NVIDIA CUDA Modules

Getting Started

Instal CUDAToolKit

- 10.1.x is required. Don't install higher/latest versions
- Download from this link
    - If the above doesn't work, search for updated link from the archives
        - Search for 10.1.x
        - Download for windows 64 bit
- Install the CudaTookKit from the executable.
- C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v10.1 folder should have been created

Build Solution

- double click build_cuda.bat
- it takes about an hour for the build to complete first time
- _cudabuild folder should be created
- _cudabuild/ECSIPFW_CUDA.sln visual studio solution is ready to use

Run Tests

- _build/Release/ecsiput.exe –run_test=decode_resize_encode_mono_1920x960
- check the output jpeg image - data/testOutput/nvjpeg_combo_tests_decode_resize_encode_mono_1920x960_to_960x480.jpg
- if the above image is valid, then the setup is complete

Nvidia CUDA Modules

cudaStream_t

Checkout CUDA DOCUMENTATION and CUDA WEBINAR

- A sequence of operations that execute in issue-order on the GPU
- Programming model used to effect concurrency
- CUDA operations in different streams may run concurrently
- CUDA operations from different streams may be interleaved

cudaStream_t has to be passed as props to all the CUDA modules. cudaStreamCreate and cudaStreamDestroy is mandatory.

CudaMemCopyProps

- cudaMemcpyKind memcpyKind - cudaMemcpyDeviceToHost
    - Input is CUDA_DEVICE memory
    - Output is HOST memory
    - sync is set to true internally
- cudaMemcpyKind memcpyKind - cudaMemcpyHostToDevice
    - Input is HOST memory
    - Output is CUDA_DEVICE memory
- sync - if sync is true, stream will be synchronized - don't set it to true until you know what you are doing- |
- cudaStream_t has to be passed

JPEGDecoderNVJPEG

- Input is HOST memory
- Output is CUDA_DEVICE memory
- Output ImageMetadata::ImageType - MONO, YUV420, YUV444 is currently supported
- cudaStream_t has to be passed

JPEGEncoderNVJPEG

- Input is CUDA_DEVICE memory
- Output is HOST memory
- quality - default is 90
- Input ImageMetadata::ImageType - MONO, YUV420, YUV444 is currently supported
- cudaStream_t has to be passed

ResizeNPPI

- Input is CUDA_DEVICE memory
- Output is CUDA_DEVICE memory
- output width and height has to be passed
- Input ImageMetadata::ImageType - MONO, YUV444, YUV420, BGR, BGRA, RGB, RGBA
- cudaStream_t has to be passed

CCNPPI

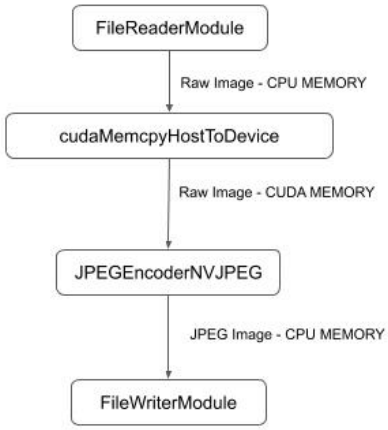
- Input is CUDA_DEVICE memory
- Output is CUDA_DEVICE memory
- Supported Color Conversions
    - MONO, YUV420 To BGRA
    - YUV420 To BGRA
    - YUV411_I To YUV444
- cudaStream_t has to be passed

OverlayNPPI

- Input is CUDA_DEVICE memory
- Output is CUDA_DEVICE memory
- Input ImageMetadata::ImageType - BGRA
- Accepts two images - Source and Overlay
- OVERLAY_HINT has to be set to tell the module which of the two input images is the overlay image
- Overlay image width and height is expected to be less than or equal to the Source image
- offsetX and offsetY can be given
- cudaStream_t has to be passed

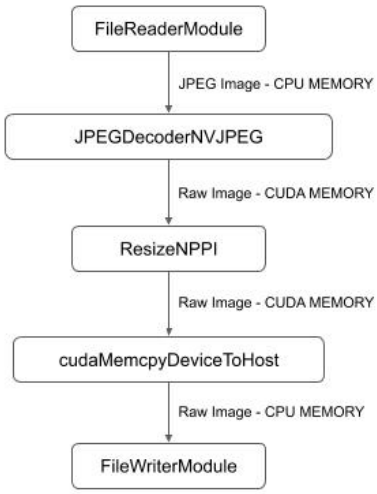
Building Pipelines

Example 1



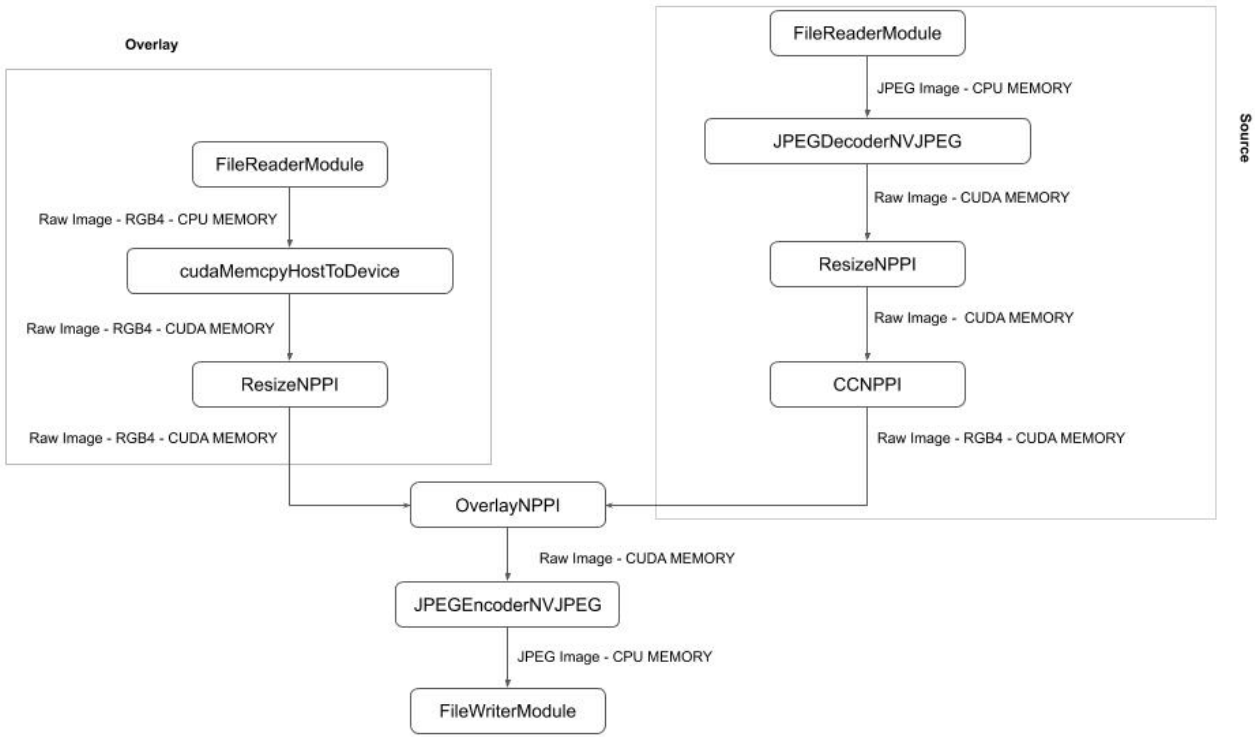

```
auto width = 1920;
auto height = 1080;

auto fileReader = boost::shared_ptr<FileReaderModule>(new
FileReaderModule(FileReaderModuleProps("./data/mono_1920x1080.raw")));
auto metadata = framemetadata_sp(new RawImageMetadata(width, height,
ImageMetadata::ImageType::MONO, CV_8UC1, 0, CV_8U, FrameMetadata::HOST, true));
fileReader->addOutputPin(metadata);

cudaStream_t stream;
cudaStreamCreate(&stream);
auto copy = boost::shared_ptr<Module>(new CudaMemCopy(CudaMemCopyProps(cudaMemcpyHostToDevice,
stream)));
fileReader->setNext(copy);

auto encoder = boost::shared_ptr<JPEGEncoderNVJPEG>(new
JPEGEncoderNVJPEG(JPEGEncoderNVJPEGProps(stream)));
copy->setNext(encoder);

auto fileWriter = boost::shared_ptr<Module>(new
FileWriterModule("./data/testOutput/mono_1920x1080.jpg"));
encoder->setNext(fileWriter);

...

cudaStreamDestroy(stream);
```

Example 2




```
auto fileReader = boost::shared_ptr<Module>(new
FileReaderModule(FileReaderModuleProps("./data/mono_1920x960.jpg")));
auto metadata = framemetadata_sp(new FrameMetadata(FrameMetadata::ENCODED_IMAGE));
fileReader->addOutputPin(metadata);

cudaStream_t stream;
cudaStreamCreate(&stream);

auto decoder = boost::shared_ptr<JPEGDecoderNVJPEG>(new
JPEGDecoderNVJPEG(JPEGDecoderNVJPEGProps(stream)));
fileReader->setNext(decoder);

auto resize = boost::shared_ptr<Module>(new ResizeNPPI(ResizeNPPIProps(960, 480, stream)));
decoder->setNext(resize);

auto copy = boost::shared_ptr<Module>(new CudaMemCopy(CudaMemCopyProps(cudaMemcpyDeviceToHost,
stream)));
resize->setNext(copy);

auto fileWriter = boost::shared_ptr<Module>(new
FileWriterModule("./data/testOutput/mono_1920x960_to_mono_960x480.raw"));
encoder->setNext(fileWriter);

...

cudaStreamDestroy(stream);
```

Example 3


Overlay
FileReaderModule
Raw Image - RGB4 - CPU MEMORY
cudaMemcpyHostToDevice
Raw Image - RGB4 - CUDA MEMORY
ResizeNPPI
Raw Image - RGB4 - CUDA MEMORY
Source
FileReaderModule
JPEG Image - CPU MEMORY
JPEGDecoderNVJPEG
Raw Image - CUDA MEMORY
ResizeNPPI
Raw Image - CUDA MEMORY
CCNPPI
Raw Image - RGB4 - CUDA MEMORY
OverlayNPPI
Raw Image - CUDA MEMORY
JPEGEncoderNVJPEG
JPEG Image - CPU MEMORY
FileWriterModule

```
cudaStream_t stream;
cudaStreamCreate(&stream);

// making source pipeline - JPEGDecoderNVJPEG -> ResizeNPPI -> CCNPPI
auto fileReader = boost::shared_ptr<Module>(new
FileReaderModule(FileReaderModuleProps("./data/mono_1920x960.jpg")));
auto metadata = framemetadata_sp(new FrameMetadata(FrameMetadata::ENCODED_IMAGE));
fileReader->addOutputPin(metadata);

auto decoder = boost::shared_ptr<JPEGDecoderNVJPEG>(new
JPEGDecoderNVJPEG(JPEGDecoderNVJPEGProps(stream)));
fileReader->setNext(decoder);

auto resize = boost::shared_ptr<Module>(new ResizeNPPI(ResizeNPPIProps(960, 480, stream)));
decoder->setNext(resize);

auto source_cc = boost::shared_ptr<Module>(new CCNPPI(CCNPPIProps(ImageMetadata::BGRA,
stream)));
resize->setNext(source_cc);

// making overlay pipeline - CudaMemCopy -> ResizeNPPI
auto overlay_host = boost::shared_ptr<FileReaderModule>(new
FileReaderModule(FileReaderModuleProps("./data/overlay_1920x960_BGRA.raw")));
auto overlay_metadata = framemetadata_sp(new RawImageMetadata(1920, 960,
ImageMetadata::ImageType::BGRA, CV_8UC4, 0, CV_8U, FrameMetadata::HOST, true));
overlay_metadata->setHint(OVERLAY_HINT);
overlay_host->addOutputPin(overlay_metadata);

auto overlay_copy = boost::shared_ptr<Module>(new
CudaMemCopy(CudaMemCopyProps(cudaMemcpyHostToDevice, stream)));
overlay_host->setNext(overlay_copy);

auto overlay_resize = boost::shared_ptr<Module>(new ResizeNPPI(ResizeNPPIProps(960, 480,
stream)));
overlay_copy->setNext(overlay_resize);

// Connecting source and overlay pipelines
auto overlay = boost::shared_ptr<Module>(new OverlayNPPI(OverlayNPPIProps(stream)));
overlay_resize->setNext(overlay);
source_cc->setNext(overlay);

auto encoder = boost::shared_ptr<Module>(new
JPEGEncoderNVJPEG(JPEGEncoderNVJPEGProps(stream)));
overlay->setNext(encoder);

auto fileWriter = boost::shared_ptr<Module>(new FileWriterModule("./data/testOutput/out.jpg"));
encoder->setNext(fileWriter);

...

cudaStreamDestroy(stream);
```