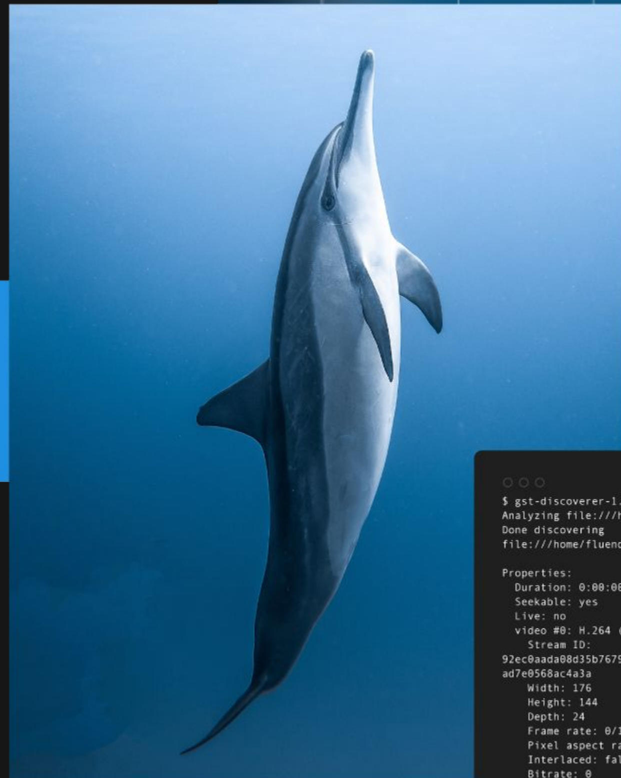
fluendo

GStreamer NVIDIA memory in Jetson

looking for an efficient way to process video frames when handling raw CUDA kernels



```
$ gst-discoverer-1.0 AUD_MW_E.264
Analyzing file:///home/fluendo/Videos/AUD_MW_E.264
Done discovering
file:///home/fluendo/Videos/AUD_MW_E.264

Properties:
  Duration: 0:00:00.000000000
  Seekable: yes
  Live: no
  video #0: H.264 (Constrained Baseline Profile)
    Stream ID:
92ec0aada08d35b7679f87a98465b4cf955fbad11d2c8535ec
ad7e0568ac4a3a
    Width: 176
    Height: 144
    Depth: 24
    Frame rate: 0/1
    Pixel aspect ratio: 1/1
    Interlaced: false
    Bitrate: 0
    Max bitrate: 0
```

Oct 7th, 2024 – Diego Nieto

Index

The project architecture

The Jetson - GStreamer memory flow

Understanding the pipeline cost

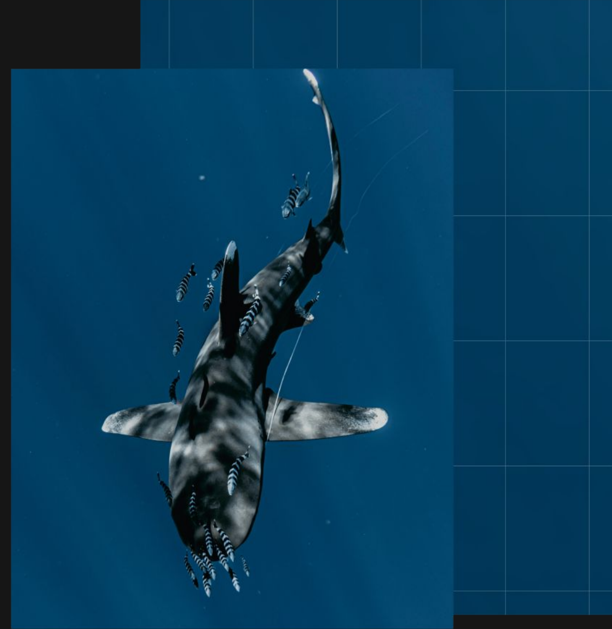
Implementing a simple CUDA resource pool

Memory layout matters

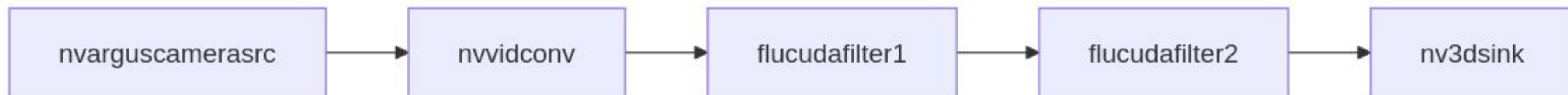
Picking the right sink

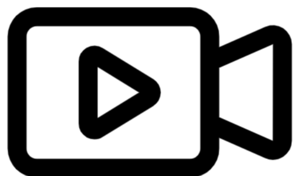
All together

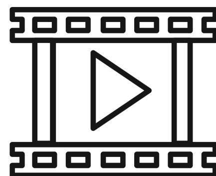
The project architecture

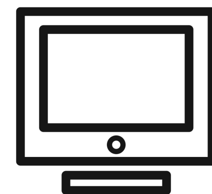


A simplified version of the pipeline









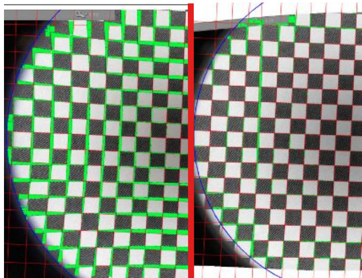
NVIDIA Jetson TX2 with Jetpack R32.7.4
Custom GStreamer built 1.20

The GStreamer filters developed

original | corrected

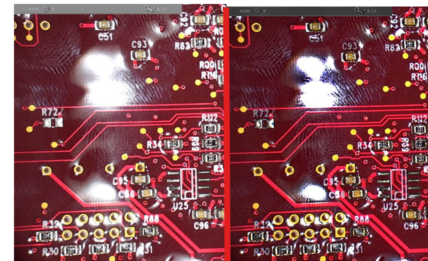
Chromatic aberration correction based on CUDA kernel
per channel correction

Spatial distortion correction based in NVIDIA VPI remap
per pixel remapping



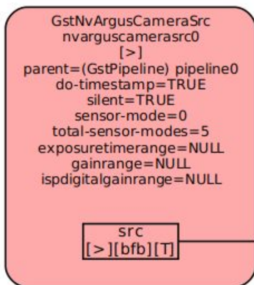
original | corrected

Glare correction (dehaze) based on CUDA kernel
per channel correction

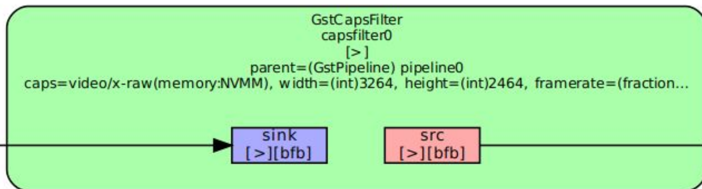


original | corrected

A quick picture of what we are talking about

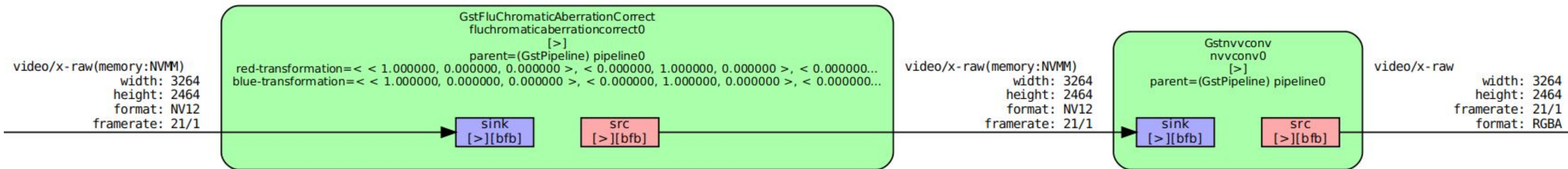


video/x-raw(memory:NVMM)
width: 3264
height: 2464
format: NV12
framerate: 21/1

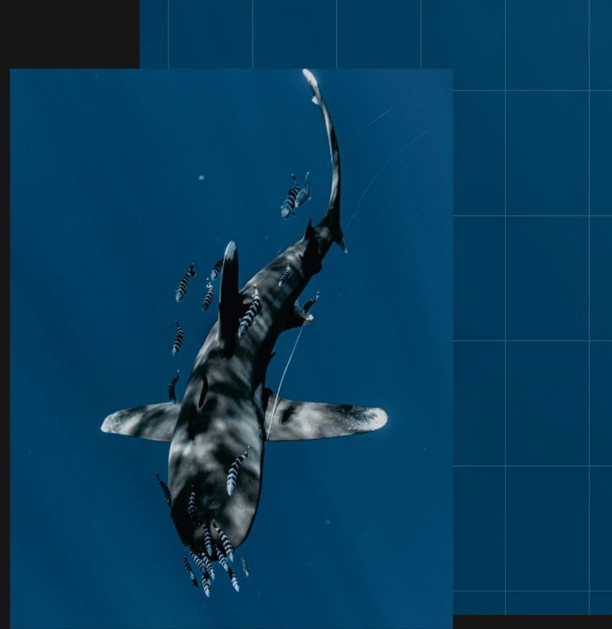


video/x-raw(memory:NVMM)
width: 3264
height: 2464
format: NV12
framerate: 21/1

Legend
Element-States: [~] void-pending, [0] null, [-] ready, [=] paused, [>] playing
Pad-Activation: [-] none, [>] push, [<] pull
Pad-Flags: [b]locked, [f]lushing, [b]locking, [E]OS; upper-case is set
Pad-Task: [T] has started task, [t] has paused task



The Jetson - GStreamer memory flow



How to represent the same in different objects

- NVMM
- DMA
- EGLImageKHR
- CUgraphicsResource
- **CUeglFrame**

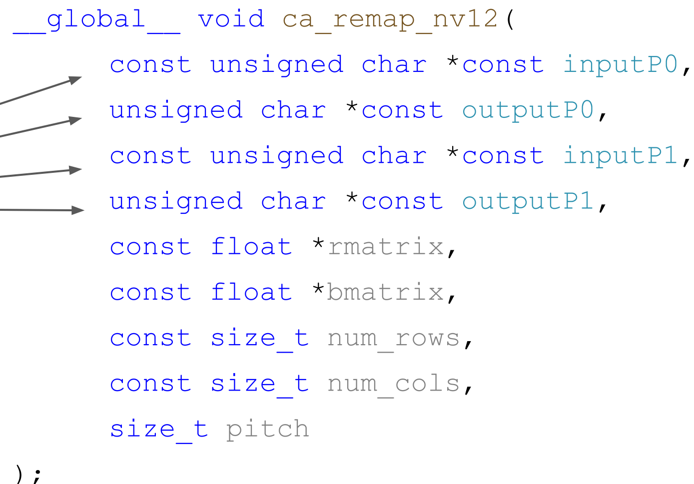
Creating NVMM memory in Jetson

```
NvBufferCreateParams input_params = { 0 };
input_params.width = video_info.width;
input_params.height = video_info.height;
input_params.layout = NvBufferLayout_Pitch;
input_params.colorFormat =
   gst_video_format_to_nv_buffer_color_format (video_info.finfo->format);
input_params.payloadType = NvBufferPayload_SurfArray;
input_params.nvbuf_tag = NvBufferTag_VIDEO_CONVERT;

if (NvBufferCreateEx (dmabuf_fd, &input_params)) {
 GST_ERROR_OBJECT (self, "NvBufferCreateEx Failed");
 return false;
}
```

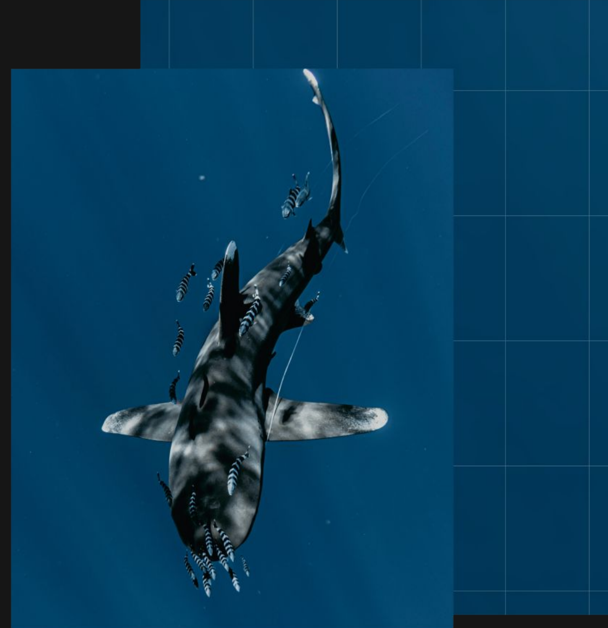
The relationship between CUeglFrame and a CUDA kernel

```
CUarray_format cuFormat
unsigned int  depth
CUeglColorFormat eglColorFormat
CUeglFrameType frameType
unsigned int  height
unsigned int  numChannels
CUarray pArray[MAX_PLANES]
void [MAX_PLANES] * pPitch
unsigned int  pitch
unsigned int  planeCount
unsigned int  width
```

The CUDA kernel memory parameter, e.g:



Understanding the pipeline cost



What is free and what is not

int NvBufferCreateEx (int *dmabuf_fd, NvBufferCreateParams *input_params) → reused

2. **DMA**

   int ← ExtractFdFromNvBuffer (void *nvbuf, int *dmabuf_fd) → free

3. **EGLImageKHR**

   EGLImageKHR ← NvEGLImageFromFd (EGLDisplay display, int dmabuf_fd) → free

4. **CUgraphicsResource**

   CUresult ← cuGraphicsEGLRegisterImage (CUgraphicsResource *pCudaResource, EGLImageKHR image, unsigned int flags) → free

5. **CUeglFrame**

   CUresult ← cuGraphicsResourceGetMappedEglFrame (CUeglFrame * eglFrame, CUgraphicsResource resource, unsigned int index, unsigned int mipLevel) → 3ms at 4K

GStreamer latency stats are our friend

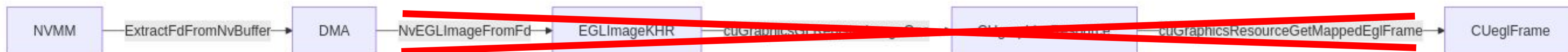
Without buffer pool

```
Latency Statistics:
	0x55acb580e0.nvarguscamerasrc0.src|0x55acc12120.xvimagesink0.sink: mean=0:00:00.028224954
min=0:00:00.026479589 max=0:00:00.093086526

Element Latency Statistics:
	0x55acc1c290.capsfilter0.src: mean=0:00:00.000133794 min=0:00:00.000079071 max=0:00:00.000223230
	0x55ac8f34d0.nvvconv0.src: mean=0:00:00.002378882 min=0:00:00.002296039 max=0:00:00.005295816
	0x55acc1c5d0.capsfilter1.src: mean=0:00:00.000140948 min=0:00:00.000104383 max=0:00:00.002090762
	0x55acc0bcc0.fluchromaticaberrationcorrect0.src: mean=0:00:00.014428116 min=0:00:00.012778967
max=0:00:00.051928885
	0x55acc1c910.capsfilter2.src: mean=0:00:00.000155358 min=0:00:00.000133471 max=0:00:00.000393212
	0x55ac8bacd0.nvvconv1.src: mean=0:00:00.010987853 min=0:00:00.010537807 max=0:00:00.036408347
```

With buffer pool

```
Latency Statistics:
	0x55cc28cd70.nvarguscamerasrc0.src|0x55cc356120.xvimagesink0.sink: mean=0:00:00.021942232
min=0:00:00.020832060 max=0:00:00.083427083

Element Latency Statistics:
	0x55cc3602d0.capsfilter0.src: mean=0:00:00.000131937 min=0:00:00.000083423 max=0:00:00.000315901
	0x55cc1784d0.nvvconv0.src: mean=0:00:00.002362198 min=0:00:00.002289319 max=0:00:00.005618050
	0x55cc360610.capsfilter1.src: mean=0:00:00.000138629 min=0:00:00.000100543 max=0:00:00.000489082
	0x55cc350440.fluchromaticaberrationcorrect0.src: mean=0:00:00.008072703 min=0:00:00.007248112
max=0:00:00.046158788
	0x55cc360950.capsfilter2.src: mean=0:00:00.000171415 min=0:00:00.000136799 max=0:00:00.000937750
	0x55cc13fcd0.nvvconv1.src: mean=0:00:00.011065348 min=0:00:00.010579820 max=0:00:00.032982550
```

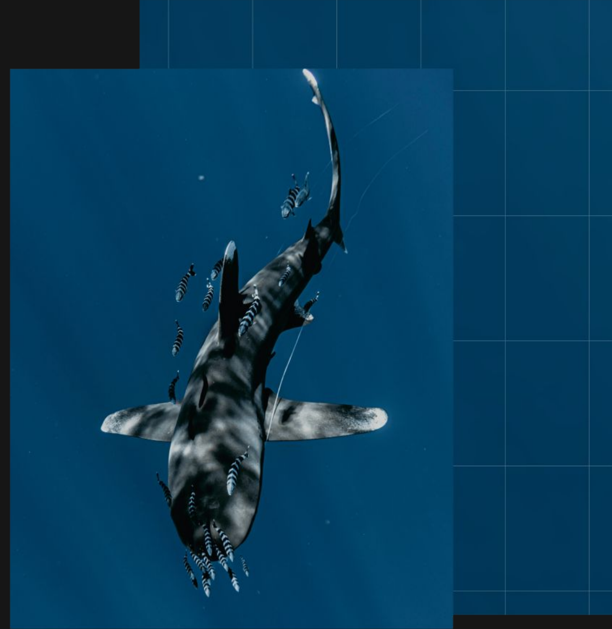
Get rid of the mapping for each frame is key



- Without buffer pool
  - Full pipeline: 28ms
  - Only the filter element: 14ms

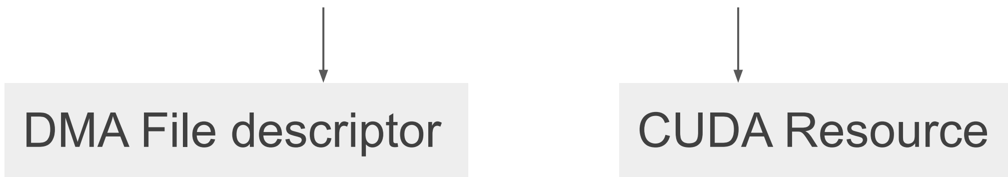
- With buffer pool
  - Full pipeline: 21ms
  - Only the filter element: 8ms

Implementing a simple CUDA resource pool



The Jetson - A simple CUDA - NVMM Buffer pool

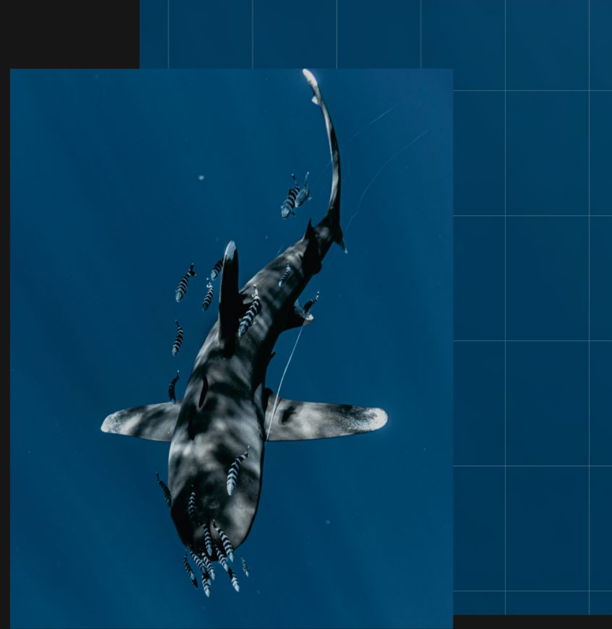
```
using CUeglFrameResource = std::unique_ptr<CUeglFrame>;
using CUgraphicsResourceResource = std::unique_ptr<CUgraphicsResource, CUgraphicsResourceDeleter>;

static std::unordered_map<unsigned, CUeglFrameResource> dmaCUEglMap;
static std::unordered_map<unsigned, CUgraphicsResourceResource> dmaCUResourceMap;
```



1 FD is associate with 1 CUeglFrame, which is associated with 1 CUgraphicsResource

All three represent the same

Memory layout matters



Internal format conversions to manipulate each color channel

nvarguscamerasrc → { **NV12**, P010_10LE }

nnvidconv { **NV12** → RGBA } →

CUDA filters → { RGBA }

nvvidconv { RGBA, **NV12** } →

nv3dsink → { **NV12**, RGBA, … }

Internal format conversions to manipulate each color channel

nvarguscamerasrc → { **NV12**, P010_10LE }

~~nnvideconv { **NV12** → RGBA } →~~

CUDA filters → { **NV12**, RGBA } → Implicit color conversion in CUDA kernel

Color conversion computation costs less than memory transfers

~~nvvideconv { RGBA, **NV12** } →~~

nv3dsink → { **NV12**, RGBA, … }

Where is the time?

```
Latency Statistics:
	0x559fb4b390.nvarguscamerasrc0.src|0x559fc17e20.xvimagesink0.sink: mean=0:00:00.037013052 min=0:00:00.031175391
max=0:00:00.351914664

Element Latency Statistics:
	0x559fc222e0.capsfilter0.src:                          mean=0:00:00.000122682 min=0:00:00.000082559 max=0:00:00.000272957
	0x559f8afcd0.nvvconv0.src:                             mean=0:00:00.002407058 min=0:00:00.002276289 max=0:00:00.005223577
	0x559fc22620.capsfilter1.src:                          mean=0:00:00.000373864 min=0:00:00.000099967 max=0:00:00.074877139
	0x559fc0eb10.fluspatialdistortioncorrect0.src:         mean=0:00:00.009992136 min=0:00:00.006858596 max=0:00:00.231934100
	0x559f971cd0.nvvconv1.src:                             mean=0:00:00.003503333 min=0:00:00.003276308 max=0:00:00.020730281
	0x559fc22960.capsfilter2.src:                          mean=0:00:00.000172287 min=0:00:00.000095807 max=0:00:00.001533836
	0x559fc11f50.fluchromaticaberrationcorrect0.src:       mean=0:00:00.008769558 min=0:00:00.007352990 max=0:00:00.057988277
	0x559f96fcd0.nvvconv2.src:                             mean=0:00:00.011665914 min=0:00:00.010606322 max=0:00:00.037716677
```

RGBA

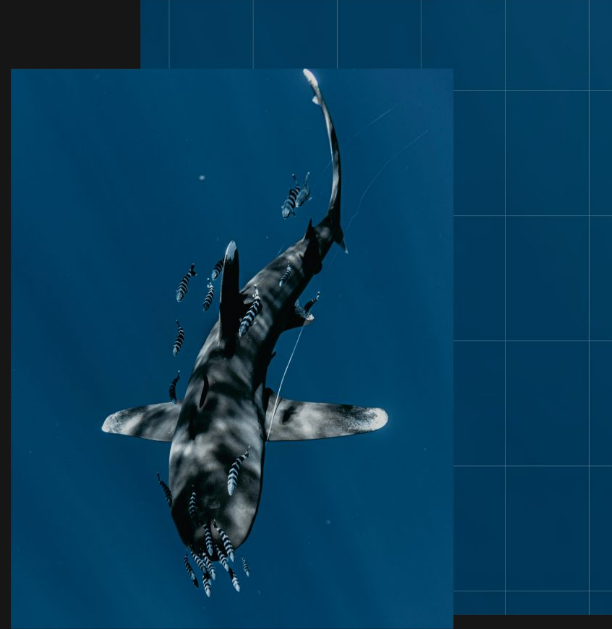
```
Latency Statistics:
	0x557014d370.nvarguscamerasrc0.src|0x55702199b0.xvimagesink0.sink: mean=0:00:00.030696960 min=0:00:00.028436184
max=0:00:00.274659614

Element Latency Statistics:
	0x55702222e0.capsfilter0.src:                          mean=0:00:00.000128253 min=0:00:00.000080735 max=0:00:00.000224285
	0x556feb1cd0.nvvconv0.src:                             mean=0:00:00.002346500 min=0:00:00.002275488 max=0:00:00.004628416
	0x5570222620.capsfilter1.src:                          mean=0:00:00.000140502 min=0:00:00.000101535 max=0:00:00.000363930
	0x5570210c90.fluspatialdistortioncorrect0.src:         mean=0:00:00.008124714 min=0:00:00.006649508 max=0:00:00.230229253
	0x5570214340.fluchromaticaberrationcorrect0.src:       mean=0:00:00.008412824 min=0:00:00.007854099 max=0:00:00.013519909
	0x556ff73cd0.nvvconv1.src:                             mean=0:00:00.011543717 min=0:00:00.010851913 max=0:00:00.027692896
```

NV12

Get rid of additional color conversions

- In RGBA
  - Full pipeline: 37ms
  - Only the filter element: 8ms
- In NV12
  - Full pipeline: 30ms
  - Only the filter element: 8ms

Picking the right sink



What are the available choices?

x11

- **xvimagesink**: does nvvidconv CPU copies → 10ms@4k
- **ximagesink**: does nvvidconv CPU copies → 10ms@4k
- nvegltransform ! **nveglglessink**: does GPU copies and transformation → 10ms@4k. It is compatible with NVMM, but not for the Jetson GPU memory
  - NVBUF_MEM_SURFACE_ARRAY memory is not compatible nveglglessink, only dGPU memory is actually compatible. So, nvegltransform is needed, then it takes a lot of time anyway
- **nv3sink**: It is compatible with NVMM memory and does not require copies
  - It only works in x11
  - This video sink element works with NVMM buffers and renders using the 3D graphics rendering API. It performs better than nveglglessink with NVMM buffers
- **nvdrmvideosink**: it requires to use DRM, so gdm or any other backend must be stopped to work. It can be useful to research when doing compositions:
  - There is an element called nvdrmvideosink which directly interacts with the Display and it is more efficient than using nveglglessink and waylandsink This does not work out of the box in the Jetpack 32.7.1

Wayland

- **waylandsink**: does nvvidconv CPU copies → 10ms@4k
- nvegltransform ! **nveglglessink**: same issue as x11

Where is the time?

```
Latency Statistics:
        0x557913c0e0.nvarguscamerasrc0.src|0x5579207b60.xvimagesink0.sink: mean=0:00:00.030714655 min=0:00:00.028569533
max=0:00:00.266009013

Element Latency Statistics:
        0x5579212140.capsfilter0.src:                         mean=0:00:00.000142151 min=0:00:00.000075008 max=0:00:00.000527009
        0x5578fc3cd0.nvvconv0.src:                            mean=0:00:00.002344703 min=0:00:00.002275877 max=0:00:00.004012456
        0x5579212480.capsfilter1.src:                         mean=0:00:00.000145054 min=0:00:00.000100384 max=0:00:00.000625185
        0x5579200180.fluspatialdistortioncorrect0.src:        mean=0:00:00.007830025 min=0:00:00.006674671 max=0:00:00.218922897
        0x5579202810.fluchromaticaberrationcorrect0.src:      mean=0:00:00.008514082 min=0:00:00.007824752 max=0:00:00.013942782
        0x55792127c0.capsfilter2.src:                         mean=0:00:00.000144500 min=0:00:00.000097920 max=0:00:00.001081506
        0x5578fb84d0.nvvconv1.src:                            mean=0:00:00.011594695 min=0:00:00.010991608 max=0:00:00.028704285
```

xvimagesink

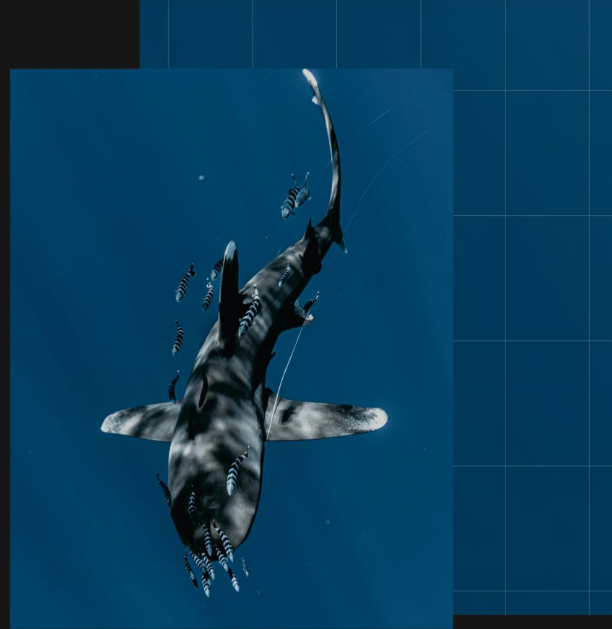
```
Latency Statistics:
        0x556dd511f0.nvarguscamerasrc0.src|0x556de1c740.nv3dsink0.sink: mean=0:00:00.019868694 min=0:00:00.017158788
max=0:00:00.217951719

Element Latency Statistics:
        0x556de24130.capsfilter0.src:                         mean=0:00:00.000143625 min=0:00:00.000063936 max=0:00:00.000712961
        0x556dbfb4d0.nvvconv0.src:                            mean=0:00:00.002406852 min=0:00:00.002264580 max=0:00:00.008775443
        0x556de24470.capsfilter1.src:                         mean=0:00:00.000162815 min=0:00:00.000103328 max=0:00:00.000714178
        0x556de15210.fluspatialdistortioncorrect0.src:        mean=0:00:00.008124731 min=0:00:00.006724783 max=0:00:00.195528409
        0x556de177b0.fluchromaticaberrationcorrect0.src:      mean=0:00:00.008863787 min=0:00:00.007748785 max=0:00:00.098920622
        0x556de247b0.capsfilter2.src:                         mean=0:00:00.000166882 min=0:00:00.000093888 max=0:00:00.001982244
```

nv3dsink

Keep the memory in the GPU all the time

- xvimagesink
  - Full pipeline: 30ms
    - nvvidconv: 11ms
- nv3dsink
  - Full pipeline: 19ms
    - ~~nvvidconv: 11ms~~

All together



Deploying the solution on the client embedded device with a different camera

```
GST_PLUGIN_PATH=/tmp:$GST_PLUGIN_PATH GST_DEBUG="GST_TRACER:7" GST_TRACERS="latency(flags=pipeline+element)" GST_DEBUG_FILE=trace.log \
gst-launch-1.0 \
        nvv4l2camerasrc device=/dev/video0 ! 'video/x-raw(memory:NVMM), format=(string)UYVY, width=(int)3840, height=(int)2160' ! \
        nvvidconv  bl-output=false ! "video/x-raw(memory:NVMM),format=NV12" ! \
        fluspatialdistortioncorrect ! \
        fluchromaticaberrationcorrect ! \
        fluglarecorrect ! \
        nv3dsink

…

Latency Statistics:
        0x557da6a0e0.nvv4l2camerasrc0.src|0x557da97b90.nv3dsink0.sink: mean=0:00:00.030956279 min=0:00:00.027534081
max=0:00:00.137824612

Element Latency Statistics:
        0x557daa42b0.capsfilter0.src:                mean=0:00:00.000097974 min=0:00:00.000070303 max=0:00:00.000518528
        0x557d652cd0.nvvconv0.src:                           mean=0:00:00.003939072 min=0:00:00.003841012 max=0:00:00.007119752
        0x557daa45f0.capsfilter1.src:                mean=0:00:00.000162690 min=0:00:00.000080704 max=0:00:00.000417376
        0x557da40380.fluspatialdistortioncorrect0.src:       mean=0:00:00.008543783 min=0:00:00.005281835 max=0:00:00.108734500
        0x557da3d780.fluchromaticaberrationcorrect0.src:     mean=0:00:00.007792492 min=0:00:00.007659260 max=0:00:00.012034112
        0x557da9c1c0.fluglarecorrect0.src:                   mean=0:00:00.010420265 min=0:00:00.010239976 max=0:00:00.045371138
```

With all the filters enabled, we are still under 33ms in 4K

For what is all time reduction?

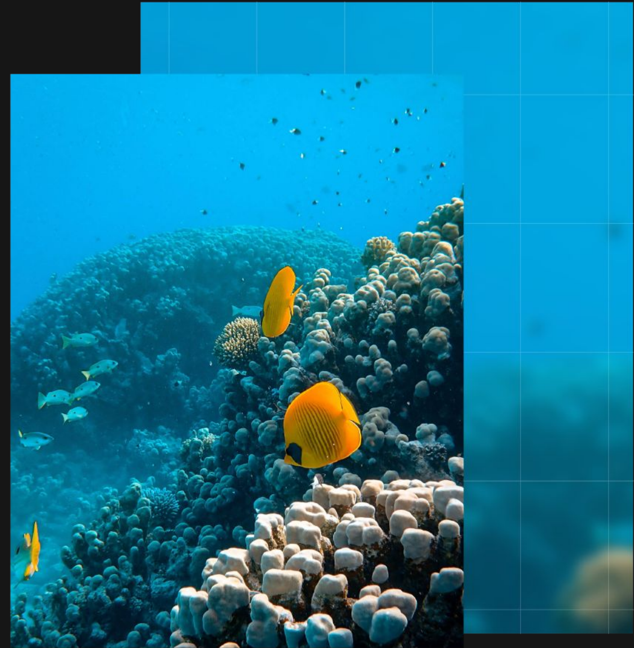
	Pipeline time	G2G approx.	Frames duplicated
Baseline	-	168ms	N.A.
Original GL	28.3ms	190ms	N.A.
Improved GL	22.6ms	N.A.	N.A.
Original CA	27.9ms	203ms	N.A.
Improved CA	22.8ms	N.A.	N.A.
Original SD + CA + GL	51.5ms	~600ms	347
Improved SD + CA + GL	36.2ms	~250-300ms	211
Improved SD + CA + GL + nv3dsink	30.9ms	~200ms	12

SD: Spatial distortion – CA: Chromatic aberration – GL: Glare

And that's all?

- Why the device baseline G2G is about 160ms at 4K?
  - Can we work at NVIDIA driver-kernel side (Jetpack sources)
- Is it possible to re-write these algorithms differently, in a way which performs better?
- Are there newer Jetpack versions which help to reduce even more the latency?

Questions?



Stop taking risks. Start finding solutions.

Thank you!

```
$ gst-discoverer-1.0 AUD_MW_E.264
Analyzing file:///home/fluendo/Videos/AUD_MW_E.264
Done discovering
file:///home/fluendo/Videos/AUD_MW_E.264

Properties:
  Duration: 0:00:00.000000000
  Seekable: yes
  Live: no
  video #0: H.264 (Constrained Baseline Profile)
    Stream ID:
92ec0aada08d35b7679f87a98465b4cf955fbad11d2c8535ec
ad7e0568ac4a3a
    Width: 176
    Height: 144
    Depth: 24
    Frame rate: 0/1
    Pixel aspect ratio: 1/1
    Interlaced: false
    Bitrate: 0
    Max bitrate: 0
```

fluendo