frame size. Although it is used in JPEG compression as well but can pertain to video frames.

2.4.1 Colour space conversion

Colour space conversion is a basic method to reduce number of bytes needed to store a video on a device with a guarantee that there will not be much degradation in the visual quality of the same video. This method is not a part of compression technique as such but more related to video processing pipeline. Assume a video is recorded for one second at 30 frames per second of framerate with Full-HD resolution. How much size does it need to store that one second worth of raw video data in device memory? Answer is 1920x1080x3x30 = 186 624 000 bytes (~177 Mega Bytes (MB)). This is just one second of raw video. Now it can be imagined that how much memory would be needed to store for hours of long videos. However, there will be compression applied to it with the help of latest video encoding standards such as HEVC/H.265 to reduce the size down, so that it does not take much storage on the device, as well as makes it suitable for real time video streaming purposes. But even before actual compression, there is a possibility to reduce the raw video frame size by taking advantage of the human visual system. The human eye is more perceptive towards black and white colours (often called brightness or luminance or luma) than colours such as read, green and blue (often called chrominance or chroma). Hence, a conversion is applied to RGB video data, so that chrominance can be extracted from the RGB components for each pixel and kept in dominance. Whereas luminance portion is reduced to half. YUV420 model is very commonly used standard which operates on chroma subsampling, where chrominance is stored at half of the lower resolution than the luminance. Which sums up to that the YUV420 model takes half size for each pixel than RGB model (1.5 bytes, compared to 3 bytes taken by RGB), i.e., ~ 88 MB for one second of raw video data.

Following figure depicts how the same frame can be represented in two colour formats. The top one is the RGB format with red, green and blue colour components whereas the bottom one is the YUV format with luminance (Y) and chrominance (UV) components.

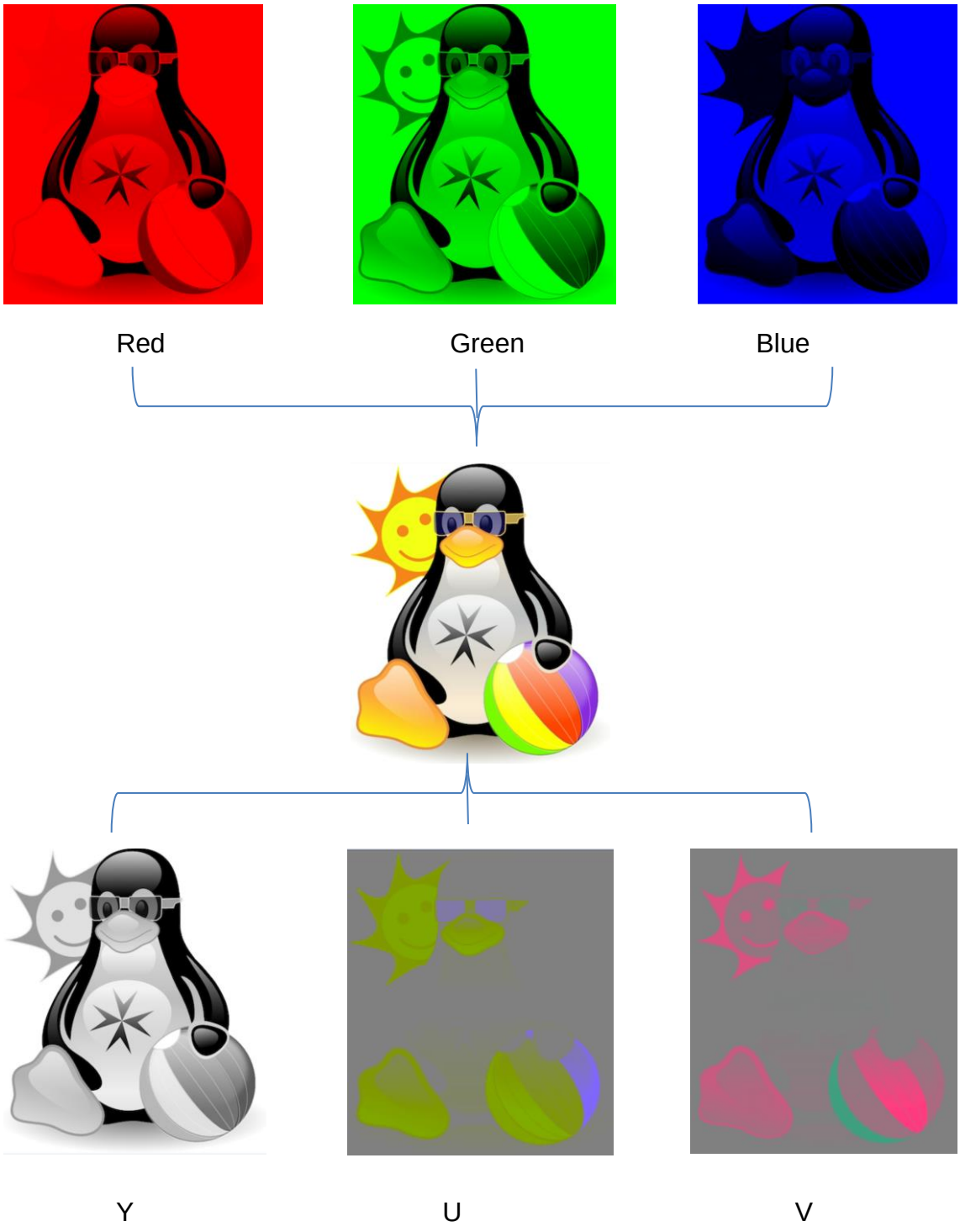
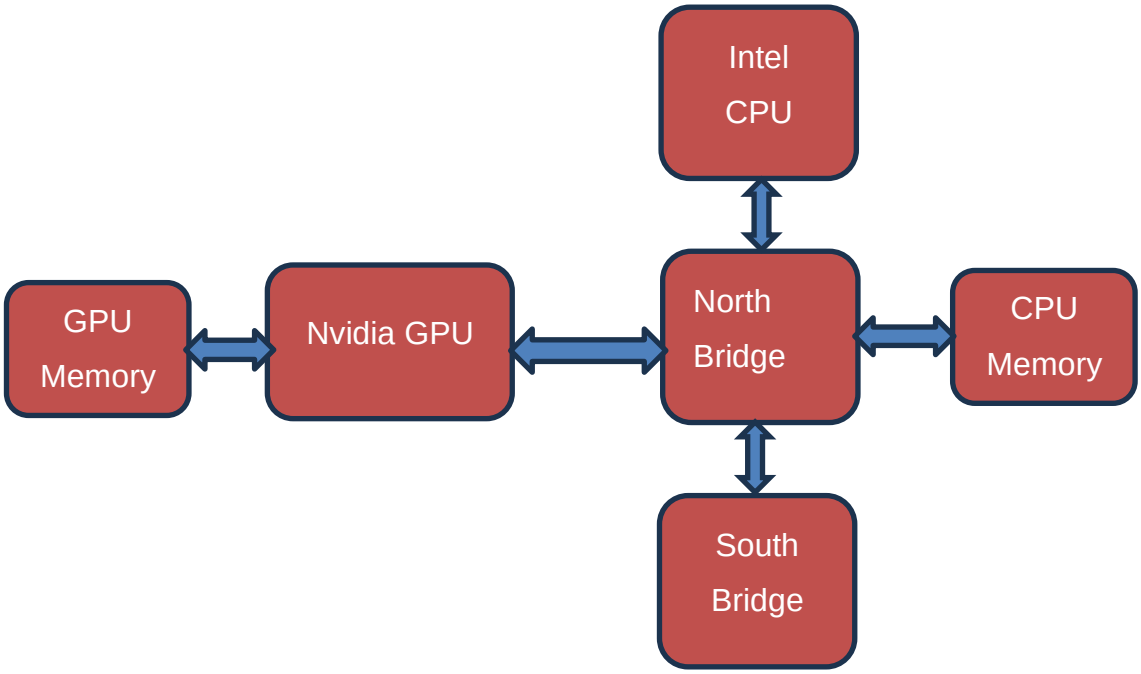


Figure 6. Different colour formats represent the same frame [11].

3 The CPU, the GPU and the key differences

This section describes the basic design and functioning of the CPU and the GPU. What are the key differences what makes them suitable for their own applications. Although the CPU and the GPU are connected to each other in modern computers, the communication between them occurs via a PCI-Express Link. This link provides a peak transfer rate of 16GB per second, meaning 8GB per second in each direction. Still this speed is much lower than the speed when they access their own respective memories. On the Intel CPU side, North Bridge manages communication between the CPU and other components of the motherboard, in this case the GPU and the CPU memory. South Bridge mainly provides connectivity with the input and output devices such as disk interface.



Picture 7. Modern computers contain Intel CPU and Nvidia GPU, which are connected with each other via a high-speed PCI-Express link.

3.1 The CPU

CPU is the main component of an electronic device. Basically, it consists of control unit and arithmetic & logic unit. The main task of control unit to supervise the operations undertaken by the CPU, that includes controlling the transfer of data and fetching data from the memory unit. Arithmetic & Logic Unit (ALU), as name suggests, does arithmetic operations such as addition, subtraction, multiplication and division, also it performs logical operations such as comparison and selection. CPU works closely with the Random Access Memory (RAM) for storing instructions, data and results.

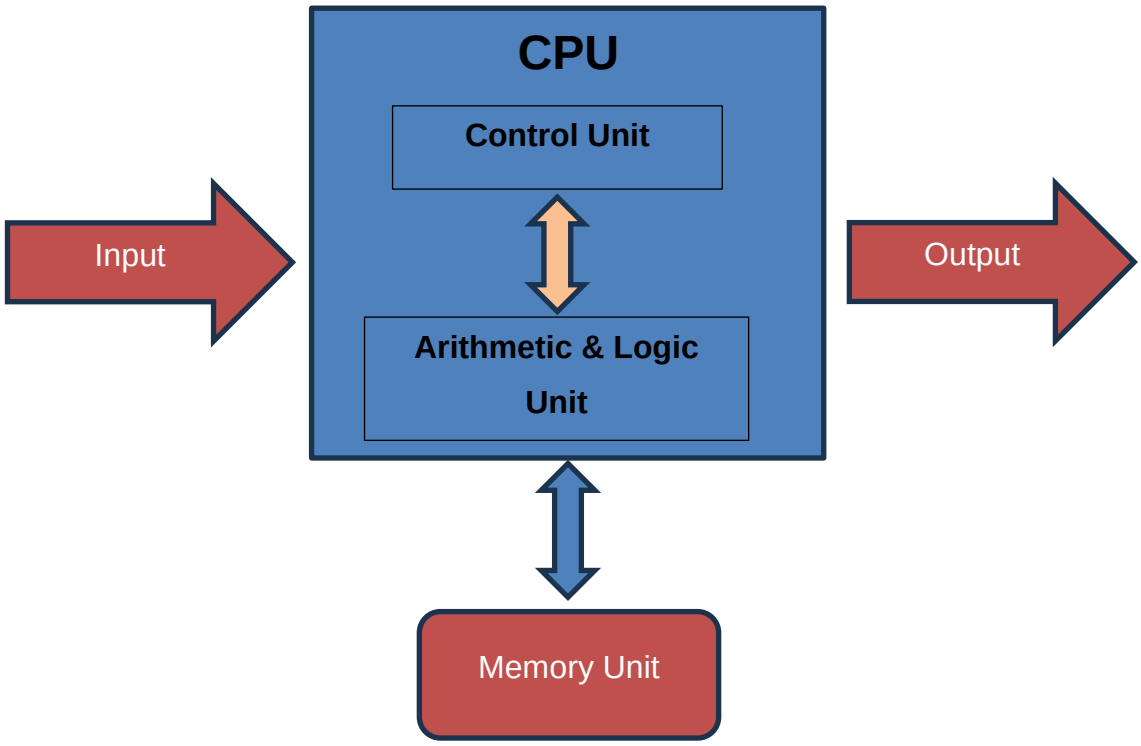


Figure 8. Main components of the CPU and memory units.

Processing power of the CPU is measured based on how many cycles it can run in a second, often it is denoted by clock speed. In modern CPU's the unit of clock speed is gigahertz (GHz). CPUs developed in earlier days could perform some basic arithmetic operations, but modern CPU's have become powerful and can

perform complex data processing tasks. This has become viable because of word size has become larger from 4 bit to 64 bit these days. Word size determines that how much data the CPU can process each time when it performs a certain operation. Nowadays CPUs are equipped with multiple cores as well, hence can perform multitasking efficiently. With time modern CPUs have evolved since their launch in 1970s they have come a long way. Their evolution is described in following table.

Table 1. The evolution of the CPUs

Decade	Company	CPU(s)	Main features
1970	Intel	4004, 8008, 8080, 8086	4-bit and 8-bit
	Motorola	68000	32-bit
1980	Intel	80386	32-bit, 16 MB RAM
	ARM	ARM1	
	Motorola	68020	32-bit
1990	Intel	I386, Pentium 2 and 3, Celeron	1 GHz speed
	AMD	AM386, Athlon	1 GHz speed
2000	Intel	Core 2 Duo, Atom	Dual core
	AMD	Phenom	64-bit, Quad-core

2010	Intel	Core i3, Core i5, Core i7, Core i9	Hyperthreading
	AMD	Zen	
2020	Intel	Core i9 14900K	24 cores
	AMD	Ryzen for laptops, EPYC for servers	16 cores, 128 cores

Most of the high-end desktop computers these days are powered by Intel i7 and i9 core processor. It's Intel's 3rd generation CPU which comes with 64-bit instruction set, can contain two to four cores. When it comes to executing instructions, it can hit 3.5 GHz of clock speed. It comprises a cache memory of 4MB or 6MB which makes data access fast. [12]

3.2 The GPU

GPU is a specialized processor meant for executing certain specific tasks which are independent of each other so that high level of parallelism can be achieved. GPU comprise thousands of lightweight processor cores (or ALUs) so that multiple tasks can be handled simultaneously. These ALUs are tailored to perform rapid graphical and math operations.

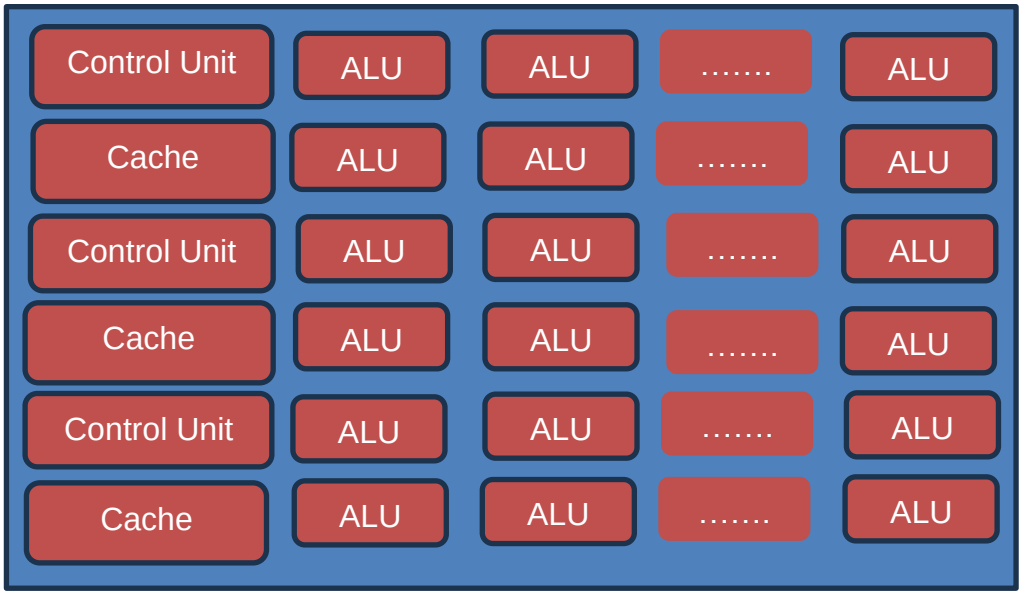


Figure 9. GPU's building blocks.

Not only in image and video processing, but GPUs are also used in several other applications such as 3D graphics rendering and cryptocurrency mining where high processing throughput is required. The performance of a GPU is measured in Floating Point Operations Per Second (FLOPS), in modern days GPUs, it in teraflops (TFLOPS), that is tens of thousands of billions of FLOPS. In the very early days of GPUs, they were intended to be used in 2D video gaming, where fast rendering of the scene is required on the display monitor.

The modern era in GPUs started with the introduction of 3D gaming in the year of 1995. This is the year when Nvidia launched its first graphic chip (or GPU), the NV1, and it became the first commercial graphics processor capable of 3D rendering, video acceleration and integrated GUI acceleration. It operates at a frequency of 75MHz. [13] The revolution embarked on in the year 2001 when Nvidia launched first ever GPU, named GetForce3, which is capable of programming the shaders. Shader is a piece of code which runs on the GPU. With the help of these shader program, each pixel of an image or video can be accessed and processed in a parallel manner. This design had offered engineers a great amount of flexibility. Additionally, this kind of GPUs are started to be called as General Purpose GPU (GPGPU), which can be programmed to perform scientific computation apart from its traditional job which was rendering graphics and images.

3.3 The key differences

Following tables summarises the key difference between CPU and the GPU.

Table 2. Key differences between the CPU and GPU

Feature	CPU	GPU
Type	Generalist processing unit	Specialised processing unit

Number of cores	Ranges from 2 to 64	Thousands
Program execution	Sequential	Parallel
Applications	Database, Encryption	Gaming, Computer Vision
Memory requirement	High	Low
Power consumption	Low	High
Price	Low	High
Programming	C, C++, Java, Python and many more	CUDA C/C++, OpenGL Shading Language (GLSL)

4 Processing Performance Comparison

How the CPU and the GPU behave differently on the same processing task will be demonstrated in this section. A couple of very basic processing techniques used in video domain, are video encoding and decoding. Encoding is done to compress the video so that it requires less bits to stream video over the network, decoding is just opposite operation than encoding, where compressed video is decoded so that display driver can understand it and render it. These two operations are at the core of video streaming services available these days as they perform crucial operation but the same time, they are computationally very expensive, therefore they contribute the most when it comes to the latency of the streaming system. Latency is the time difference between when video frame is captured and when video frame is rendered. In real time video streaming it is expected to have very minimal latency, as it is impossible to remove it completely, because captured content needs to go through several rounds of processing. But if latency increases too much then it ruins the experience of video watching, one of the most potential issues observed is audio-video synchronisation goes out. As audio data is usually very small compared to the video data and takes lesser time to transmit. So, the user will hear the audio first and video will be seen after some delay, this kind of phenomenon is called “lip-sync error”. Therefore, it is quite apparent that the latency plays major role in video streaming, and it can be made a vital criterion to determine the performance of the system.

4.1 Setup

To measure the latency caused by the video encoding and decoding operations, two tests are performed. The tests are run on Microsoft’s Windows powered Lenovo ThinkPad P16 Gen1 Laptop computer, which consists of 12th generation Intel i7 Core CPU and Nvidia RTX A3000 GPU. Both the CPU and the GPU are the flagship products of the respective companies in contemporary time.

In the first scenario, the video encoding and decoding operations are run only on the CPU. In the second scenario the same operation is performed on the GPU.

To achieve this, two different video processing pipelines need to be created and run. These pipelines can be created using GStreamer Multimedia Framework. [14] GStreamer is a widely used cross platform open-source framework to create multimedia applications. Any given functionality can be wrapped in a form of a GStreamer plugin. Plugins have pads which are used when one plugin has to connect and communicate with other plugins. Sink pad of a plugin receives the data and source pad pushes the processed data out of the plugin. Multiple plugins are inserted in a sequence inside the GStreamer pipeline. All the plugins work in synchronized manner within the pipeline because of the availability of the clock component in GStreamer. Default installation of GStreamer comes with several plugins, which can cater multiple application areas such as video, image, audio, streaming etc. Although GStreamer provides flexibility to write a custom plugin which contains functionality of one's wish, but for the comparation testing activity, already available plugins were used.

The laptop computer is connected to a monitor screen and a web camera.
The web camera is placed in front of the monitor screen in such a way that it captures only the left half of the monitor screen, which displays an online stopwatch.[15] The right half of the monitor screen renders the processed content (processed means the captured video frames are first encoded and then decoded). The video capture resolution is set to 640x480 and framerate is 30 frames per second.

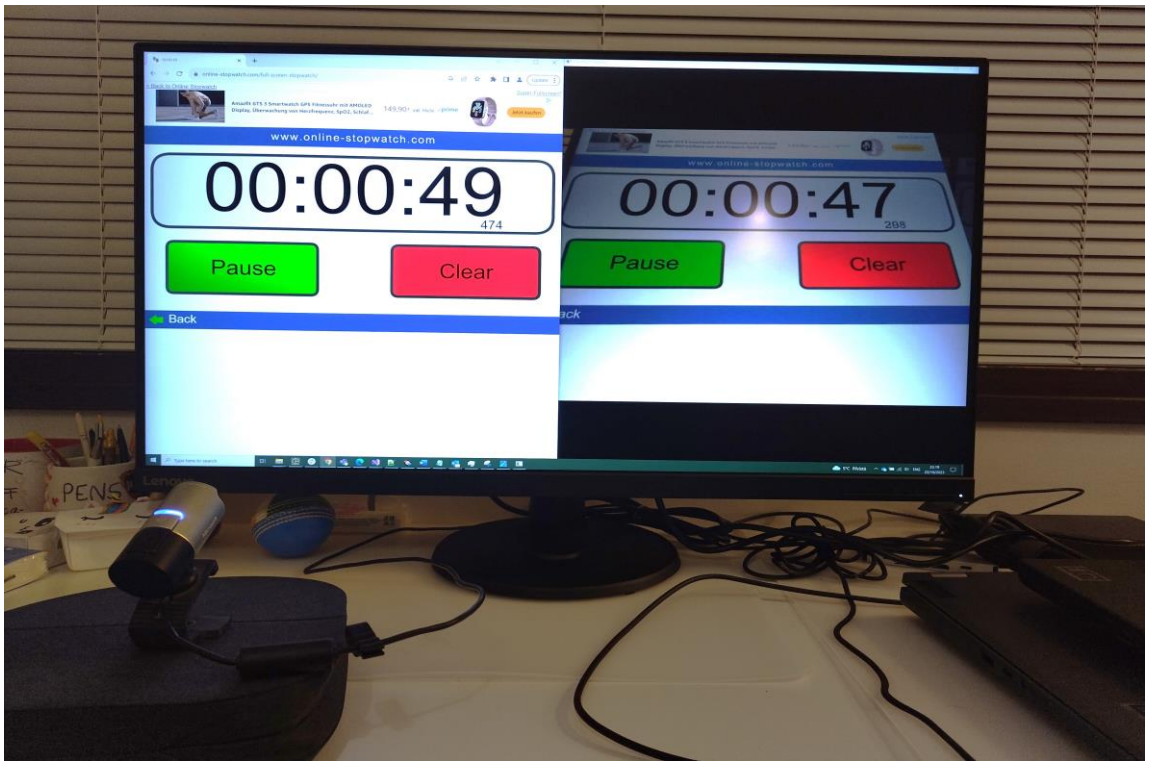


Figure 10. Test system contains a laptop computer, a monitor and a web camera.

4.2 Video encoding and decoding on the CPU

The first pipeline where software-based video encoder (x264enc) and decoder(avdec_h264) plugins are being used. When plugins are denoted as software-based that means, they do not exploit the full power of the dedicated hardware, rather most of the processing takes place in the software, therefore the CPU is being used solely. The pipeline looks as follows (Pipeline 1):

"*gst-launch-1.0.exe ksvideosrc ! video/x-raw,width=640,height=480,framerate=30/1 ! videoconvert ! queue ! x264enc ! queue ! h264parse ! queue ! avdec_h264 ! queue ! autovideosink*"

gst-launch-1.0.exe is command line-based tool provided by GStreamer to create and run abovementioned pipeline.

A brief description about each plugin is provided in following table.

Table 3. List of the CPU based GStreamer plugins with their functionalies.

Plugin	Description
Ksvideosrc	Uses web camera's driver APIs to grab the video frames
video/x-raw,width=640,height=480, framerate=30/1	Configures web camera's video recording parameters
videoconvert	Colour space conversion from YUY2 to Y42B
x264enc	H264 encoder
h264parse	H264 parser
avdec_h264	H264 Decoder
Autovideosink	Renders uncompressed video frames

Note: queue plugin is used to control the number of buffers in the pipeline to avoid any buffer drop when pipeline is running.

When pipeline is run, the usage of the CPU and the GPU can be seen from the task manager, screenshot is depicted below. Red rectangle shows the CPU usage (19%), green rectangle shows the GPU usages (2%) and yellow rectangle depicts that the GPU is not involved in video encoding and decoding operations.

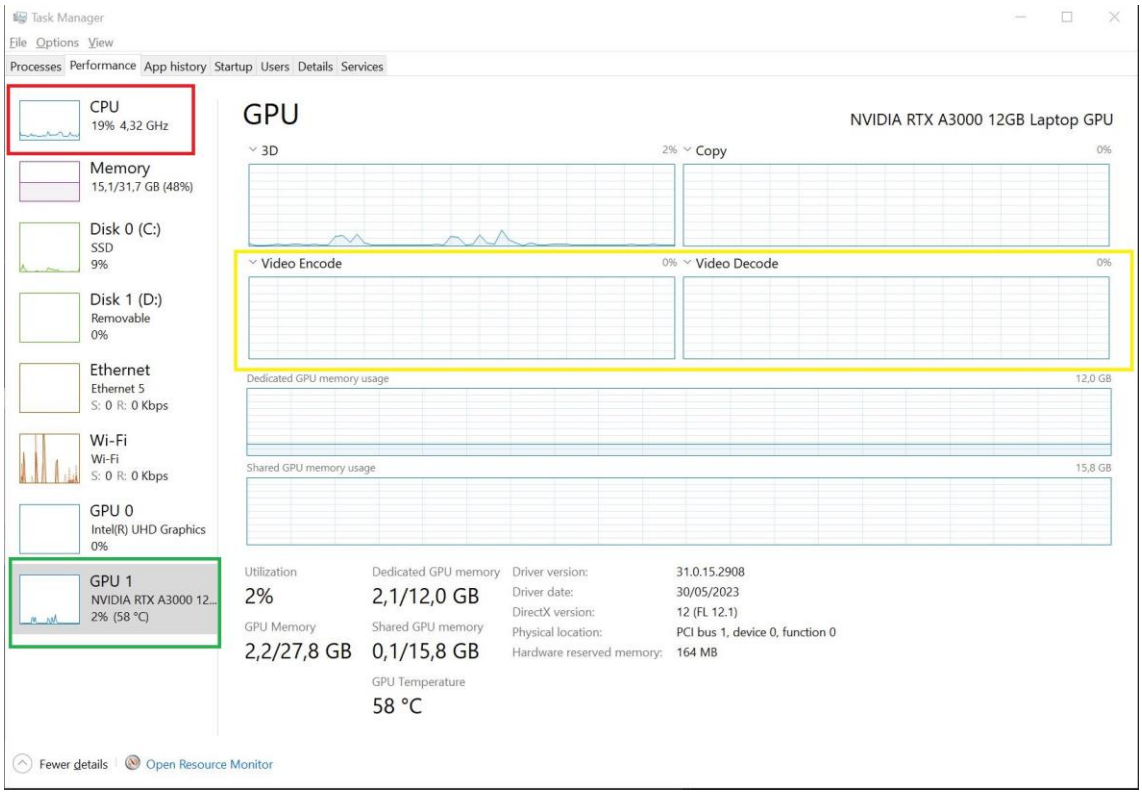


Figure 11. CPU based encoding and decoding.

From the task manager's screenshot its quite evident that the CPU is doing most of the processing whereas GPU is idle. This screenshot is taken to make sure that GPU is not involved in the video encoding and decoding operations. That is the intention of this experiment.

While pipeline was running, multiple screenshots of the test system's laptop computer's screen are captured to determine the latencies caused by the video encoding and decoding processes. Latency is calculated simply by performing subtraction between stopwatch counters of left half and right half of the screen. Following figure contains screenshots of stopwatch counters from left half (i.e., captured frame) and right half (i.e., rendered frame) of the screen as well as the latency value for each set.

Captured frame	Rendered frame	Latency
		2112 ms
		2113 ms
		2081 ms
		2112 ms

Figure 12: CPU based encoding and decoding latencies.

The average latency value gathered from the multiple screenshots is **2104.5ms**.

4.3 Video encoding and decoding on the GPU

The second pipeline, whose focus is to exploit the GPU's processing power to encode and decode captured content looks like this (Pipeline 2):

gst-launch-1.0.exe ksvideosrc ! videoconvert ! video/x-raw,width=640,height=480,framerate=30/1 ! queue ! nvh264enc ! queue ! h264parse ! queue ! nvh264dec ! queue ! autovideosink

Most of the plugins used in the above-mentioned pipeline are already discussed in the earlier section (4.2). The only difference this pipeline has is that it uses nvh264enc and nvh264dec plugins. The special attribute of these plugins is that they both use a library called nvcodec. [16] This library takes advantage of the hardware-based encoder and decoder blocks present in the GPU. [17] The sole purpose of these blocks to perform encode and decode operations, they do not perform any other operations.

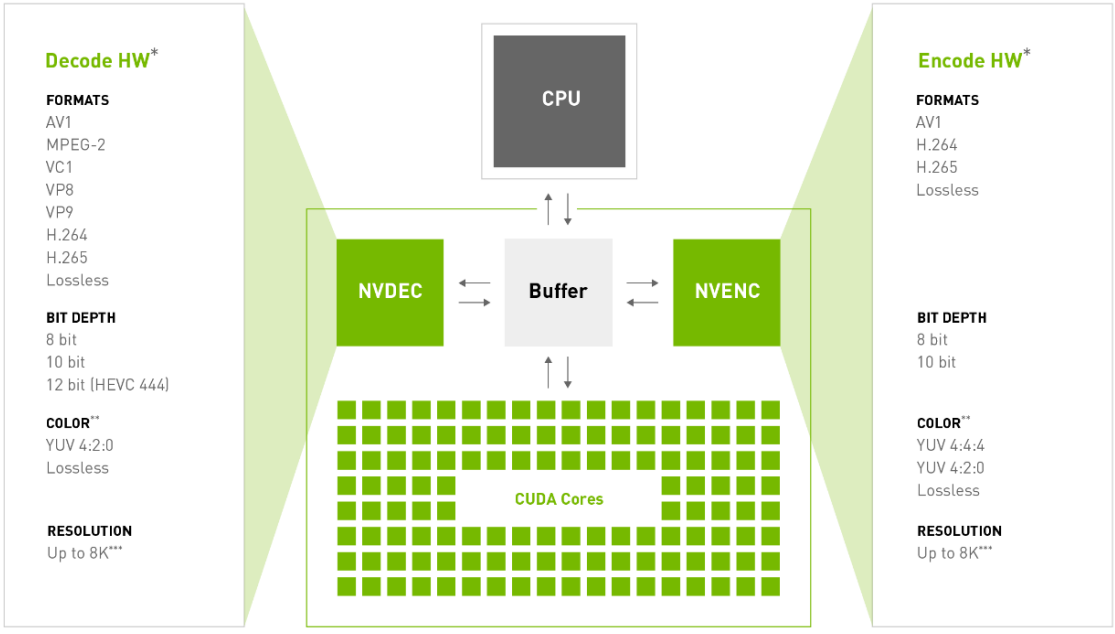


Figure 13: Hardware blocks of Nvidia GPU [17]

When the pipeline runs, the GPU usages goes up to 6% (depicted by the red rectangle in following figure), whereas the CPU usages stays on the lower side, i.e., 4% (green rectangle), as the GPU has offloaded the CPU. The GPU's hardware based encode and decode acceleration is active as they both consume 4% each (purple rectangle).

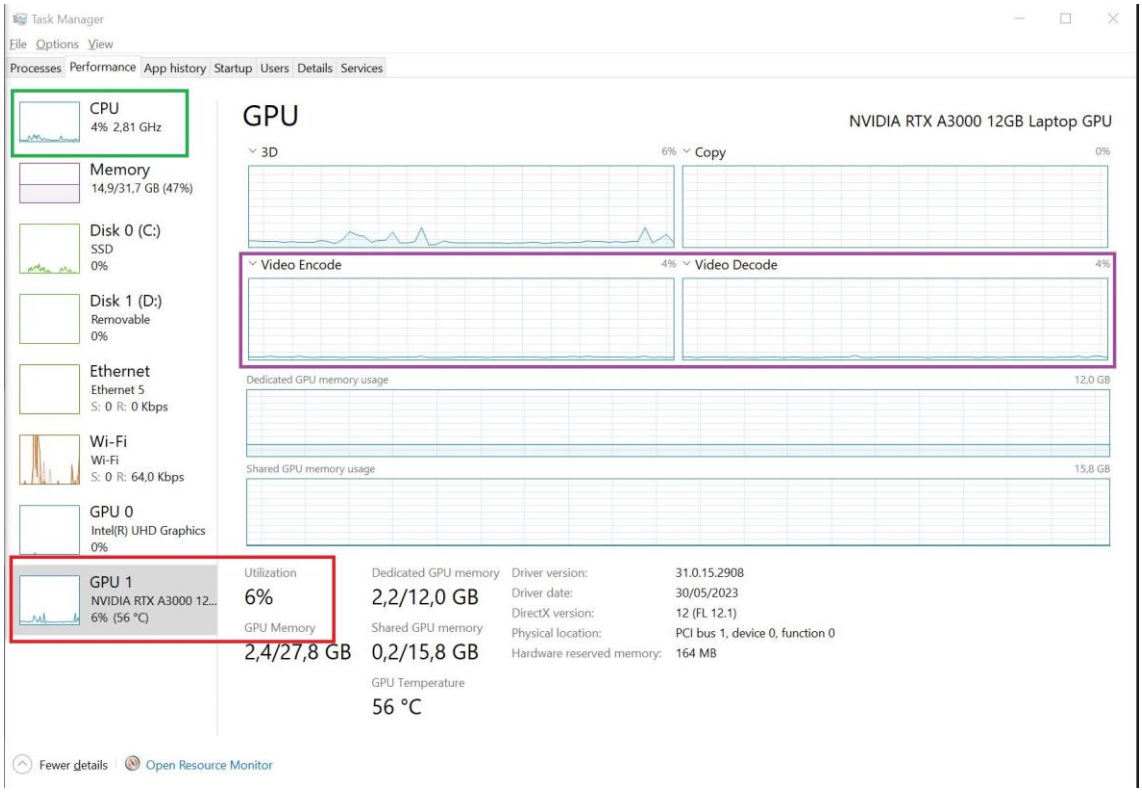


Figure 14: The GPU based encoding and decoding.

Few screenshots are taken while pipeline was running to determine the latencies caused by video encoding and decoding processes when run on the GPU. Latency is calculated simply by performing subtraction between stopwatch counters of left half and right half of the screen. Following figure contains screenshots of stopwatch counters from left half (i.e., captured frame) and right half (i.e., rendered frame) of the screen as well as the latency value for each set.

Captured frame	Rendered frame	Latency
		351 ms
		319 ms
		352 ms
		352s ms

Figure 15: Multiple screenshots showing actual stopwatch counters and rendered stopwatch counters, as well as the respective latency values.

The average latency value gathered from the multiple screenshots is **343.5ms**.

4.4 Miscellaneous effects on simulated video

Apart from encoding and decoding, multiple tests are performed where certain effects are applied on the input video frames. Videos are captured with resolution of 1920x1080 at 30 frames per second. Number of video frames are kept limited in order to profile the time taken in these operations. There are two versions of effects, out of these two, one is made to run on the CPU and another on the GPU. GStreamer already has these effects implemented in the form of plugins. Plugins which run on the GPU, use OpenGL interface to access the GPU. OpenGL provides a set of standard APIs to access the GPU on the computer. [18] A logical graphics/rendering pipeline which runs within the supervision of OpenGL performs processing on data stream. The graphics pipeline is made of several discrete stages. In the figure below, the stages denoted with yellow colour are the non-programmable, whereas the stages marked in blue colour are programmable. However, the most important ones for the software engineers are Vertex Shader and Fragment Shader sages as they are programmable therefore can be used to change the data which goes in the pipeline.

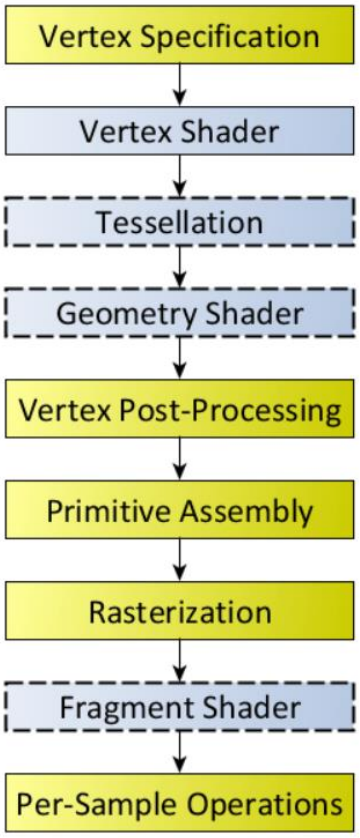


Figure 16. OpenGL supported logical graphics/rendering pipeline stages [19]

GStreamer plugins, which take advantage of the GPU's processing power use OpenGL to access these shaders with the help of GLSL. GLSL is C-like programming language which contains variables, data types and functions which are meant to be run on the GPU. Vertex shader can be considered as a program implemented in GLSL to manipulate vertex data, which defines the final position of the vertices of the object, eventually decides the shape and the position of the rendered object. Fragment Shader gets written in GLSL also and determines the colour of each fragment or pixel. Once implemented and executed these shaders run parallelly for all the pixels in a given frame. Therefore, all the pixels get processed in single cycle rather than being processed one by one, as typically done by the CPU.

The same laptop computer is used here as mentioned in the previous video encoding and decoding latency measurement test.

Instead of feeding web camera output to processing pipeline, a simulated video feed is being used, which looks like this:

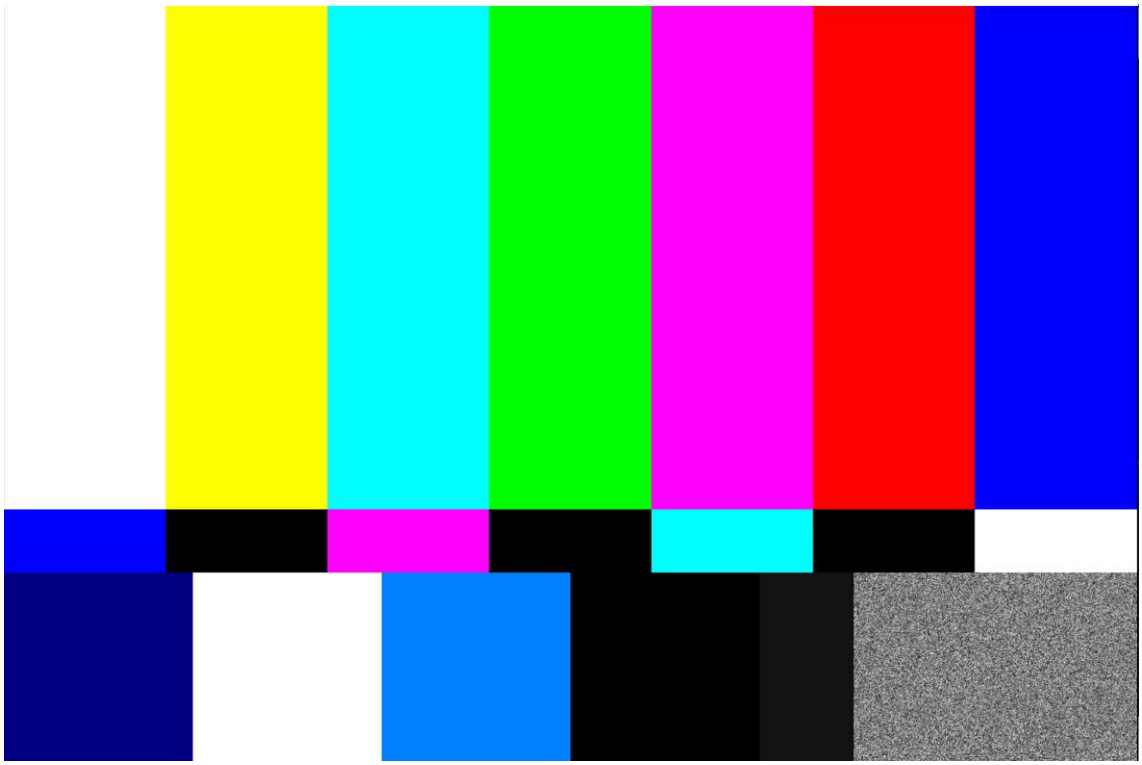


Figure 17: Simulated input video feed

After 100 frames are pushed to the pipeline and different effects are applied each time, the difference can be easily observed how the CPU and the GPU has performed. Following figure depicts the effect name, the visual output after effect has been applied to simulated video feed, the time consumed by the CPU based plugins and the GPU based plugins respectively.

Effect name	**Effect visual**	**CPU latency**	**GPU latency**
Twirl		1250 ms	300 ms

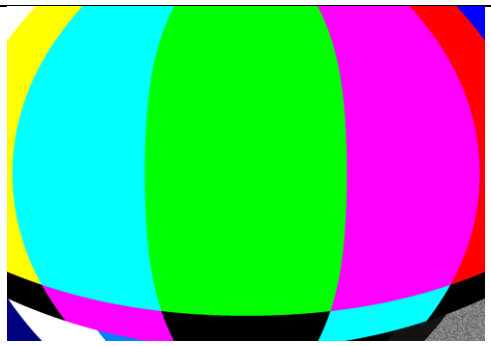
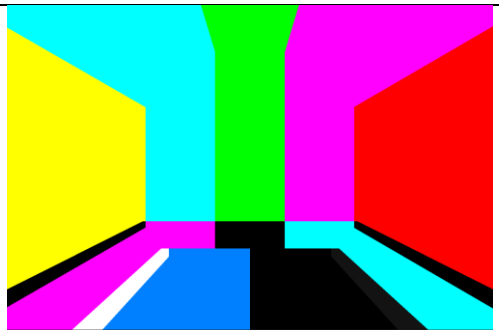
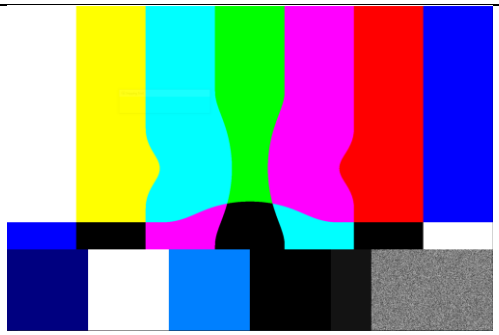
Fisheye		1300 ms	320 ms
Bulge		1350 ms	330 ms
Tunnel		1250 ms	29 ms
Stretch		1300 ms	310 ms

Figure 18: Miscellaneous effects applied on simulated video feed.

From conducted tests, it is quite evident that the CPU took more time to process the video frames. Difference is clearly seen to be almost four times for every effect applied on the simulated video.

5 Conclusion

The average latency value received from the first test, where the CPU was used to encode and decode the video captured from the web camera, is **2104.5ms** whereas from the second test, where the GPU was used to perform the same operation, was **343.5ms**. This demonstrates that how much impact the GPU can bring in when it comes to process the videos. Although the use of the CPU is still essential in order to provide a stage for the GPU to work. The CPU helps storing the captured video frame into the memory and does other suitable operations.

When several other effects such as twirl, fisheye, bulge, tunnel and stretch are applied on a simulated video feed which consists of 100 frames of resolution 1920x1080, the difference in the latency is clearly observed. When these effects are applied with the help of the CPU only, the average latency was approximately **1290ms**, but when the same effects are applied leveraging GPU power the average latency goes significantly down to approximately **310ms**.

Abovementioned experiments clearly suggest that the use of the GPU is so important when it comes to process the images and the videos. This makes latency down enough to perform video and image-based operations in real time. But the same time the essential role of the CPU can not be denied as the GPU processing can only be accessed via the CPU. Developers who are writing code to access GPU can not access it directly, the Application Programming Interface (API) are available only for the CPU. Software engineers need to write code on the platform provided by the CPU. Hence, the CPU and the GPU must work in tandem to get best out of these two processing units. On the contrary side the addition of the GPU to the computer system, increases the cost. They are expensive piece of hardware, an average GPU (Graphics Card) could cost thousands of euros. Therefore, one must think about the requirements clear enough, be precise about the priority and then decide whether the GPU is needed in their work.

References

1 How many YouTube video watched a day
https://techjury.net/blog/how-many-youtube-videos-watched-a-day/
Accessed 25.9.2023

2 How many pictures are on Instagram in 2023
https://earthweb.com/how-many-pictures-are-on-instagram
Accessed 25.9.2023

3 Facebook users are uploading 350 million new photos each day
https://businessinsider.com/facebook-350-million-photos-each-day-2013-9?r=US&IR=T
Accessed 25.9.2023

4 TikTok users and growth statistics (2023)
https://www.usesignhouse.com/blog/tiktok-stats#:~:text=How%20many%20videos%20are%20uploaded,videos%20was%20removed%20in%202021
Accessed 25.9.2023

5 Indians top users of WhatsApp video calls daily
https://www.livemint.com/Industry/Y0iFXtmb3tYNx5w3nDRXcP/Indians-top-users-of-WhatsApp-video-calls-daily.html#:~:text=minutes%20per%20day.-,Globally%2C%20users%20make%20over%2055%20million%20video%20calls%20per%20day,over%20a%20billion%20users%20globally
Accessed 25.9.2023

6 JPEG image compression
https://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/jpegcompression/#:~:text=The%20compression%20ratio%20is%20dependent,with%20a%20particular%20compression%20level
Accessed 25.9.2023

7 H.264 vs H.265: An analytical breakdown of video streaming codecs
https://imagekit.io/blog/h264-vs-h265/
Accessed 25.9.2023

8 Advanced image processing systems
https://hal.science/hal-03168963/document
Accessed 4.10.2023

9 Digital Video Processing
https://www.sciencedirect.com/topics/computer-science/frequency-coefficient
Accessed 4.10.2023

10 Video Processing
https://resources.riches-project.eu/glossary/video-processing/
Accessed 08.10.2023

11 Lossless Compression
https://github.com/huuuuusy/Lossless-Compression
Accessed 08.10.2023

12 Intel 3rd generation i7 core processors
https://www.mouser.fi/new/intel/intel-ivybridge-i7-core-processors/
Accessed 10.10.2023

13 NV1, the first commercial GPU
https://en.wikipedia.org/wiki/NV1
Accessed 15.10.2023

14 GStremaer Open Source Multimedia Framework
https://gstreamer.freedesktop.org
Accessed on 26.10.2023

15 Online stop watch
https://www.online-stopwatch.com/full-screen-stopwatch/
Accessed on 26.10.2023

16 GStreamer's bad plugins
https://gstreamer.freedesktop.org/documentation/nvcodec/index.html?gi-language=c
Accessed on 27.19.2023

17 NVIDIA Video Codec SDK
https://developer.nvidia.com/video-codec-sdk
Accessed on 27.10.2023

18 OpenGL API Documentation Overview
https://www.opengl.org/Documentation/Specs.html
Accessed on 03.11.2023

19 Rendering Pipeline Overview
https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
Accessed on 18.11.2023

Appendix 1: GStreamer pipeline commands for miscellaneous effects

Effects are applied using the CPU

Following GStreamer based pipelines use plugin code which runs on the CPU.

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! fisheye ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! stretch ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! tunnel ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! bulge ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! twirl ! fakesink async=false

Effects are applied using the GPU

Following GStreamer based pipelines use plugin code which runs on the GPU. OpenGL is used to access the GPU functionality.

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! glupload ! gleffects_fisheye ! gldownload ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! stretch ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! tunnel ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! bulge ! fakesink async=false

gst-launch-1.0.exe videotestsrc num-buffers=100 ! "video/x-raw, width=1920, height=1280" ! twirl ! fakesink async=false

Appendix 2: Source code for miscellaneous effects

Fragment shader code written in GSLS for fisheye effect.

```
const gchar *fisheye_fragment_source_gles2 =
  "varying vec2 v_texcoord;"
  "uniform sampler2D tex;"
  "void main () {"
  "  vec2 texturecoord = v_texcoord.xy;"
  "  vec2 normcoord;"
  "  normcoord = texturecoord - 0.5;"
  "  float r = length (normcoord);"
  "  normcoord *= r * 1.41421;" /* sqrt (2) */
  "  texturecoord = normcoord + 0.5;"
  "  gl_FragColor = texture2D (tex, texturecoord);"
  "}";
```

Fragment shader code written in GSLS for stretch effect.

```
const gchar *stretch_fragment_source_gles2 =
  "varying vec2 v_texcoord;"
  "uniform sampler2D tex;"
  "void main () {"
            "  vec2 texturecoord = v_texcoord.xy;"
  "  vec2 normcoord;"
  "  normcoord = texturecoord - 0.5;"
  "  float r = length (normcoord);"
  "  normcoord *= 2.0 - smoothstep(0.0, 0.35, r);"
  "  texturecoord = normcoord + 0.5;"
  "  gl_FragColor = texture2D (tex, texturecoord);"
"}";
```

Fragment shader code written in GSLS for tunnel effect.

```
const gchar *tunnel_fragment_source_gles2 =
  "varying vec2 v_texcoord;"
  "uniform sampler2D tex;"
  "void main () {"
```

```
  "  vec2 texturecoord = v_texcoord.xy;"
  "  vec2 normcoord;"
  "  normcoord = (texturecoord - 0.5);"
  "  float r = length(normcoord);"
  "  if (r > 0.0)"
  "    normcoord *= clamp (r, 0.0, 0.275) / r;"
  "  texturecoord = normcoord + 0.5;"
  "  gl_FragColor = texture2D (tex, texturecoord);"
"}";
```

Fragment shader code written in GSLS for bulge effect.

```
const gchar *bulge_fragment_source_gles2 =
  "varying vec2 v_texcoord;"
  "uniform sampler2D tex;"
  "void main () {"
  "  vec2 texturecoord = v_texcoord.xy;"
  "  vec2 normcoord;"
  "  normcoord = texturecoord - 0.5;"
  "  float r =  length (normcoord);"
  "  normcoord *= smoothstep (-0.05, 0.25, r);"
  "  texturecoord = normcoord + 0.5;"
  "  gl_FragColor = texture2D (tex, texturecoord);"
"}";
```

Fragment shader code written in GSLS for twirl effect.

```
const gchar *twirl_fragment_source_gles2 =
  "varying vec2 v_texcoord;"
  "uniform sampler2D tex;"
  "void main () {"
  "  vec2 texturecoord = v_texcoord.xy;"
  "  vec2 normcoord;"
  "  normcoord = texturecoord - 0.5;"
  "  float r = length (normcoord);"
  "  float phi = (1.0 - smoothstep (0.0, 0.3, r)) * 1.6;"
  "  float s = sin(phi);"
  "  float c = cos(phi);"
  "  normcoord *= mat2(c, s, -s, c);"
  "  texturecoord = normcoord + 0.5;"
  "  gl_FragColor = texture2D (tex, texturecoord);"
"}";
```

CPU code for fisheye effect.

```
static gboolean
fisheye_map (GstGeometricTransform * gt, gint x, gint y, gdouble * in_x,
```

```
    gdouble * in_y)
{
  gdouble norm_x;
  gdouble norm_y;
  gdouble r;

  gdouble width = gt->width;
  gdouble height = gt->height;

  norm_x = 2.0 * x / width - 1.0;
  norm_y = 2.0 * y / height - 1.0;

  r = sqrt ((norm_x * norm_x + norm_y * norm_y) / 2.0);

  norm_x *= (0.33 + 0.1 * r * r + 0.57 * pow (r, 6.0));
  norm_y *= (0.33 + 0.1 * r * r + 0.57 * pow (r, 6.0));

  *in_x = 0.5 * (norm_x + 1.0) * width;
  *in_y = 0.5 * (norm_y + 1.0) * height;

  GST_DEBUG_OBJECT (fisheye, "Inversely mapped %d %d into %lf %lf",
      x, y, *in_x, *in_y);

  return TRUE;
}
```

CPU code for stretch effect.

```
static gboolean
stretch_map (GstGeometricTransform * gt, gint x, gint y, gdouble * in_x,
    gdouble * in_y)
{
  GstCircleGeometricTransform *cgt = GST_CIRCLE_GEOMETRIC_TRANSFORM_CAST
(gt);
  GstStretch *stretch = GST_STRETCH_CAST (gt);

  gdouble norm_x, norm_y;
  gdouble r;

  gdouble width = gt->width;
  gdouble height = gt->height;
  gdouble a, b;

  norm_x = 2.0 * (x / width - cgt->x_center);
  norm_y = 2.0 * (y / height - cgt->y_center);

  r = sqrt (0.5 * (norm_x * norm_x + norm_y * norm_y));

  a = 1.0 + (MAX_SHRINK_AMOUNT - 1.0) * stretch->intensity;
  b = a - 1.0;

  norm_x *= a - b * gst_gm_smoothstep (0.0, cgt->radius, r);
  norm_y *= a - b * gst_gm_smoothstep (0.0, cgt->radius, r);

  *in_x = (0.5 * norm_x + cgt->x_center) * width;
  *in_y = (0.5 * norm_y + cgt->y_center) * height;

  GST_DEBUG_OBJECT (stretch, "Inversely mapped %d %d into %lf %lf",
      x, y, *in_x, *in_y);
```

```
  return TRUE;
}
```

CPU code for tunnel effect.

```
static gboolean
tunnel_map (GstGeometricTransform * gt, gint x, gint y, gdouble * in_x,
    gdouble * in_y)
{
  GstCircleGeometricTransform *cgt = GST_CIRCLE_GEOMETRIC_TRANSFORM_CAST
(gt);

  gdouble norm_x, norm_y;
  gdouble width = gt->width;
  gdouble height = gt->height;
  gdouble r;


  norm_x = 2.0 * (x - cgt->x_center * width) / MAX (width, height);
  norm_y = 2.0 * (y - cgt->y_center * height) / MAX (width, height);

  r = sqrt (0.5 * (norm_x * norm_x + norm_y * norm_y));

  norm_x *= CLAMP (r, 0.0, cgt->radius) / r;
  norm_y *= CLAMP (r, 0.0, cgt->radius) / r;

  *in_x = 0.5 * (norm_x) * MAX (width, height) + cgt->x_center * width;
  *in_y = 0.5 * (norm_y) * MAX (width, height) + cgt->y_center * height;

  GST_DEBUG_OBJECT (tunnel, "Inversely mapped %d %d into %lf %lf",
      x, y, *in_x, *in_y);

  return TRUE;
}
```

CPU code for bulge effect.

```
static gboolean
bulge_map (GstGeometricTransform * gt, gint x, gint y, gdouble * in_x,
    gdouble * in_y)
{
  GstCircleGeometricTransform *cgt = GST_CIRCLE_GEOMETRIC_TRANSFORM_CAST
(gt);
  GstBulge *bulge = GST_BULGE_CAST (gt);

  gdouble norm_x, norm_y;
  gdouble r;
  gdouble scale;

  gdouble width = gt->width;
  gdouble height = gt->height;

  norm_x = 2.0 * (x / width - cgt->x_center);
  norm_y = 2.0 * (y / height - cgt->y_center);

  r = sqrt (0.5 * (norm_x * norm_x + norm_y * norm_y));
```

```
  scale =
      1.0 / (bulge->zoom + ((1.0 - bulge->zoom) * gst_gm_smoothstep (0,
              cgt->radius, r)));

  norm_x *= scale;
  norm_y *= scale;

  *in_x = (0.5 * norm_x + cgt->x_center) * width;
  *in_y = (0.5 * norm_y + cgt->y_center) * height;

  GST_DEBUG_OBJECT (bulge, "Inversely mapped %d %d into %lf %lf",
      x, y, *in_x, *in_y);

  return TRUE;
}
```

CPU code for twirl effect.

```
static gboolean
twirl_map (GstGeometricTransform * gt, gint x, gint y, gdouble * in_x,
    gdouble * in_y)
{
  GstCircleGeometricTransform *cgt = GST_CIRCLE_GEOMETRIC_TRANSFORM_CAST
(gt);
  GstTwirl *twirl = GST_TWIRL_CAST (gt);
  gdouble distance;
  gdouble dx, dy;

  dx = x - cgt->precalc_x_center;
  dy = y - cgt->precalc_y_center;
  distance = dx * dx + dy * dy;

  if (distance > cgt->precalc_radius2) {
    *in_x = x;
    *in_y = y;
  } else {
    gdouble d = sqrt (distance);
    gdouble a = atan2 (dy,
        dx) + twirl->angle * (cgt->precalc_radius - d) / cgt-
>precalc_radius;

    *in_x = cgt->precalc_x_center + d * cos (a);
    *in_y = cgt->precalc_y_center + d * sin (a);
  }

  GST_DEBUG_OBJECT (twirl, "Inversely mapped %d %d into %lf %lf",
      x, y, *in_x, *in_y);

  return TRUE;
}
```

Source codes are referenced from

https://gitlab.freedesktop.org/gstreamer/gstreamer