Class FormatConverterOp

Table of contents

- Defined in File format_converter.hpp

Inheritance Relationships

Base Type

- `public holoscan::Operator` (Class Operator)

Class Documentation

class FormatConverterOp : public holoscan::Operator

Operator class to convert the data format of the input data.

==Named Inputs==

- **source_video** : `nvidia::gxf::Tensor` or `nvidia::gxf::VideoBuffer`
  - The input video frame to process. If the input is a VideoBuffer it must be in format GXF_VIDEO_FORMAT_RGBA, GXF_VIDEO_FORMAT_RGB or GXF_VIDEO_FORMAT_NV12. If a video buffer is not found, the input port message is searched for a tensor with the name specified by `in_tensor_name`. This must be a tensor in one of several supported formats (unsigned 8-bit int or float32 graycale, unsigned 8-bit int RGB or RGBA YUV420 or NV12). The tensor or video buffer may be in either host or device memory (a host->device copy is performed if needed).

==Named Outputs==

- **tensor** : `nvidia::gxf::Tensor`
  - The output video frame after processing. The shape, data type and number of channels of this output tensor will depend on the specific parameters that were set for this operator. The name of the Tensor transmitted on this port is determined by `out_tensor_name`.

==Parameters==

- **pool**: Memory pool allocator (holoscan::Allocator) used by the operator.

- **out_dtype**: Destination data type. The available options are:
  - `"rgb888"`
  - `"uint8"`
  - `"float32"`
  - `"rgba8888"`
  - `"yuv420"`
  - `"nv12"`
- **in_dtype**: Source data type. The available options are:
  - `"rgb888"`
  - `"uint8"`
  - `"float32"`
  - `"rgba8888"`
  - `"yuv420"`
  - `"nv12"` Optional (default: `"rgb888"`).
- **in_tensor_name**: The name of the input tensor. Optional (default: `""`).
- **out_tensor_name**: The name of the output tensor. Optional (default: `""`).
- **scale_min**: Output will be clipped to this minimum value. Optional (default: `0.0`).
- **scale_max**: Output will be clipped to this maximum value. Optional (default: `1.0`).

- **alpha_value**: Unsigned integer in range [0, 255], indicating the alpha channel value to use when converting from RGB to RGBA. Optional (default: `255`).
- **resize_height**: Desired height for the (resized) output. Height will be unchanged if `resize_height` is `0`. Optional (default: `0`).
- **resize_width**: Desired width for the (resized) output. Width will be unchanged if `resize_width` is `0`. Optional (default: `0`).
- **resize_mode**: Resize mode enum value corresponding to NPP's NppiInterpolationMode. Values available at: https://docs.nvidia.com/cuda/npp/nppdefs.html?highlight=Two%20parameter%20cubic%20filter#c.NppiInterpolationMode
    - NPPI_INTER_UNDEFINED (`0`): Undefined filtering interpolation mode.
    - NPPI_INTER_NN (`1`): Nearest neighbor filtering.
    - NPPI_INTER_LINEAR (`2`): Linear interpolation.
    - NPPI_INTER_CUBIC (`4`): Cubic interpolation.
    - NPPI_INTER_CUBIC2P_BSPLINE (`5`): Two-parameter cubic filter (B=1, C=0)
    - NPPI_INTER_CUBIC2P_CATMULLROM (`6`): Two-parameter cubic filter (B=0, C=1/2)
    - NPPI_INTER_CUBIC2P_B05C03 (`7`): Two-parameter cubic filter (B=1/2, C=3/10)
    - NPPI_INTER_SUPER (`8`): Super sampling.
    - NPPI_INTER_LANCZOS (`16`): Lanczos filtering.
    - NPPI_INTER_LANCZOS3_ADVANCED (`17`): Generic Lanczos filtering with order 3.
    - NPPI_SMOOTH_EDGE (`0x8000000`): Smooth edge filtering.

Optional (default: `0`). The default value `0` (NPPI_INTER_UNDEFINED) which would be equivalent to `4` (NPPI_INTER_CUBIC).

- **channel_order**: Sequence of integers describing how channel values are permuted. Optional (default: `[0, 1, 2]` for 3-channel images and `[0, 1, 2, 3]` for 4-channel images).

- **cuda_stream_pool**: `holoscan::CudaStreamPool` instance to allocate CUDA streams. Optional (default: `nullptr`).

==Device Memory Requirements==

When using this operator with a `BlockMemoryPool`, between 1 and 3 device memory blocks (`storage_type` = 1) will be required based on the input tensors and parameters:

- 1.) In all cases there is a memory block needed for the output tensor. The size of this block will be `out_height * out_width * out_channels * out_element_size_bytes` where (out_height, out_width) will either be (in_height, in_width) (or (resize_height, resize_width) a resize was specified). `out_element_size` is the element size in bytes (e.g. 1 for RGB888 or 4 for Float32).

- 2.) If a resize is being done, another memory block is required for this. This block will have size `resize_height * resize_width * in_channels * in_element_size_bytes`.

- 3.) If the input tensor will be in host memory, a memory block is needed to copy the input to the device. This block will have size `in_height * in_width * in_channels * in_element_size_bytes`.

Thus when declaring the memory pool, `num_blocks` should be between 1-3 and `block_size` should be set to the maximum of the individual blocks sizes described above.

Public Functions

HOLOSCAN_OPERATOR_FORWARD_ARGS (FormatConverterOp)
FormatConverterOp()=default

virtual void setup(OperatorSpec &spec) override

Define the operator specification.

Parameters

spec – The reference to the operator specification.

virtual void initialize() override

Initialize the operator.

This function is called when the fragment is initialized by Executor::initialize_fragment().

virtual void start() override

Implement the startup logic of the operator.

This method is called multiple times over the lifecycle of the operator according to the order defined in the lifecycle, and used for heavy initialization tasks such as allocating memory resources.

virtual void compute(InputContext &op_input, OutputContext &op_output, ExecutionContext &context) override

Implement the compute method.

This method is called by the runtime multiple times. The runtime calls this method until the operator is stopped.

Parameters

- **op_input** – The input context of the operator.
- **op_output** – The output context of the operator.
- **context** – The execution context of the operator.

virtual void stop() override

Implement the shutdown logic of the operator.

This method is called multiple times over the lifecycle of the operator according to the order defined in the lifecycle, and used for heavy deinitialization tasks such as deallocation of all resources previously assigned in start.

nvidia::gxf::Expected<void*> resizeImage(const void *in_tensor_data, const std::vector<nvidia::gxf::ColorPlane> &in_color_planes, const int32_t rows, const int32_t columns, const int16_t channels, const nvidia::gxf::PrimitiveType primitive_type, const int32_t resize_width, const int32_t resize_height)

void convertTensorFormat(const void *in_tensor_data, const std::vector<nvidia::gxf::ColorPlane> &in_color_planes, void *out_tensor_data, const int32_t rows, const int32_t columns, const int16_t out_channels)