Summary of working pipelines

**Display pipelines used to access the camera USB or integrated camera**

//shows the original stream from the camera that is located at /dev/video0 mostly used for USB devices, will not work if we are connecting the the Xavier camera port

// you can check to see if the camera Is accessible by running **gst-device-monitor-1.0** which will show all available devices in linux

//integrated camera

gst-launch-1.0 v4l2src device="/dev/video0" ! video/x-raw,width=640,height=480 ! autovideosink

//USB camera (Logitech C920)

gst-launch-1.0 v4l2src device="/dev/video0" ! "video/x-raw, width=640, height=480, format=(string)YUY2" ! xvimagesink -e

**Display pipelines used to access the camera Xavier camera connector**

//stream video from Jetson Xavier camera that is attached to the Xavier camera port

gst-launch-1.0 nvarguscamerasrc ! nvvidconv ! xvimagesink

// stream video from Jetson Xavier camera that is attached to the Xavier camera port at a certain frame rate and quality

gst-launch-1.0 nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=640, height=480, framerate=30/1, format=NV12' ! nvvidconv flip-method=2 ! nvegltransform ! nveglglessink -e

**.H264 Encoding from stream to .mp4 container**

gst-OMX

//works but doesn’t save video to the file

//problem with the camera probally

FILE=filename.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1'! nvvidconv ! 'video/x-raw(memory:NVMM), format=(string)I420' ! omxh264enc bitrate=8000000 ! 'video/x-h264, stream-format=(string)byte-stream'

! h264parse ! qtmux ! filesink location=$FILE -e

Dual .H264 encoding omx

//hasn’t been tested

FILE\_A=filenameA.mp4

FILE\_B=filenameB.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' **\**

! nvvidconv ! 'video/x-raw(memory:NVMM), format=(string)I420' ! tee name=streams **\**

streams. ! omxh264enc bitrate=8000000 ! 'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! qtmux ! filesink location=$FILE\_A **\**

streams. ! omxh264enc bitrate=8000000 ! 'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! qtmux ! filesink location=$FILE\_B

NOTE: This is not terminating the pipeline I think when it finishes, (probably because of the camera we are using. The –e at the end is supposed to properly terminate the command.

Gst-V4L2

//error saying “Failed to create capture session”

FILE=filename.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' **\**

! nvv4l2h264enc bitrate=8000000 ! h264parse ! qtmux ! filesink location=$FILE -e

Dual .h264 encoding V4L2

//haven’t tested

FILE\_A=filenameA.mp4

FILE\_B=filenameB.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' ! tee name=streams **\**

streams. ! nvv4l2h264enc bitrate=8000000 ! h264parse ! qtmux ! filesink location=$FILE\_A **\**

streams. ! nvv4l2h264enc bitrate=8000000 ! h264parse ! qtmux ! filesink location=$FILE\_B

NOTE: This is not terminating the pipeline I think when it finishes, (probally because of the camera we are using. The –e at the end is supposed to properly terminate the command.

**.H264 Encoding from stream to .h264 file**

//Xavier encoding for .264 using omx

//takes a stream from the camera and creates a .264 file from it

//works

gst-launch-1.0 nvarguscamerasrc ! nvvidconv ! 'video/x-raw, format=(string)I420, width=(int)640, height=(int)480' ! omxh264enc ! filesink location=test.h264 -e

**.H264 Decoding from .mp4 container to display**

gst-OMX

//works

//HW Decode

FILE=filename.mp4

gst-launch-1.0 filesrc location=$FILE ! qtdemux name=demux demux.video\_0 ! queue ! h264parse ! omxh264dec ! nveglgessink -e

gst-V4L2

//works

//HW decode

FILE=filename.mp4

gst-launch-1.0 filesrc location=$FILE ! qtdemux ! queue ! h264parse ! nvv4l2decoder ! nv3dsink -e

**.H264 Decoding from .h264 file to display**

//V412

//no valid frames before end of stream

//probably from camera or not ending the .h264 file correctly

gst-launch-1.0 filesrc location= test.h264! h264parse ! nvv4l2decoder ! nvoverlaysink –e

//omx

//no valid frames before end of stream

//probably from camera or not ending the .h264 file correctly

gst-launch-1.0 filesrc location=test.h264 ! h264parse ! omxh264dec ! nveglglessink –e

**.H265 Encoding from stream to .mp4 container**

gst-OMX

FILE=filename.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' ! nvvidconv ! 'video/x-raw(memory:NVMM), format=(string)I420' ! omxh265enc bitrate=8000000 ! 'video/x-h265, stream-format=(string)byte-stream' ! h265parse ! qtmux ! filesink location=$FILE -e

dual h265 encoding omx

FILE\_A=filenameA.mp4

FILE\_B=filenameB.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' **\**

! nvvidconv ! 'video/x-raw(memory:NVMM), format=(string)I420' ! tee name=streams **\**

streams. ! omxh265enc bitrate=8000000 ! 'video/x-h265, stream-format=(string)byte-stream' ! h265parse ! qtmux ! filesink location=$FILE\_A **\**

streams. ! omxh265enc bitrate=8000000 ! 'video/x-h265, stream-format=(string)byte-stream' ! h265parse ! qtmux ! filesink location=$FILE\_B

gst-V4L2

FILE=filename.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' **\**

! nvv4l2h265enc bitrate=8000000 ! h265parse ! qtmux ! filesink location=$FILE -e

dual h265 encoding V412

FILE\_A=filenameA.mp4

FILE\_B=filenameB.mp4

gst-launch-1.0 -e nvarguscamerasrc ! 'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, format=(string)NV12, framerate=(fraction)30/1' ! tee name=streams **\**

streams. ! nvv4l2h265enc bitrate=8000000 ! h265parse ! qtmux ! filesink location=$FILE\_A **\**

streams. ! nvv4l2h265enc bitrate=8000000 ! h265parse ! qtmux ! filesink location=$FILE\_B

**.H265 Encoding from stream to .h265 file**

//Xavier encoding for .265 using omx //this works

//takes a stream from the camera and creates a .265 file from it

gst-launch-1.0 nvarguscamerasrc! nvvidconv ! 'video/x-raw, format=(string)I420, width=(int)640, height=(int)480' ! omxh265enc ! filesink location=test.h265 –e

**.H265 Decoding from .mp4 to display**

gst-OMX

FILE=filename.mp4

gst-launch-1.0 filesrc location=$FILE ! qtdemux name=demux demux.video\_0 ! queue ! h265parse ! omxh265dec ! nvoverlaysink -e

gst-V4L2

FILE=filename.mp4

gst-launch-1.0 filesrc location=$FILE ! qtdemux ! queue ! h265parse ! nvv4l2decoder ! nv3dsink -e

**.H265 Decoding from .h265 to display**

//Xavier decoding isn’t working but I think its because we have a problem with encoding in correct format maybe? Not sure. I get an error that says frames //Xavier decoding for .264 using omx

//omx

gst-launch-1.0 filesrc location=test.h265 ! h265parse ! omxh265dec ! nvoverlaysink -e

//412

gst-launch-1.0 filesrc location=test.h265 ! h265parse ! nvv4l2decoder ! nv3dsink -e

//another way to display

gst-launch-1.0 filesrc location=test.h265 ! h265parse ! nvv4l2decoder ! nvoverlaysink -e

**Encoding a set of images into a .avi container**

//used to take a set of images and decode them into a video

//need to test with images, don’t have images to test with right now

gst-launch-1.0 multifilesrc location="./images/image-%06d.jpg" ! "image/jpeg,framerate=12/1" jpegparse ! jpegdec ! x264enc avimux ! filesink location=video.avi

//a pipeline to mux 5 JPEG frames per second into a 10 sec. long motion jpeg avi.

gst-launch-1.0 videotestsrc num-buffers=50 ! video/x-raw, framerate='(fraction)'5/1 ! jpegenc ! avimux ! filesink location=mjpeg.avi

**Display device resolutions**

//shows formats that the device can use

v4l2-ctl -d /dev/video1 --list-formats-ext

**Kill any process that is holding a camera up**

//kills camera device if it gets hung up

fuser /dev/video1 -k

**If there is a memory leak from a plugin it will most likely be located a temp/ syslog**

**Remove that file**

**NVGSTCAPTURE**

**.H264 encode video from CSI Stream**

//nvgstcapture-1.0 –-help

//shows all the options for the command to capture video and perform operations on it

//.h264 encode

nvgstcapture-1.0 --video-res=2 --camsrc=1 --mode=2 --video-enc=0 --file-type=0 –overlayConfig=”0,0,0,640,480”

**.H264 encode video from USB Stream**

//nvgstcapture-1.0 –-help

//shows all the options for the command to capture video and perform operations on it

//works

nvgstcapture-1.0 --video-res=2 --camsrc=0 --mode=2 --video-enc=0 --hw-enc-path=1 --cap-dev-node=1 --color-format=2

**.H265 encode video from CSI Stream**

//nvgstcapture-1.0 –-help

//shows all the options for the command to capture video and perform operations on it

nvgstcapture-1.0 --video-res=2 --camsrc=1 --mode=2 --video-enc=2 --file-type=0

**.H265 encode video from USB Stream**

//nvgstcapture-1.0 –-help

//shows all the options for the command to capture video and perform operations on it

nvgstcapture-1.0 --video-res=2 --camsrc=0 --mode=2 --video-enc=2 --hw-enc-path=1 --cap-dev-node=1 –file-type=0 --color-format=2

**Encode Sequence of images**

//nvgstcapture-1.0 –-help

//needs work may be able to use

nvgstcapture-1.0 --video-res=2 --camsrc=0 –mode=2 --video-enc=2 –hw-enc-path=1 --cap-dev-node=1 --file\_type=0 –color-format=2 –-streaming-file=filelocation

**Jetson Stats**

//used to show stats of the Xavier as we run it

//pip install jetson-stats

sudo jtop

**Run Deepstream Examples**

//used to run deepstream examples

1. Go to location of the samples /opt/nvidia/deepstream/deepstream-4.0/sources/objectDetector\_Yolo

2. $ deepstream-app –c deepstream\_app\_config\_yoloV3\_tiny.txt

Note: -c is the location of the config file

Note: sources/app name will change based on model you want to run

Note: Read the README may have to install models and weights located under the sources/appname…

**L4T Multi Media API**

**Documentation** [**https://docs.nvidia.com/jetson/l4t-multimedia/index.html**](https://docs.nvidia.com/jetson/l4t-multimedia/index.html)

**Encode stream to H264 and H265 files**

//use a sample called 10\_camera\_recording

//located at : /usr/src/jetson\_multimedia\_api/samples/10\_camera\_recording

//make

//./camera\_recording to run

//Options

printf("Usage: camera\_recording [OPTIONS]\n"

"Options:\n"

" -r Set output resolution WxH [Default 640x480]\n"

" -f Set output filename [Default output.h264]\n"

" -t Set encoder type H264 or H265 [Default H264]\n"

" -d Set capture duration [Default 5 seconds]\n"

" -s Enable profiling\n"

" -v Enable verbose message\n"

" -c Enable demonstration of CPU processing\n"

" -h Print this help\n");

//H264 Pipeline

//works

$ ./camera\_recording -r 640x480 –f home/output.h264 –t H264 –d 10

//H265 Pipeline

//works

$ ./camera\_recording -r 640x480 –f /home/output.h265 –t H265 –d 10

**Decode H264 and H265 files to mp4**

//

video\_dec\_cuda <in-file> <in-format> [options]

Supported formats:

H264

H265

OPTIONS:

-h,--help Prints this text

--dbg-level <level> Sets the debug level [Values 0-3]

--disable-rendering Disable rendering

--fullscreen Fullscreen playback [Default = disabled]

-ww <width> Window width in pixels [Default = video-width]

-wh <height> Window height in pixels [Default = video-height]

-wx <x-offset> Horizontal window offset [Default = 0]

-wy <y-offset> Vertical window offset [Default = 0]

-fps <fps> Display rate in frames per second [Default = 30]

-o <out-file> Write to output file

-f <out\_pixfmt> 1 NV12, 2 I420 [Default = 1]

--input-nalu Input to the decoder will be nal units

--input-chunks Input to the decoder will be a chunk of bytes [Default]

--bbox-file bbox file path

--display-text <string> enable nvosd text overlay with input string

//decodes a .h264 file

$ ./video\_dec\_cuda /home/output.h264 H264 –o newfile.mp4

//decodes a .h265 file

$ ./video\_dec\_cuda /home/output.h265 H265 –o newfile.mp4

**Encode file to H264 and H265 files**

//video\_encode <in-file> <in-width> <in-height> <encoder-type> <out-file> [OPTIONS]

Encoder Types:

H264

H265

OPTIONS:

-h,--help Prints this text

--dbg-level <level> Sets the debug level [Values 0-3]

-br <bitrate> Bitrate [Default = 480000000]

-pbr <peak\_bitrate> Peak bitrate [Default = 1.2\*bitrate]

NOTE: Peak bitrate takes effect in VBR more; must be >= bitrate

-p <profile> Encoding Profile [Default = baseline]

-rc <rate-control> Ratecontrol mode [Default = cbr]

--elossless Enable Lossless encoding [Default = disabled]

-ifi <interval> I-frame Interval [Default = 30]

-idri <interval> IDR Interval [Default = 256]

--insert-spspps-idr Insert SPS PPS at every IDR [Default = disabled]

--insert-vui Insert VUI [Default = disabled]

--enable-extcolorfmt Set Extended ColorFormat (Only works with insert-vui) [Default = disabled]

--insert-aud Insert AUD [Default = disabled]

-fps <num> <den> Encoding fps in num/den [Default = 30/1]

-tt <level> Temporal Tradeoff level [Default = 0]

-vbs <size> Virtual buffer size [Default = 0]

-nrf <num> Number of reference frames [Default = 1]

-slt <type> Slice length type (1 = Number of MBs, 2 = Bytes) [Default = 1]

-hpt <type> HW preset type (1 = ultrafast, 2 = fast, 3 = medium, 4 = slow)

-slen <length> Slice length [Default = 0]

-sir <interval> Slice intrarefresh interval [Default = 0]

-nbf <num> Number of B frames [Default = 0]

-rpc <string> Change configurable parameters at runtime

-goldcrc <string> GOLD CRC

--rcrc Reconstructed surface CRC

-rl <cordinate> Reconstructed surface Left cordinate [Default = 0]

-rt <cordinate> Reconstructed surface Top cordinate [Default = 0]

-rw <val> Reconstructed surface width

-rh <val> Reconstructed surface height

-rcrcf <reconref\_file\_path> Specify recon crc reference param file

--report-metadata Print encoder output metadata

--blocking-mode <val> Set blocking mode, 0 is non-blocking, 1 for blocking (Default)

--input-metadata Enable encoder input metadata

--copy-timestamp <st> Enable copy timestamp with start timestamp(st) in seconds

--mvdump Dump encoded motion vectors

--eroi Enable ROI [Default = disabled]

-roi <roi\_file\_path> Specify roi param file

--erps Enable External RPS [Default = disabled]

--egdr Enable GDR [Default = disabled]

--gif Enable Gaps in FrameNum [Default = disabled]

-fnb <num\_bits> H264 FrameNum bits [Default = 0]

-plb <num\_bits> H265 poc lsb bits [Default = 0]

--ni No I-frames [Default = disabled]

-rpsf <rps\_file\_path> Specify external rps param file

--erh Enable External picture RC [Default = disabled]

-mem\_type\_oplane <num> Specify memory type for the output plane to be used [1 = V4L2\_MEMORY\_MMAP, 2 = V4L2\_MEMORY\_USERPTR, 3 = V4L2\_MEMORY\_DMABUF]

-gdrf <gdr\_file\_path> Specify GDR Parameters filename

-gdrof <gdr\_out\_file\_path> Specify GDR Out filename

-smq <max\_qp\_value> Max QP per session when external picture RC enabled

-hf <hint\_file\_path> Specify external rate control param file

-MinQpI Specify minimum Qp Value for I frame

-MaxQpI Specify maximum Qp Value for I frame

-MinQpP Specify minimum Qp Value for P frame

-MaxQpP Specify maximum Qp Value for P frame

-MinQpB Specify minimum Qp Value for B frame

-MaxQpB Specify maximum Qp Value for B frame

NOTE: roi parameters need to be feed per frame in following format

<no. of roi regions> <Qpdelta> <left> <top> <width> <height> ...

e.g. [Each line corresponds roi parameters for one frame]

2 -2 34 33 16 19 -3 68 68 16 16

1 -5 40 40 40 40

3 -4 34 34 16 16 -5 70 70 18 18 -3 100 100 34 34

Supported Encoding profiles for H.264:

baseline main high

Supported Encoding profiles for H.265:

main

Supported Encoding rate control modes:

cbr vbr

Supported Temporal Tradeoff levels:

0:Drop None 1:Drop 1 in 5 2:Drop 1 in 3

3:Drop 1 in 2 4:Drop 2 in 3

Runtime configurable parameter string should be of the form:

"f<frame\_num1>,<prop\_id1><val>,<prop\_id2><val>;f<frame\_num1>,<prop\_id1><val>,<prop\_id2><val>;..."

e.g. "f20,b8000000,i1;f300,b6000000,r40/1"

Property ids:

b<bitrate> Bitrate

p<peak\_bitrate> Peak Bitrate

r<num/den> Framerate

i1 Force I-frame

NOTE: These encoding parameters are slightly imprecisely updated depending upon the number of

frames in queue and/or processed already.

//h264

$ ./video\_encode /home/testfile.yuv 480 360 H264 newfile.h264

//h265

$ ./video\_encode /home/testfile.yuv 480 360 H265 newfile.h265

**Encode JPEG images**

jpeg-encode <in-file> <in-width> <in-height> <out-file> [OPTIONS]

OPTIONS:

-h,--help Prints this text

--dbg-level <level> Sets the debug level [Values 0-3]

--encode-fd Uses FD as input to encoder [DEFAULT]

--encode-buffer Uses buffer as input to encoder

-f <pixfmt> Color format of input to encoder (works only for --encode-fd) [1=YUV420(Default), 2=NV12]

-crop <left> <top> <width> <height> Cropping rectangle for JPEG encoder

-scale\_encode <scale\_width> <scale\_height> Scale encoding with given scaled width and height encoder

-quality <value> Sets the image quality [75(default)]

//encode into a jpeg

$ ./jpeg-encode /home/testfile.yuv 480 360 testfile.jpg

**Decode JPEG images to .yuv**

jpeg-decode num\_files <num\_files> <in-file1> <out-file1> <in-file2> <out-file2> [OPTIONS]

OPTIONS:

-h,--help Prints this text

num\_files number of files to decode simultaneously

--dbg-level <level> Sets the debug level [Values 0-3]

--decode-fd Uses FD as output of decoder [DEFAULT]

--decode-buffer Uses buffer as output of decoder

-s <loop-count> Stress test [Default = 1]

$ ./jpeg-decode num\_files 1 /home/testpic1.jpg /home/testconverted.yuv

**Pipelines for encoding images and decoding**

IMAGE DECODE

gst-launch-1.0 filesrc location=<filename.jpg> ! nvjpegdec ! \ imagefreeze ! xvimagesink -e

**USE FFMEG T CONVERT ALL OUR VIDEOS TO YUV FORMAT**

Almost all video decodes to YCbCr 4:2:0, aka IYUV ([fourcc](https://www.fourcc.org/pixel-format/yuv-i420/)), hence

**ffmpeg -i sample.mp4 out.yuv**

# Using ffmpeg

ffplay -s WxH file.yuv

**//data that I got back after converting**

**Output #0, rawvideo, to 'out.yuv':**

**Metadata:**

**major\_brand : isom**

**minor\_version : 512**

**compatible\_brands: isomiso2mp41**

**encoder : Lavf58.29.100**

**Stream #0:0(und): Video: rawvideo (I420 / 0x30323449), yuv420p, 720x480 [SAR 1:1 DAR 3:2], q=2-31, 124416 kb/s, 30 fps, 30 tbn, 30 tbc (default)**

Power, heat, GPU memory, CPU memory to the excel file

**//used to record the averages across the CPUs**

mpstat -P ALL 2 1000