configuration (e.g. width, color, and opacity) of a given bounding box. It also draws text and RoI polygons at specified locations in the frame. Text and polygon parameters are configurable through metadata.

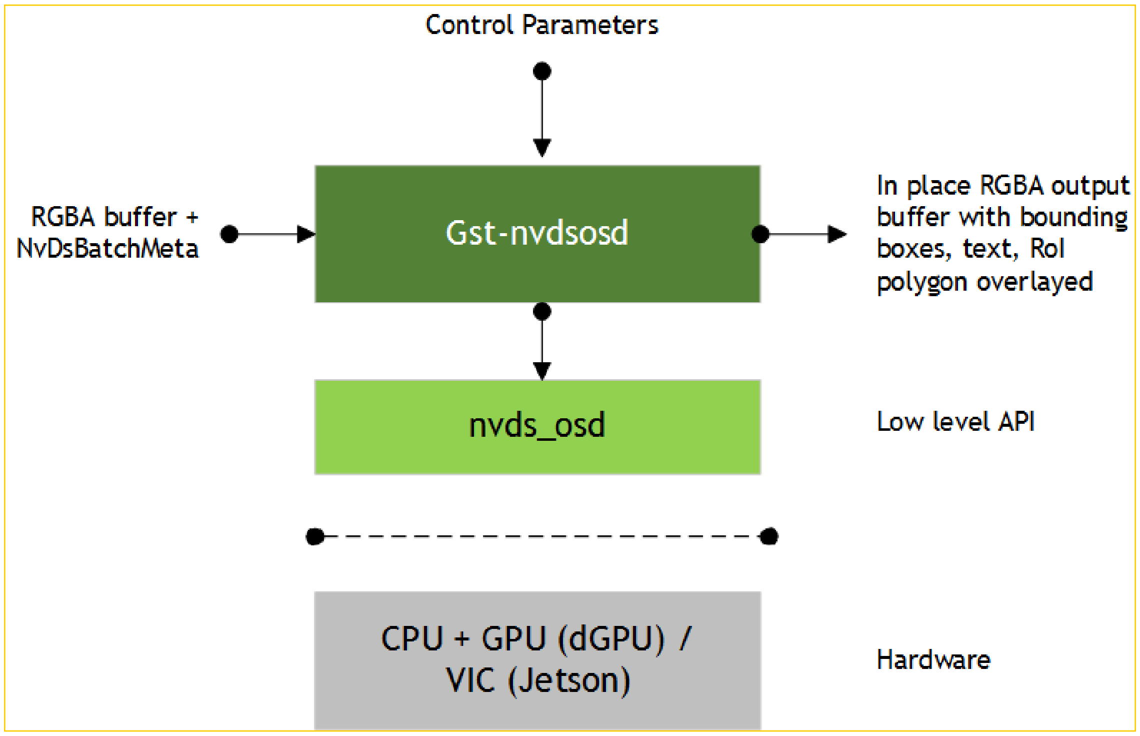
Control Parameters
RGBA buffer + NvDsBatchMeta
Gst-nvdsosd
In place RGBA output buffer with bounding boxes, text, RoI polygon overlayed
nvds_osd
Low level API
CPU + GPU (dGPU) / VIC (Jetson)
Hardware

Figure 6. The Gst-nvdsosd plugin

2.6.1 Inputs and Outputs

- Inputs
  - RGBA buffer
  - `NvDsBatchMeta` (holds `NvDsFrameMeta` consisting of bounding boxes, text parameters, and lines parameters)
  - `NvDsLineMeta` (RoI polygon)
- Control parameters
  - `gpu-id` (dGPU only)
  - `display-clock`
  - `clock-font`
  - `clock-font-size`
  - `x-clock-offset`
  - `y-clock-offset`
  - `clock-color`

- Output
  - RGBA buffer modified in place to overlay bounding boxes, texts, and polygons represented in the metadata

2.6.2 Features

Table 13 summarizes the features of the plugin.

Table 13. Features of the Gst-nvdsosd plugin

Feature	Description	Release
Supports drawing polygon lines	—	DS 3.0
Supports drawing text using Pango and Cairo libraries	—	DS 2.0
VIC (Jetson only) and GPU support for drawing bounding boxes	—	DS 2.0

2.6.3 Gst Properties

Table 14 describes the Gst properties of the `Gst-nvdsosd` plugin.

Table 14. Gst-nvdsosd plugin, Gst Properties

Property	Meaning	Type and Range	Example *Notes*
gpu-id	Device ID of the GPU to be used for operation **(dGPU only)**	Integer, 0 to 4,294,967,295	gpu-id=0
display-clock	Indicates whether to display system clock	Boolean	display-clock=0
clock-font	Name of Pango font to use for the clock	String	clock-font=Arial
clock-font-size	Font size to use for the clock	Integer, 0-60	clock-font-size=2
x-clock-offset	X offset of the clock	Integer, 0 to 4,294,967,295	x-clock-offset=100
y-clock-offset	Y offset of the clock	Integer, 0 to 4,294,967,295	y-clock-offset=50
clock-color	Color of the clock to be set while display, in the order 0xRGBA	Integer, 0 to 4,294,967,295	clock-color=0xff0000ff *(Clock is red with alpha=1)*

2.7 GST-NVVIDEOCONVERT

This plugin performs video color format conversion. It accepts NVMM memory as well as RAW (memory allocated using `calloc()` or `malloc()`), and provides NVMM or RAW memory at the output.

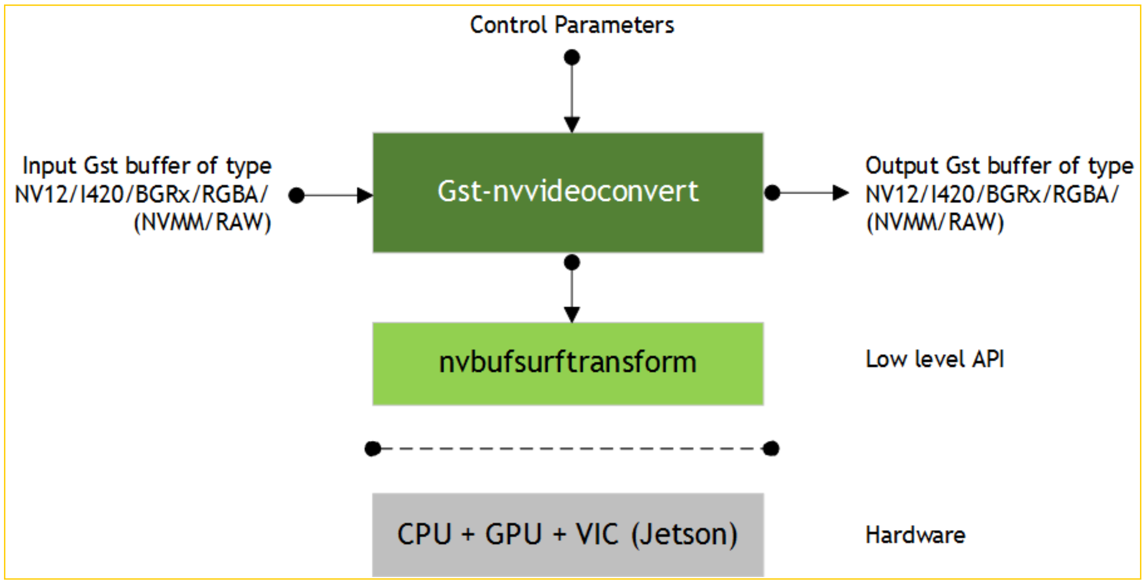
Control Parameters
Input Gst buffer of type NV12/I420/BGRx/RGBA/ (NVMM/RAW)
Gst-nvvideoconvert
Output Gst buffer of type NV12/I420/BGRx/RGBA/ (NVMM/RAW)
nvbufsurftransform
Low level API
CPU + GPU + VIC (Jetson)
Hardware

Figure 7. The Gst-nvvideoconvert plugin

2.7.1 Inputs and Outputs

- Inputs
  - Gst Buffer batched buffer
  - `NvDsBatchMeta`

  Format: NV12, I420, BGRx, RGBA (NVMM/RAW)
- Control parameters
  - `gpu-id` (dGPU only)
  - `nvbuf-memory-type`
  - `src-crop`
  - `dst-crop`
  - `interpolation-method`
  - `compute-hw`
- Output
  - Gst Buffer
  - `NvDsBatchMeta`
  - Format: NV12, I420, BGRx, RGBA (NVMM/RAW)

2.7.2 Features

This plugin supports batched scaling and conversion in single call for NVMM to NVMM, RAW to NVMM, and NVMM to RAW buffer types. It does not support RAW to RAW conversion. The plugin supports cropping of the input and output frames.

2.7.3 Gst Properties

Table 15 describes the Gst properties of the `Gst-nvvideoconvert` plugin.

Table 15. Gst-nvvideoconvert plugin, Gst Properties

Property	Meaning	Type and Range	Example *Notes*
nvbuf-memory-type	Type of memory to be allocated. **For dGPU**: 0 (`nvbuf-mem-default`): Default memory, `cuda-device` 1 (`nvbuf-mem-cuda-pinned`): Pinned/Host CUDA memory 2 (`nvbuf-mem-cuda-device`) Device CUDA memory 3 (`nvbuf-mem-cuda-unified`): Unified CUDA memory **For Jetson**: 0 (`nvbuf-mem-default`): Default memory, surface array 4 (`nvbuf-mem-surface-array`): Surface array memory	enum GstNvVidConvBufMemoryType	
src-crop	Pixel location: left:top:width:height	String	20; 40; 150; 100
dst-crop	Pixel location: left:top:width:height	String	20; 40; 150; 100

Property	Meaning	Type and Range	Example *Notes*
interpolation-method	Interpolation method. 0: Nearest 1: Bilinear 2: Algo-1 (GPU—Cubic, VIC—5 Tap) 3: Algo-2 (GPU—Super, VIC—10 Tap) 4: Algo-3 (GPU—LanzoS, VIC—Smart) 5: Algo-4 (GPU—Ignored, VIC—Nicest) 6: Default (GPU—Nearest, VIC—Nearest)	enum GstInterpolationMethod	interpolation-method=1 *Default value is 6.*
compute-hw	Type of computing hardware 0: Default (GPU for gDPU, VIC for Jetson) 1: GPU 2: VIC	enum GstComputeHW	compute-hw=0 *Default value is 0.*
gpu-id	Device ID of GPU to use for format conversion	Integer, 0 to 4,294,967,295	gpu-id=0
output-buffers	Number of Output Buffers for the buffer pool	Unsigned integer, 1 to 4,294,967,295	output-buffers=4

2.8 GST-NVDEWARPER

This plugin dewarps 360° camera input. It accepts `gpu-id` and `config-file` as properties. Based on the selected configuration of surfaces, it can generate a maximum of four dewarped surfaces. It currently supports dewarping of two projection types, `NVDS_META_SURFACE_FISH_PUSHBROOM` and `NVDS_META_SURFACE_FISH_VERTCYL`. Both of these are used in 360-D use case.

The plugin performs its function in these steps:

1. Reads the configuration file and creates a vector of surface configurations. It supports a maximum of four dewarp surface configurations.
2. Receives the 360-D frame from the decoder; based on the configuration, generates up to four dewarped surfaces.
3. Scales these surfaces down to network / selected dewarper output resolution using NPP APIs.
4. Pushes a buffer containing the dewarped surfaces to the downstream component.

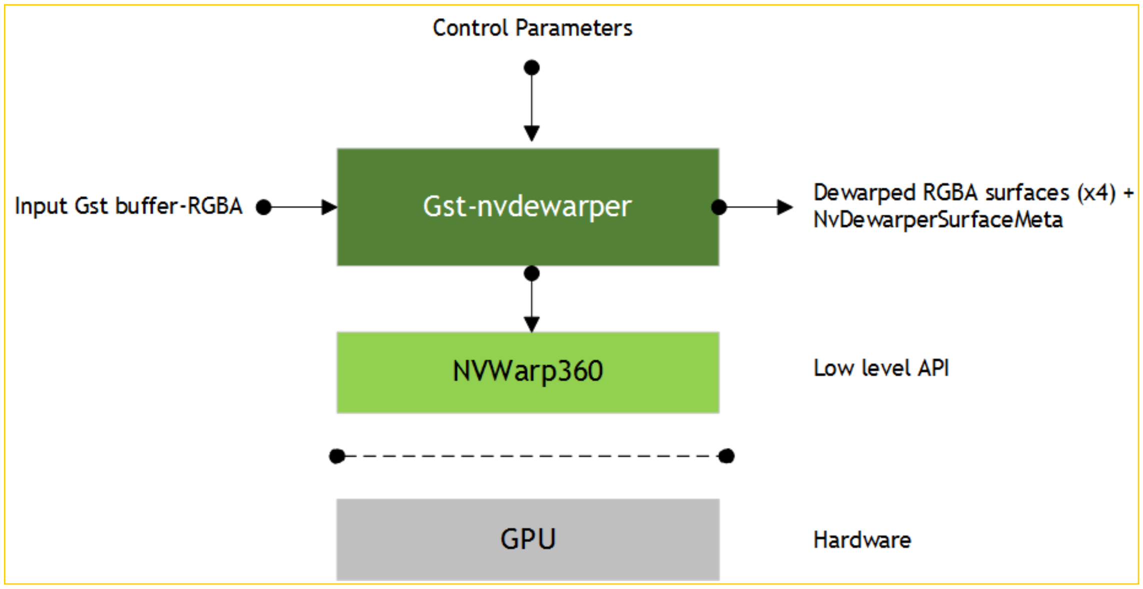

Figure 8. The Gst-nvdewarper plugin

2.8.1 Inputs and Outputs

- Inputs
  - A buffer containing a 360-D frame in RGBA format
- Control parameters
  - `gpu-id`; selects the GPU ID (dGPU only)
  - `config-file`, containing the pathname of the dewarper configuration file
- Output
  - Dewarped RGBA surfaces
  - `NvDewarperSurfaceMeta` with information associated with each surface (`projection_type`, `surface_index`, and `source_id`), and the number of valid dewarped surfaces in the buffer (`num_filled_surfaces`)

2.8.2 Features

Table 16 summarizes the features of the plugin.

Table 16. Features of the Gst-nvdewarper plugin

Feature	Description	Release
Configure number of dewarped surfaces	Supports a maximum of four dewarper surfaces.	DS 3.0

Feature	Description	Release
Configure per-surface projection type	Currently supports `FishPushBroom` and `FishVertRadCyd` projections.	DS 3.0
Configure per-surface index	Surface index to be set in case of multiple surfaces having same projection type.	DS 3.0
Configure per-surface width and height		DS 3.0
Configure per-surface dewarping parameters	Per-surface configurable yaw, roll, pitch, top angle, bottom angle, and focal length dewarping parameters.	DS 3.0
Configurable dewarper output resolution	Creates a batch of up to four surfaces of a specified output resolution; internally scales all dewarper surfaces to output resolution.	DS 3.0
Configurable NVDS CUDA memory type	—	DS 3.0
Multi-GPU support	—	DS 3.0
Aisle view CSV calibration file support	If set, properties in the `[surface<n>]` group are ignored.	DS 3.0
Spot view CSV calibration file support	If set, properties in the `[surface<n>]` group are ignored.	DS 3.0
Configure source id	Sets the source ID information in the `NvDewarperSurfaceMeta`.	DS 4.0
Configurable number of output buffers	Number of allocated output dewarper buffers. Each buffer contains four dewarped output surfaces.	DS 4.0

2.8.3 Configuration File Parameters

The configuration file specifies per-surface configuration parameters in `[surface<n>]` groups, where <n> is an integer from 0 to 3, representing dewarped surfaces 0 to 3.

Table 17. Gst-nvdewarper plugin, configuration file, [surface<n>] parameters

Property	Meaning	Type and Range	Example *Notes*
output-width	Scale dewarped surfaces to specified output width	Integer, >0	output-width=960
output-height	Scale dewarped surfaces to specified output height	Integer, >0	output-height=752
dewarp-dump-frames	Number of dewarped frames to dump.	Integer, >0	dewarp-dump-frames=10
projection-type	Selects projection type. Supported types are: 1: PushBroom 2: VertRadCyl	Integer, 1 or 2	projection-type=1

Property	Meaning	Type and Range	Example *Notes*
surface-index	An index that distinguishes surfaces of the same projection type.	Integer, ≥0	surface-index=0
width	Dewarped surface width.	Integer, >0	width=3886
height	Dewarped surface height.	Integer, >0	height=666
top-angle	Top field of view angle, in degrees.	Float, −180.0 to 180.0	top-angle=0
bottom-angle	Bottom field of view angle, in degrees.	Float, −180.0 to 180.0	bottom-angle=0
pitch	Viewing parameter pitch in degrees.	Float, 0.0 to 360.0	pitch=90
yaw	Viewing parameter yaw in degrees.	Float, 0.0 to 360.0	yaw=0
roll	Viewing parameter roll in degrees.	Float, 0.0 to 360.0	roll=0
focal-length	Focal length of camera lens, in pixels per radian.	Float, >0.0	focal-length=437
aisle-calibration-file	Pathname of the configuration file for aisle view. Set for the 360-D application only. If set, properties in the `[surface<n>]` group are ignored. The configuration file is a CSV file with columns like `sensorId` and `cameraId`, and dewarping parameters like `top-angle`, `bottom-angle`, `yaw`, `roll`, `pitch`, `focal-length`, `width`, and `height`.	String	aisle-calibration-file= csv_files/nvaisle_2M.csv
spot-calibration-file	Pathname of the configuration file for spot view. Set for the 360-D application only. If set, properties in the `[surface<n>]` group are ignored. The configuration file is a CSV file with columns like `sensorId` and `cameraId`, and dewarping parameters like `top-angle`, `bottom-angle`, `yaw`, `roll`, `pitch`, `focal-length`, `width`, and `height`.	String	spot-calibration-file= csv_files/nvspot_2M.csv *For an example of a spot view configuration file, see the file in the example above.*

This plugin can be tested with the one of the following pipelines.

- For dGPU:

```
gst-launch-1.0 filesrc location=streams/sample_cam6.mp4 ! qtdemux !
h264parse ! nvv4l2decoder ! nvvideoconvert ! nvdewarper config-
file=config_dewarper.txt source-id=6 nvbuf-memory-type=3 ! m.sink_0
nvstreammux name=m width=960 height=752 batch-size=4 num-surfaces-
per-frame=4 ! nvmultistreamtiler ! nveglglessink
```

- For Jetson:

```
gst-launch-1.0 filesrc location= streams/sample_cam6.mp4 ! qtdemux !
h264parse ! nvv4l2decoder  ! nvvideoconvert  ! nvdewarper config-
file=config_dewarper.txt source-id=6  ! m.sink_0 nvstreammux name=m
width=960 height=752 batch-size=4 num-surfaces-per-frame=4 !
nvmultistreamtiler ! nvegltransform ! nveglglessink
```

The `Gst-nvdewarper` plugin always outputs a GStreamer buffer which contains the maximum number of dewarped surfaces (currently four surfaces are supported). These dewarped surfaces are scaled to the output resolution (`output-width` × `output-height`) set in the configuration file located at `configs/deepstream-app/config_dewarper.txt`.

Also, the batch size to be set on `Gst-nvstreammux` must be a multiple of the maximum number of dewarped surfaces (currently four).

2.8.4 Gst Properties

Table 18 describes the `Gst-nvdewarper` plugin's Gst properties.

Table 18. Gst-nvdewarper plugin, Gst properties

Property	Meaning	Type and Range	Example and Notes
config-file	Absolute pathname of configuration file for the Gst-nvdewarper element	String	config-file= configs/deepstream-app/config_dewarper.txt
gpu-id	Device ID of the GPU to be used (dGPU only)	Integer, 0 to 4,294,967,295	gpu-id=0
source-id	Source ID, e.g. camera ID	Integer, 0 to 4,294,967,295	source-id=6
num-output-buffers	Number of output buffers to be allocated	Integer, 0 to 4,294,967,295	num-output-buffers=4

Property	Meaning	Type and Range	Example and Notes
nvbuf-memory-type	Type of memory to be allocated. **For dGPU**: 0 (`nvbuf-mem-default`): Default memory, `cuda-device` 1 (`nvbuf-mem-cuda-pinned`): Pinned/Host CUDA memory 2 (`nvbuf-mem-cuda-device`) Device CUDA memory 3 (`nvbuf-mem-cuda-unified`): Unified CUDA memory **For Jetson**: 0 (`nvbuf-mem-default`): Default memory, surface array 4 (`nvbuf-mem-surface-array`): Surface array memory	Integer, 0 to 4	nvbuf-memory-type=3

2.9 GST-NVOF

NVIDIA GPUs, starting with the dGPU Turing generation and Jetson Xavier generation, contain a hardware accelerator for computing optical flow. Optical flow vectors are useful in various use cases such as object detection and tracking, video frame rate up-conversion, depth estimation, stitching, and so on.

The gst-nvof plugin collects a pair of NV12 images and passes it to the low-level optical flow library. The low-level library returns a map of flow vectors between the two frames as its output.

The map of flow vectors is encapsulated in the `NvDsOpticalFlowMeta` structure and is added as a user meta with `meta_type` set to `NVDS_OPTICAL_FLOW_META`. The user meta is added to the `frame_user_meta_list` member of `NvDsFrameMeta`.

For guidance on how to access user metadata, see User/Custom Metadata Addition inside NvDsBatchMeta and Tensor Metadata.

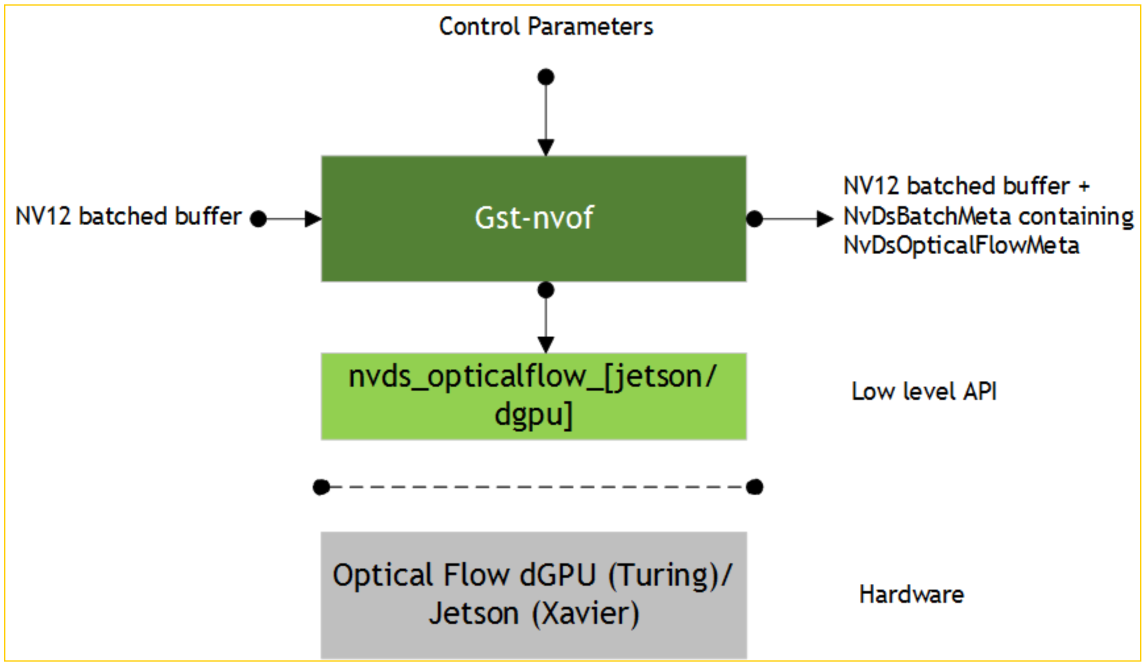
Control Parameters
NV12 batched buffer
Gst-nvof
NV12 batched buffer + NvDsBatchMeta containing NvDsOpticalFlowMeta
nvds_opticalflow_[jetson/dgpu]
Low level API
Optical Flow dGPU (Turing)/ Jetson (Xavier)
Hardware

Figure 9. The Gst-nvof plugin

2.9.1 Inputs and Outputs

- Inputs
  - GStreamer buffer containing NV12 frame(s)
- Control parameters
  - `gpu-id`: selects the GPU ID (valid only for dGPU platforms)
  - `dump-of-meta`: enables dumping of optical flow map vector into a `.bin` file
  - `preset-level`: sets the preset level
  - `pool-size`: sets the pool size
  - `grid-size`: sets the grid size
- Outputs
  - GStreamer buffer containing NV12 frame(s)
  - `NvDsOpticalFlowMeta` metadata

2.9.2 Features

Table 19 summarizes the features of the plugin.

Table 19. Features of the Gst-nvof plugin

Feature	Description	Release
Configure GPU selection	Sets the gpu ID to be used for optical flow operation (valid only for dGPU platforms)	DS 4.0
Configure dumping of optical flow metadata	Enables dumping of optical flow output (motion vector data)	DS 4.0
Configure preset level	Sets the desired preset level	DS 4.0
Configure grid size	Sets the flow vector block size	DS 4.0

2.9.3 Gst Properties

Table 20 describes the Gst properties of the `Gst-nvof` plugin.

Table 20. Gst-nvof plugin, Gst properties

Property	Meaning	Type and Range	Example *Notes*
gpu-id	Device ID of the GPU to be used for decoding (dGPU only).	Integer, 0 to 4,294,967,295	gpu-id=0
dump-of-meta	Dumps optical flow output into a `.bin` file.	Boolean	dump-of-meta=1
preset-level	Selects a preset level, default preset level is 0 i.e. NV_OF_PERF_LEVEL_FAST Possible values are: 0 (`NV_OF_PERF_LEVEL_FAST`): high performance, low quality. 1 (`NV_OF_PERF_LEVEL_MEDIUM`): intermediate performance and quality. 2 (`NV_OF_PERF_LEVEL_SLOW`): low performance, best quality (valid only for dGPU platforms).	Enum, 0 to 2	preset-level=0
grid-size	Selects the grid size. The hardware generates flow vectors blockwise, one vector for each block of 4×4 pixels. Currently only the 4x4 grid size is supported.	Enum, 0	grid-size=0
pool-size	Sets the number of internal motion vector output buffers to be allocated.	Integer, 1 to 4,294,967,295	pool-size=7

2.10 GST-NVOFVISUAL

The `Gst-nvofvisual` plugin is useful for visualizing motion vector data. The visualization method is similar to the OpenCV reference source code in:

```
https://github.com/opencv/opencv/blob/master/samples/gpu/optical_flow.c
pp
```

The plugin solves the optical flow problem by computing the magnitude and direction of optical flow from a two-channel array of flow vectors. It then visualizes the angle (direction) of flow by hue and the distance (magnitude) of flow by value of Hue Saturation Value (**HSV**) color representation. The strength of HSV is always set to a maximum of 255 for optimal visibility.

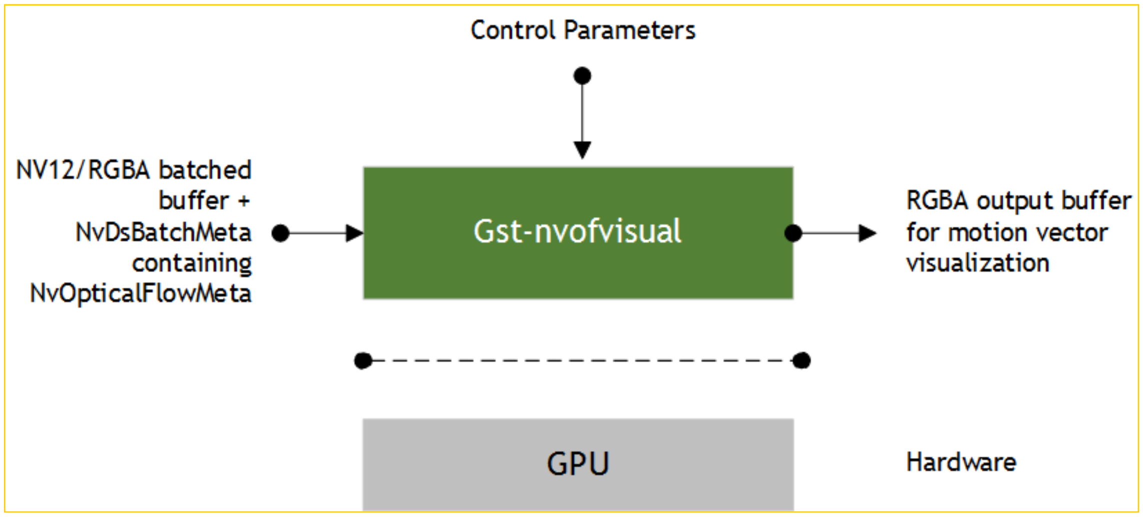
Control Parameters
NV12/RGBA batched buffer + NvDsBatchMeta containing NvOpticalFlowMeta
Gst-nvofvisual
RGBA output buffer for motion vector visualization
GPU
Hardware

Figure 10. The Gst-nvofvisual plugin

2.10.1 Inputs and Outputs

- Inputs
  - GStreamer buffer containing NV12/RGBA frame(s)
  - `NvDsOpticalFlowMeta` containing the motion vector (MV) data generated by the gst-nvof plugin
- Control parameters
  - `gpu-id,` selects the GPU ID
- Output
  - GStreamer buffer containing RGBA frame(s)

- RGBA buffer generated by transforming MV data into color-coded RGBA image reference

2.10.2 Features

Table 21 summarizes the features of the plugin.

Table 21. Features of the Gst-nvofvisual plugin

Feature	Description	Release
Configure GPU selection	Sets the GPU ID to be used for optical flow visualization operations (valid only for dGPU platforms)	DS 4.0

2.10.3 Gst Properties

Table 22 describes the Gst properties of the `Gst-nvofvisual` plugin.

Table 22. Gst-nvofvisual plugin, Gst Properties

Property	Meaning	Type and Range	Example *Notes*
gpu-id	Device ID of the GPU to be used **(dGPU only)**	Integer, 0 to 4,294,967,295	gpu-id=0

2.11 GST-NVSEGVISUAL

The `Gst-nvsegvisual` plugin visualizes segmentation results. Segmentation is based on image recognition, except that the classifications occur at the pixel level as opposed to the image level as with image recognition. The segmentation output size is generally same as the input size.

For more information, see the segmentation training reference at:

```
https://github.com/qubvel/segmentation_models
```

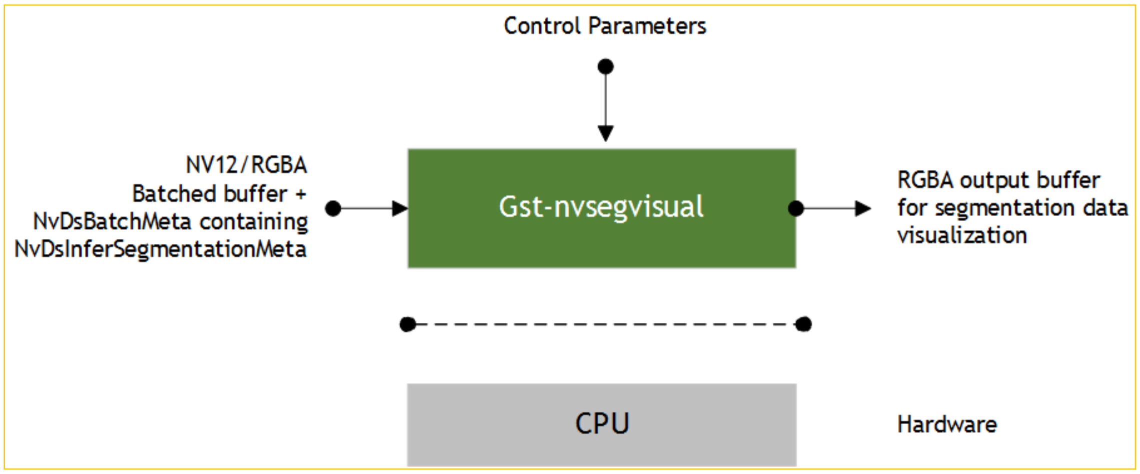
Control Parameters
NV12/RGBA Batched buffer + NvDsBatchMeta containing NvDsInferSegmentationMeta
Gst-nvsegvisual
RGBA output buffer for segmentation data visualization
CPU
Hardware

Figure 11. The Gst-nvsegvisual plugin

2.11.1 Inputs and Outputs

- Inputs
  - GStreamer buffer containing NV12/RGBA frame(s)
  - `NvDsInferSegmentationMeta` containing class numbers, pixel class map, width, height, etc. generated by gst-nvinfer.
  - `gpu-id`: selects the GPU ID
  - width, set according the segmentation output size
  - height, set according the segmentation output size
- Output

  This plugin allocates different colors for different classes. For example, the industrial model's output has only one representing defective areas. Thus defective areas and background have different colors. The semantic model outputs four classes with four different colors: car, pedestrian, bicycle, and background.

  This plugin shows only the segmentation output. It does not overlay output on the original NV12 frame.

Table 23 summarizes the features of the plugin.

Table 23. Features of the Gst-nvsegvisual plugin

Feature	Description	Release
Configure GPU selection	Sets the GPU ID to be used for segmentation visualization operations (valid only for dGPU platforms)	DS 4.0
Configure width	Sets width according to the segmentation output size	DS 4.0

Feature	Description	Release
Configure height	Sets height according to the segmentation output size	DS 4.0

2.11.2 Gst Properties

Table 24 describes the Gst properties of the `Gst-nvsegvisual` plugin.

Table 24. Gst-nvsegvisual plugin, Gst Properties

Property	Meaning	Type and Range	Example and Notes
gpu-id	Device ID of the GPU to be used for decoding	Integer, 0 to 4,294,967,295	gpu-id=0
width	Segmentation output width	Integer, 0 to 4,294,967,295	width=512
height	Segmentation output height	Integer, 0 to 4,294,967,295	height=512

2.12 GST-NVVIDEO4LINUX2

DeepStream extends the open source V4L2 codec plugins (here called Gst-v4l2) to support hardware-accelerated codecs.

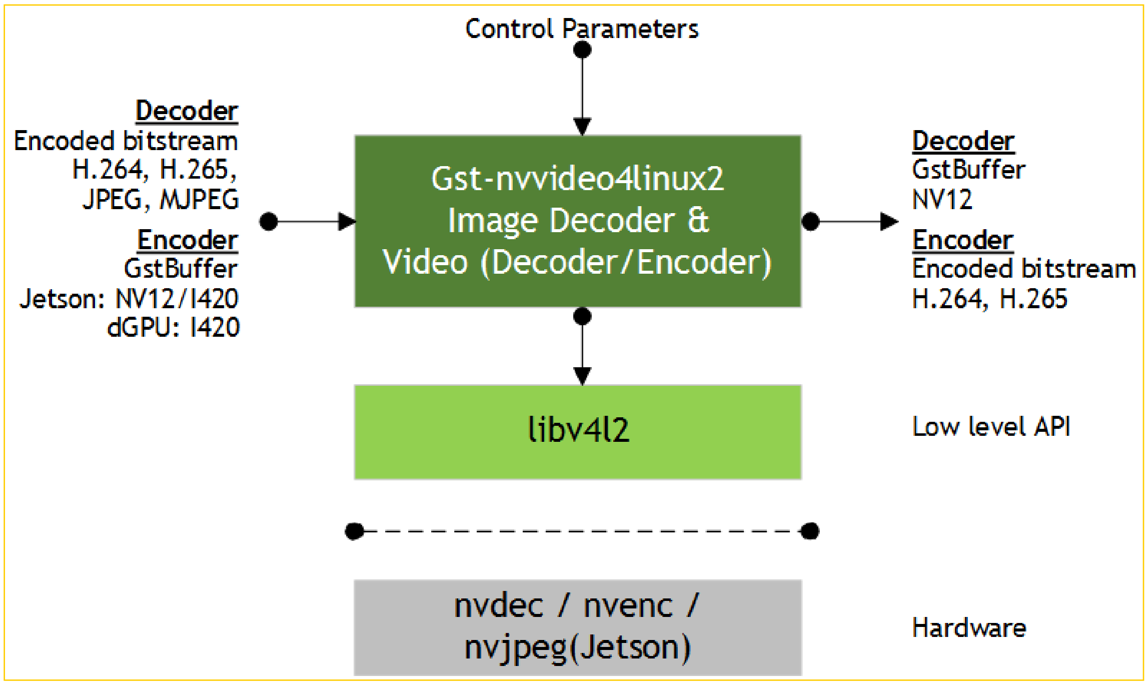

Figure 12. The Gst-nvvideo4linux2 decoder plugin

2.12.1 Decoder

The OSS Gst-nvvideo4linux2 plugin leverages the hardware decoding engines on Jetson and DGPU platforms by interfacing with `libv4l2` plugins on those platforms. It supports H.264, H.265, JPEG and MJPEG formats.

The plugin accepts an encoded bitstream & NVDEC h/w engine to decoder the bitstream. The decoded output is in NV12 format.

Note: **When you use the v4l2 decoder use for decoding JPEG images, you must use the open source `jpegparse` plugin before the decoder to parse encoded JPEG images.**

2.12.1.1 Inputs and Outputs

- Inputs
  - An encoded bitstream. Supported formats are H.264, H.265, JPEG and MJPEG
- Control Parameters
  - `gpu-id`
  - `num-extra-surfaces`
  - `skip-frames`
  - `cudadec-memtype`

- `drop-frame-interval`

- Output
  - Gst Buffer with decoded output in NV12 format

2.12.1.2 Features

Feature	Description	Release
Supports H.264 decode	—	DS 4.0
Supports H.265 decode	—	DS 4.0
Supports JPEG/MJPEG decode	_	DS 4.0
User-configurable CUDA memory type (Pinned/Device/Unified) for output buffers	—	DS 4.0

2.12.1.3 Configuration Parameters

Property	Meaning	Type and Range	Example *Notes*	Platforms
gpu-id	Device ID of GPU to use for decoding.	Integer, 0 to 4,294,967,295	gpu-id=0	dGPU
num-extra-surfaces	Number of surfaces in addition to min decode surfaces given by the V4L2 driver.	Integer, 1 to 24	num-decode-surfaces=24 *Default: 0*	dGPU Jetson
skip-frames	Type of frames to skip during decoding. Represented internally by enum *SkipFrame*. 0 (`decode_all`): decode all frames 1 (`decode_non_ref`): decode non-ref frames 2 (`decode_key`): decode key frames	Integer, 0, 1, or 2	skip-frames=0 *Default: 0*	dGPU Jetson
drop-frame-interval	Interval to drop the frames, e.g. a value of 5 means the decoder receives every fifth frame, and others are dropped.	Integer, 1 to 30	*Default: 0*	dGPU Jetson
cudadec-memtype	Memory type for CUDA decoder buffers. Represented internally by enum `CudaDecMemType`. 0 (`memtype_device`): Device 1 (`memtype_pinned`): Host Pinned 2 (`memtype_unified`): Unified	Integer, 0, 1, or 2	cuda-memory-type=1 *Default: 2*	dGPU

2.12.2 Encoder

The OSS Gst-nvvideo4linux2 plugin leverages the hardware accelerated encoding engine available on Jetson and dGPU platforms by interfacing with `libv4l2` plugins on those platforms. The plugin accepts RAW data in I420 format. It uses the NVENC hardware engine to encode RAW input. Encoded output is elementary bitstream supported formats.

2.12.2.1 Inputs and Outputs

- Inputs
  - RAW input in I420 format
- Control parameters
  - gpu-id (dGPU only)
  - `profile`
  - `bitrate`
  - `control-rate`
  - `iframeinterval`
- Output
  - Gst Buffer with encoded output in H264, H265, VP8 or VP9 format.

2.12.2.2 Features

Feature	Description	Release
Supports H.264 encode	—	DS 4.0
Supports H.265 encode	—	DS 4.0

2.12.2.3 Configuration Parameters

Property	Meaning	Type and Range	Example *Notes*	Platforms
gpu-id	Device ID of GPU to used.	Integer, 0 to 4,294,967,295	gpu-id=0	dGPU
bitrate	Sets bitrate for encoding, in bits/seconds.	Integer, 0 to 4,294,967,295	Default:4000000	dGPU Jetson
iframeinterval	Encoding intra-frame occurrence frequency.	Unsigned integer, 0 to 4,294,967,295	*Default: 30*	dGPU Jetson

Profile	H.264/H.265 encoder profile; represented internally by enum `GstV4l2VideoEncProfileType`. **For H.264:** 0 (`GST_V4L2_H264_VIDENC_BASELINE_PROFILE`): Baseline 2 (`GST_V4L2_H264_VIDENC_MAIN_PROFILE`): Main 4 (`GST_V4L2_H264_VIDENC_HIGH_PROFILE`): High **For H.265:** 0 (`GST_V4L2_H265_VIDENC_MAIN_PROFILE`): Main 1 (`GST_V4L2_H265_VIDENC_MAIN10_PROFILE`): Main10	Values of enum `GstV4l2VideoEncProfileType`	Default Baseline *Default: 0*	dGPU Jetson

2.13 GST-NVJPEGDEC

The Gst-nvjpegdec plugin decodes images on both dGPU and Jetson platforms. It is the preferred method for decoding JPEG images.

On the dGPU platform this plugin is based on the `libnvjpeg` library, part of the CUDA toolkit. On Jetson it uses a platform-specific hardware accelerator.

The plugin uses an internal software parser to parse JPEG streams. Thus there is no need to use a `jpegparse` open source plugin separately to parse the encoded frame.

The plugin accepts a JPEG encoded bitstream and produces RGBA output on the dGPU platform, and produces I420 output on the Jetson platform.

2.13.1 Inputs and Outputs

- Inputs
  - Elementary JPEG
- Control parameters
  - `gpu-id` (dGPU only)
  - DeepStream (Jetson only)
- Output
  - Gst Buffer with decoded output in RGBA format