Program Listing for File format_converter.hpp

Return to documentation for file (include/holoscan/operators/format_converter/format_converter.hpp)

/* * SPDX-FileCopyrightText: Copyright (c) 2022-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved. * SPDX-License-Identifier: Apache-2.0 * * Licensed under the Apache License, Version 2.0 (the "License"); * you may not use this file except in compliance with the License. * You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ #ifndef HOLOSCAN_OPERATORS_FORMAT_CONVERTER_FORMAT_CONVERTER_HPP #define HOLOSCAN_OPERATORS_FORMAT_CONVERTER_FORMAT_CONVERTER_HPP #include <npp.h> #include <memory> #include <string> #include <vector> #include "holoscan/core/io_context.hpp" #include "holoscan/core/io_spec.hpp" #include "holoscan/core/operator.hpp" #include "holoscan/core/operator_spec.hpp" #include "holoscan/utils/cuda_stream_handler.hpp" namespace holoscan::ops { enum class FormatDType { kUnknown, kRGB888, kRGBA8888, kUnsigned8, kFloat32, kYUV420, kNV12 }; enum class FormatConversionType { kUnknown, kNone, kUnsigned8ToFloat32, kFloat32ToUnsigned8, kRGB888ToRGBA8888, kRGBA8888ToRGB888, kRGBA8888ToFloat32, kRGB888ToYUV420, kYUV420ToRGBA8888, kYUV420ToRGB888, kNV12ToRGB888 }; class FormatConverterOp : public holoscan::Operator { public: HOLOSCAN_OPERATOR_FORWARD_ARGS(FormatConverterOp) FormatConverterOp() = default; *// TODO(gbae): use std::expected* void setup(OperatorSpec& spec) override; void initialize() override; void start() override; void compute(InputContext& op_input, OutputContext& op_output, ExecutionContext& context) override; void stop() override; 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); private:
Parameter<holoscan::IOSpec*> in_; Parameter<holoscan::IOSpec*> out_;
Parameter<std::string> in_tensor_name_; Parameter<std::string> out_tensor_name_;
Parameter<float> scale_min_; Parameter<float> scale_max_; Parameter<uint8_t>
alpha_value_; Parameter<int32_t> resize_width_; Parameter<int32_t> resize_height_;
Parameter<int32_t> resize_mode_; Parameter<std::vector<int>>
out_channel_order_; std::unique_ptr<nvidia::gxf::MemoryBuffer> resize_buffer_;
std::unique_ptr<nvidia::gxf::MemoryBuffer> channel_buffer_;
std::unique_ptr<nvidia::gxf::MemoryBuffer> device_scratch_buffer_;
Parameter<std::shared_ptr<Allocator>> pool_; Parameter<std::string> in_dtype_str_;
Parameter<std::string> out_dtype_str_; // internal state FormatDType in_dtype_ =
FormatDType::kUnknown; FormatDType out_dtype_ = FormatDType::kUnknown;
nvidia::gxf::PrimitiveType in_primitive_type_ = nvidia::gxf::PrimitiveType::kCustom;
nvidia::gxf::PrimitiveType out_primitive_type_ = nvidia::gxf::PrimitiveType::kCustom;
FormatConversionType format_conversion_type_ =
FormatConversionType::kUnknown; NppStreamContext npp_stream_ctx_{};
CudaStreamHandler cuda_stream_handler_; }; } // namespace holoscan::ops #endif/*
HOLOSCAN_OPERATORS_FORMAT_CONVERTER_FORMAT_CONVERTER_HPP */
```