Program Listing for File holoviz.hpp

Return to documentation for file (include/holoscan/operators/holoviz/holoviz.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 INCLUDE_HOLOSCAN_OPERATORS_HOLOVIZ_HOLOVIZ_HPP #define INCLUDE_HOLOSCAN_OPERATORS_HOLOVIZ_HOLOVIZ_HPP #include <array> #include <memory> #include <optional> #include <string> #include <vector> #include "holoscan/core/conditions/gxf/boolean.hpp" #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/core/resources/gxf/allocator.hpp" #include "holoscan/utils/cuda_stream_handler.hpp" namespace holoscan::viz { typedef void* InstanceHandle; } // namespace holoscan::viz namespace holoscan::ops { // forward declaration struct BufferInfo; class HolovizOp : public Operator { public: HOLOSCAN_OPERATOR_FORWARD_ARGS(HolovizOp) HolovizOp() = default; 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; enum class InputType { UNKNOWN, COLOR, COLOR_LUT, POINTS, LINES, LINE_STRIP, TRIANGLES, CROSSES, RECTANGLES, OVALS, TEXT, DEPTH_MAP, DEPTH_MAP_COLOR, POINTS_3D, LINES_3D, LINE_STRIP_3D, TRIANGLES_3D, }; enum class DepthMapRenderMode { POINTS, LINES, TRIANGLES }; struct InputSpec { InputSpec() = default; InputSpec(const std::string& tensor_name, InputType type) : tensor_name_(tensor_name), type_(type) {} InputSpec(const std::string& tensor_name, const std::string& type_str); explicit InputSpec(const std::string& yaml_description); explicit operator bool() const noexcept { return !tensor_name_.empty(); } std::string description() const; std::string tensor_name_; InputType type_ = InputType::UNKNOWN; float opacity_ = 1.f; int32_t priority_ = 0;
```

```
std::vector<float> color_{1.f, 1.f, 1.f, 1.f}; float line_width_ = 1.f; float point_size_ = 1.f;
std::vector<std::string> text_; DepthMapRenderMode depth_map_render_mode_ =
DepthMapRenderMode::POINTS; struct View { float offset_x_ = 0.f, offset_y_ = 0.f;
float width_ = 1.f, height_ = 1.f; std::optional<std::array<float, 16>> matrix_; };
std::vector<View> views_; }; private: bool enable_conditional_port(const std::string&
name, bool set_none_condition_on_disabled = false); void set_input_spec(const
InputSpec& input_spec); void set_input_spec_geometry(const InputSpec&
input_spec); void read_frame_buffer(InputContext& op_input, OutputContext&
op_output, ExecutionContext& context); void render_color_image(const InputSpec&
input_spec, BufferInfo& buffer_info); void render_geometry(const
ExecutionContext& context, const InputSpec& input_spec, BufferInfo& buffer_info);
void render_depth_map(InputSpec* const input_spec_depth_map, const
BufferInfo& buffer_info_depth_map, InputSpec* const input_spec_depth_map_color,
const BufferInfo& buffer_info_depth_map_color);
Parameter<std::vector<holoscan::IOSpec*>> receivers_;
Parameter<holoscan::IOSpec*> render_buffer_input_;
Parameter<holoscan::IOSpec*> render_buffer_output_;
Parameter<holoscan::IOSpec*> camera_pose_output_;
Parameter<std::vector<InputSpec>> tensors_;
Parameter<std::vector<std::vector<float>>> color_lut_; Parameter<std::string>
window_title_; Parameter<std::string> display_name_; Parameter<uint32_t> width_;
Parameter<uint32_t> height_; Parameter<float> framerate_; Parameter<bool>
use_exclusive_display_; Parameter<bool> fullscreen_; Parameter<bool> headless_;
Parameter<std::shared_ptr<BooleanCondition>> window_close_scheduling_term_;
Parameter<std::shared_ptr<Allocator>> allocator_; Parameter<std::string>
font_path_; Parameter<std::string> camera_pose_output_type_;
Parameter<std::array<float, 3>> camera_eye_; Parameter<std::array<float, 3>>
camera_look_at_; Parameter<std::array<float, 3>> camera_up_; // internal state
viz::InstanceHandle instance_ = nullptr; std::vector<float> lut_;
std::vector<InputSpec> initial_input_spec_; CudaStreamHandler
cuda_stream_handler_; bool render_buffer_input_enabled_; bool
render_buffer_output_enabled_; bool camera_pose_output_enabled_; bool
is_first_tick_ = true; std::array<float, 3> camera_eye_cur_; //< current camera eye
position std::array<float, 3> camera_look_at_cur_; //< current camera look at position
std::array<float, 3> camera_up_cur_; //< current camera up vector }; } // namespace
```

holoscan::ops #endif/* INCLUDE_HOLOSCAN_OPERATORS_HOLOVIZ_HOLOVIZ_HPP */