Analysis

Concept

Analysis Concept

The analysis module allows the user to conduct experiments and thereby discover performance bottlenecks.

These experiments can be performed on the host in a simulation or on the target.
During an experiment, the host and target have to be connected. Currently, Analyzer only supports a connection via TCP/IP and Serial Port.

Preparations for Analysis

Build Candera with Monitor enabled

Candera, as well as the application you are writing, have to be configured via CMake. Point CMake to the source directory of your application and indicate a directory for binary generation. Then fire up configuration. For further instructions see Establishing connection.

For configuration of CGI Monitor for a binary Candera distribution, see section 0.

Provide CPU and/or GPU load

In order to display a trend with key performance indicators (CPU load, GPU load, frame rate) on Linux, the proc file system has to be mounted.

This can be accomplished (if not already mounted) by adjusting / adding the corresponding entry under /etc/mtab or mounting the proc file system by hand with the following command:

```
mount /proc
```

CPU and GPU information have to be provided by drivers / kernel modules and are thus not available on every (Linux) platform.

Provide communication infrastructure

Candera is only pre-configured for communication with Analyzer, if you call the function Renderer::RenderAllCameras or Renderer2D::RenderAllCameras (or both).

Otherwise, you have to ensure, that

- Connection parameters are set correctly
- A connection is established
- A Candera application can be paused by Analyzer

Setting connection parameters

Communication parameters can be set, if necessary, using the following macro:

```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_PORT_CONFIGURE(address, port)
```

> This macro name has changed with v3.2. Previous version were named MONITOR_TCPIP_CONFIGURE.

The standard Player for use with the tunnel application under Windows presents a dialog for setting communication parameters after the asset library to load has been chosen. The tunnel sample application delivered with Candera now takes two additional parameters for TCP/IP address and port or for Serial Communication baudrate and com port. This macro has to be used before either MONITOR_TRYCONNECT() or MONITOR_HANDLE_PAUSED(), otherwise the standard configuration will still remain.

Establishing a connection

A connection can be established explicitly, if necessary, from application code, by using the following macro

```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_TRYCONNECT()
```

Enabling the pausing of a Candera application

For Analyzer to be able to pause your application and the key performance indicators to be updated, the following macro has to be called every time, the drawing of a new frame begins:

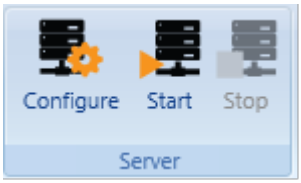
```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_HANDLE_PAUSED()
```

Connecting with a Candera application

The procedure for setting up a connection between host and target is as follows:

1. Start Analyzer
2. Configure the communication server
3. Start the server via the Server Ribbon Group on the Online Ribbon Tab
4. Start the Candera application, it will try to connect to the pre-configured port and IP-address

Configuration, starting, and stopping of the analysis server are triggered from the server ribbon group as depicted in the following figure.

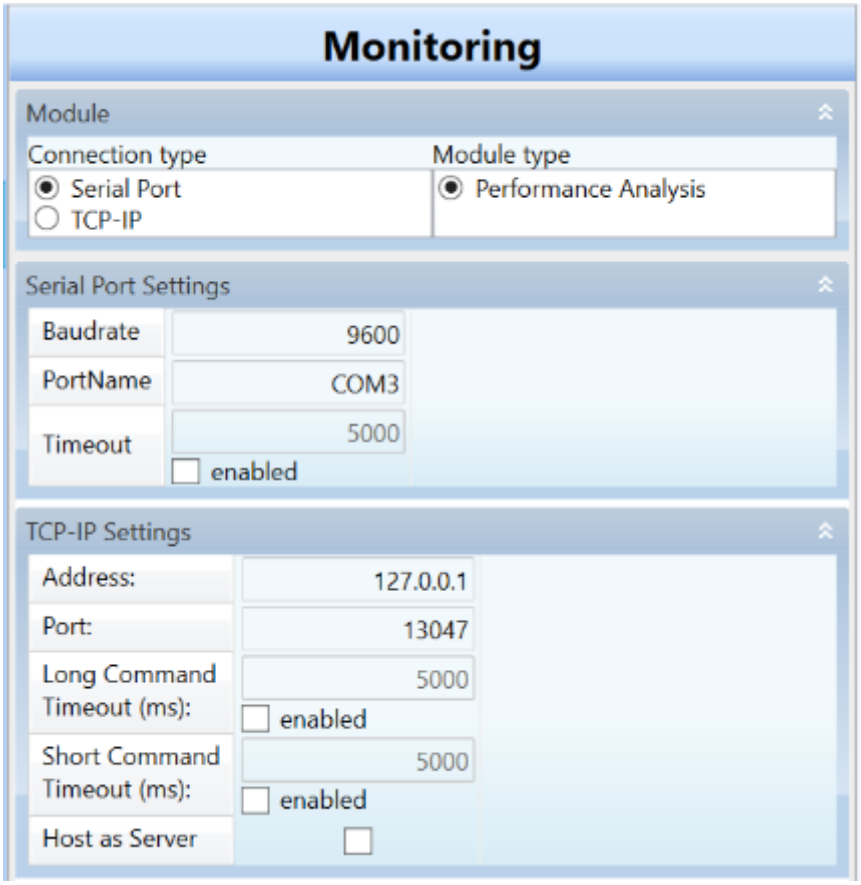


Upon successful connection between Analyzer and the Candera application, the Candera application is in Running Mode. A successful connection is indicated by the status bar. The server state is displayed as Connected and the application state as "Running".

Configuration

The communication server is configured using a configuration dialog accessed through the "Configure" button on the Server Ribbon Group. Figure 20 shows the configuration dialog for TCP/IP connections.

> IP-address and port, where Analyzer should listen for incoming connection requests have to be set.



Timeouts can be set in order to prevent the user interface from freezing indefinitely if the communication fails. Disabling these timeouts completely can lead to freezes and communication failures depending on application type, communication type and stability of connection.

Analysis Views

Trend View

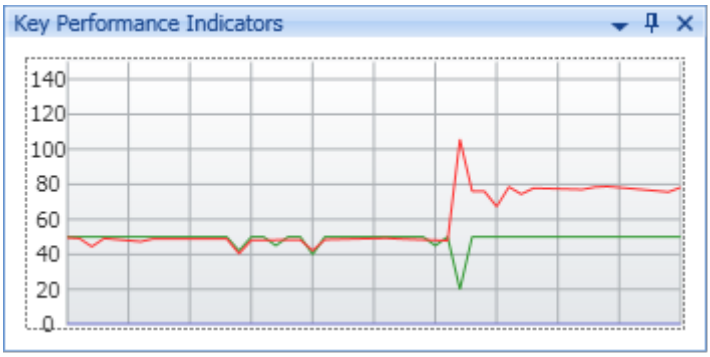
The Trend View shows successively recorded key performance indicators such as

- CPU load (if available)
- GPU load (if available)
- Frame Rate

It can be displayed via the View Ribbon Group of the Online Ribbon Tab.

The following shows a Trend View for a host simulation. The color assignment for the performance indicators is currently fixed and as follows:

- Frame Rate (red)
- CPU load (green)
- GPU load (blue)



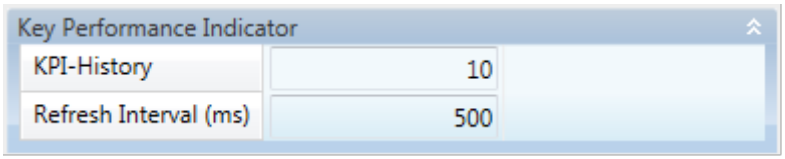
Mouse over the Trend View shows a clear button which resets the view.

Right-clicking the Trend View opens a context menu for

- closing it and
- displaying it always on top of every other window.

> GPU load availability is dependent on device capabilities.

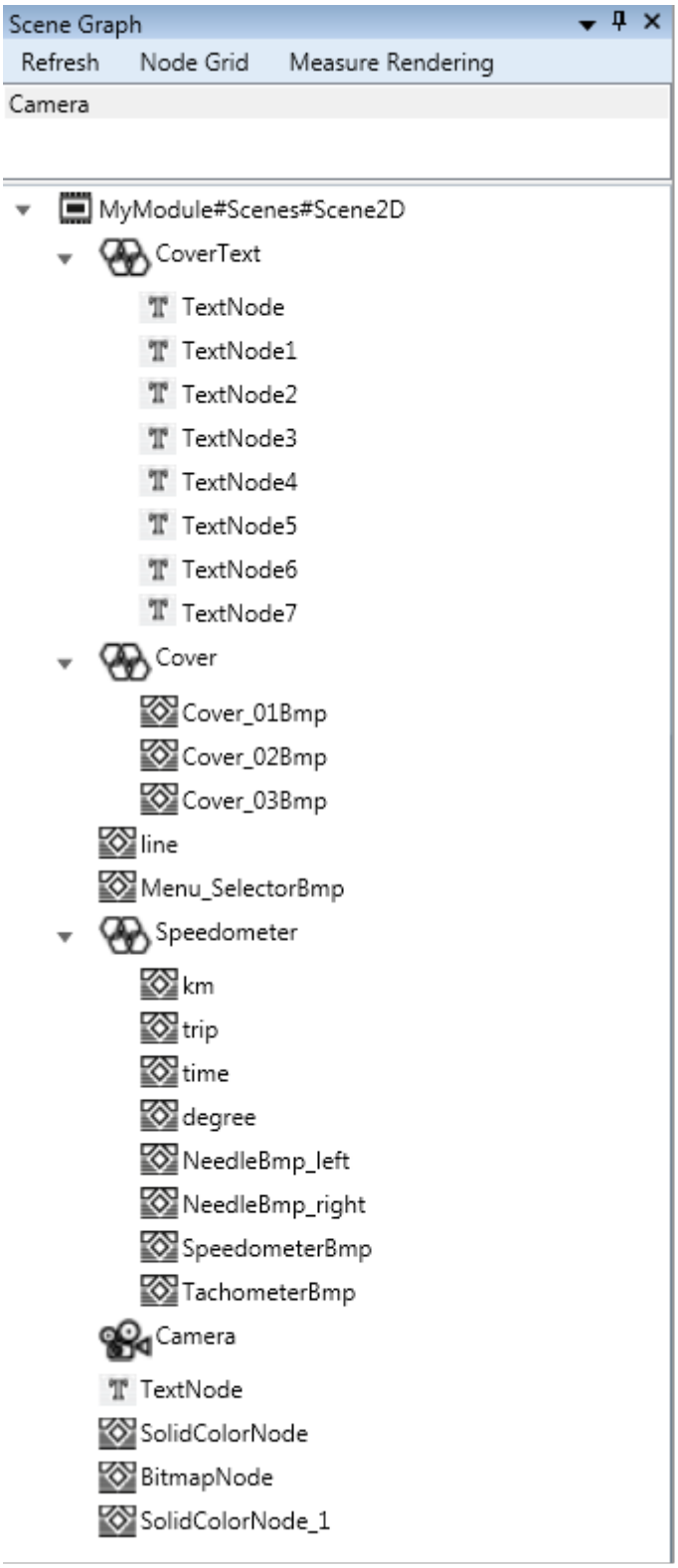
The settings menu also provides settings for trends:



These settings allow to set the update rate and the history length of trends. IT is possible to change these values on the fly without restarting, pausing or changing the communication

Scene Graph View

The Scene Graph View allows analysis of the scene graph during paused mode.



This view is divided vertically by a splitter into two sections:

- A camera list on top and
- A tree view of a scene graph below.

The tree view shows the scene graph associated with the selected camera in the camera list.

The following commands are available via a toolbar on top of the Scene Graph View:

Refresh:

- This command is always available in paused mode.
- Fetches the scene graphs from the Candera application and displays it. Node expansion and selected nodes are preserved.

Node Grid:

- This command is available in paused mode, if a camera was selected from the camera list.
- Displays all information transferred for the scene graph of the selected camera in a grid.

Measure Rendering:

- This command is available in paused mode, if a camera was selected from the camera list.
- Measures the rendering times of individual nodes and displays the render times along with static scene graph information in a grid

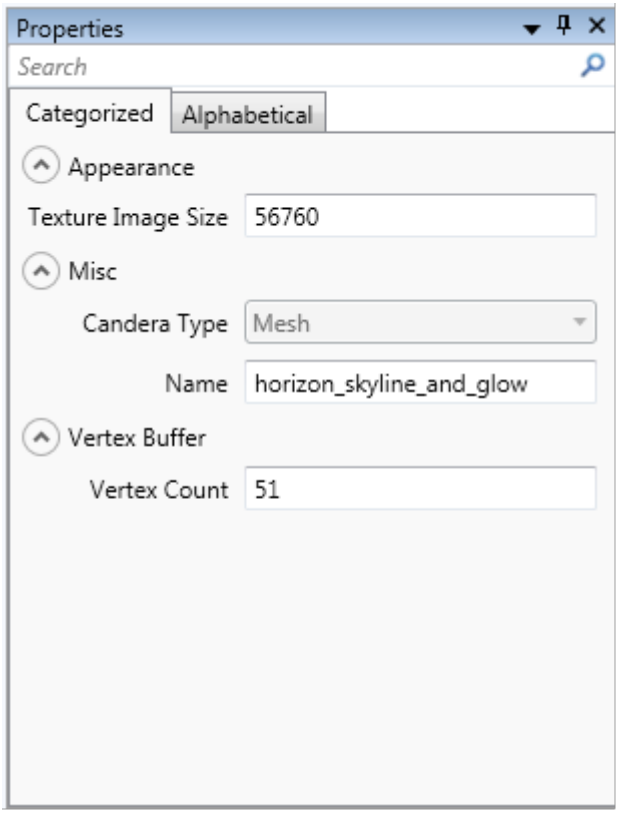
The properties of a selected node are displayed in the Properties View.

Properties View

Depending on the scene graph node selected in the Scene Graph View, different properties are displayed.

For each node, the following properties are displayed:

- **Name:** The name of the node, if available, a blank field otherwise
- **Candera Type:** The type of the node (e.g. Mesh or Group)



Guards

Guards are a means to switch a Candera application from running to paused mode.

We differentiate between two flavors of guards:

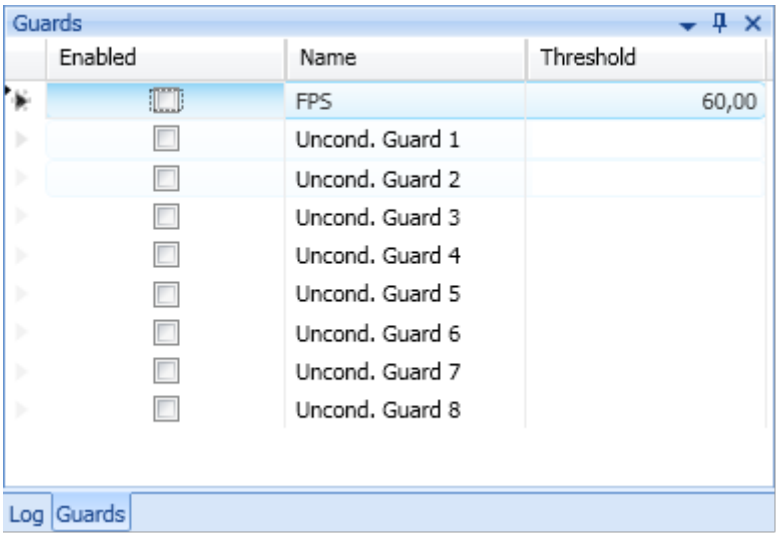
- Conditional Guards
- Unconditional Guards

Conditional guards are triggered, when a condition is met. Currently, only one conditional guard (based on measured frames per second) is supported. If the measured frames per second fall below a specified value, the guard is triggered, if it has been enabled, and the Candera application enters the paused mode.

Unconditional guards can be set in application code via the use of the following macros:

```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_FORCE_PAUSED(groupId)
```

Guards are combined in guard groups. Each guard group has a unique id (the guard group ids are defined in the include file MonitorTypes.h).



All guards can be en- and disabled. Additionally, the number of frames per second can be set for a FPS guard.

Performance Log Transfer

During an active connection to a Candera application, a performance log can be transferred from the target to the host via the communication channel used.

There are two flavors of performance log transfer:

- Recording until the recording buffer is full and explicit transfer of the buffer content
- Continuous streaming of performance log data to a specified file

Single Buffer Recording and Transfer

Caution has to be taken to use only one of the following means of performance log recording

- Record to file on device
- Record to memory buffer on device

If both flavors of recording are enabled, unpredictable effects may occur.

Recording to memory buffer is enabled via the Enable Recording checkbox of the Performance Log Ribbon Group.

After the memory buffer has been filled (approx. 1 sec), the buffer contents can be transferred to the host and stored in a file by clicking the "Get Log" button, also located in the Performance Log Ribbon Group.

Performance Log Streaming

Performance log streaming is the continuous recording and transfer for performance log data from a Candera application to Analyzer.
Streaming is enabled by first enabling recording via the Performance Log Ribbon Group and then enabling streaming via the "Enable Streaming" checkbox. A storage location for the performance log data (file) is asked.

After provision of this information, performance log data is continuously streamed from target to host and appended to the specified file.

Global Experiments

Experiments allow the user to locate performance bottlenecks by intervening in the rendering process. By targeted change of rendering parameters, conclusions about the origins of performance issues can be drawn.

There are three different groups of experiments:

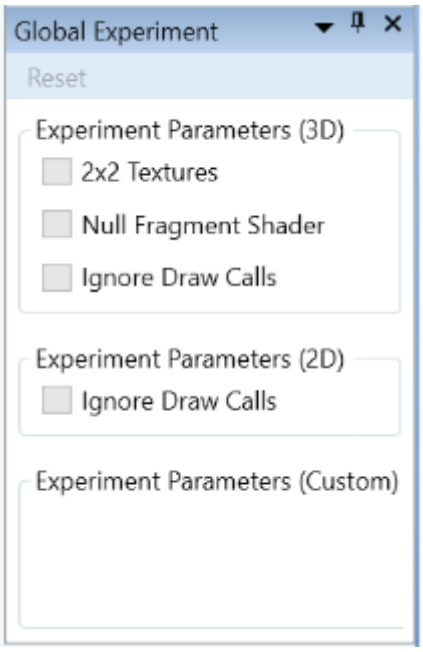
- 3D experiments
- 2D experiments
- Custom experiments

2D/3D Experiments

The following experiments are available:

- 2x2 textures (3D only)
- Null Fragment Shader (3D only)
- Ignore Draw Calls

The following figure shows the dialog for setting experiments. Currently, experiments can only be en- or disabled in Paused Mode.



Using 2x2 textures

All textures are replaces with a 2x2 texture using a chess pattern.
If the frame rate rises significantly after enabling 2x2 textures, there are likely too big textures in use.

Null Fragment Shader Program

All shader programs are replaces with a default shader program whose fragment shader always produces the color red.
If the frame rate rises significantly after enabling the null fragment shader, fragment shader may be to complex.

Ignore Draw Calls

All draw calls are ignored, thereby simulating an infinitely fast GPU. If the frame rate does not rise significantly, the problem may be CPU-bound.

Custom Experiments

Custom experiments are universally usable experiments. In general it is a table of functions which have a parameter (enable/disable). The analyzer can trigger function calls on these specified functions. An experiment state is false/disabled per default.

Custom Experiments - Target side:

To use this feature, the following flag has to be set:

```
// Enables custom experiments
#define CANDERA_MONITOR_CUSTOM_EXPERIMENTS
```

This flag can be set within code when it were not enabled by cmake configuration.

> Using this flag requires init-Call (next step). Compiler produces error without this init-Call.

The following sample code snippet has to be used in order to register a function as experiment:

```
//Adds macros for global experiments
#include <Candera/System/Monitor/GlobalExperimentPublicIF.h>
…
// Function which can be triggered by analyzer
void SampleFunction(bool flag);

// Initializes list – has to be outside of any namespace
//First param is amount of function pointer
//following parameters are function pointers
GLOBAL_EXPERIMENT_INIT_CUSTOM_LIST(1, SampleFunction)
```

The include adds the appropriate macros. To enable these custom experiments previously mentioned define has to be set.

> The Init-Call has to be placed outside of any namespace and function.

A callable function’s declaration is defined as:

```
void FunctionName(bool flag);
```

In general: A static function with no return value. The flag is set by analyzer. A function within namespaces and classes can be added but need full declaration. This function should never be called by custom code - otherwise undefined behavior. The next point explains a way to indirectly call such a function.

Calling a function pointer by target application
If it is required to call a registered function pointer, use the following macro:

```
GLOBAL_EXPERIMENT_CALL_CUSTOM_EXPERIMENT(fct_ptr, value);
```

Global Experiments - Analyzer:

The analyzer provides a view for global experiments. It contains three different areas for 2D, 3D and custom experiments. To enable or disable an experiment toggle the checkbox. Custom experiments are listed in the same order as they were added to list on target side (see Custom

Experiments - Target side: GLOBAL_EXPERIMENT_INIT_CUSTOM_LIST). If the list is wrong or not present, pause the application to update the experiments list.

Additionally it provides a reset button which is used to disable all experiments.

Revision #8
Created 2023-02-16 05:15:48 UTC by Tsuyoshi.Kato
Updated 2025-01-10 08:04:41 UTC by Elisabeth Zauner