## Frame Analysis

CodeXL Frame Analysis allows the developer to generate an API Trace with CPU side timing data and a GPU Trace with GPU side timing data. Both sets of data are displayed in a unified timeline which quickly allows the user to identify expensive GPU executions and the CPU side API calls which generate them. CodeXL allows you capture one or more frames from your running application and the captures are automatically saved to disk. The captured frames can be inspected offline using the CodeXL client (your application does not need to be running).

Frame Analysis supports capturing frame from applications using Microsoft DirectX12 (on Windows) or Vulkan API (on Windows and Linux) to render their graphics.

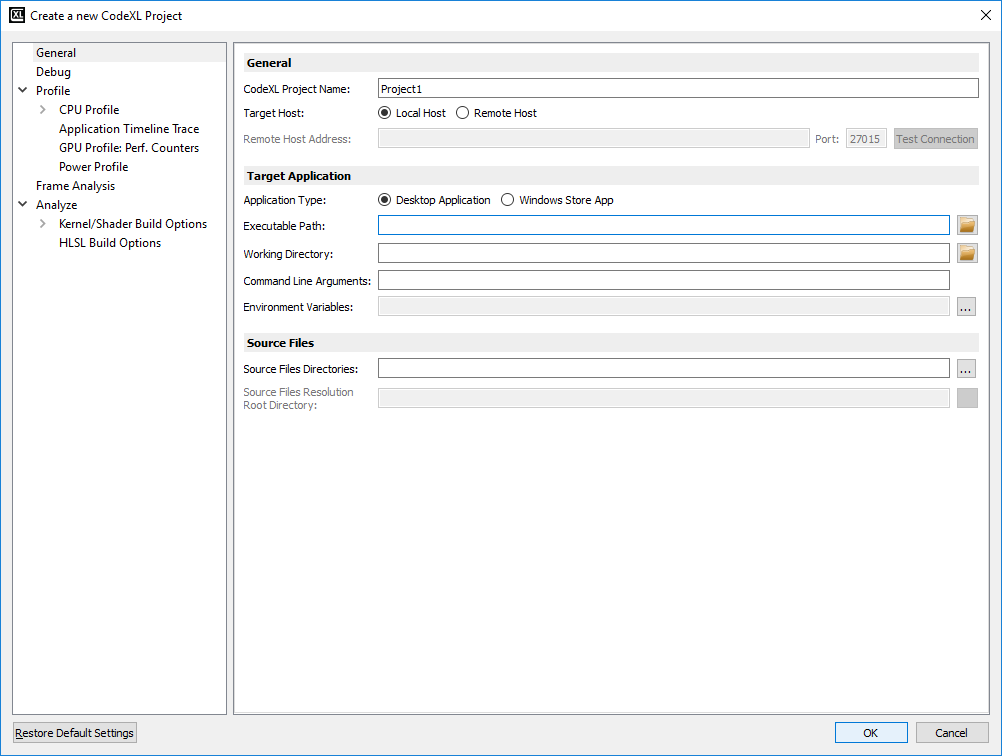
There are 4 stages to viewing the API and GPU trace data from your application.

1. **Creating a New Project**
2. **Starting Your Application**
3. **Capturing Frames**
4. **Viewing Capture Data**

### Creating a New Project

The CodeXL project allows you to define your application executable, its working directory, and any command line arguments it requires. CodeXL will automatically save your project settings so the next time you want to analyze your application you simply load the project to get started.

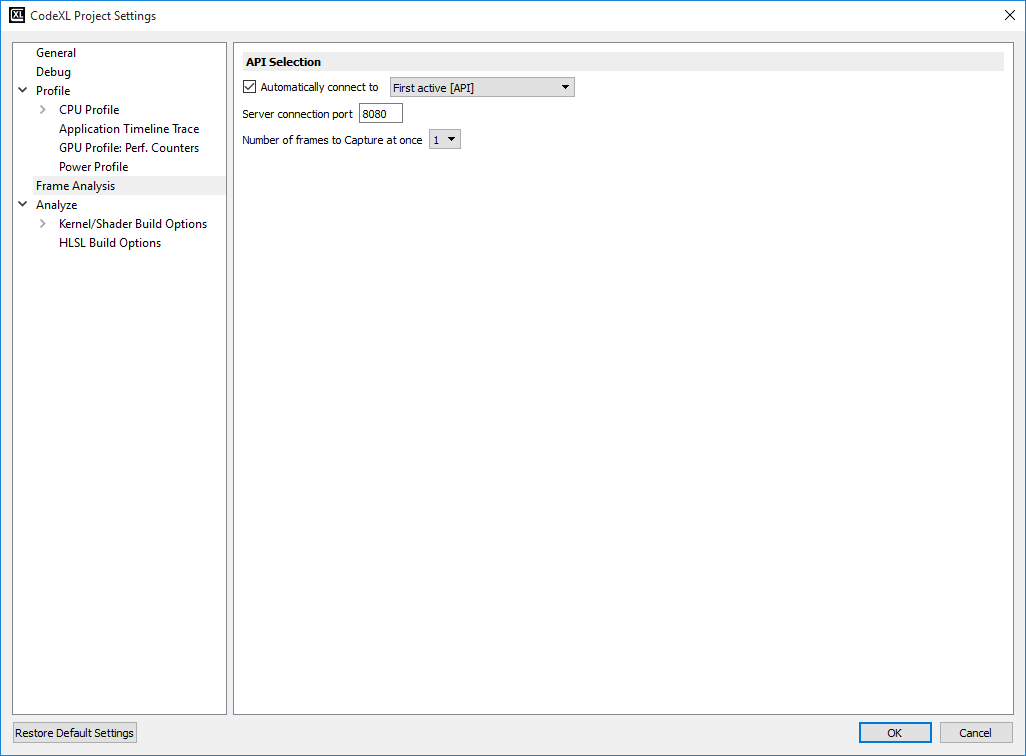
1. Click “File -> New Project” to open the “Create a new CodeXL Project” dialog.
2. Browse to the executable you wish to analyze by clicking on the folder to the right of the “Executable Path:” data entry field.
3. The project will automatically be given the same name as the chosen executable. If you wish to use a different project name, enter a new name under the “CodeXL Project Name:” field **after selecting the executable**.
4. Enter any command line arguments, with which to run the executable, in the “Command Line Arguments:” data entry field.



Frame Analysis settings

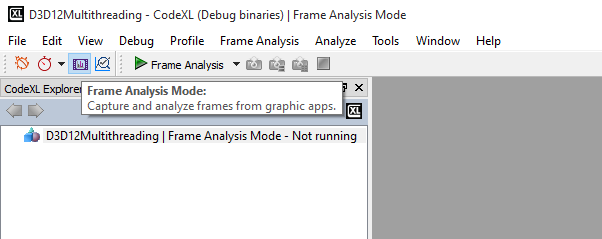
1. In the ‘CodeXL Project Settings’ dialog or the ‘Create a new CodeXL Project’ dialog, click on the ‘Frame Analysis’ tree node to display the ‘Frame Analysis Settings’.
2. By default the “Automatically connect to first active API” option is turned on. When checked, this setting makes CodeXL automatically connect to the first DX12 or Vulkan application it finds when launching an analysis session. There are a couple circumstances where this behavior is undesired:
   1. If the executable launches multiple DX12 or Vulkan processes and you want to be able to choose which one to connect to.
   2. If you have trouble connecting to the DX12 or Vulkan application because CodeXL times out before the program has fully launched.

NOTE: If you have auto-connect disabled, the API selection window will remain open until you select an API context to connect to. Frame capture cannot begin until a connection has been made to a DX12 or Vulkan application.

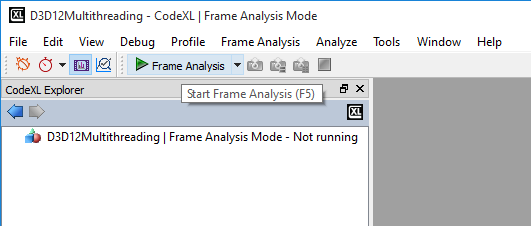


### Starting Your Application

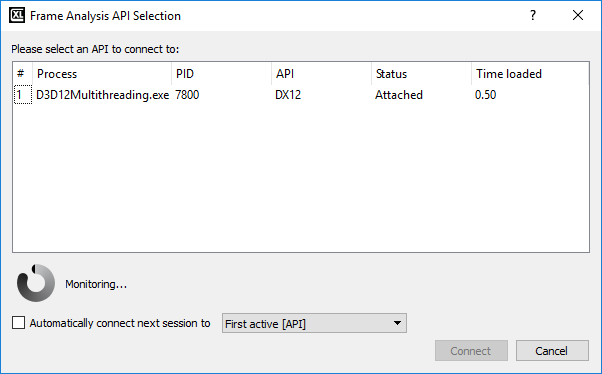
1. Switch to Frame Analysis Mode either by clicking the Frame Analysis button  in the toolbar, or by selecting ‘Frame Analysis’ -> ‘Switch to Frame Analysis Mode’ from the menu bar.



1. Begin a frame capture session either by:
   1. Clicking the Play button  in the toolbar, or
   2. Selecting ‘Frame Analysis’ -> ‘Start Frame Analysis’ from the menu bar, or
   3. Pressing F5.



1. At this point, your application will launch and a selection box will appear with a list of all the API contexts found so far.

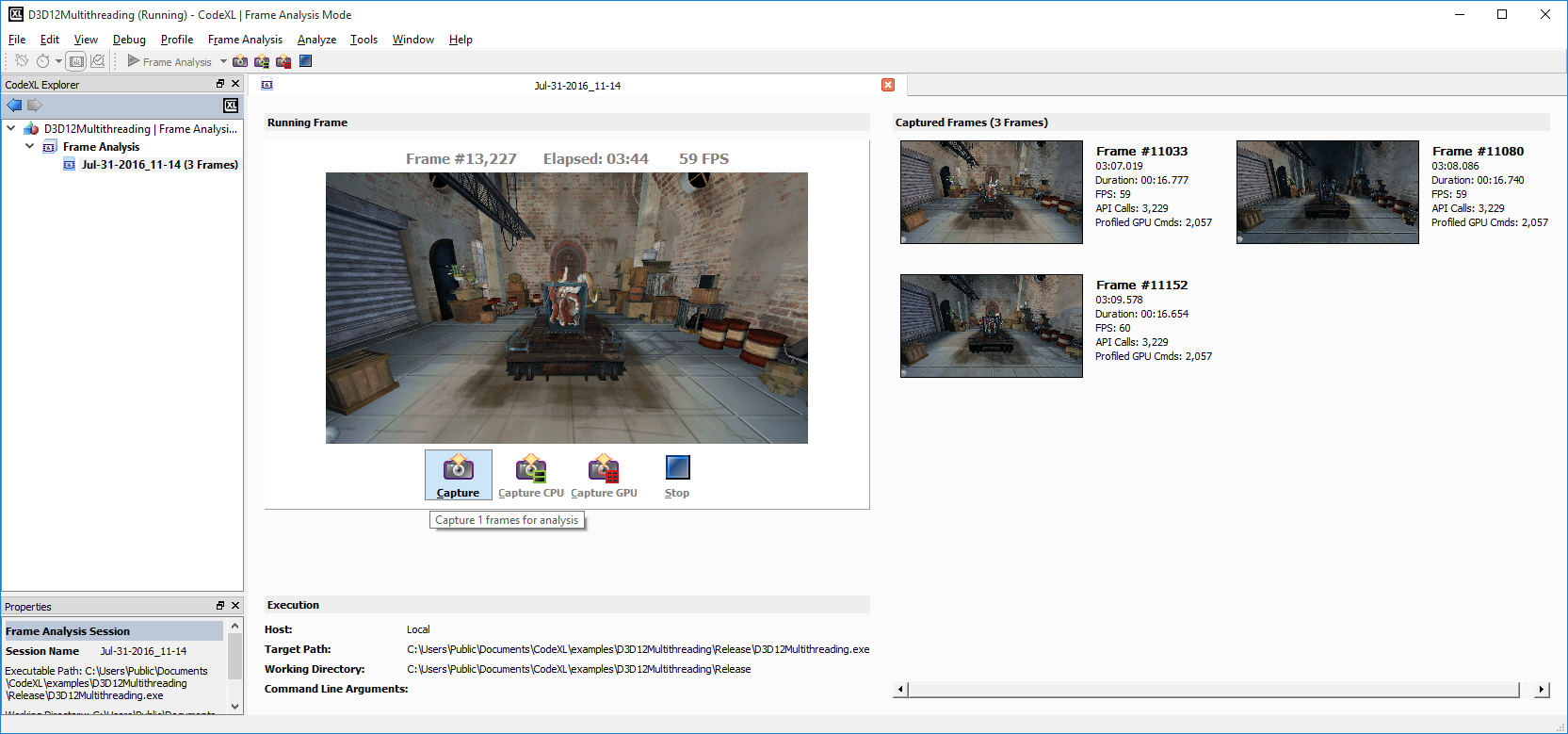


The above image shows the connection dialog with auto-connect disabled. Auto connect will be enabled by default.

Once a connection is established to a DX12 or Vulkan application, focus will switch from the launched program to CodeXL, and you will see a central preview image of the frames being rendered.

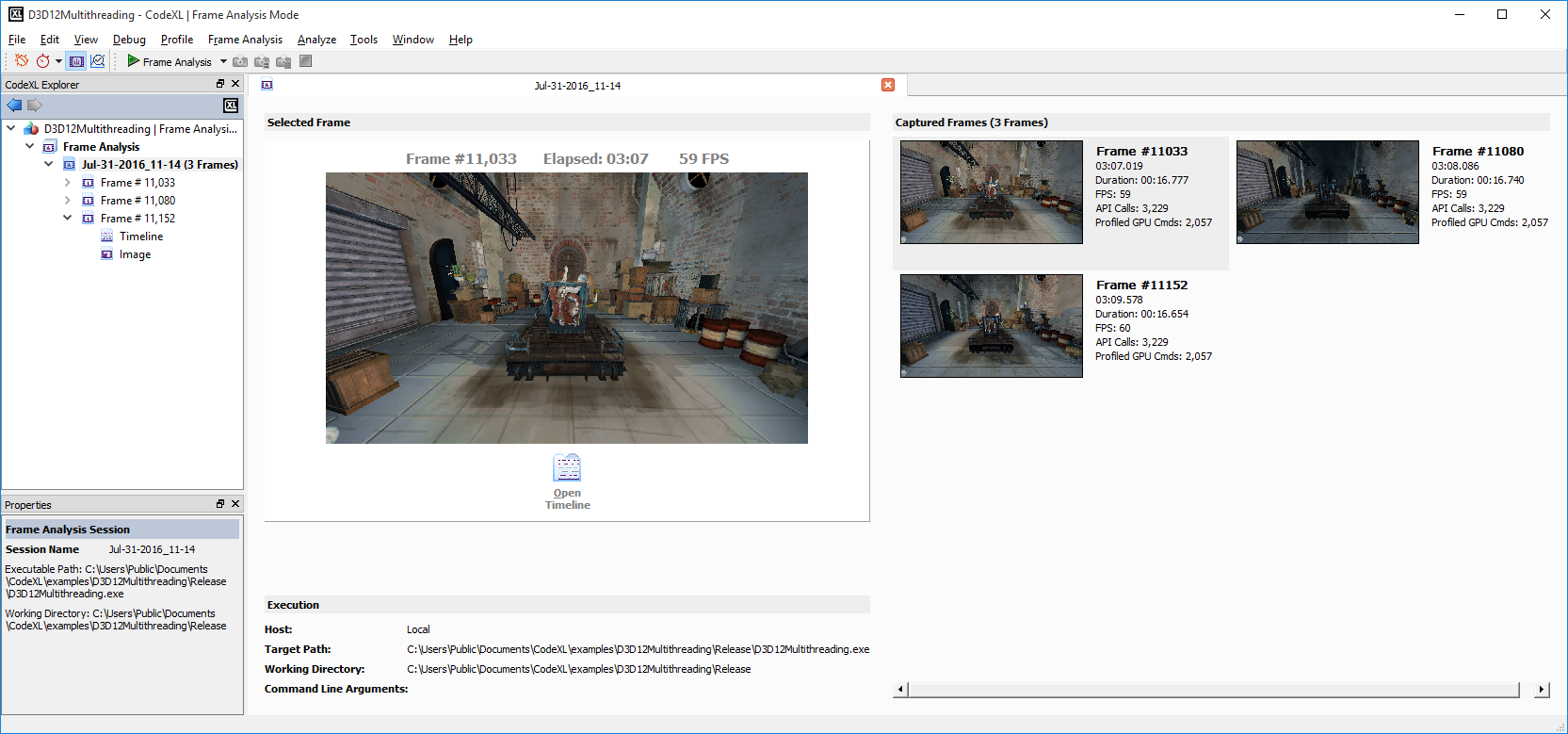
### Capturing Frames

1. Once a frame capture session is running, press one of the Capture buttons  either underneath the central preview image or in the toolbar to capture a frame.
   1.  - capture a linked trace containing CPU API calls and GPU profiled commands.
   2.  - capture only the CPU API calls without profiling GPU commands.
   3. - capture only the GPU profiled commands.
2. Once a frame capture has been initiated, it may take a few seconds to complete. The capture is complete when the capture button returns to an enabled state and a small preview image of the captured frame appears in the margin on the right hand side of the window.
3. Continue to capture as many frames as desired, waiting for a previous capture to finish before initiating a new one. The right hand margin will provide a running count of the captured frames, as well as a small preview image and basic data for each.
4. When stopping the capture session, press the Stop button  either underneath the central preview image or in the toolbar.

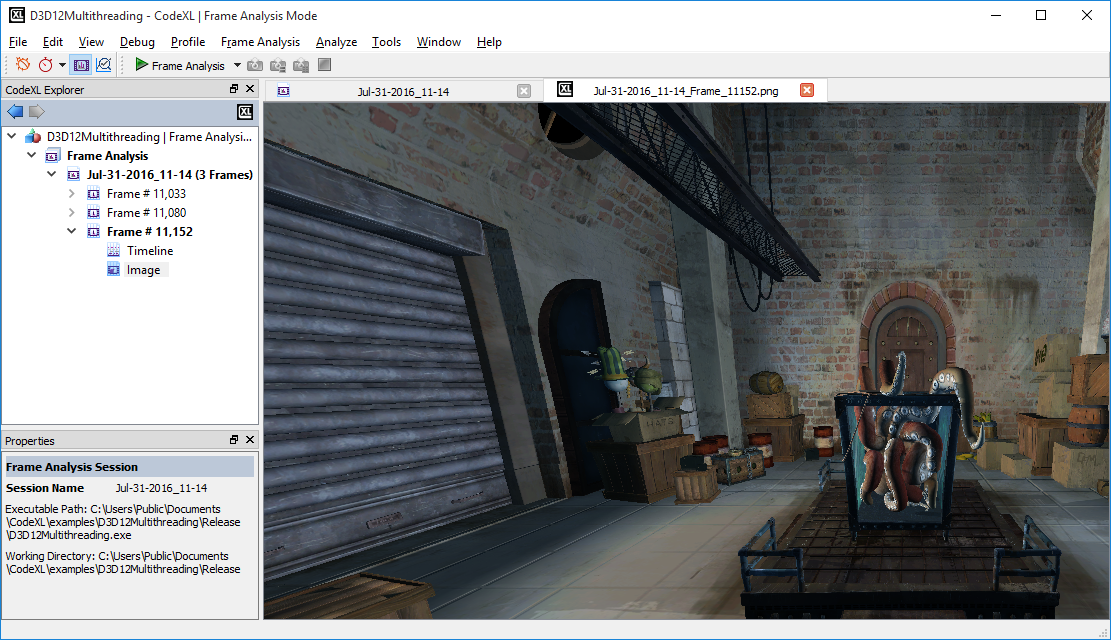


### Viewing Capture Data

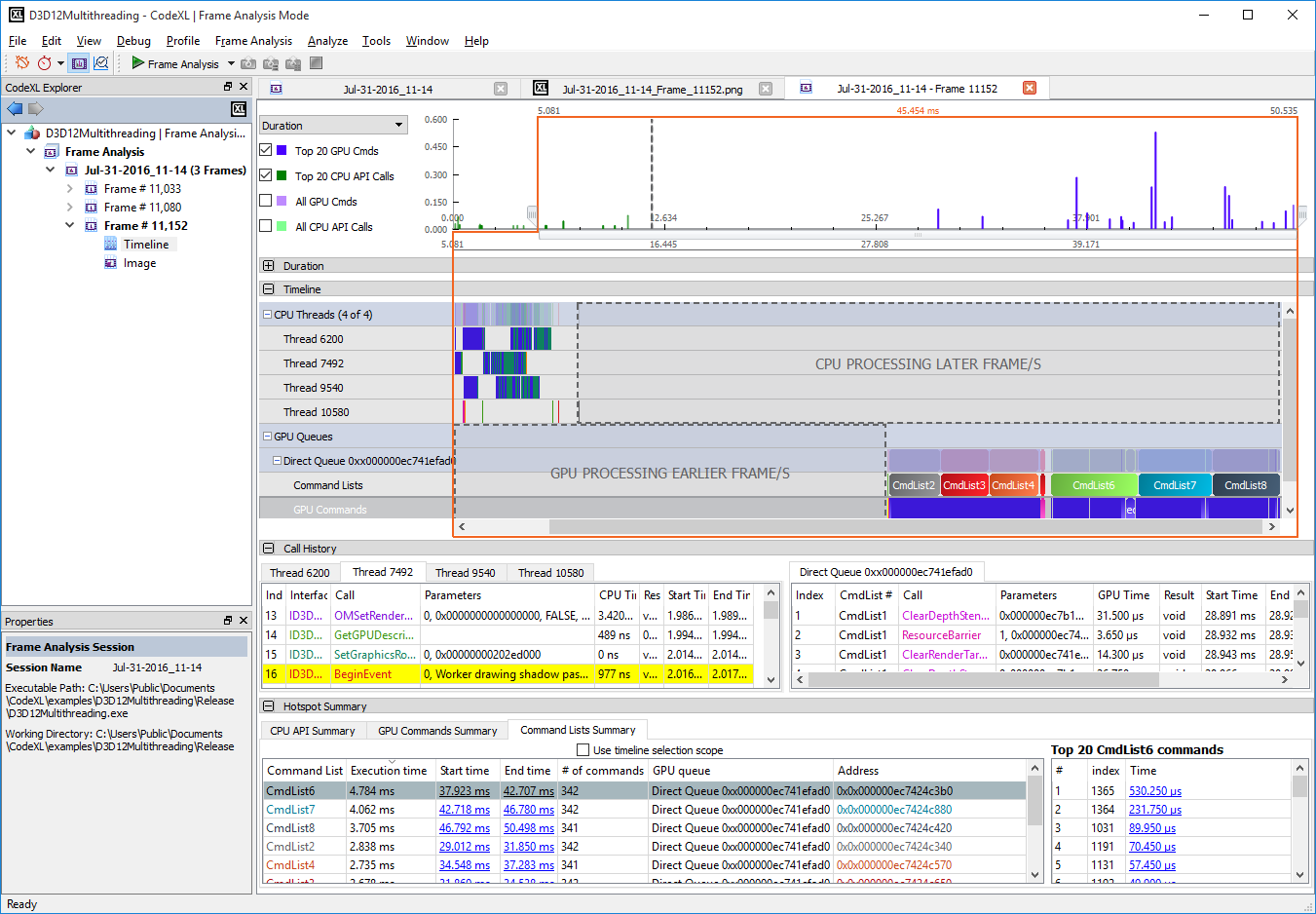
1. After stopping a capture session, the captured frames will appear in the left hand pane CodeXL Explorer Tree under “(**Project name here)** | Frame Analysis Mode -> Frame Analysis”.
2. Expand the desired capture session node (capture sessions are named with the date and time of capture).
3. Select the desired frame sub-node. There should be a Timeline and Image item for each frame.



1. Double click the Image item to bring up a full resolution image of the captured frame.



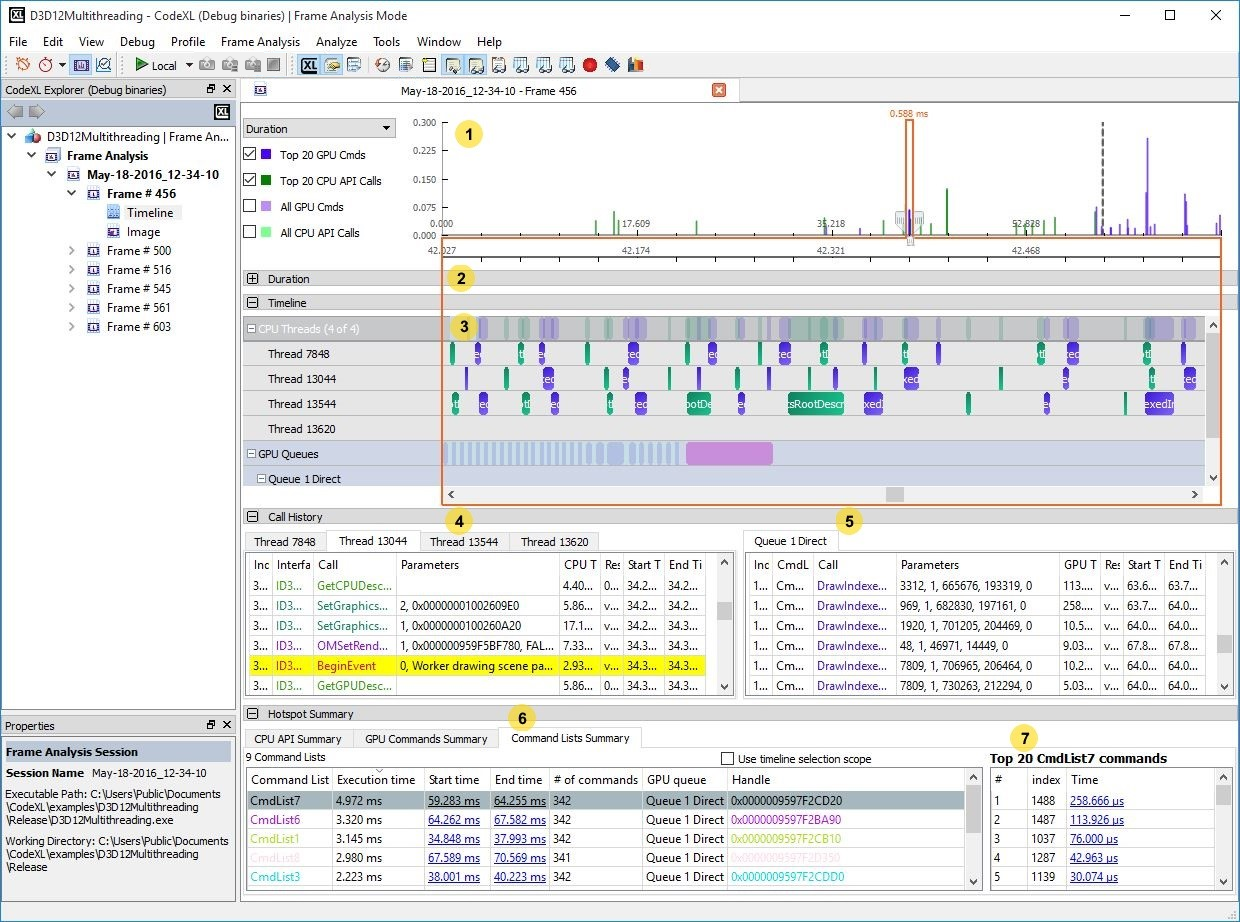
1. Double click the timeline item to bring up GPU and CPU trace data (depending on the type of capture you invoked earlier).



The timeline view displays the collected frame data. See:

* **The Frame Timeline View**
* **Navigating the Frame Timeline View**

#### The Frame Timeline View



The frame timeline view contains the following elements (from bottom to top). Each of the elements’ numbers is in the screenshot above.

1. A navigation chart that allows you to zoom in and out, view the whole frame timeline or focus on a fragment of it, and display API calls duration, count and concurrency.

2. A collapsed detailed view of the API calls and their durations/count/concurrency.

3. A timeline chart visualizing the API calls over the frame timeline.

4. An API calls table for each of the CPU threads.

5. A commands table for each of the GPU command queues.

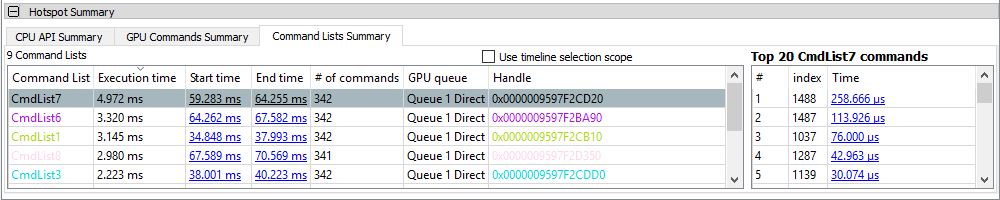
6. Summary tables that summarize the time consumption for each of the API types in the frame.

7. A Top 20 calls table for the currently selected API in the summary table.

#### View and analyze data in API Summary tables

The bottom pane of the trace view contain hotspot Summary tables. The table are summarizing the API call, and display each of the API types significant times. These tables can be used for quick identification of performance bottlenecks in the analyzed frame.

The table below specified each of the API table columns.



CPU \ GPU properties:

|  |  |
| --- | --- |
| **Name** | **Description** |
| Call | The API call name |
| Max Time | The maximum duration for this call type. Clicking on the link in this column will select the specific call in the API table, and the timeline. |
| Min Time | The minimum duration for this call type. Clicking on the link in this column will select the specific call in the API table, and the timeline. |
| Average Time | The average duration for this API type. |
| Cumulative Time | The cumulative duration for all calls with this API type. |
| % of total Time | The percentage of all this API type calls, from the frame duration. |
| # of Calls | The calls count for this API type |

CodeXL 2.1 supports both DX12 and Vulkan API. The hotspot summary table shows DX12 Command list \ Vulkan command buffer properties:

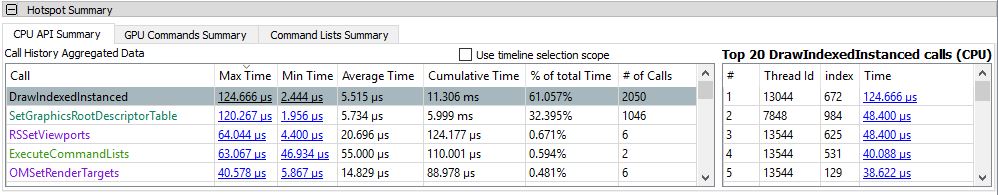
|  |  |
| --- | --- |
| **Name** | **Description** |
| Command List\Buffer | The command list \ buffer name |
| Execution Time | The duration for this command. |
| Start Time | The start time of this command list \ buffer. Clicking on the link in this column will select the specific item in the timeline. |
| End Time | The end time of this command list \ buffer. Clicking on the link in this column will select the specific item in the timeline. |
| # of commands | The number of commands in this command list \ buffer. |
| GPU Queue | The GPU queue associated with this command list \ buffer. |
| Handle | The address of this command list \ buffer. |

* Use timeline selection scope: when checked, the summary tables will reflect the selected timeline frame (which is painted within the red boundaries above). When un-checked, the summary tables will display the whole frame time range.

Selection of a line in the API summary table, will fill the top 20 calls, sorted by the time.

#### View data in the Summary Top 20 Table

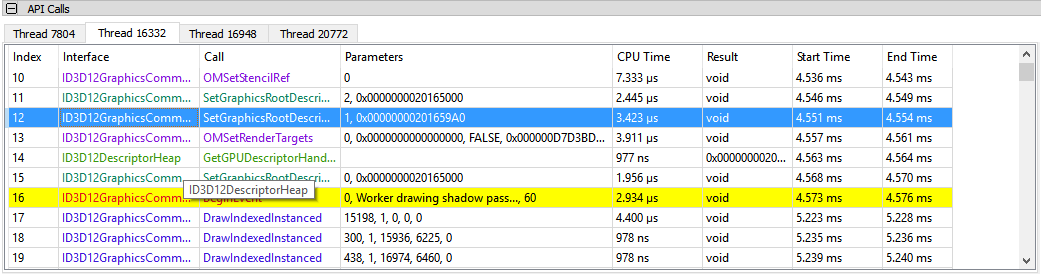
The screenshot below shows the API calls table, with “DrawIndexedInstanced” selected. The Top 20 table will display the top 20 time consuming calls to DrawIndexedInstanced.



|  |  |
| --- | --- |
| **Name** | **Description** |
| # | The number of the call (1 for the top time consuming call) |
| Thread Id | The thread on which the call was executed |
| Index | The index within the thread |
| Time | The call time in ms. Clicking on the link in this column will select the specific call in the API table, and the timeline. |

#### View data in CPU API Tables

The CPU and GPU API table monitor each of the frames API calls. The table below specified each of the API table columns.

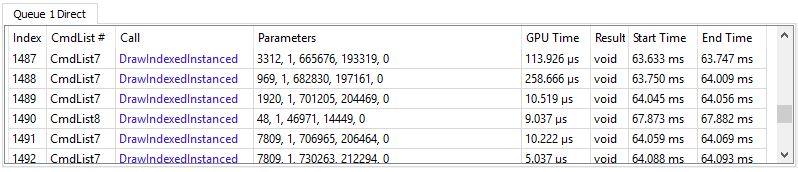


|  |  |
| --- | --- |
| **Name** | **Description** |
| Index | The API call index for this thread. |
| Interface | The API interface. |
| Call | The API call name |
| Parameters | A string describing the parameters for this call. |
| CPU Time | The duration of this API call on the CPU |
| Result | The API return value |
| Start Time | The time (from the frame start time in ms) the API started |
| End Time | The time (from the frame start time in ms) the API ended |

\* Notice: BeginEvent, EndEvent, and SetMarker API calls are marked in yellow for easy navigation.

#### View data in GPU API Tables

The CPU and GPU API table monitor each of the frames API calls. The table below specified each of the API table columns.



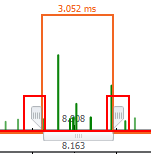
|  |  |
| --- | --- |
| **Name** | **Description** |
| Index | The API call index for this thread. |
| Cmd List # | The index of the command list \ buffer. |
| Call | The API call name. |
| Parameters | A string describing the parameters for this call. |
| GPU Time | The duration of this API call on the GPU |
| Result | The API return value |
| Start Time | The time (from the frame start time in ms) the API started |
| End Time | The time (from the frame start time in ms) the API ended |

#### Navigating the Frame Timeline View

A graphics frame can be very busy and contains tens of thousands of API calls. The following buttons and UI elements are useful in navigating the frame timeline and highlighting API calls of interest:

Focused Timeline Fragment

Use the left and right handles to expand/reduce the focused timeline fragment in and out:

****

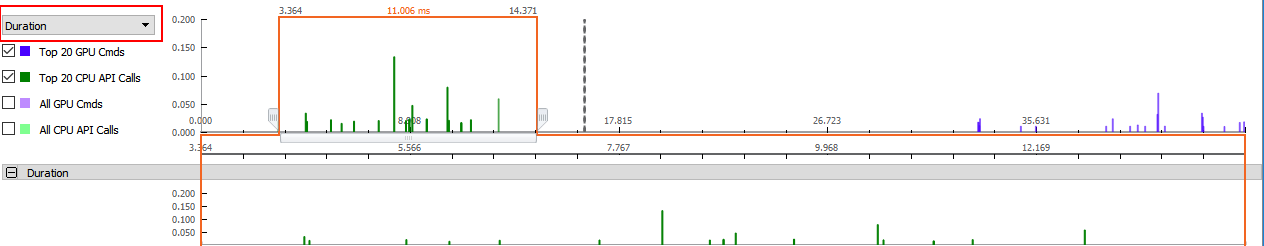
Navigation Bar Visualizations

The navigation bar drop-list contains 3 visualization aids:

* Duration
* Count
* Concurrency

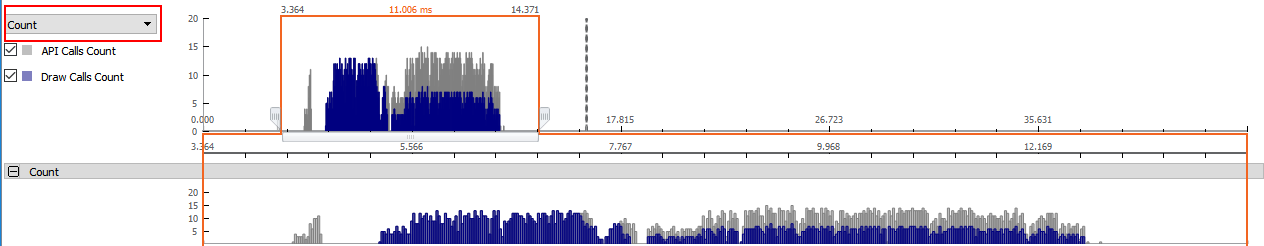
API Calls Duration

Select “Duration” in the top left combo box, to view the duration of API calls:



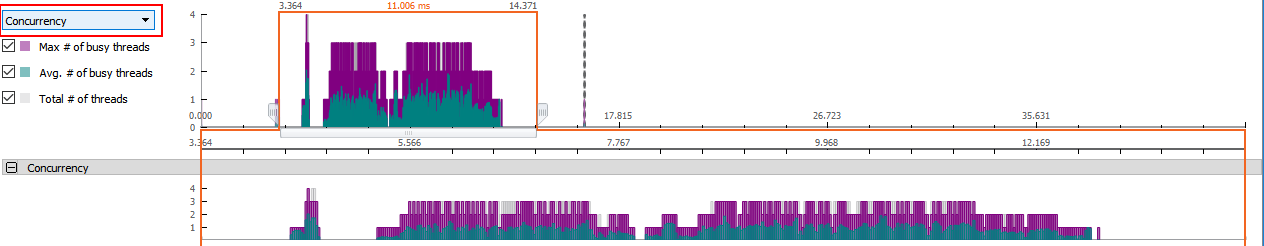
API Calls Count

Select “Count” in the top left combo box, to view the API calls count for each time fragment:



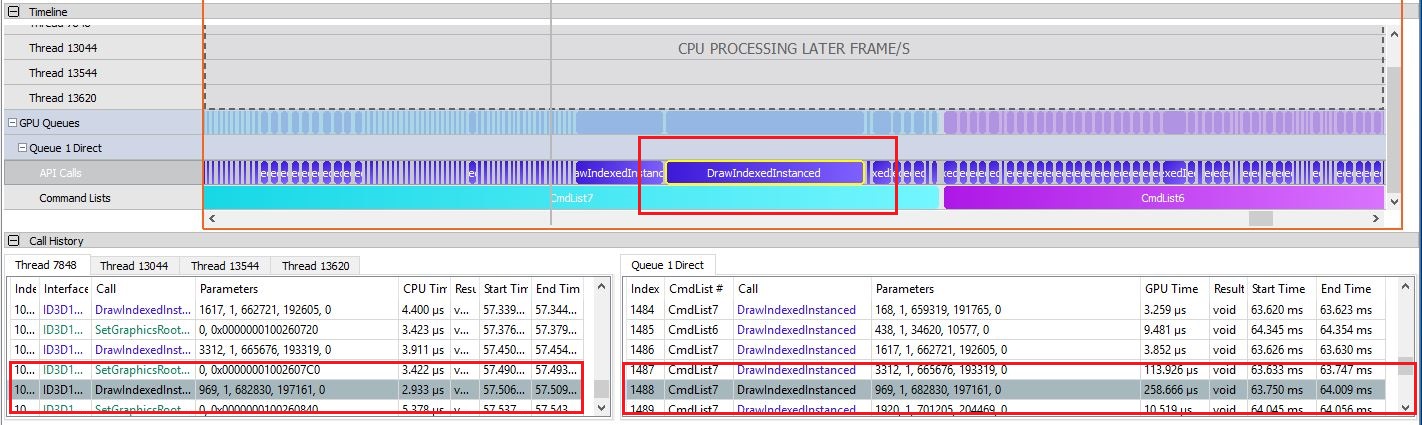
API Calls Concurrency

Select “Concurrency” in the top left combo box, to view the max / average busy threads concurrency over the frame timeline:



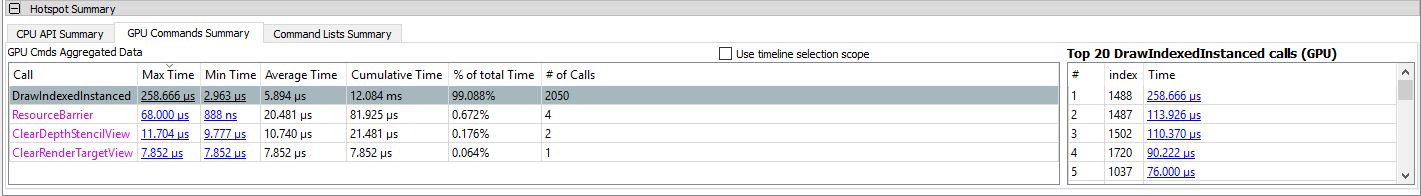
Timeline chart and API Calls tables

Double click a CPU API table item, and the timeline chart will zoom to the corresponding timeline item and highlight it. If there is a linked GPU API item, it will also be highlighted in the GPU API table



Hotspot Summary Tables

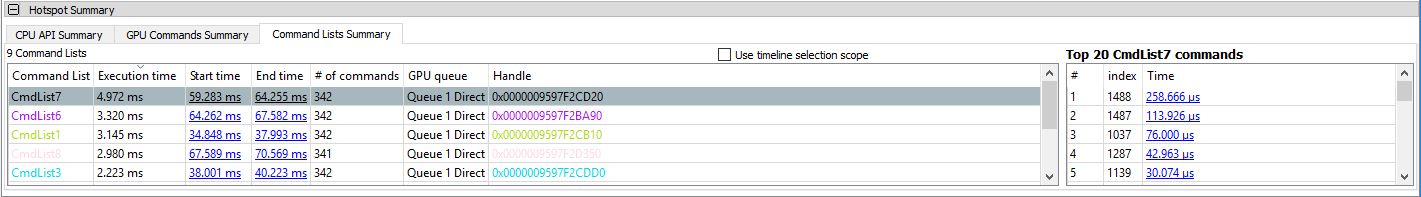
The bottom ribbon of the timeline view contains the Hotspot Summary tables. These tables display aggregated data for API calls and command lists \ buffers, showing statistics for each type of API and the top individual calls to that API type.



The Max Time and Min Time columns display the execution time of the longest and shortest API call of the selected API type. These are also direct links – clicking them causes the API tables and timeline chart to display the individual call item.

The ‘Top 20’ table is automatically populated with the 20 longest calls of the API type selected in the API summary table.

Performance tip: The longest GPU command in the frame, is always the first API call in the Top 20 table when the timeline view is opened.



For command lists\buffers the hotspot table displays the command lists\buffers which were executed during the captured frame.

The Execution Time column displays the execution time of the longest and shortest command list\buffer. The start and end time are also direct links – clicking them causes the timeline chart to display the individual command list\buffer.

The ‘Top 20’ table is automatically populated with the 20 longest GPU call which belongs to the command list\buffer which is selected in the hotspot summary table.

Performance tip: The longest GPU command in the frame, is always the first API call in the Top 20 table when the timeline view is opened.

### Analyzing frames from Steam titles

To prevent intervention with CodeXL’s capture capabilities, verify the following conditions prior to starting the frame analysis session:

* Make sure Gaming Evolved (raptr.exe) is not installed
* Make sure FRAPS is not installed
* ​​Make sure that Steam in-game overlay is disabled (Steam Settings menu, In-Game tab, top 3 checkboxes should be unchecked)

To analyze a steam title, select Steam as the executable that CodeXL will launch, start the Frame Analysis session and use Steam UI to launch the game.