



# CodeXL Quick Start Guide

Version 2.2

Revision 1

## Table of Contents

<b>INTRODUCTION .....</b>	<b>3</b>
<b>LATEST VERSION OF THIS DOCUMENT .....</b>	<b>3</b>
<b>PREREQUISITES.....</b>	<b>3</b>
<b>DOWNLOAD AND INSTALL CODEXL .....</b>	<b>4</b>
<b>CODEXL HELP .....</b>	<b>5</b>
<b>SYSTEM INFORMATION .....</b>	<b>6</b>
<b>TEAPOT SAMPLE PROJECT .....</b>	<b>8</b>
Debug the Teapot Sample Application .....	9
Basic Debugging .....	10
Source Code View .....	11
Breakpoint View .....	12
Watch and Locals Views .....	13
In kernel debugging mode .....	13
In Host debugging mode .....	15
Multi-Watch view .....	15
Explorer View .....	15
Call Stack View .....	18
Function Calls History View .....	19
Debugged Process Events View .....	19
Memory View .....	20
Statistics View .....	20
<b>MATRIX MULTIPLICATION PROJECT .....</b>	<b>21</b>
Perform CPU Profile for the Matrix Multiply Sample Application .....	23
CPU Time Based Profile Navigation .....	23
Source Code View .....	24
Run the classic textbook sample .....	25
Analyzing the classic implementation .....	26
Analyzing the improved implementation .....	29
<b>D3DMULTITHREADING PROJECT.....</b>	<b>30</b>
Open CodeXL D3DMultiThreading sample .....	30
Start a Frame Analysis session .....	31
Capture frame for analysis .....	33
Analyze a captured frame .....	33
The frame timeline.....	34
Navigating the frame timeline.....	35
<b>PROFILE MODE .....</b>	<b>36</b>
<b>CPU Profiling .....</b>	<b>37</b>
Overview Tab .....	37
Modules Tab.....	38
Call Graph Tab .....	39
Functions Tab .....	40
<b>GPU Profiling.....</b>	<b>41</b>
Summary Tab .....	42
Performance Counters View .....	43
CodeXL Explorer Tree .....	44
<b>Power Profiling .....</b>	<b>46</b>
Switching to Power Profiling mode .....	46



Starting a new Power Profiling session .....	46
Setting the Sampling Interval .....	47
Stopping a Power Profiling session .....	47
Power Profiling Real-Time Values .....	48
Power Profiling Timeline View .....	48
Power Profiling Summary View .....	49
Configuring Power Profiler Sessions .....	50
<b>ANALYZE MODE .....</b>	<b>50</b>
<b>Static Kernel and Shader Analysis .....</b>	<b>50</b>
Switching to Analyze mode .....	50
Creating a new project for Analysis .....	51
Working with the new CodeXL Analyzer Explorer Tree .....	51
Working with Programs .....	54
Working with Folders .....	55
Selecting target devices .....	57
Build Options - defining kernel/shader compilation options .....	58
Output Tab .....	62
Statistics Tab .....	62
Viewing compilation output: ISA and IL .....	63
Navigating through ISA code with the Enhanced ISA View .....	64
<b>KNOWN ISSUES .....</b>	<b>65</b>
<b>SUPPORT .....</b>	<b>65</b>



## Introduction

CodeXL™ is a tool suite with a unified user interface that lets you harness the benefits of CPUs, GPUs, and APUs. It has powerful capabilities for

- APU/GPU debugging,
- CPU and GPU profiling,
- DirectX® 12 and Vulkan® frame analysis,
- and static OpenCL™/DirectX/OpenGL®/Vulkan kernels and shaders analysis.

These features let you find bugs, optimize application performance, and easily access heterogeneous computing.

CodeXL source code and released binaries are publically available on GitHub as part of the [GPUOpen](#) initiative. CodeXL is available as a stand-alone application for Windows® and Linux®, as well as a Microsoft® Visual Studio® extension for Windows.

Getting the most out of the CodeXL tool suite requires a relatively recent AMD APU, a recent version of Radeon Software, and the OpenCL APP SDK.

This document describes how to

- get started using CodeXL
- find information about known CodeXL issues
- contact AMD for support

## Latest Version of This Document

- For the latest and greatest version of the documentation, go to the [CodeXL Website](#).

## Prerequisites

### Operating Systems

- Microsoft Windows 7 64-bit
- Microsoft Windows 8.1 64-bit
- Microsoft Windows 10 64-bit
- Linux 64-bit (Red Hat, Ubuntu, SUSE)

For detailed system requirements see the CodeXL Release Notes in the CodeXL installation folder or on the Documentation section of the [CodeXL web page](#).

### CodeXL Visual Studio Extension

- [Optional] Microsoft Visual Studio 2010 (Standard/Professional/Team System Edition)



- [Optional] Microsoft Visual Studio 2012 (Professional/Premium/Ultimate Edition)
- [Optional] Microsoft Visual Studio 2013 (Professional/Premium/Ultimate Edition)
- [Optional] Microsoft Visual Studio 2015 (Community/Professional/EnterpriseEdition)

### Profiling OpenCL™ Applications

- [GPU device] AMD Catalyst driver with OpenCL™ GPU support
- [GPU device] AMD Radeon™ HD 5000 series or newer
- AMD APP SDK ([requirements](#))

For detailed system requirements see the CodeXL Release Notes in the CodeXL installation folder or on the Documentation section of the [CodeXL web page](#).

## Download and Install CodeXL

Installation is system-specific (Windows or Linux); but once installed and started, the CodeXL operation is system-independent.

Download the CodeXL installation package from the [CodeXL GitHub repo](#).

### For Windows

1. Download the **.exe** file **CodeXL\_Win\*.exe**.
2. When the download completes, double-click the **.exe** file to install CodeXL.  
The installer guides you through the installation process.  
The CodeXL Visual Studio 2010, 2012, 2013 and 2015 extensions are part of the installer package and are installed by default.
3. In the configuration dialog, de-select the Visual Studio extensions if you do not want to install them.
4. Launch CodeXL from C:\Program Files (x86)\CodeXL\CodeXL.exe or from the created Desktop shortcut.

### For Linux

Either install the dedicated distribution package or use the tar archive.

The default installation folder

### For Red Hat and other Fedora based Linux distributions

1. Download the 64-bit Linux **.rpm** package **CodeXL\_Linux\*.rpm**.
2. If installed version 1. at the machine, remove it first:  

```
$ rpm -qa | grep -i codexl
```

```
$ sudo rpm -e <package name>
```



3. Install the RPM package:  
`$ sudo rpm -Uvh CodeXL_Linux*.rpm`
4. Navigate to `/opt/CodeXL_X.Y-ZZZZ/`
5. Launch CodeXL using `./CodeXL`.

### **For Ubuntu and other Debian based Linux distributions**

1. Download the 64-bit Linux **.deb** package **codexl\*.deb**.
2. If installed version 1. at the machine, remove it first:  
`$ dpkg -l |grep -i codexl`  
`$ sudo dpkg -r <package name>`
3. Install the DEB package:  
`$ sudo dpkg -i codexl_x.x-x_amd64.deb`  
`$ sudo apt-get -f install`
4. Navigate to `/opt/CodeXL_X.Y-ZZZZ/`
5. Launch CodeXL using `./CodeXL`.

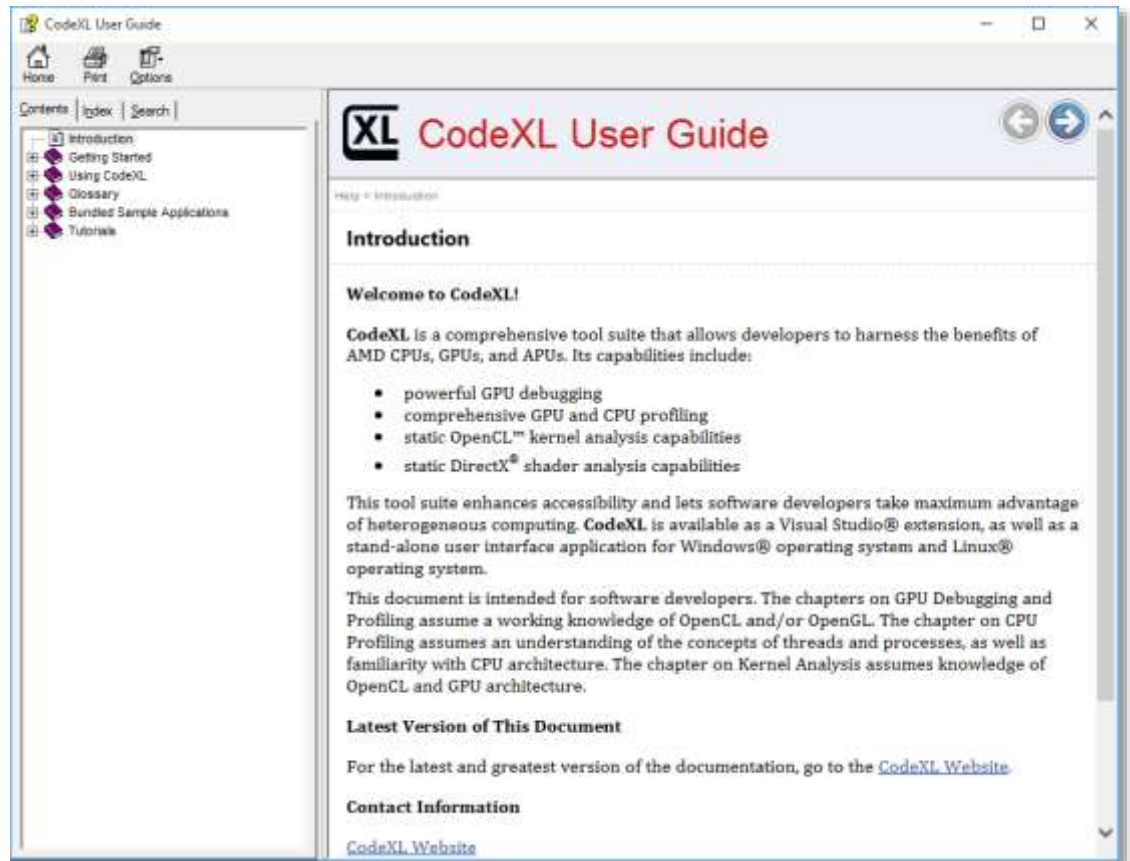
### **Use the tar archive**

1. Download the 64-bit Linux **.tar** package **CodeXL\_Linux\*.tar.gz**.
2. Extract the tar package:  
`$ tar -xvzf CodeXL_Linux*.tar.gz`
3. Install the PowerProfile driver:  
`$ sudo <codexl-install-dir>/ CXLTPwrProfDriver.sh`  
`install`
6. Navigate to `<codexl-install-dir>`
7. Launch CodeXL using `./CodeXL`.

## **CodeXL Help**

To bring up a CodeXL Help window:

1. Click on the [CodeXL User Guide](#) link on the CodeXL startup Home Page.  
OR
2. Select Help >> View Help from the CodeXL toolbar.



CodeXL Help provides some of the same information provided in this document, but also includes additional details about CodeXL views and modules.

To bring up the CodeXL Help window for the Visual Studio extension:

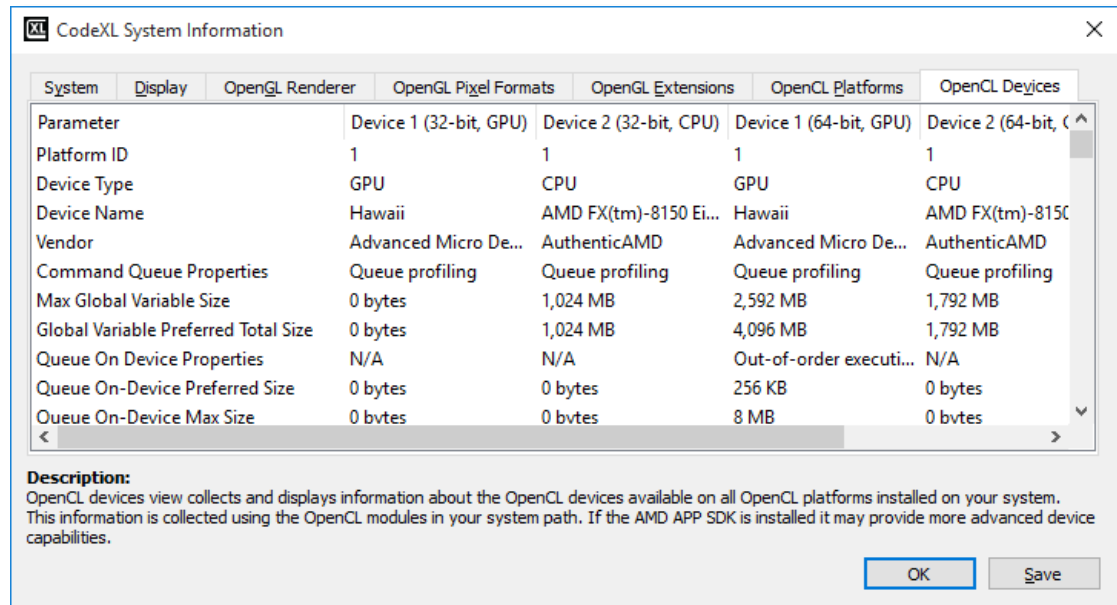
1. Select CodeXL >> Help >> View Help from the VS menu.

## System information

To display system information:

1. Select Tools >> System Information from the CodeXL toolbar.

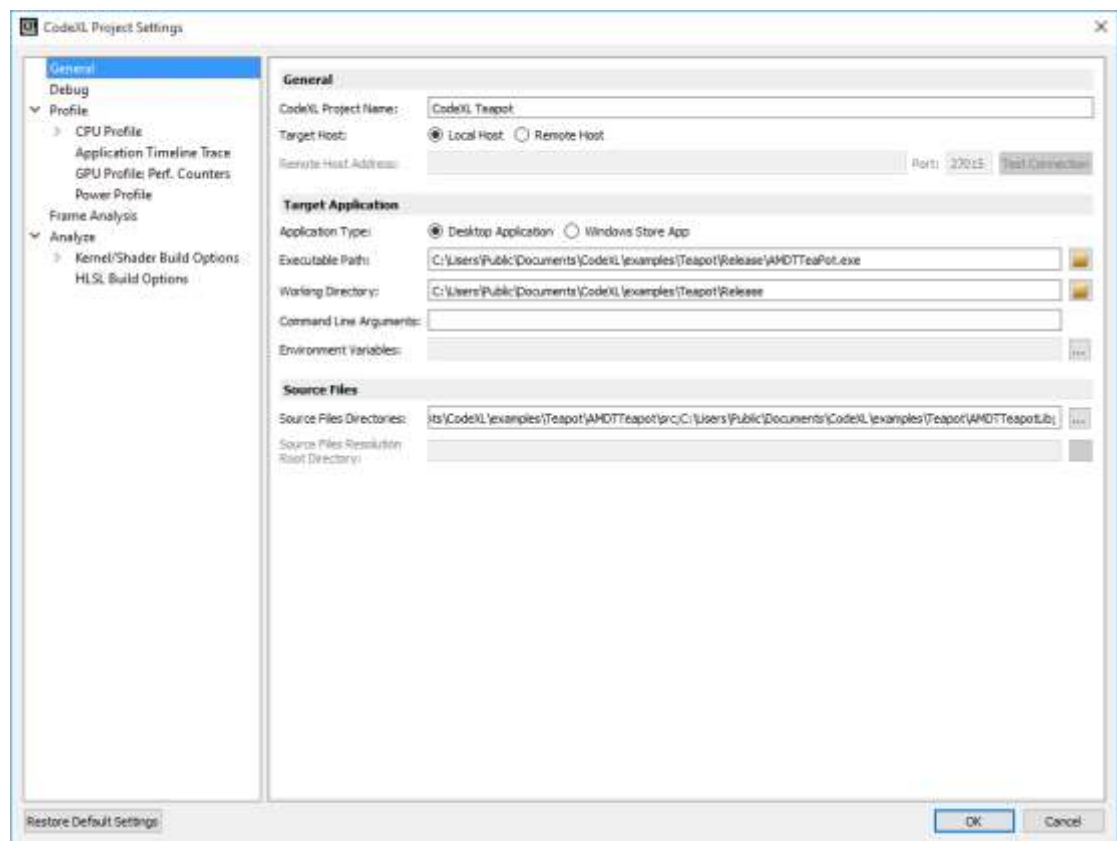
The tabs let you select a category of information. The following screenshot shows OpenCL device information for a GPU device and a CPU device on the runtimes available locally - a 32-bit and 64-bit runtime.



To display project settings, the project must be stopped.

To edit the settings of a project:

1. Select File >> Project Settings from the drop-down File menu.



See the CodeXL Help for more details about project settings.

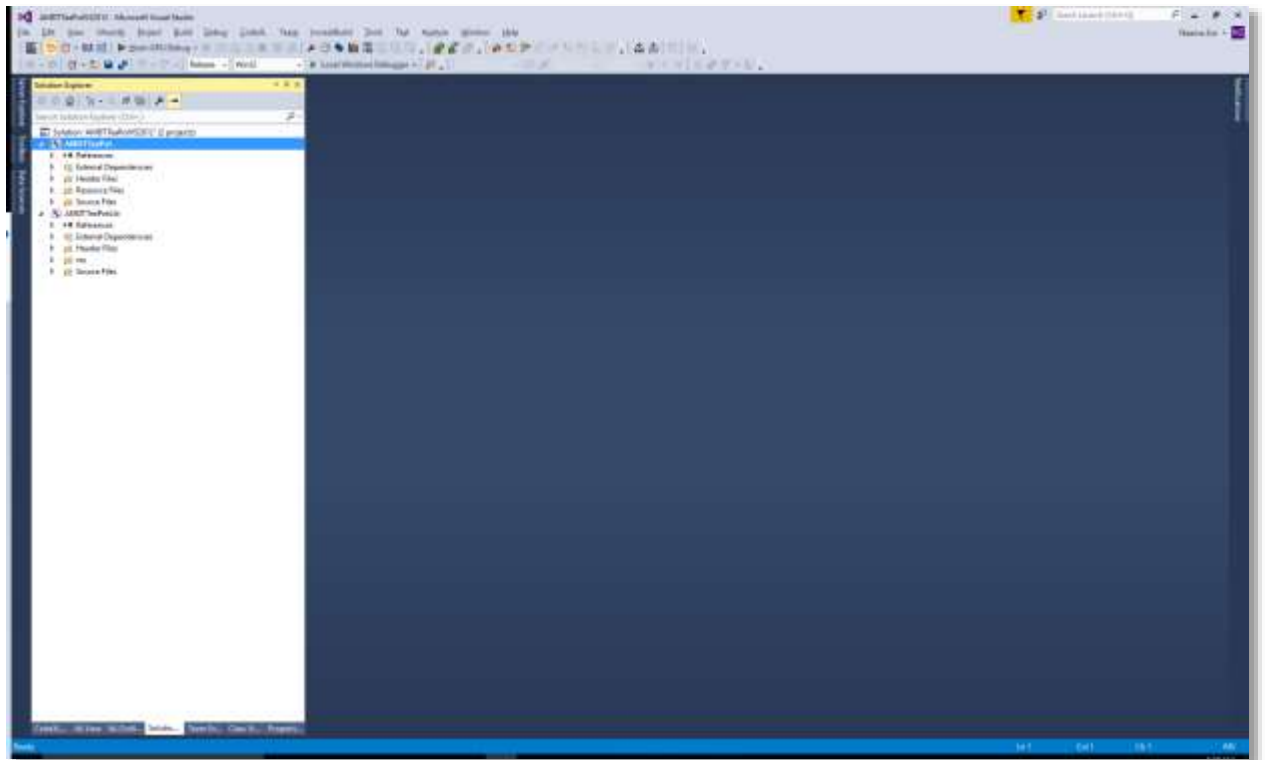


## Teapot Sample Project

The CodeXL distribution includes a sample project that displays a smoking teapot. The project uses OpenCL kernels to solve Navier-Stokes equations. It shares a 3D texture between OpenCL and OpenGL, copies a density field grid into the 3D texture, and renders the smoke using OpenGL.

For the Visual Studio extension:

1. Select CodeXL >> Open Teapot Sample Project from the VS menu.  
Visual Studio displays the teapot sample project.

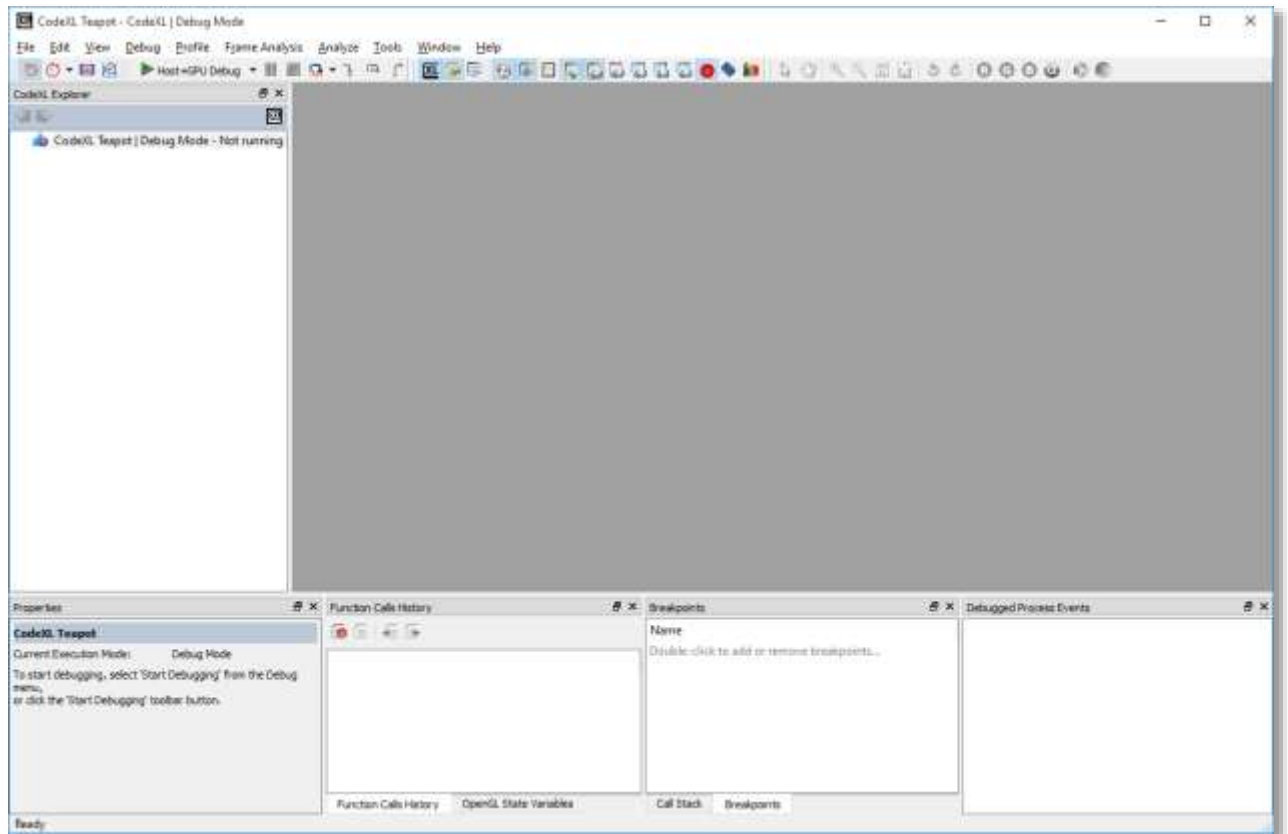


Screenshots in the remainder of this document show the standalone version of CodeXL. The Visual Studio version is similar, but contains a VS window rather than a CodeXL window.

For Windows or Linux:

1. In the CodeXL welcome page (in the CodeXL menu bar, click on File->Welcome Page), Under the Samples header, click the [CodeXL Teapot](#) link.





The CodeXL Explorer view now shows:

CodeXL TeaPot | Debug Mode - Not running

The CodeXL window also displays several other views, but since the program is not running, those views do not display any information.


### Debug the Teapot Sample Application

**Note:** Before debugging the Teapot sample application, you must load it (see the previous section).

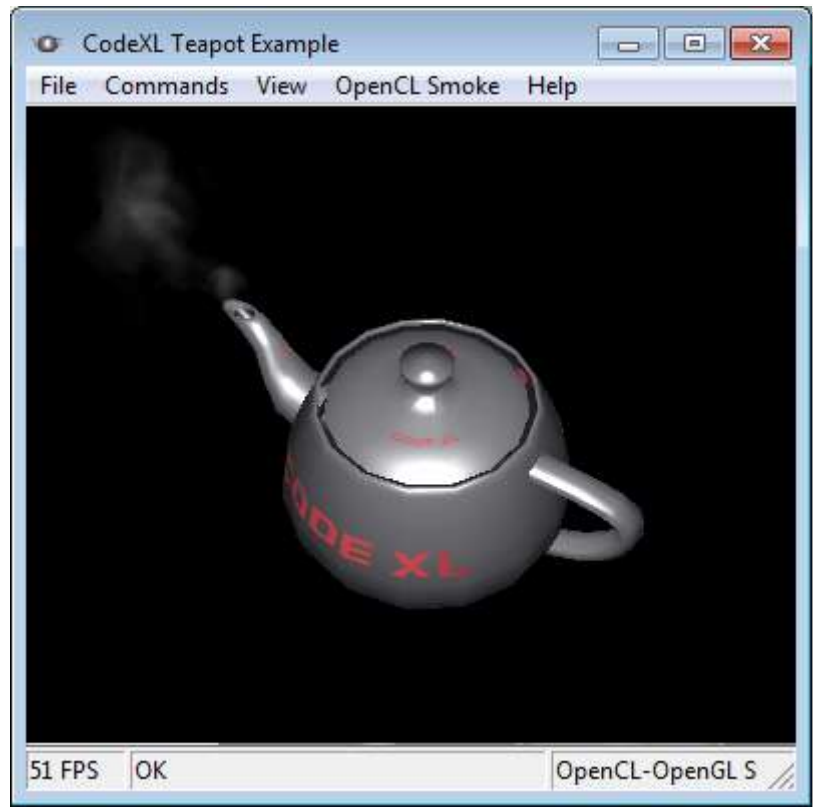
After the teapot sample is loaded, run the debug program:

1. Select Debug >> Start Debugging from the taskbar,


or

2. Click on the green right arrow  taskbar.

The program begins execution, and soon displays a rotating smoking teapot in a separate window.



To stop the program:

1. Select Debug >> Stop Debugging from the taskbar,  
or
2. Click the black square taskbar Stop button ,  
or
3. Click the close button in the upper-right corner of the teapot window.

## Basic Debugging

The CodeXL GPU Debugger lets you examine the runtime behavior of your OpenCL/OpenGL application in detail. You can use the information it provides to find bugs and to improve application performance. You can debug OpenCL kernels, inspect variable values across different work items and work groups, and inspect call stacks, among other things.

This quick start guide presumes you are familiar with the use of a GUI debugger; so the guide provides only a quick introduction to the basic CodeXL debugging features.

The following four buttons, at the far left of the CodeXL taskbar, let you select Debug mode, Profile mode, Frame Analysis or Analyze mode.



Hovering over a taskbar button displays a pop-up help description.



The following taskbar buttons control program execution during debugging.



These controls are (left to right): start, pause and stop debugging, frame / api / draw step (in drop-down menu), step in, step over, step out. You can also perform these actions from the taskbar Debug pull-down menu, or by using function keys.

**Note:** host code debugging (stepping and breaking in C/C++ code) is currently only available in the Linux CodeXL standalone application, and in Windows CodeXL Visual Studio extension (32-bit native C/C++ target applications only). In all other configurations, the step commands will only be enabled in kernel debugging mode.

The following taskbar buttons show, or hide, various views.





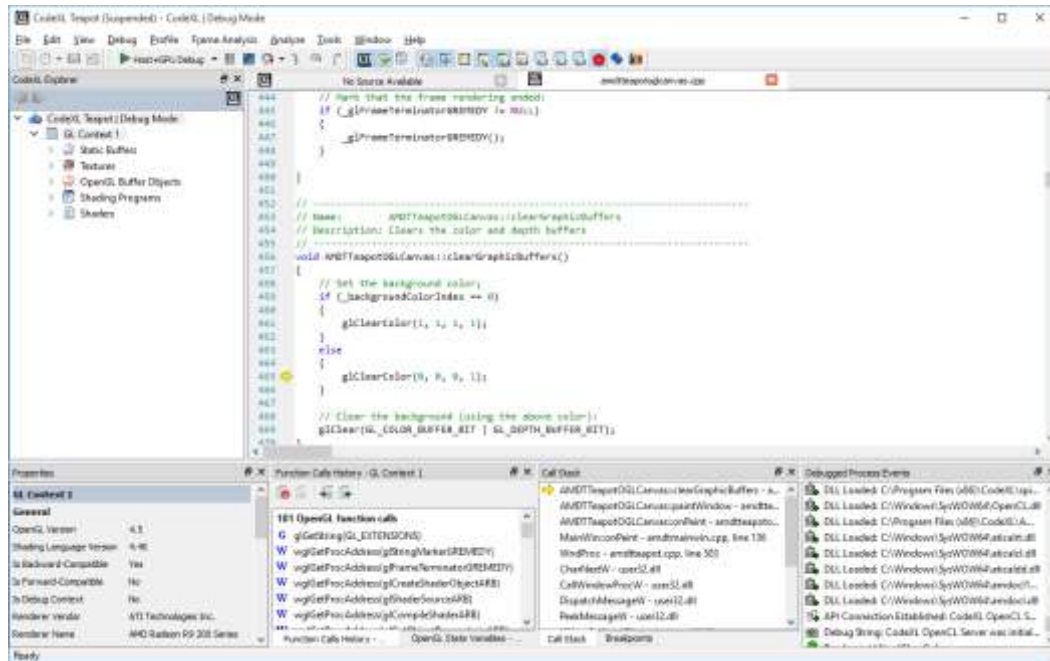
These buttons are (left to right): CodeXL Explorer, Properties, Function Calls History, Debugged Process Events, Call Stack, Locals, Watch, OpenGL™ State Variables, OpenCL Multi-Watch (1,2,3), Breakpoints, Memory, and Statistics.

You can resize views, drag, and drop views to rearrange them, or move them to a separate window. The next sections of this guide describe individual CodeXL views in more detail.

### Source Code View

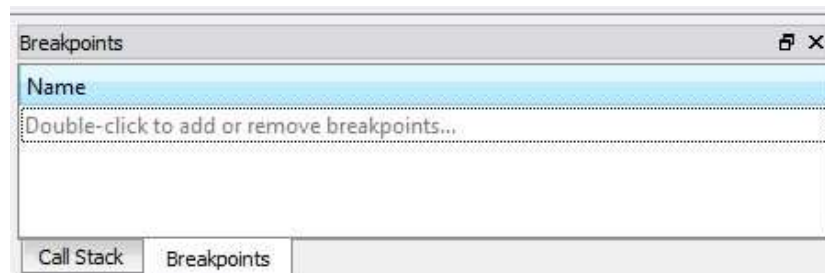
Source Code views display C, C++, or OpenCL code. To display the Source Code view:

1. Start the teapot program, as described above.
2. Hit the Break button  to interrupt it.  
A Source Code view displays the source file where the break occurred, with a yellow arrow  indicating the current line number. In the following screenshot, it is line 431 in the `amdtteapottoglcanvas.cpp` file.
3. In all configurations, you can now step to the next API call, OpenGL draw function call, or Frame Terminator function call by choosing the appropriate command from the API steps drop-down button (your last choice will be remembered for quick access).
4. In configurations that allow Host debugging (see note above), you can also step through the host code with the step in, step over and step out commands.



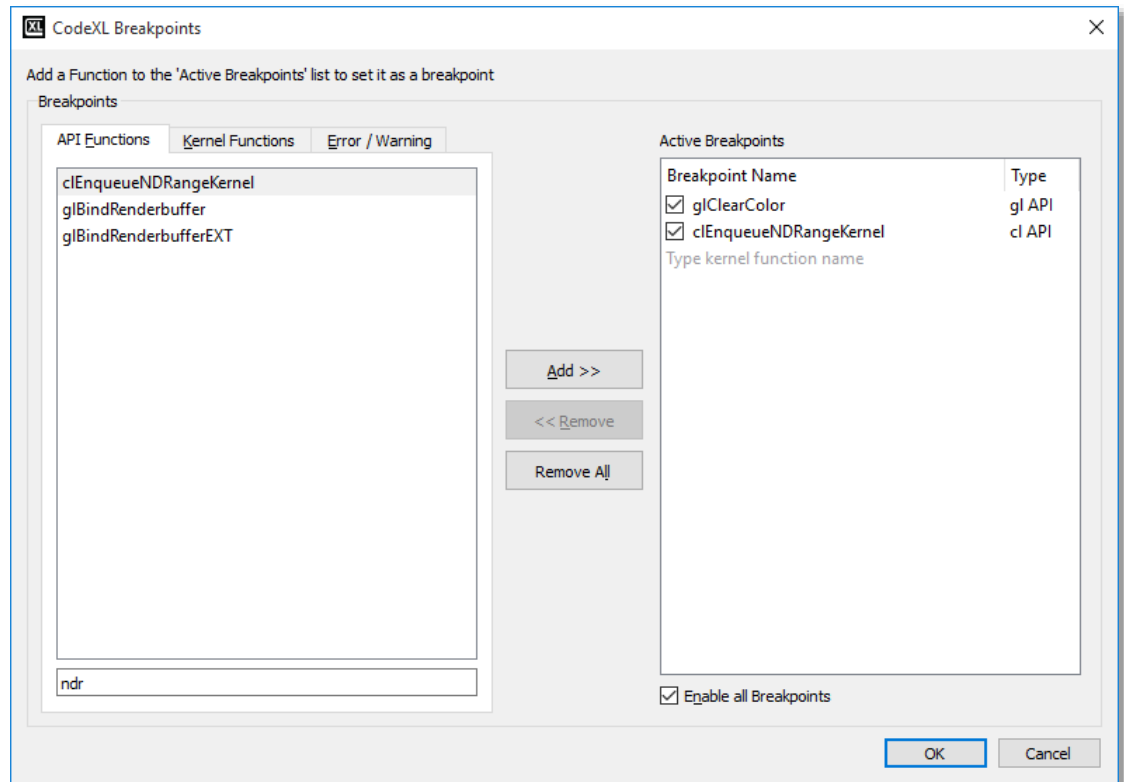
### Breakpoint View

The Breakpoint view shows active breakpoints. Initially, the Breakpoint view shows no breakpoints:



To add a breakpoint:

1. Double-click “Double-click to add or remove breakpoints...”  
A new Breakpoints window appears.

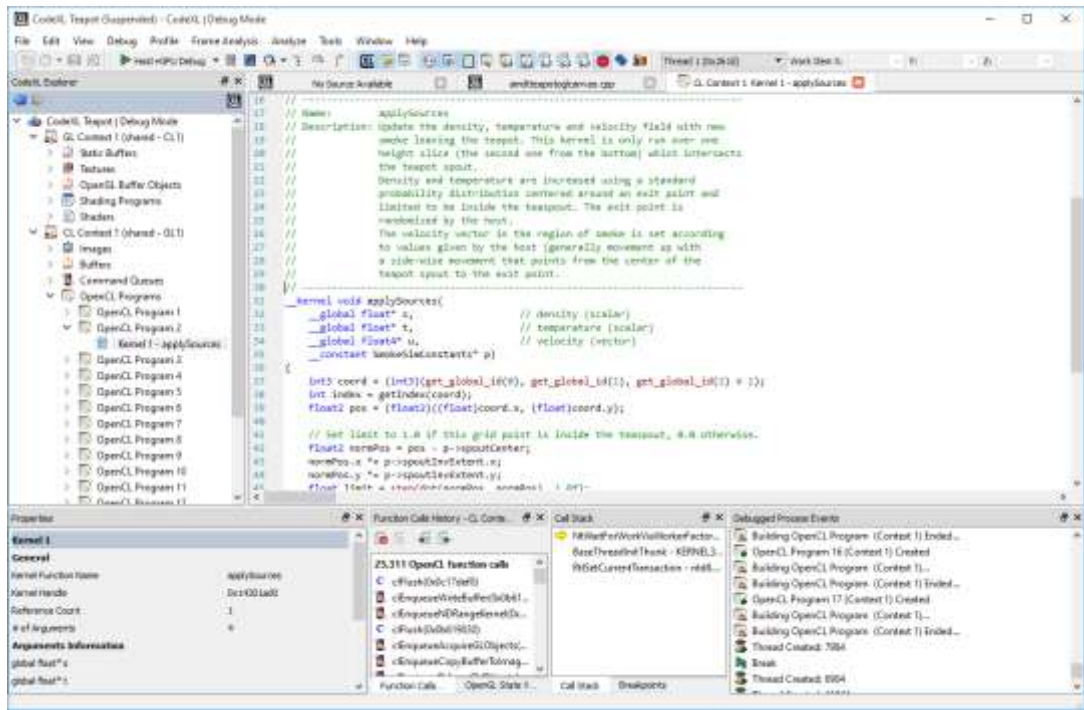


2. Select the API Functions tab to set a breakpoint on an API function, or select the Kernel Functions tab to set a breakpoint on a kernel function.  
When program execution hits a breakpoint, the Source view displays the line where the breakpoint occurs. A yellow arrow indicates the current location.  
A red dot next to the line number indicates a set breakpoint.

## Watch and Locals Views

### In kernel debugging mode

The Watch view shows the values and types of program variables you specify.  
The Locals view displays the values and types of local variables in a kernel.



In the image above, the Watch view displays the value of variable **normPos**. The Locals view displays the values of all local variables in the current kernel (in this case, **applySources** in **tpApplySources.c1**). For a structured variable, click on the triangle to the left of the variable name to see the name and value of each member.

When CodeXL is in Kernel Debugging mode, move the mouse cursor to hover over any variable name in the OpenCL kernel source code to display a tooltip with the variable value. This is demonstrated in the screenshot below.



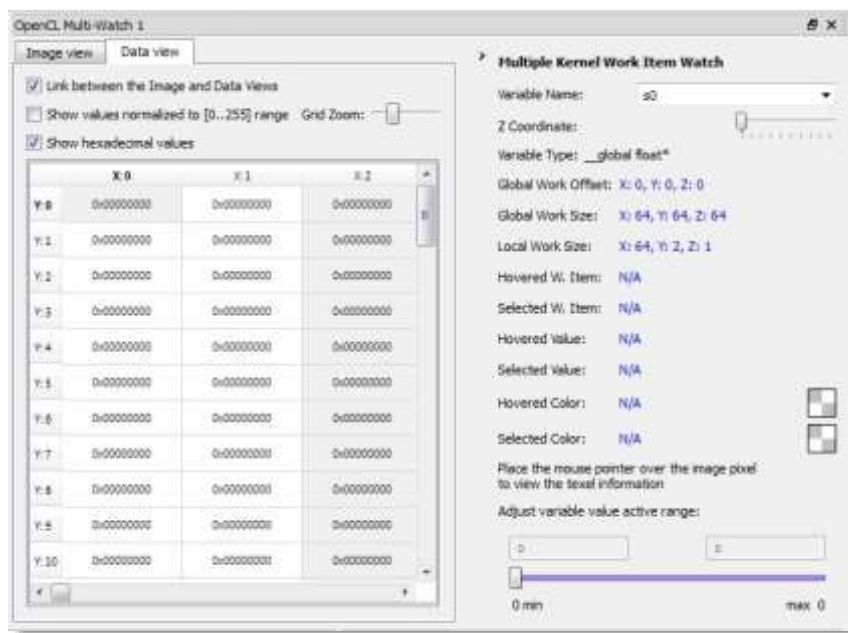


### In Host debugging mode

In configurations where host debugging is supported (see note above), the Watch and Locals views will show the currently selected host thread call stack frame's variables. You can even view host variables during kernel debugging, by switching to a host thread in the Kernel Work Item toolbar's Threads and Wavefronts combobox.

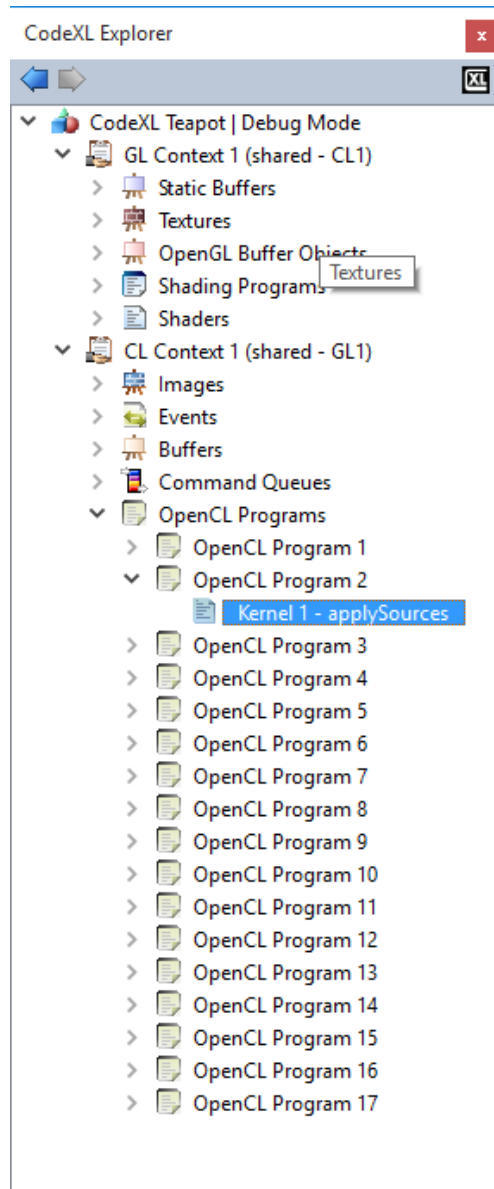
### Multi-Watch view

The Multi-Watch view lets you compare the values of an OpenCL kernel variable across work items and work groups.



### Explorer View

The Explorer view displays OpenCL-allocated objects and OpenCL/OpenGL shared contexts.

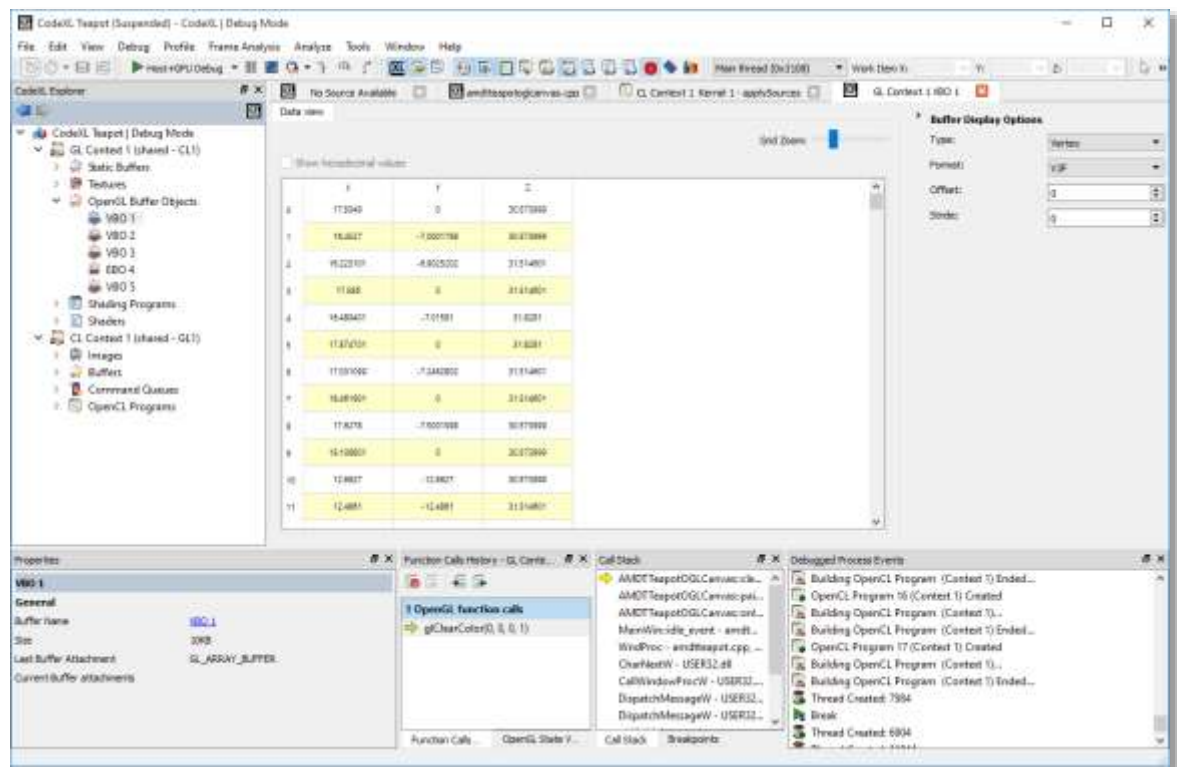


1. Click on an object to bring up information about the object in the Properties view.  
For example, clicking on Texture 2 in the view above brings up its properties, as shown in the next screenshot.

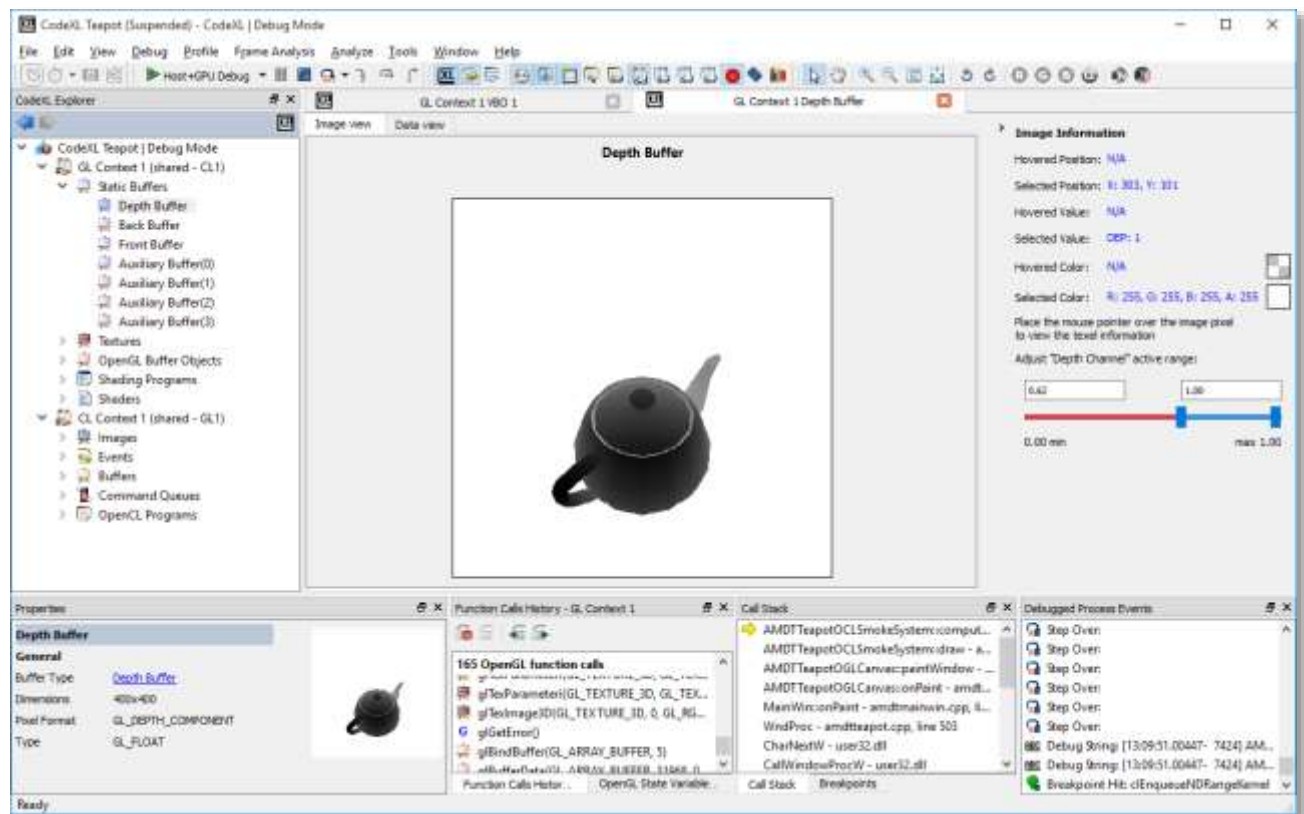




- Click on Vertex Buffer object VBO 1 to display its data, with a variety of available drop-down menu display and format options in the right-hand panel.



- Double-click on an object to display an appropriate view. For example, double-click on Vertex Shader 1 under Shaders to bring up a Source Code view of its source file `tpVertexSharder.glsl`. Alternatively, double-click on Depth buffer to bring up an Image view of the depth buffer.



You can manipulate an Image view with the following image manipulation buttons on the CodeXL toolbar:

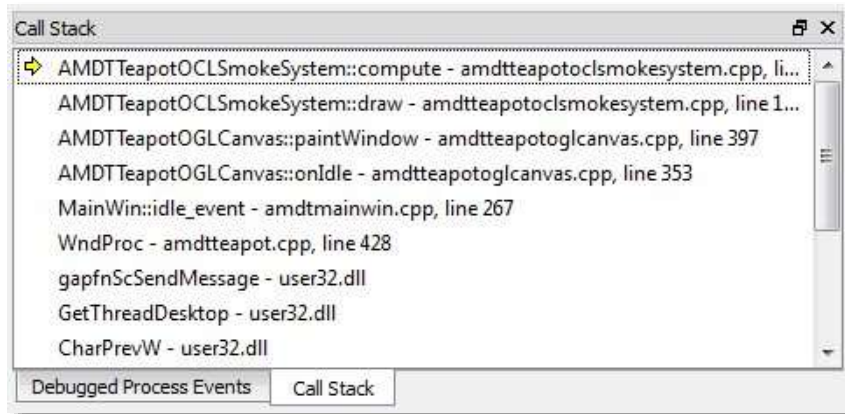


These buttons let you select, zoom in, zoom out, pan, enable R/G/B/alpha channels, enable grayscale mode, enable color invert mode, original size, best fit, and rotate CCW/CW. Hovering over the image displays pixel-specific information (position and color) in the Image Information panel.

Alternatively, select the Data view tab of the depth buffer to display the buffer as raw spreadsheet data rather than as an image.

### Call Stack View

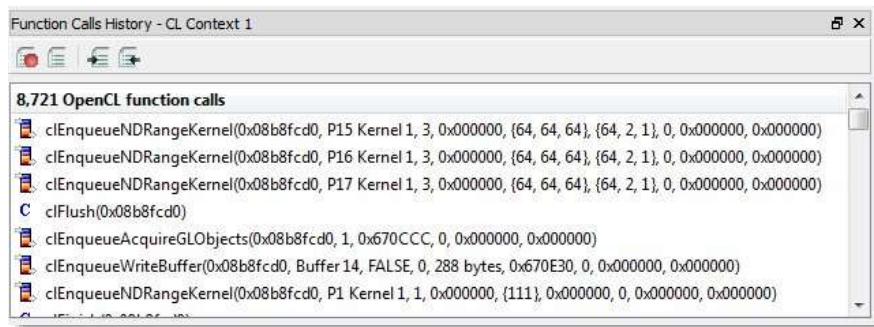
The Call Stack view displays a combined C/C++/OpenCL call stack.



### Function Calls History View

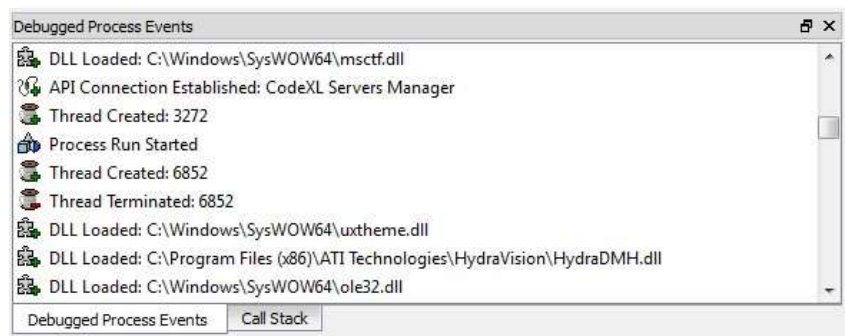
The Function Calls History view displays a log of OpenCL API calls.

1. Click on a function call to display call details in a Properties view.



### Debugged Process Events View

The Debugged Process Events view displays process events.





## Memory View

The Memory view summarizes memory use.

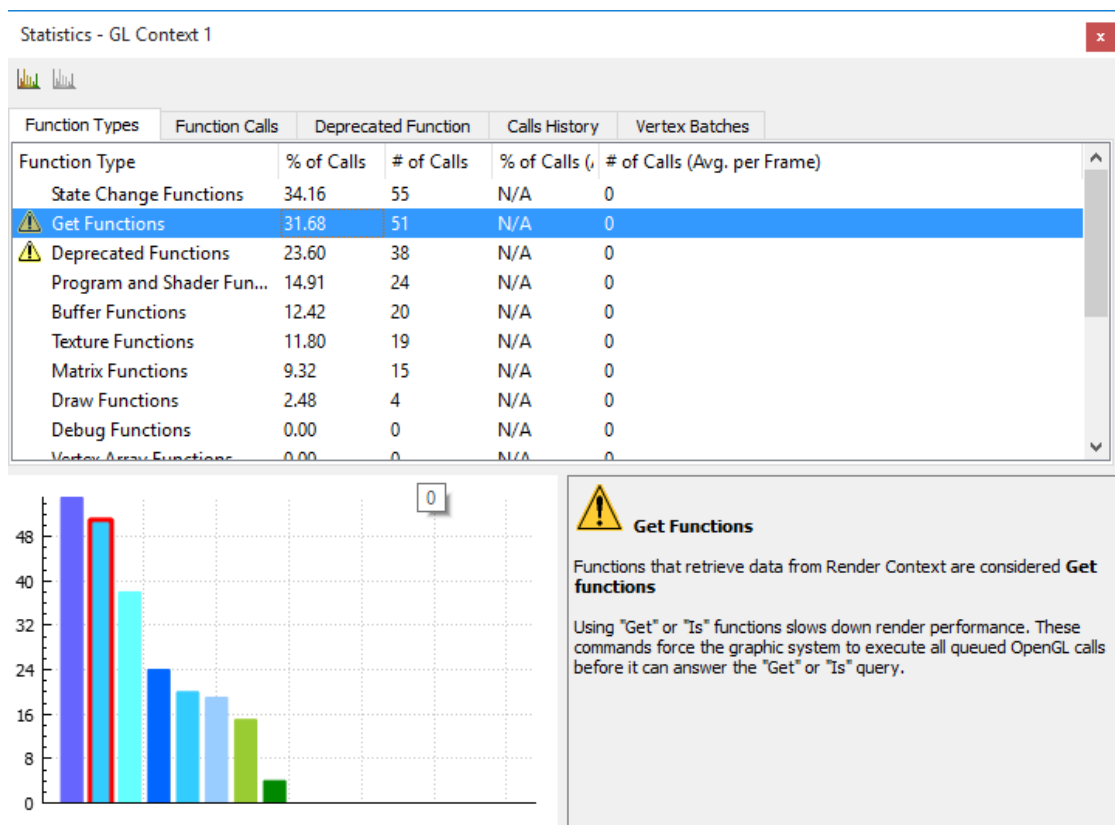
Object Type	Memory Size	# of Objects
Shading Programs	Insignificant	1
Shaders	11 KB	3
Static Buffers	3,596 KB	8
Textures	5,462 KB	2
VBOs	2,084 KB	5
<b>Total</b>	<b>11,153 KB</b>	<b>19</b>

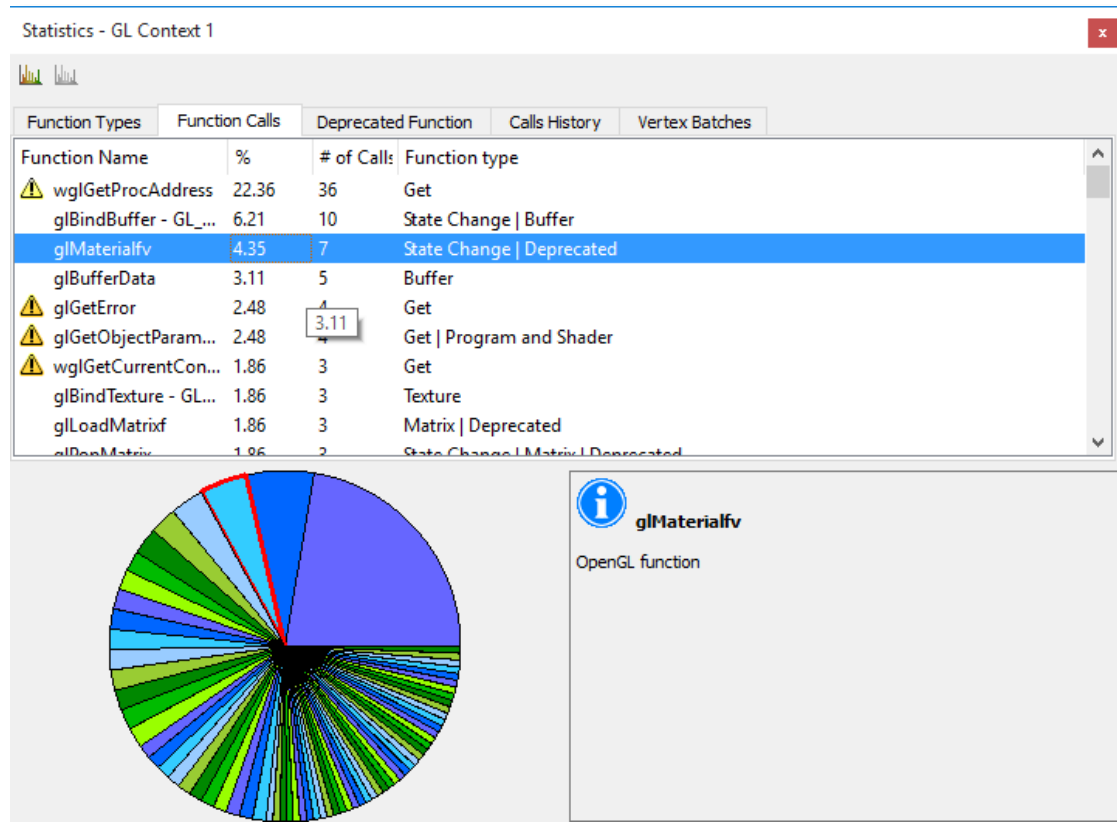
MainWin::EnableOpenGL - amdtmainwin.cpp, li...
MainWin::init - amdtmainwin.cpp, line 98
wWinMain - amdtteapot.cpp, line 220
_tmainCRTStartup - crtexe.c, line 547
BaseThreadInitThunk - kernel32.dll
RtlInitializeExceptionChain - ntdll.dll

## Statistics View

The Statistics view provides statistical information about the program. Select a tab to choose among options, such as Function Types:



or Function Calls:



## Matrix Multiplication Project

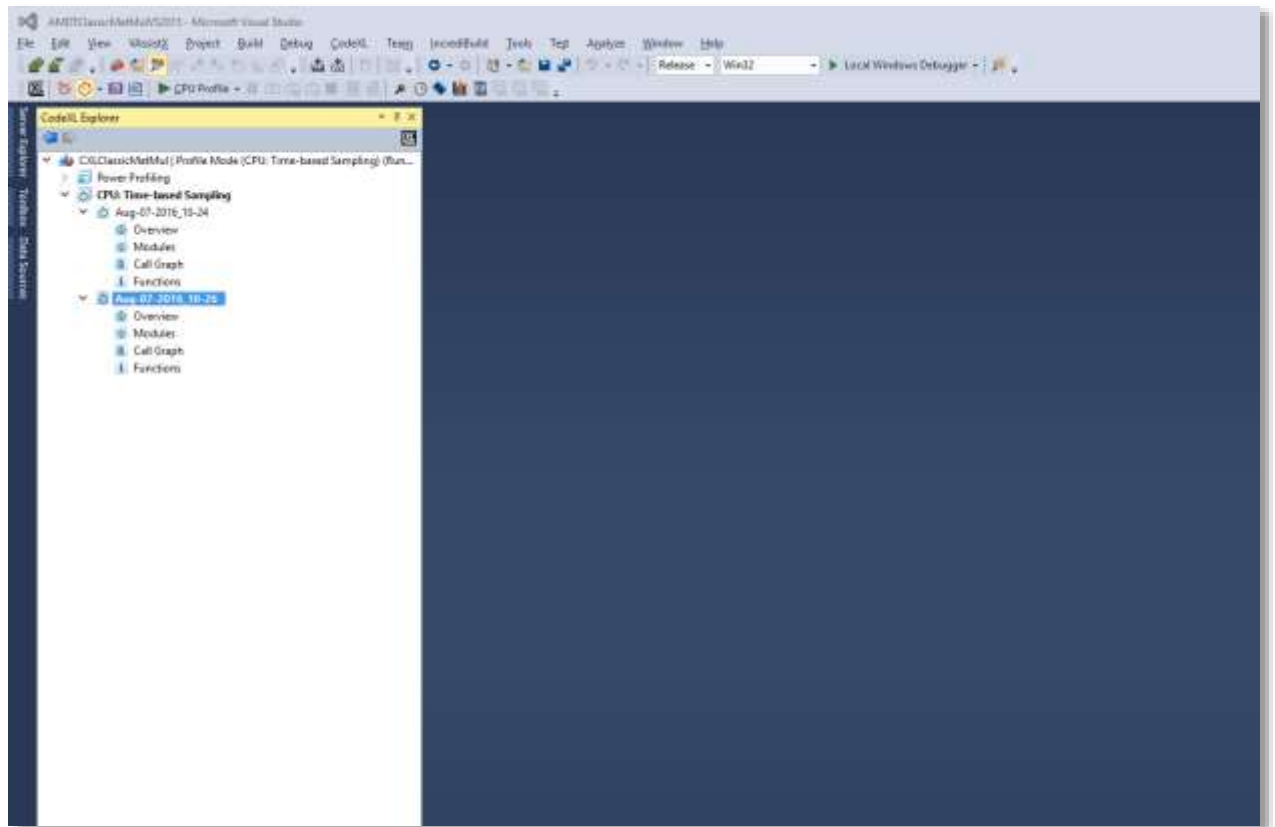
The CodeXL distribution includes a CPU Profile sample. The sample includes 3 functions that implement matrix multiplication.

Use command line arguments to call each of the 3 implementations:

- Running the sample **without any argument** will invoke inefficient implementation of matrix multiplication.
- Running the sample with **-c** will invoke the classic textbook implementation of matrix multiplication.
- Running the sample with **-i** will invoke improved implementation of matrix multiplication.

For the Visual Studio extension:

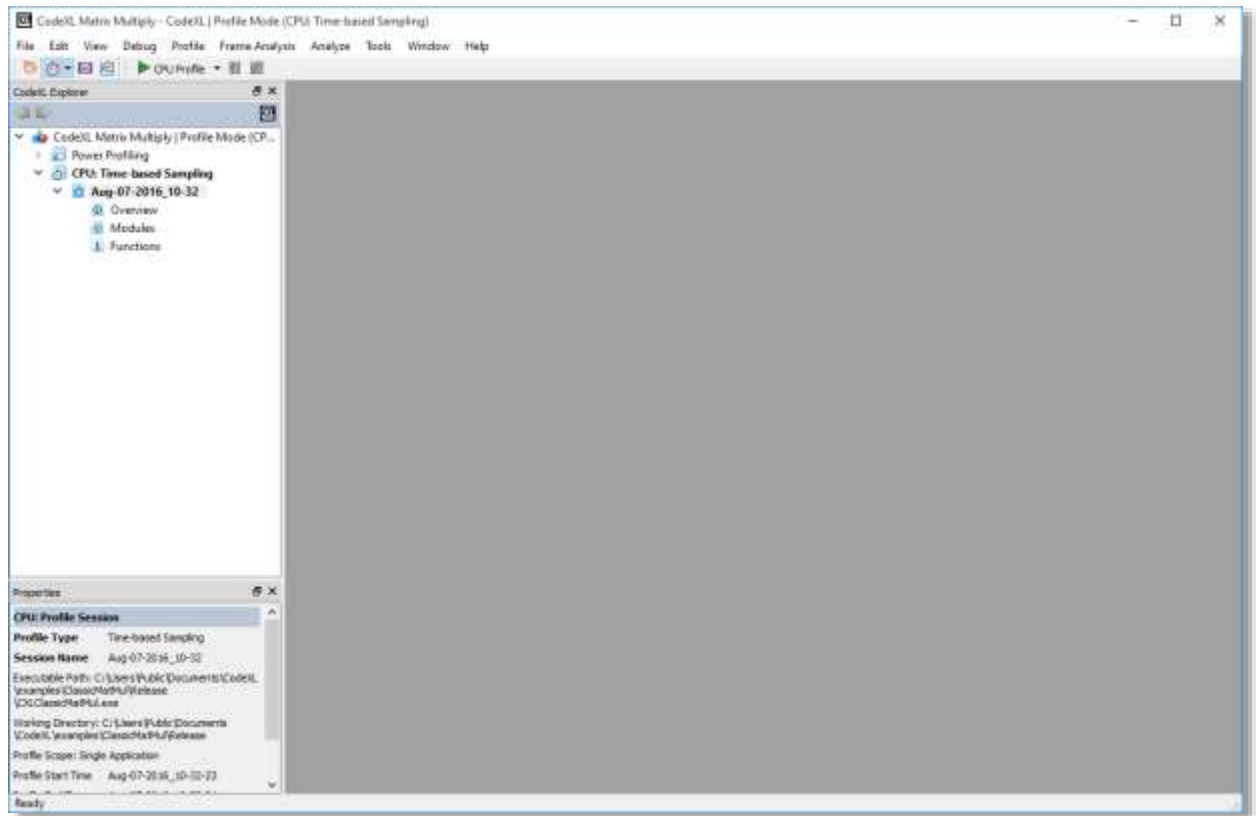
1. Select CodeXL >> Open Matrix Multiplication Sample Project from the VS menu. Visual Studio displays the matrix multiply sample project.



Screenshots in the remainder of this document show the standalone version of CodeXL. The Visual Studio version is similar, but contains a VS window rather than a CodeXL window.

For Windows or Linux:

2. In the CodeXL welcome page (in the CodeXL menu bar, click on File->Welcome Page), Under the Samples header, click the [CodeXL Matrix Multiply](#) link.



The CodeXL Explorer view now shows:

CodeXL Matrix Multiply | Profile Mode (CPU:Time-Based Sampling) - Not running

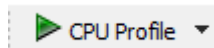
### Perform CPU Profile for the Matrix Multiply Sample Application

**Note:** Before profiling the matrix multiply sample application, you must load it (see the previous section).

After the sample is loaded, run a Time-Based profile session for the sample:

1. Select CodeXL >> Profile >> Time-Based Sampling to switch to Time Based CPU Profile mode.
2. Start Profile the inefficient implementation:

Click CodeXL -> Start Profiling or Click on the green right arrow



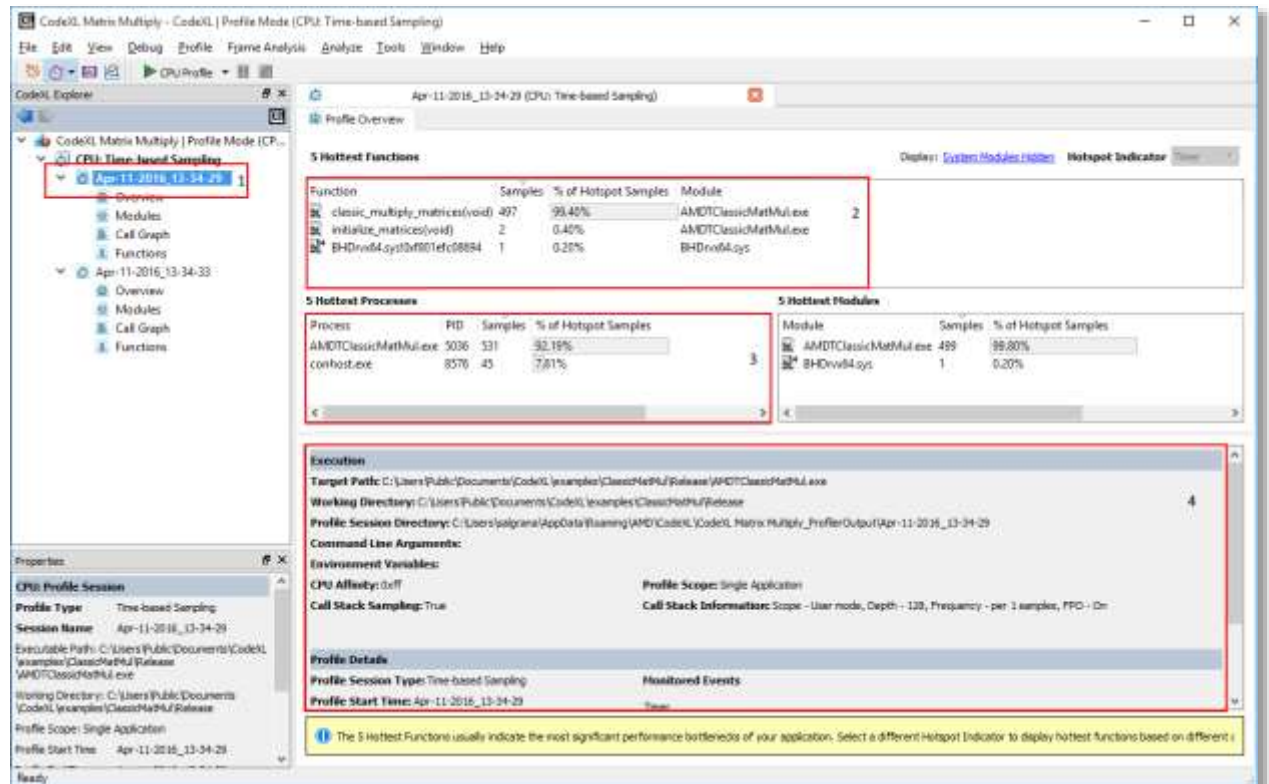
taskbar. (Local means local host profiling. Use the right black arrow to configure remote host settings)

The program begins execution, and soon displays a command line window that will run the matrix multiplication executable. Once the sample execution completes, CodeXL will open a CPU Profile session overview that displays the profile results.

### CPU Time Based Profile Navigation

After the execution is complete, CodeXL display a profile session overview window.



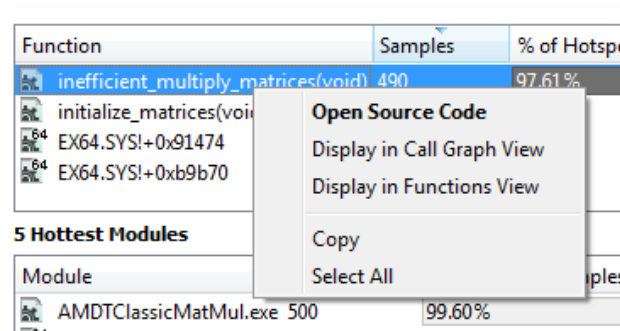


This screenshot displays the session overview window. See marked red rectangles:

1. **CodeXL Explorer** - The current profile session selected in CodeXL explorer. Double click on this node in the tree will open the session after it is closed.
2. **Functions view** - displays the 5 most sampled functions. See that the function “inefficient\_multiply\_matrices” was sampled 490 times.
3. **Modules view** - displays the 5 most sampled modules. In this example this table is not useful. Use it to find inefficient modules in multiple modules executables.
4. **Profile Overview** - displays general information of the session. Executable path, working directory, etc’.

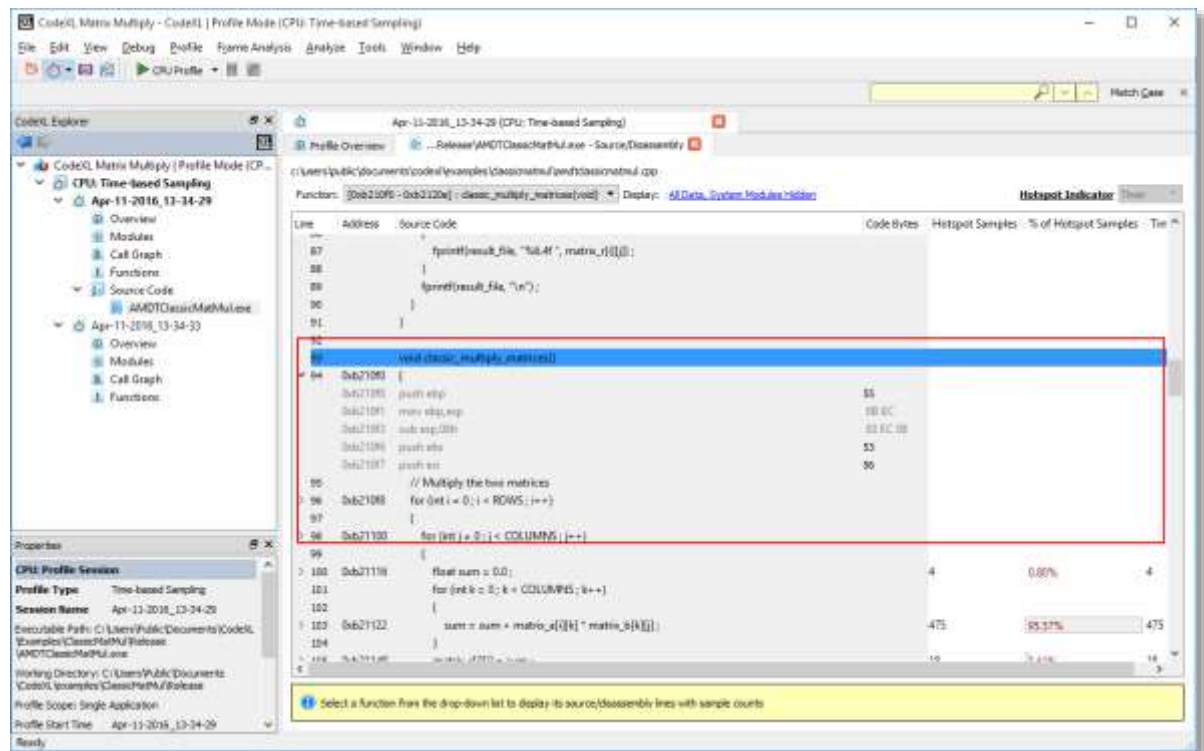
### Source Code View

While looking at the overview of this session, we can see that the function “inefficient\_multiply\_matrices” is consuming resources. Right click on this function, and click “**Open Source Code**”.





Clicking this will open the source code view for the file containing “inefficient\_multiply\_matrices”.



The source code view display a line-by-line performance table for the requested function. In this sample, we can see, marked in red in the above screenshot, that **line 126** had 100% of the samples for this function. Looking at the marked comment we can see that the function is called 3 times, which is redundant.

To call the “classic\_multiply\_matrices” only once, we will change the command line arguments.

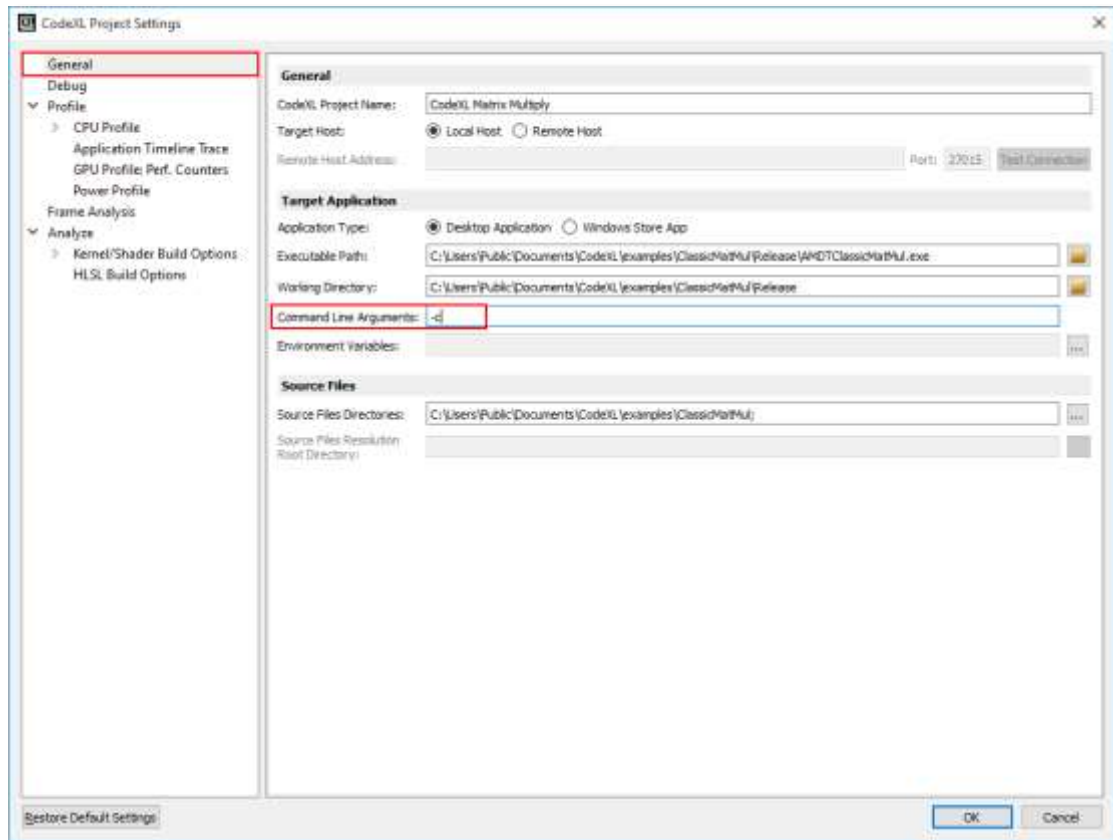
## Run the classic textbook sample

In order to call “classic\_multiply\_matrices” in this sample, user should give “-c” command line argument. In order to configure the command line argument sent for the session, open project session window.

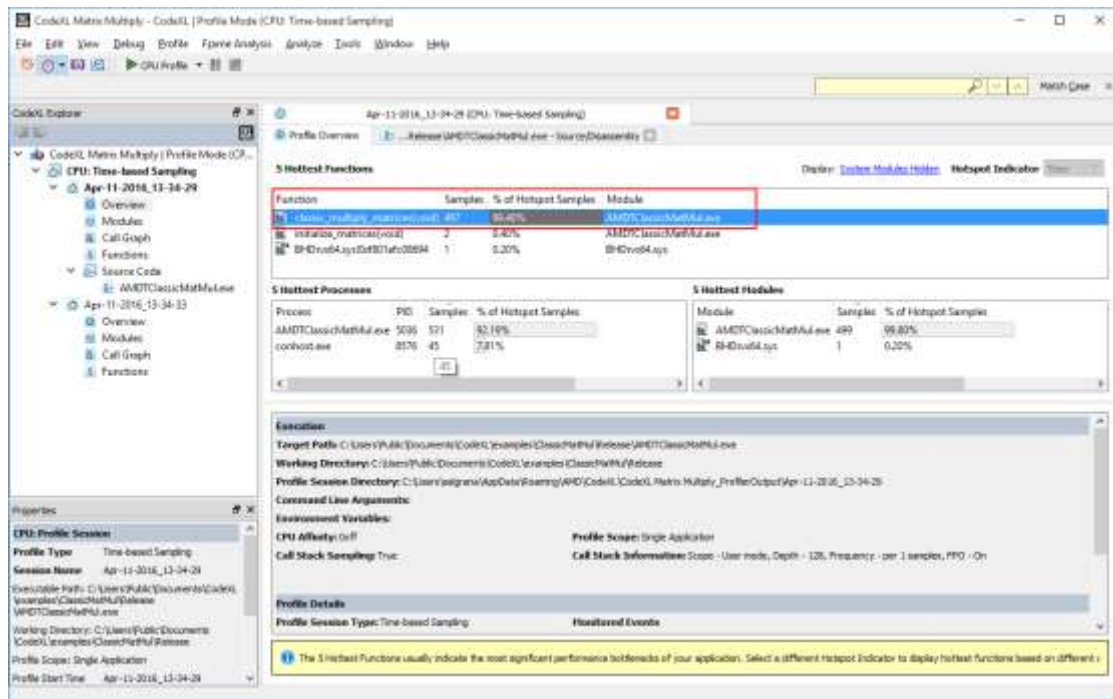
1. Click File->Project Settings
2. Go to the "General" page
3. Click "-c" in the command line argument text box
4. Click OK.
5. Run the profile session again (click Profile -> Start Profile)



## Getting Started with CodeXL



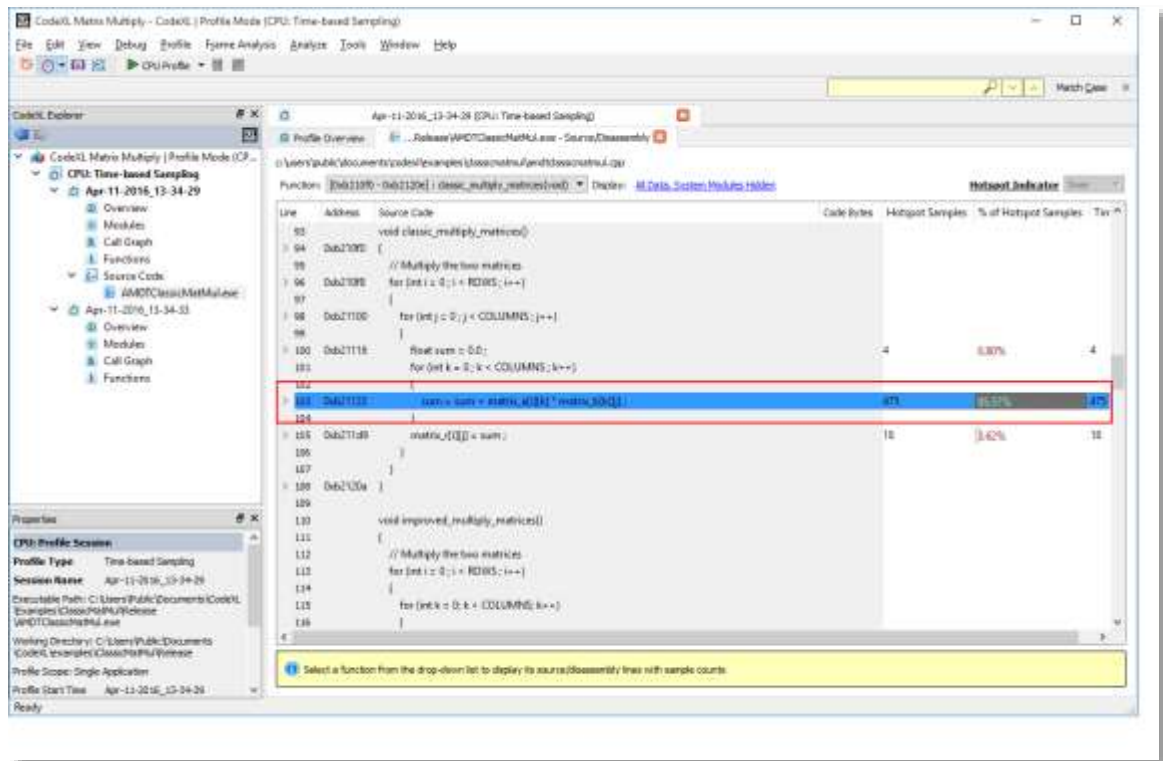
## Analyzing the classic implementation





## Getting Started with CodeXL

Looking at the session again, we can see that “classic\_multiply\_matrices” was sampled 497 times (which is expected, since we call it once, instead of three times). Right click on “classic\_multiply\_matrices”, and click “Open Source Code” to open the source code again for further analysis.



The source code shows the following line was sampled 475 times.

```
sum = sum + matrix_a[row][inCol] * matrix_b[inCol][outCol];
```

In order to look for ideas how to improve this line, we can run an Assess Performance session, to see how our sample is consuming the current system resources.

1. Click on Profile -> CPU: Assess Performance to select this profile type.
2. Click on Profile -> Start Profile to run an assess performance session.
3. Look at the displayed session.
4. Right click on “classic\_multiply\_matrices” to see a line-by-line display of the system resources.



## Getting Started with CodeXL

CodeXL: Matrix Multiply - CodeXL | Profile Mode (CPU: Assess Performance)

File Edit View Debug Profile Frame Analysis Analyze Tools Window Help

CodeXL Explorer

Apr-11-2016\_13-34-29 (CPU: Time-based Sampling) Apr-11-2016\_13-41-48 (CPU: Assess Performance)

Profile Overview

5 Hottest Functions

Function	Samples	% of Hotspot Samples	Module
classic_multiply_matrices(void)	1,027	99.99%	AMDTClassicMatMul.exe
improved_multiply_matrices(void)	12	0.54%	AMDTClassicMatMul.exe
BHDrv64.sys!0a01efbf223e	1	0.09%	BHDrv64.sys
BHDrv64.sys!0a01efbf223e	1	0.09%	BHDrv64.sys

5 Hottest Processes

Process	PID	Samples	% of Hotspot Samples
AMDTClassicMatMul.exe	2344	1,368	89.87%
conhost.exe	7998	154	10.13%

5 Hottest Modules

Module	Samples	% of Hotspot Samples
AMDTClassicMatMul.exe	1,099	99.92%
BHDrv64.sys	2	0.18%

Execution

Target Path: C:\Users\Public\Documents\CodeXL\Examples\ClassMatMul\Release\AMDTClassicMatMul.exe

Working Directory: C:\Users\Public\Documents\CodeXL\Examples\ClassMatMul\Release

Profile Session Directory: C:\Users\Public\Documents\CodeXL\Examples\ClassMatMul\Release\AMDTClassicMatMul\_ProfileOutput\Apr-11-2016\_13-41-48

Command Line Arguments: <

Environment Variables:

CPU Affinity: 0x0

Profile Scope: Single Application

Call Stack Sampling: True

Call Stack Information: Scope - User mode, Depth - 32, Frequency - per 1 samples, PPO - Off

Profile Details

Profile Session Type: Assess Performance

Monitored Events

The 5 Hottest Functions usually indicate the most significant performance bottlenecks of your application. Select a different hotspot indicator to display hottest functions based on different:

CodeXL: Matrix Multiply - CodeXL | Profile Mode (CPU: Assess Performance)

File Edit View Debug Profile Frame Analysis Analyze Tools Window Help

CodeXL Explorer

Apr-11-2016\_13-34-29 (CPU: Time-based Sampling) Apr-11-2016\_13-41-48 (CPU: Assess Performance)

Profile Overview Release\AMDTClassicMatMul.exe - Source\Disassembly

c:\users\public\documents\codexl\examples\classmatmul\release\amdtklassicmatmul.exe

Functions: [0x21100 : 0x21100] : classic\_multiply\_matrices(void) Display: All Data, System Modules Hidden

Hotspot Indicator Data Cache Accesses

Line	Address	Source Code	Code Type	Hotspot Samples	% of Hotspot Samples	DC access
95	0x21093	// Multiply the two matrices				
96	0x21093	for (int i = 0; i < ROWS; i++)				
97		{				
98	0x21100	for (int j = 0; j < COLUMNS; j++)				
99		{				
100	0x21116	float sum = 0.0;			0.10%	
101		for (int k = 0; k < COLUMNS; k++)				
102		{				
103	0x21122	sum = sum + matrix_a[i][k] * matrix_b[k][j];		1,026	99.99%	1,026
104		}				
105	0x211d5	matrix_c[i][j] = sum;				
106		}				
107		}				
108	0x2120e	}				
109		void improved_multiply_matrices()				
110		{				
111		// Multiply the two matrices				
112		for (int i = 0; i < ROWS; i++)				
113		{				
114		for (int k = 0; k < COLUMNS; k++)				
115		{				
116		for (int j = 0; j < COLUMNS; j++)				
117		{				
118		}				

Select a function from the drop down list to display its source/assembly lines with sample counts

In the source code view, look at the values for each of the counters for the specified line. We can see that we have many data cache misses, which might cause a performance bottle neck.



## Analyzing the improved implementation

The third sample implementation is called “improved\_matrix\_multiplication”.

This implementation is only slightly different from the classic one. We change this line:

```
sum = sum + matrix_a[row][inCol] * matrix_b[inCol][outCol];
```

to this line:

```
matrix_r[row][outCol] = matrix_r[row][outCol] + matrix_a[row][inCol] *  
matrix_b[inCol][outCol];
```

This change is constructed of 2 changes:

1. We do not use “sum” as temporary variable for the summary accumulation.
2. We change the order of the summary operation. Instead of multiplying matrix A row by a matrix B column, we go over each element of A, and multiply by a whole row of B. Since the matrix is kept as a single line in the memory, we perform much less skips in the memory, therefore expect to have less Cache Misses.

To run the improved implementation:

1. Click File->Project Settings
2. Go to the “General” page
3. Click “-i” in the command line argument text box
4. Click OK.
5. Run the profile session again (click Profile -> Start Profile)

Now open the source code view again:





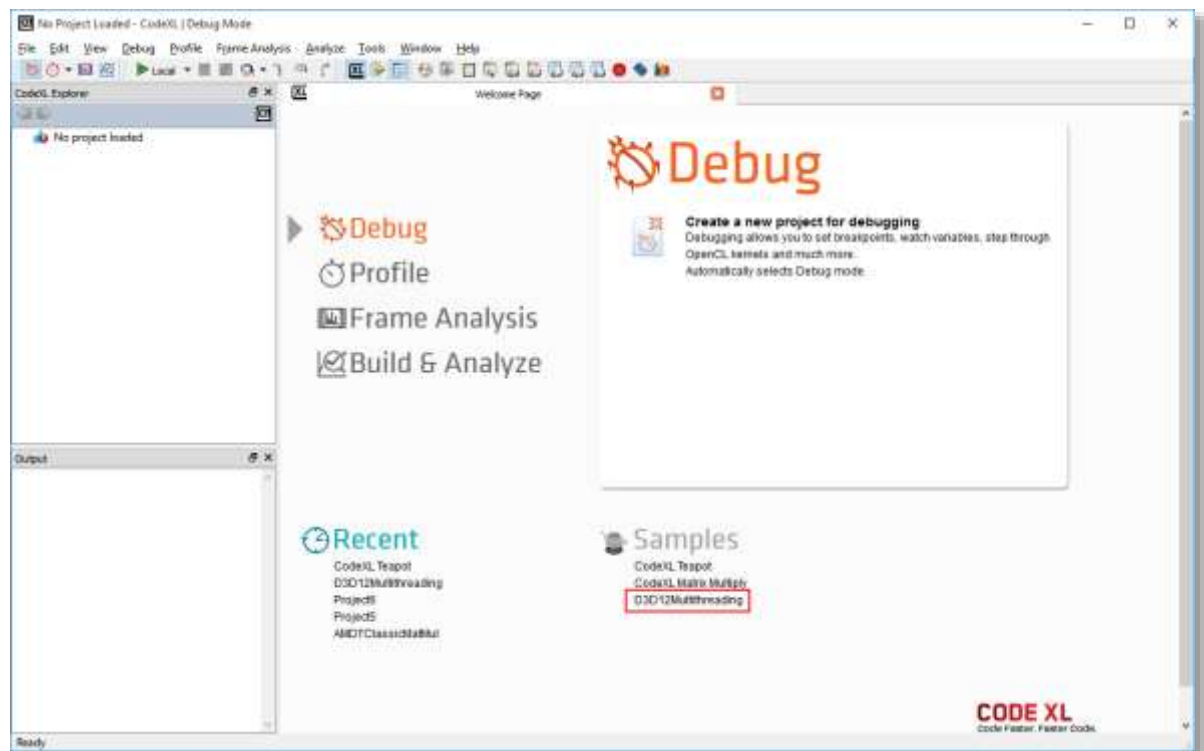
The improved function, for the same line, we only have 97 cache misses.

## D3DMultiThreading Project

### Open CodeXL D3DMultiThreading sample

The CodeXL distribution includes a DX12 Frame Analysis sample. This sample is taken from Microsoft DX12 SDK and can be used to try the Frame Analysis features in CodeXL.

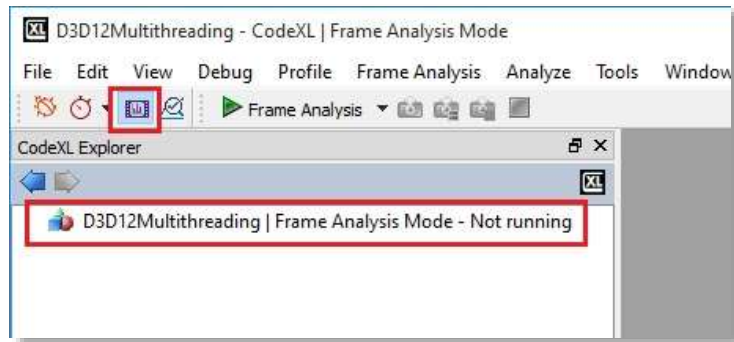
Start by clicking on the D3DMultiThreading sample link in CodeXL welcome page.



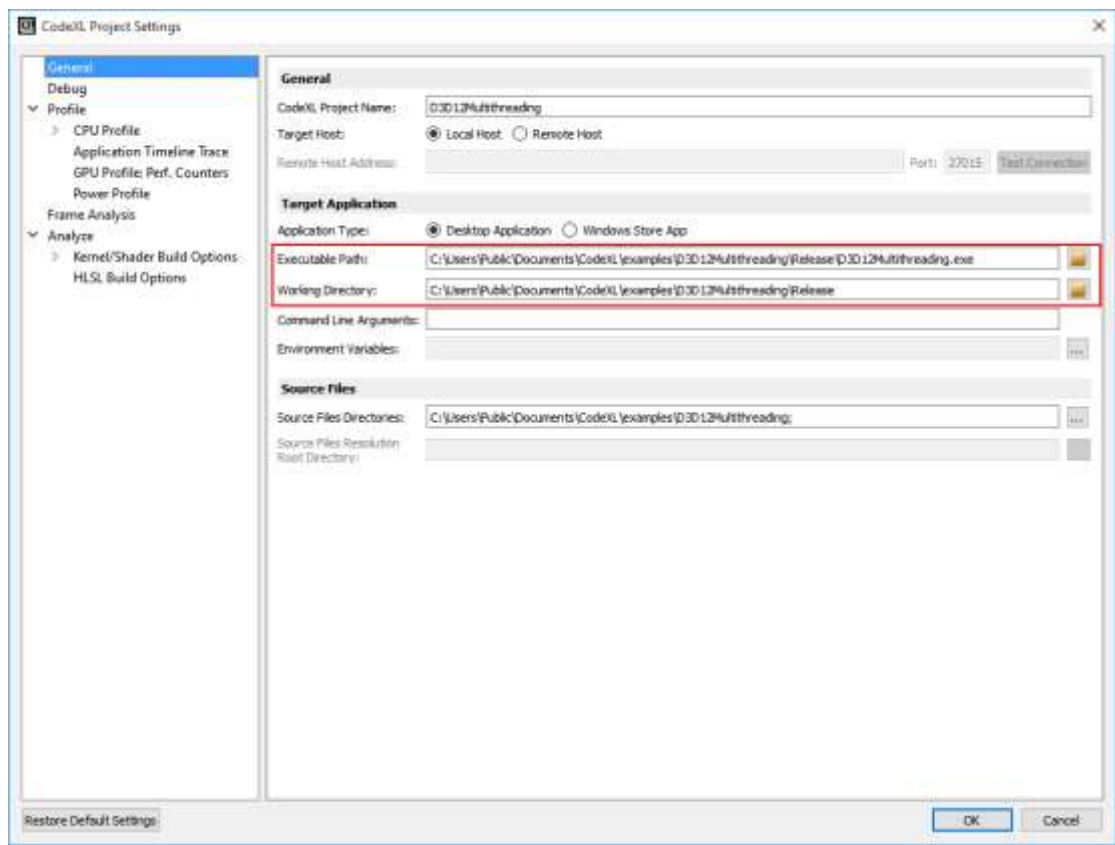
The sample project is loaded. You can see that the sample name in CodeXL Explorer. CodeXL will also select the Frame Analysis mode for this sample.



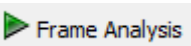
## Getting Started with CodeXL

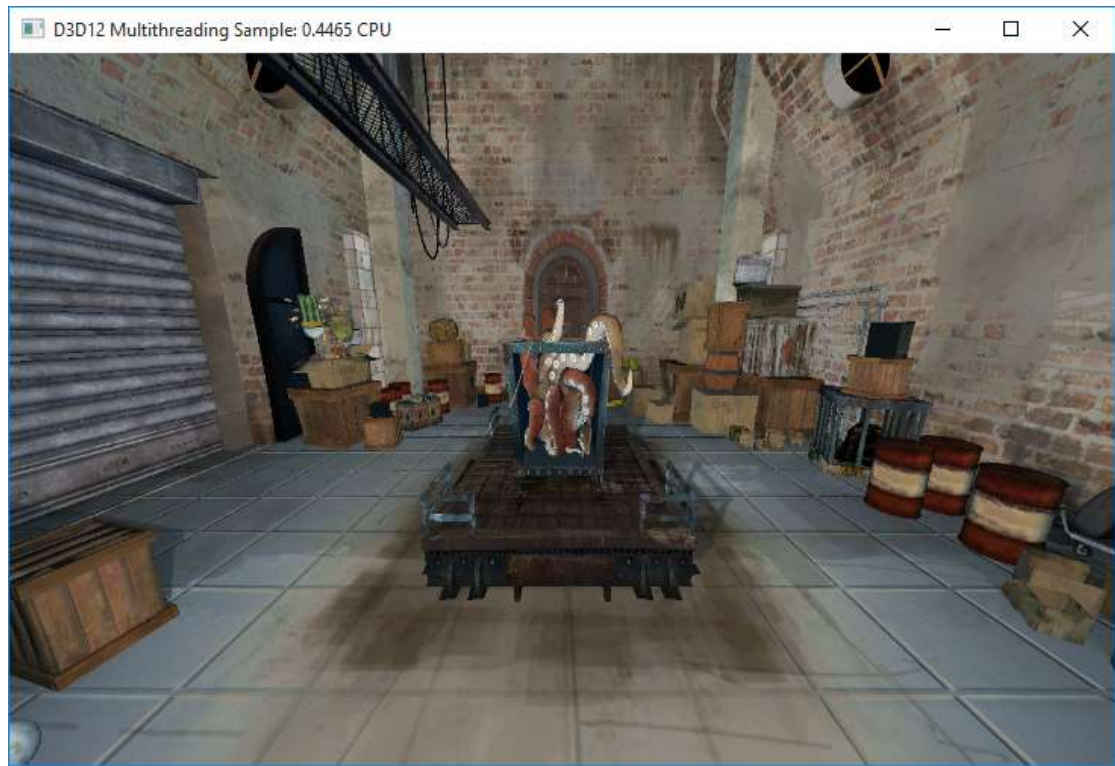


Click on the File->Project Settings command to see the project settings for the sample. The project Executable Path will point to CodeXL bundled D3DMultiThreading sample.

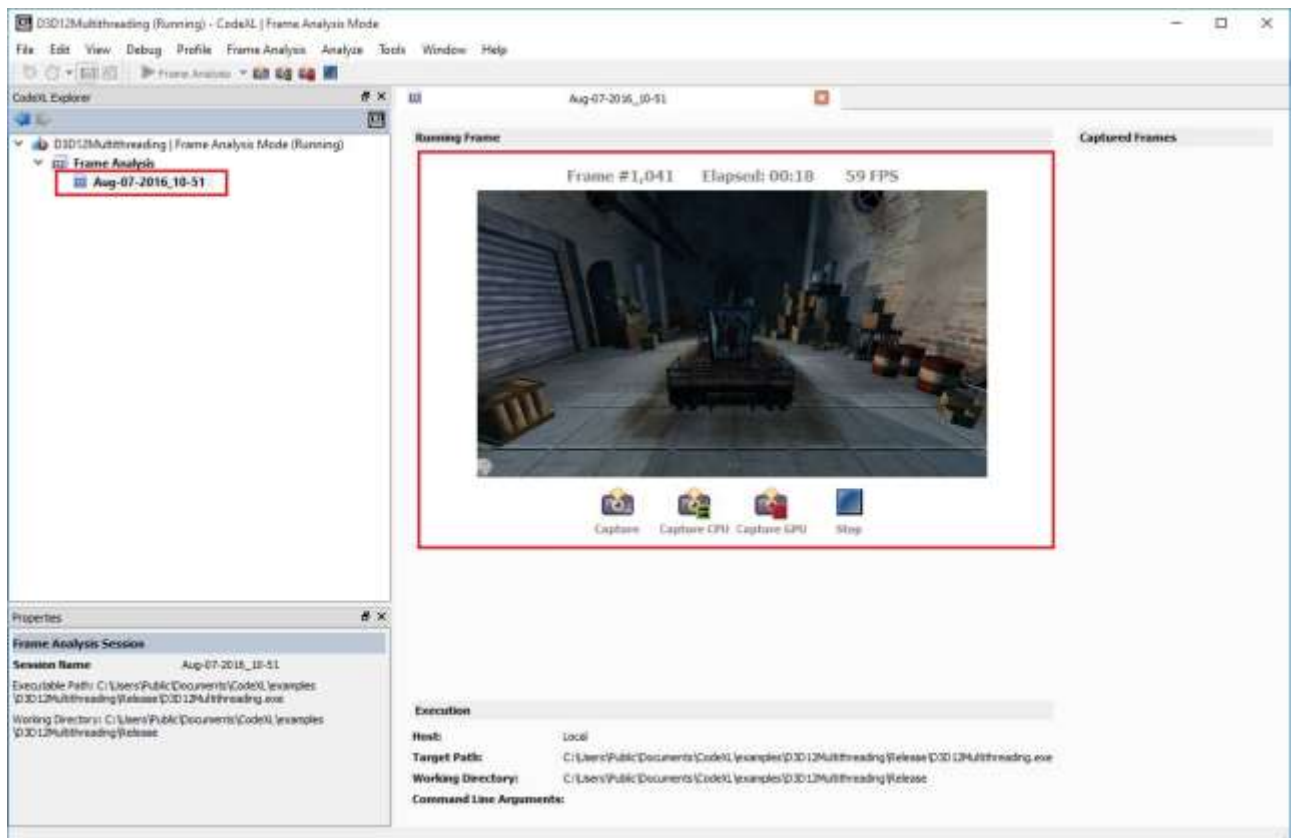


### Start a Frame Analysis session

Click the Frame Analysis  to execute a Frame Analysis session. A window with the D3DMultiThreading sample should open



- CodeXL will open a session window. The session window monitors the running application and will help you Capture the requested frames for later Frame Analysis.







## Getting Started with CodeXL

The screenshot above displays a session view, displaying the running D3DMultiThreading sample. The image in the center of the window displays a snapshot of the currently running frame.

The text above the frame image will display the following details for the current frame:

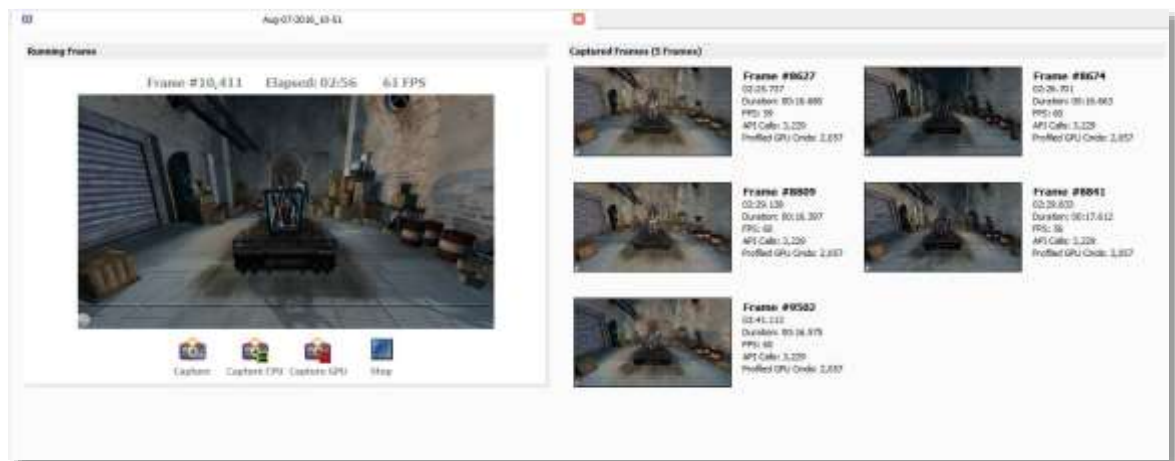
1. Frame index
2. Elapsed time since the application started
3. FPS

Below the frame image there are 2 buttons:

1. Capture – capture the current frame for later analysis
2. Stop – stop the session and start analyzing the captured frames.

### Capture frame for analysis

After clicking the “Capture” button, the current frame thumbnail and details will be added to the right panel. Each of the captured frame will be available after the session will stop for analysis.



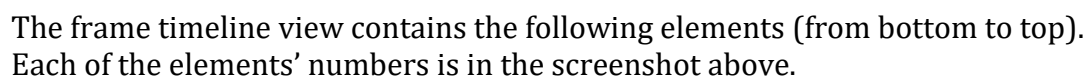
The above screenshot contains 5 captured frames.

Notice: double clicking on a frame should open it for detailed analysis. This will not be available while the session is running.

### Analyze a captured frame

- Stop the Frame Analysis session (click the stop button).
- Double click on one of the frames on the right panel.

A frame timeline will open with the detailed trace of the D3DMultiThreading



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.

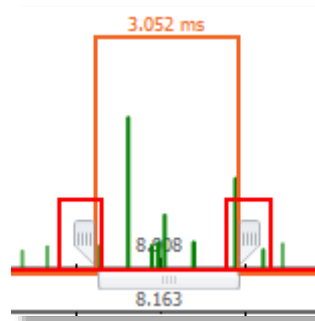


2. A collapsed detailed view of the API calls and their durations/count/concurrency.
3. A timeline chart visualizing the API calls over the frame timeline.
4. An API calls table for each of the CPU threads.
5. A commands table for each of the GPU command queues.
6. Summary tables that summarize the time 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.

## Navigating the frame timeline

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.

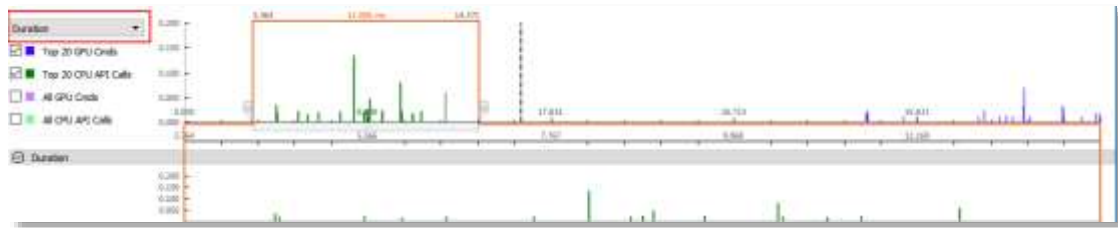
Use the left and right handles to expand/reduce the focused timeline fragment in and out:



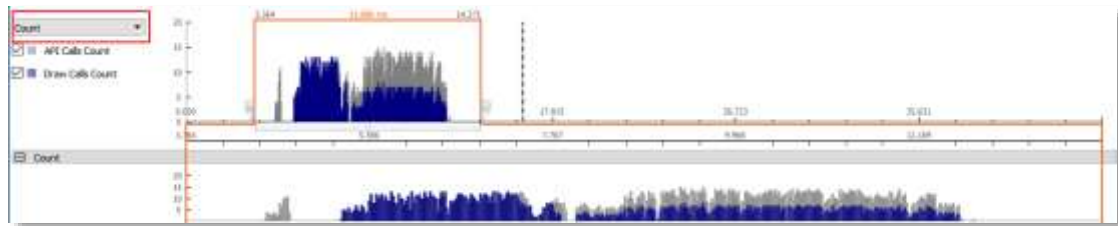
Double click a CPU API table item, and the timeline chart will zoom to the corresponding timeline item and highlight it. If there is a linked GPU API item, it will also be highlighted in the GPU API table



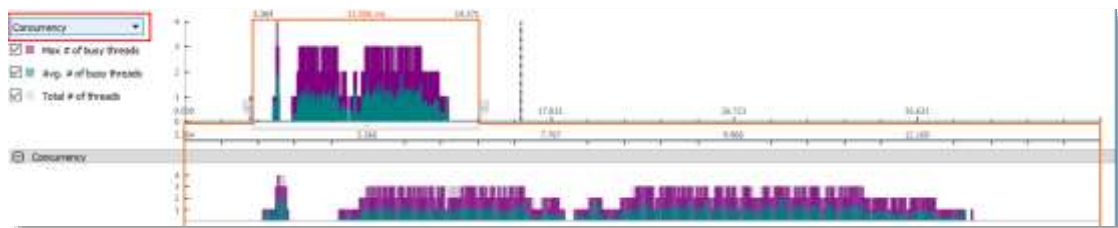
Select “Duration” in the top left combo box, to view the API call duration in details:



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



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



## Profile Mode

CodeXL profile mode is a powerful performance analysis tool that supports CPU and GPU profiling to provide program performance data. CodeXL profiling does not require modifications to your source code or project. Profiling does not require recompilation, except for CPU profiling, which requires compilation with debugging enabled. Profiling lets you find performance hotspots and issues, determine the top data transfer and kernel execution operations, and identify problems such as failed API calls and resource leaks. You can use profiling to improve application performance through proper synchronization, bottleneck elimination, and load balancing.

CodeXL provides several modes of profiling. These modes let you assess program performance, use instruction-based sampling (IBS) or time-based sampling (TBS), or investigate branching, data access, instruction access, or L2 cache access. GPU profiling provides application timeline trace and performance counter modes.



The following is a quick introduction to CPU and GPU profiling. For further details, see the CodeXL Help information.

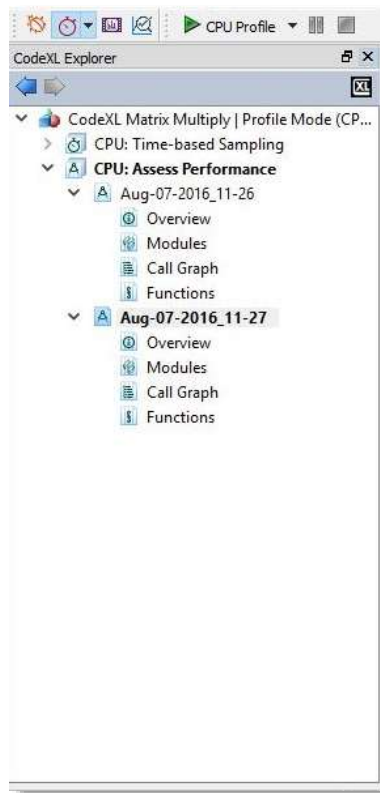
### CPU Profiling

To profile a program:

1. Click on the profiling mode taskbar button.
2. Use the Profile drop-down menu to select the profiling mode.  
For example, for CPU performance profiling, select Profile >> CPU: Assess Performance.
3. Click the start button to launch the application for profiling.
4. To stop it, use the stop button any time during profiling. The bottom of the CodeXL window displays the elapsed clock time.

Profiling is available up to the time the application is closed. For the teapot example: click on the 'x' in the upper right corner of the teapot window.

After profiling is complete and data translation is over, a node in the left session tree is added for this session.



### Overview Tab

The first page shown is the overview page. It shows the Modules and Functions tables and a brief description of the execution environment and profile detail. If multiple processes are profiled, then the Process table is shown. Each table shows the top five hot items.



**Profile Overview**

5 Hottest Functions

Display: [System Modules Hidden](#) Hotspot Indicator: Data cache accesses

Function	Samples	% of Hotspot Samples	Module
multiply_matrices(void)	4,081	97.14%	classic.exe
initialize_matrices(void)	58	1.38%	classic.exe
_getptd	36	0.86%	classic.exe
rand	26	0.62%	classic.exe

5 Hottest Modules

Module	Samples	% of Hotspot Samples
classic.exe	4,201	100.00%

**Execution**

Target Path: C:\CodeXL\classic.exe  
Working Directory: C:\CodeXL  
Data Folder: C:\Users\javadm\AppData\Roaming\WMI\CodeXL\classic\_Profiler\Output\Sep-02-2014\_04-30-43  
Command Line Arguments:  
Environment Variables:  
CPU Affinity: 0x3f  
Call Stack Sampling: True  
Profile Scope: Single Application  
Call Stack Information: Scope - User space, Depth - 32, Frequency - per 1 samples, PPO - Off

**Profile Details**

Profile Session Type: Assess Performance  
Profile Start Time: Sep-02-2014\_04-30-43  
Profile End Time: Sep-02-2014\_04-30-54  
Profile Duration: 10 seconds  
CPU Details: Family 0x10, Model 10, 6 core(s)  
Total Processes in Profile: 1  
Total Threads in Profile: 1

**Monitored Events**

Data cache accesses	Data cache misses	Misaligned accesses
CPU clock not halted (cycles)	Retired instructions	Retired branch instructions
Retired mispredicted branch instructions: L1 DTLB and L2 DTLB miss		

**Info** The 5 Hottest Functions usually indicate the most significant performance bottlenecks of your application. Select a different Hotspot Indicator to display hottest functions based on different criteria.

### Modules Tab

To open the Modules view:

1. Double click or use the “Open Modules” command from the context menu of the tree’s Modules node.





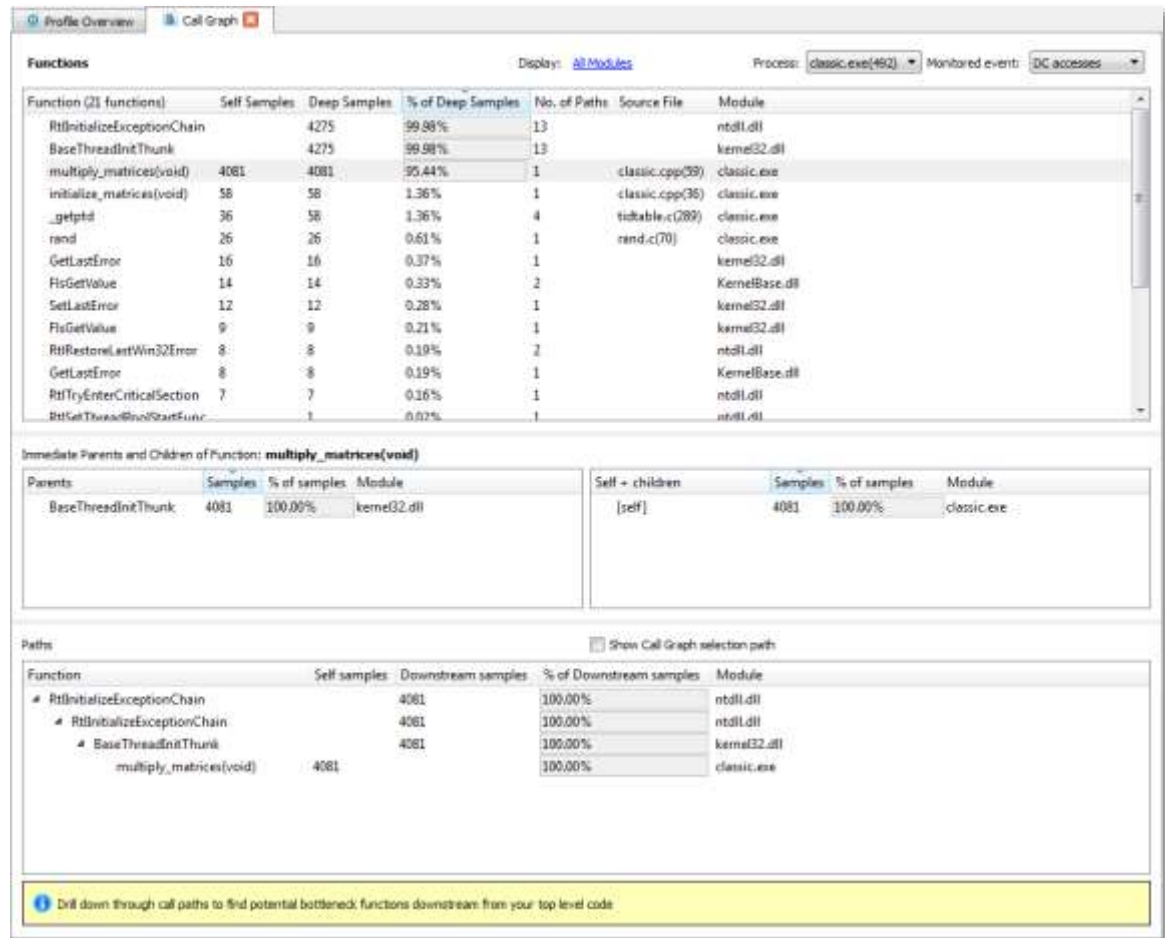
## Getting Started with CodeXL

Profile Overview									
Modules									
Processes									
Display: All Data, All Modules Display by: Processes									
Process	PID	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret miss branch	DTLB L1M L2M
classic.exe	492	4,296	3,190		66,668	7,355	2,903	34	15,844
Modules filtered by selected processes									
Module	Symbols	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret miss branch	DTLB L1M L2M
classic.exe	Loaded	4,201	3,170		66,191	7,243	2,404	27	15,840
kernel32.dll	Not Loaded	37			39	39	161		
kernelBase.dll	Not Loaded	22			26	31	258		
ntdll.dll	Not Loaded	16			34	14	32		
ntoskrnl.exe	Not Loaded	18	19		355	21	43	4	4
fltmgr.sys	Not Loaded	2			1				
Ntfs.sys	Not Loaded	1			5	2	2		
atimdag.sys	Not Loaded	1					1		
diagknl.sys	Not Loaded					1			
diagmmsl.sys	Not Loaded				1			1	
tcpip.sys	Not Loaded				1			1	
ntdll.dll	Not Loaded				3				
win32k.sys	Not Loaded		1		2				
wow64.dll	Not Loaded				1				
amdppm.sys	Not Loaded				2			1	
ataport.SYS	Not Loaded				6	3			
NETIO.SYS	Not Loaded						1		
WinP1000.sys	Not Loaded				1				
hal.dll	Not Loaded					1	1		
Modules with a high sample count usually indicate performance bottlenecks. Sort the table according to a specific metric to highlight potential bottlenecked modules									

### Call Graph Tab

To open the Call Graph view:

1. Double click the Call graph node in the Explorer tree, or use the “Open Call Graph” command from the context menu of the Call Graph node (available only if Call Stack Sampling was enabled).



## Functions Tab

To open the Functions view:

1. Double-click the Functions node in the Explorer tree, or use the “Open Functions” command from the context menu of the Function node.  
The Functions list can be filtered based on the module to which they belong. To do this, invoke a dialog from the hyperlink at the top of function table that lists the displayed and hidden modules. The Functions list also can be filtered to display functions for a specific process using the Process drop list.





The screenshot shows the 'Functions' window in Windows Performance Analyzer. The table lists various functions and their performance metrics. The columns are: Function, Module, DC accesses, DC misses, Misalign access, CPU clocks, Ret inst, and Ret branch. The functions are sorted by CPU clocks in descending order.

Function	Module	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch
multiply_matrices(void)	classic.exe	4,061	5,170		66,071	7,036	1,958
initialize_matrices(void)	classic.exe	58			61	131	224
_getpid	classic.exe	36			39	53	170
rand	classic.exe	26			20	23	52
GetLastError	kernel32.dll	16			12	18	66
FileGetFile	kernel32.dll	14			17	22	204
SetLastError	kernel32.dll	12			8	9	47
FileGetFile	kernel32.dll	9			19	12	48
GetLastError	kernel32.dll	8			9	9	54
RtlRestoreLastWin32Error	ntdll.dll	8			16	13	24
RtlTryEnterCriticalSection	ntdll.dll	7			11	1	7
KeStackAttachProcess	ntoskrnl.exe	3				2	6
KeSetTimer	ntoskrnl.exe	3	1		21	5	3
KeUpdateRunTime	ntoskrnl.exe	2	5		249	4	2
PsGetCurrentThreadWin32ThreadAndEnterCriticalRegion	ntoskrnl.exe	2			5		
memcpy	ntdll.dll	1					
Ntfs.sys\0xffff8b001220041	Ntfs.sys	1					
atikmdag.sys\0xffff8b0049609e3	atikmdag.sys	1					
FileReleasePushLock	filemgr.sys	1			1		
FileCallBackDataDirty	filemgr.sys	1					
SeAccessCheckWithHint	ntoskrnl.exe	1	1		12		6
KeGetCurrentProcessorNumberEx	ntoskrnl.exe	1			1	2	4
ExRtlLookupBaseMcbEntry	ntoskrnl.exe	1					
ExRaiseStatus	ntoskrnl.exe	1					
PsQueryWatchdogTime	ntoskrnl.exe	1					
SeQueryInformationToken	ntoskrnl.exe	1			2		
RtlLeaveCriticalSection	ntdll.dll				1		
memset	ntdll.dll				1		
LdrGetDllHandleEx	ntdll.dll				1		
LdrGetProcedureAddressEx	ntdll.dll						1
RtlCompareUnicodeStrings	ntdll.dll				1		
memmove	ntdll.dll				1		
auIrem	ntdll.dll				1		
TpAllocPool	ntdll.dll				1		
Ntfs.sys\0xffff8b00122324a	Ntfs.sys					1	
Ntfs.sys\0xffff8b001225b11	Ntfs.sys						1
Ntfs.sys\0xffff8b001226682	Ntfs.sys				1		

## GPU Profiling

For GPU application timeline trace profiling:

1. Click on the profiling mode taskbar button.
2. Select Profile >> GPU: Application Timeline Trace from the Profile drop-down menu.
3. Run the program, then let it complete, or terminate it.

An Application Timeline Trace view appears with a timeline of the program execution. This timeline shows the created OpenCL contexts and command queues, as well as the relationships between them.

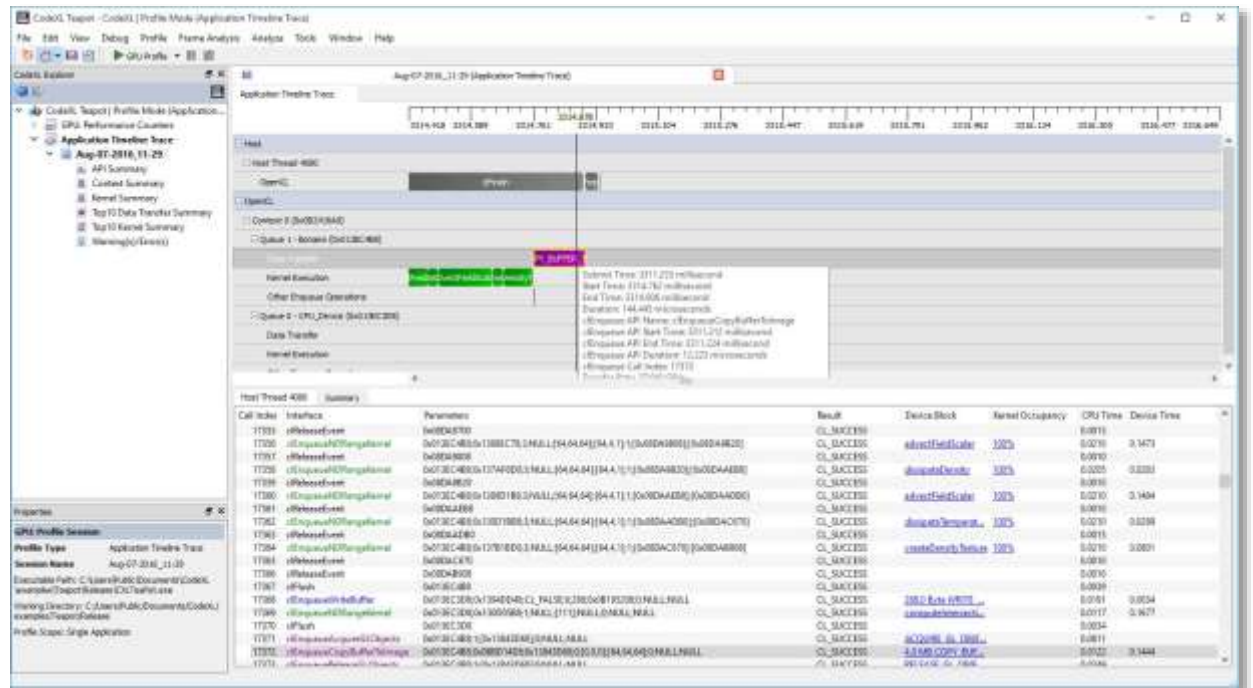
To select a subrange of the timeline, hold down <Ctrl>, and click and drag on a section of the timeline.

To shift the timeline display left or right, simply click on it and drag.

To zoom in/out, use the mouse wheel or the +/- keys. Selecting a small subrange lets you zoom in to see details about each event.

For additional information, hover over an event; this displays a pop-up.

The following screenshot is an example of a COPY\_BUFFER\_TO\_IMAGE data transfer event at 7752.980 ms on the timeline. The pop-up provides detailed timing data.



### Summary Tab

The Summary tab provides several options for viewing profiling data: API, context, kernel, top 10 data transfer, top 10 kernel, warnings/errors.

The following screenshot shows an example of a Top 10 Kernel Summary.

Kernel Name	Context ID	Command Queue ID	Device Name	Duration(ms)	Global Work Size	Work Group Size	Thread ID	Call Index
<a href="#">computeFieldPressureIter</a>	0	0	Juniper	5.00478	[64,64,64]	[64,4,1]	7756	276
<a href="#">computeFieldPressureIter</a>	0	0	Juniper	1.51300	[64,64,64]	[64,4,1]	7756	294
<a href="#">dissipateTemperature</a>	0	0	Juniper	0.84333	[64,64,64]	[64,4,1]	7756	20715
<a href="#">dissipateTemperature</a>	0	0	Juniper	0.83733	[64,64,64]	[64,4,1]	7756	2301
<a href="#">dissipateTemperature</a>	0	0	Juniper	0.83311	[64,64,64]	[64,4,1]	7756	12729
<a href="#">dissipateTemperature</a>	0	0	Juniper	0.82756	[64,64,64]	[64,4,1]	7756	6261
<a href="#">dissipateTemperature</a>	0	0	Juniper	0.81600	[64,64,64]	[64,4,1]	7756	19725
<a href="#">computeFieldPressureIter</a>	0	0	Juniper	0.81333	[64,64,64]	[64,4,1]	7756	6301
<a href="#">dissipateTemperature</a>	0	0	Juniper	0.80589	[64,64,64]	[64,4,1]	7756	17679

The Warning(s)/Error(s) summary also includes a helpful list of best practice recommendations to improve program performance. The following example indicates issues with blocking write calls and small global work size.



Index	Call Index	Thread ID	Type	Message
0	55	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3098]: Object created by clCreateKernel
1	60	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3118]: Object created by clCreateKernel
2	100	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3158]: Object created by clCreateKernel
3	95	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3218]: Object created by clCreateKernel
4	75	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3298]: Object created by clCreateKernel
5	85	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3308]: Object created by clCreateKernel
6	90	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3418]: Object created by clCreateKernel
7	80	7756	Warning	Memory leak detected [Ref = 1, Handle = 0x06CB3458]: Object created by clCreateKernel

### Performance Counters View

The Performance Counters view in a GPU Performance Counters profile provides kernel performance details, including global work size and time. This mode collects performance counters from the GPU or APU for each kernel dispatched to the device. It also displays statistics from the shader compiler for each kernel dispatched. The performance counters and statistics can be used to discover kernel bottlenecks.

To display a Code viewer with kernel code:

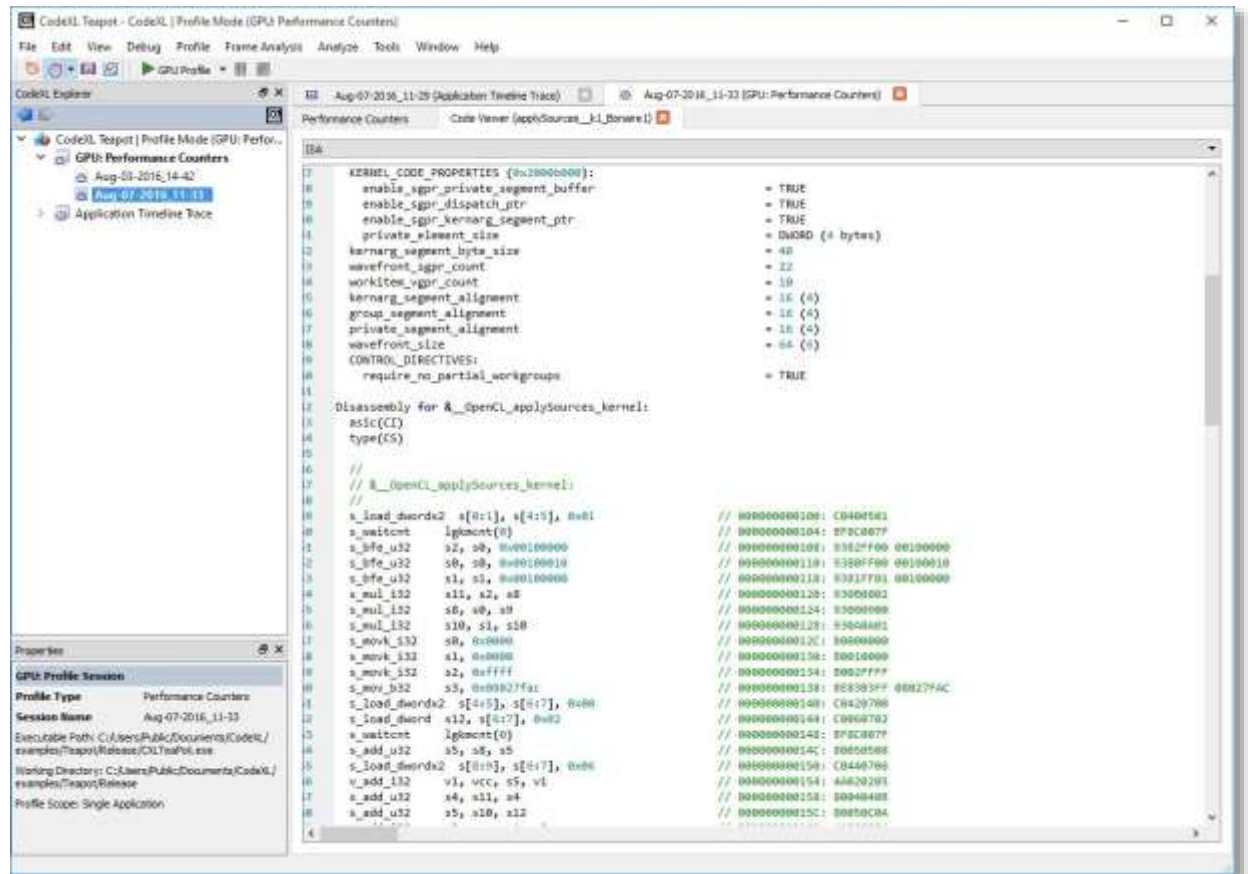
1. Click on a kernel name (Method) in the Performance Counters view.

	Method	ExecutionOrder	ThreadID	CallIndex	GlobalWorkSize	WorkGroupSize	Time	LocalMemSize	VGPRs	S
1	<a href="#">applySources_k1 Juniper1</a>	1	8056	269	{ 64 64 1 }	{ 64 4 1 }	0.01500	0	7	NL
2	<a href="#">applyBuoyancy_k2 Juniper1</a>	2	8056	270	{ 64 64 64 }	{ 64 4 1 }	0.17567	0	4	NL
3	<a href="#">calculateCutU_k3 Juniper1</a>	3	8056	271	{ 64 64 64 }	{ 64 4 1 }	0.20700	0	9	NL
4	<a href="#">applyVorticity_k4 Juniper1</a>	4	8056	272	{ 64 64 64 }	{ 64 4 1 }	0.39611	0	5	NL
5	<a href="#">advectFieldVelocity_k5 Juniper1</a>	5	8056	273	{ 64 64 64 }	{ 64 4 1 }	0.16711	0	14	NL
6	<a href="#">applyVelocityBoundaryCondition_k6 Juniper1</a>	6	8056	274	{ 64 64 64 }	{ 64 4 1 }	0.26656	0	4	NL
7	<a href="#">computeFieldPressurePrep_k7 Juniper1</a>	7	8056	275	{ 64 64 64 }	{ 64 4 1 }	0.16111	0	9	NL

A pull-down bar at the top of the window under the Code Viewer tab (see following screenshot) lets you select OpenCL source (CL), intermediate language (IL), or instruction set architecture (ISA) code.

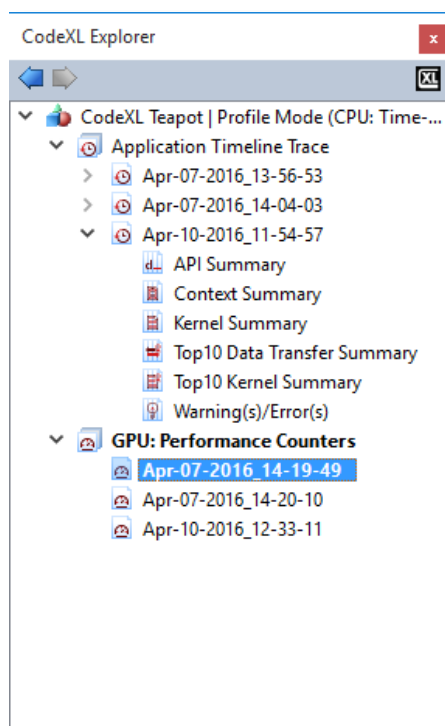


## Getting Started with CodeXL



### CodeXL Explorer Tree

The Explorer view lets you switch between profiling sessions. This view lists all profiling sessions for the current project.





## [Getting Started with CodeXL](#)

To display a session's data, double-click on it. To rename or delete it, right-click on it.

To import profiling data, right-click or drag/drop a session data file to the Explorer.



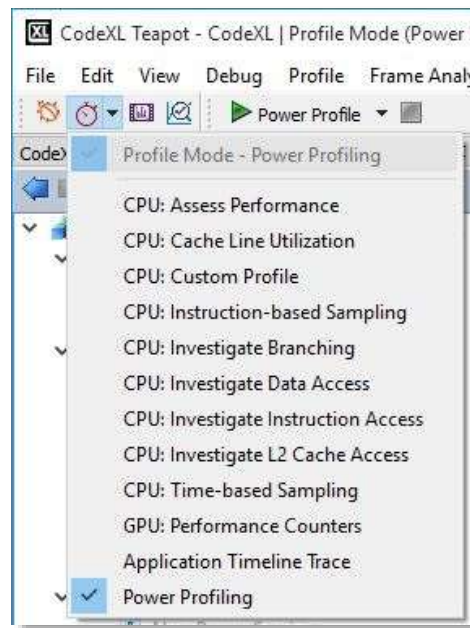
## Power Profiling

Power Profiling mode allows monitoring the power consumption of supported AMD APU's in real-time. Additional attributes can also be monitored: GPU and CPU frequencies, APU thermal trend and CPU Cores P-State.

For a list of currently supported APU's see the System Requirements section of the CodeXL Release Notes, available in the CodeXL installation folder and on the CodeXL web page.

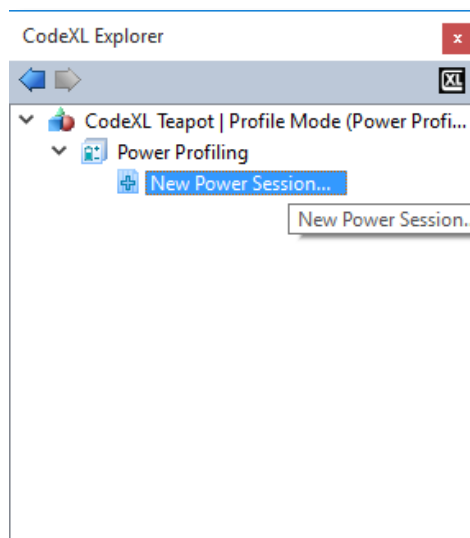
### Switching to Power Profiling mode

In the Profile Mode drop-down list, choose Power Profiling.



### Starting a new Power Profiling session

After switching to Power Profiling mode, double click New Power Session... in the CodeXL Explorer tree.



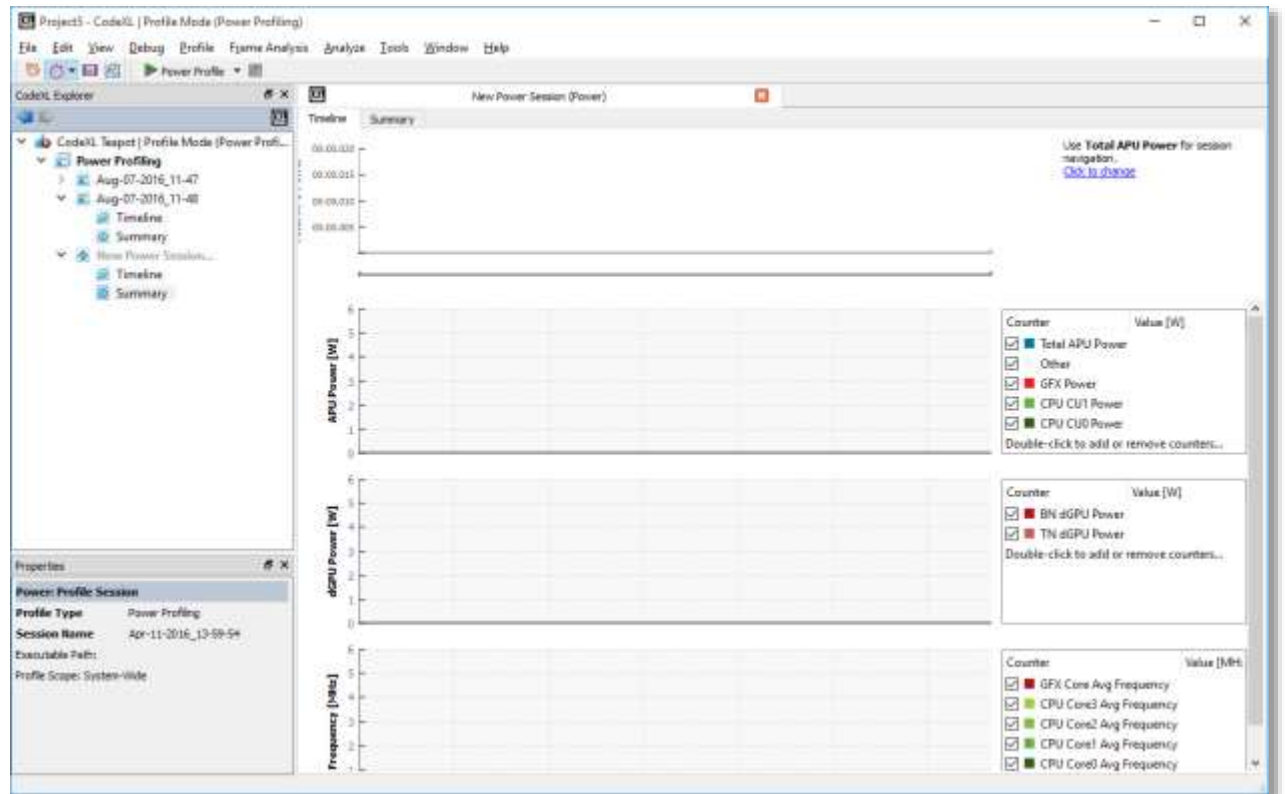





## Getting Started with CodeXL

Please note that if your hardware does not support Power Profiling, you will not be able to start the session.

After double clicking on New Power Session... in the CodeXL Explorer tree, an empty Power Profiling session window will be opened.



Clicking the Start button  will start the session with the current configuration. We will now see how to change a Power Profiling session's configurations.

### Setting the Sampling Interval


To configure the sampling interval (the time period which will pass between every two consecutive samples of the active counters), click on Profile->Profile Settings -> Power Profile:



Please note that the current minimum sampling interval is 100 milliseconds.

Click OK to apply your changes.

### Stopping a Power Profiling session

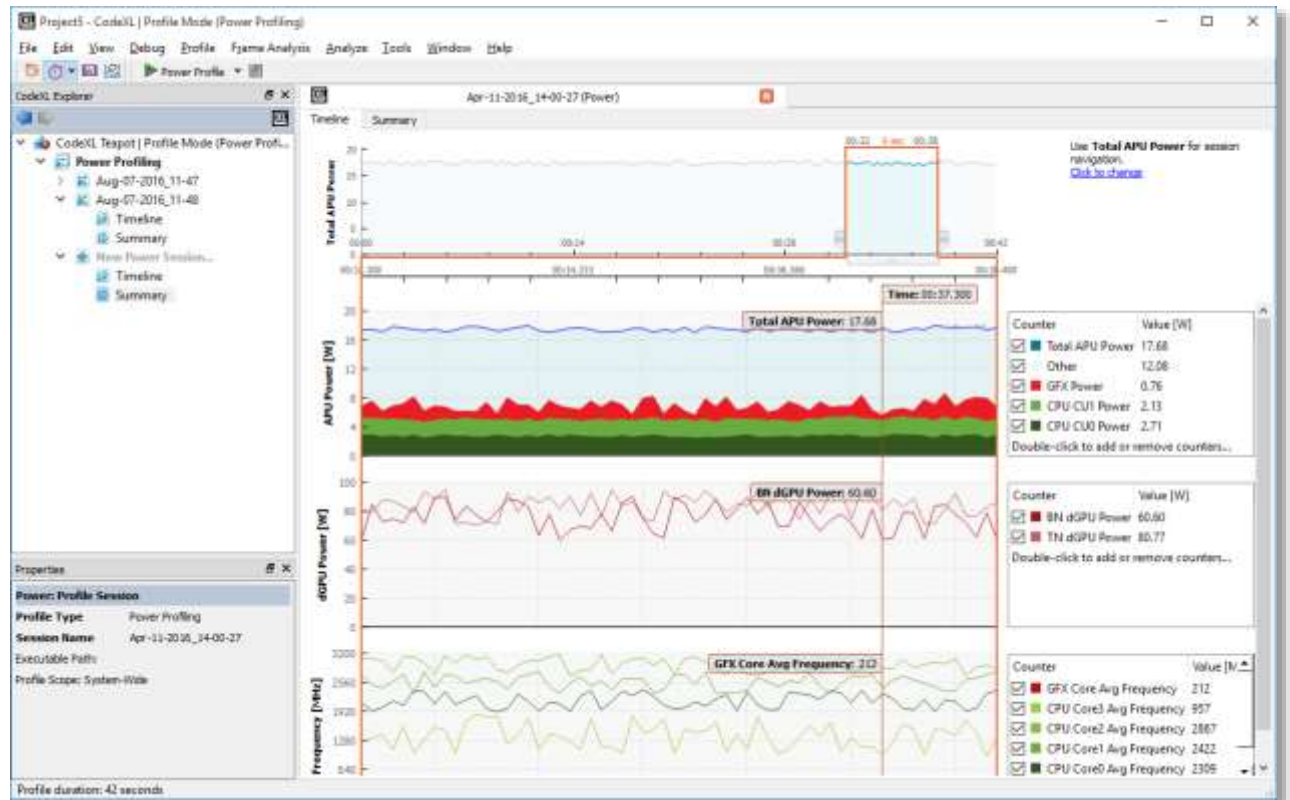
To stop a running Power Profiling session, click on the Stop button .



### Power Profiling Real-Time Values

Throughout the Power Profiling session, the collected data will be streaming in real-time to the graphs. There are two tabs that present data: Timeline View and Summary View.

### Power Profiling Timeline View



The Power Profiler Timeline View displays the measured values of the activated counters throughout the session. The horizontal axis of all charts represents the time that has passed since the beginning of the session. During profiling sessions, all of the charts in the Timeline View are being updated in real-time with the measured values which are streaming in. The uppermost ribbon (titled “Total APU Power”) displays the overall power consumption of the APU throughout the session.

The top chart has an adjustable range slider that controls the display of all the other timeline charts. By performing such actions as dragging the slider sideways, extending or retracting it, you set the scope of attention and the focus of the timeline charts. Each of the charts below displays only the data that was collected in the time range corresponding to the slider’s position and length. That is, the data in all timeline charts, except for the Total APU Power chart itself, is dictated by the time range which is selected by the Total APU Power chart’s range slider.

Below the Total APU Power ribbon, you will find additional ribbons containing more graphs, according to the set of activated counters: A Power chart which displays the power consumed by specific APU components (such as CPU cores or integrated GPU), a Frequency chart which displays the frequency of the selected

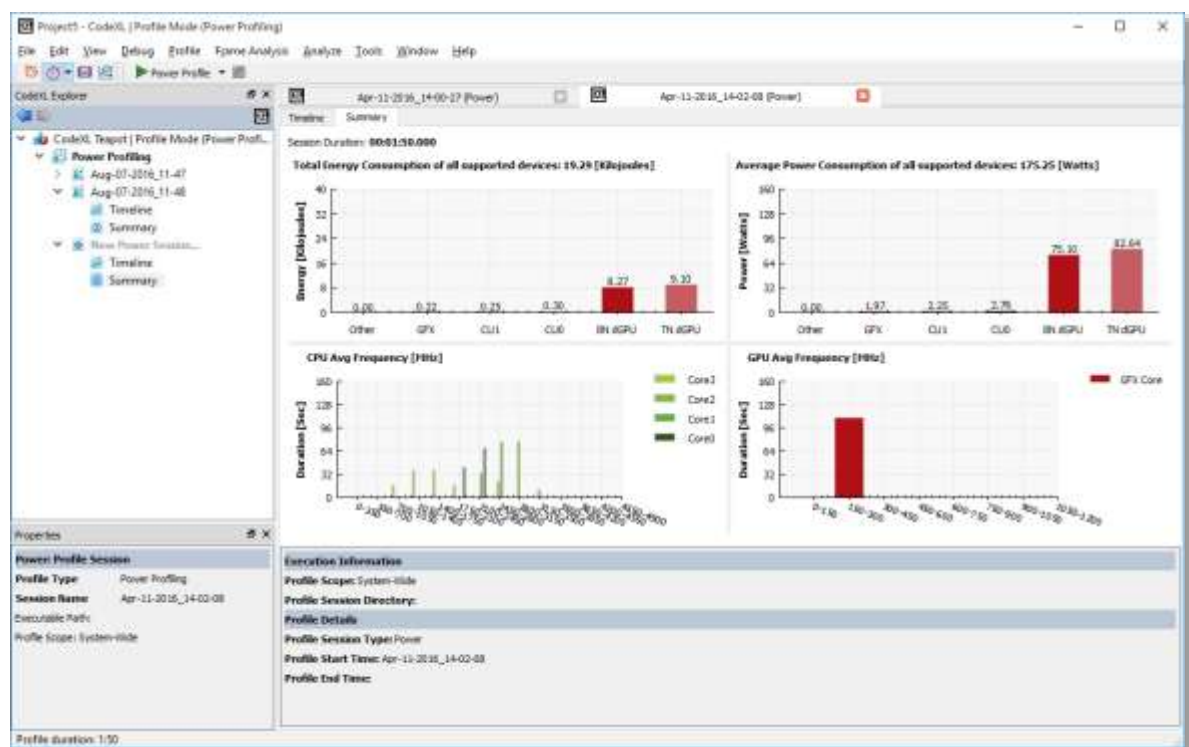




components, a Temperature chart which displays the thermal trend of the selected components, and a CPU State chart which displays the CPU core states. The APU Power graph is always displayed, since the Total APU Power counter is activated by default and cannot be deactivated. The other charts (frequency, temperature and CPU core state) are optional, and will only be displayed if the relevant counters were activated.

To the right side of each chart you will find a legend that displays the measured values at a specific point in time. To change the point in time for which the values are displayed, reposition the mouse cursor horizontally on one of the graphs. The list of counters in the legends is customizable, and specific counters can be removed/added between profile sessions.

### Power Profiling Summary View



The Power Profiler Summary View displays an analysis of the values measured throughout the session. Similarly to the Timeline View, this view is updated in real-time when power profiling sessions are running.

At the upper-left side of the summary view, you can see the session duration which is the amount of time that the profiling session was in progress.

The following Power histograms graphs are plotted in the Summary View:

- **Total Energy Consumption**  
This histogram graph displays the cumulative energy consumed by the APU and discrete GPU components, measured in Joules.
- **Average Power Consumption**  
This graph displays the average power consumption of the APU and the discrete GPU components, measured in Watts.



If CPU or GPU frequency counters were activated for the session, you will find additional histogram graphs below the Power graph in the Summary View:

- **CPU Frequency Graph**  
This stacked histograms graph displays for each CPU core how much time it spent at each frequencies range.
- **GPU Frequency Graph**  
This graph displays how much time the GPU spent at each frequencies range.

### Configuring Power Profiler Sessions

For further configuration options of the Power Profiler such as selecting which counters will be sampled and launching an application when the session begins, see the CodeXL User Guide by selecting from the menu bar Help -> View Help.

## Analyze Mode

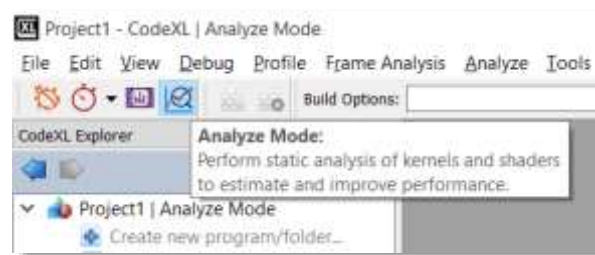
### Static Kernel and Shader Analysis

In Analyze mode, you can compile and generate performance statistics for OpenCL kernels, DirectX Shaders, OpenGL Programs and Vulkan Programs. The compilation and statistics generation process can be targeted at a variety of AMD GPUs and APUs, independent from the actual type of GPU/APU that is installed on your system. Using the Analyzer, you can generate and inspect performance statistics, ISA code, IL code and D3D ASM code.

### Switching to Analyze mode

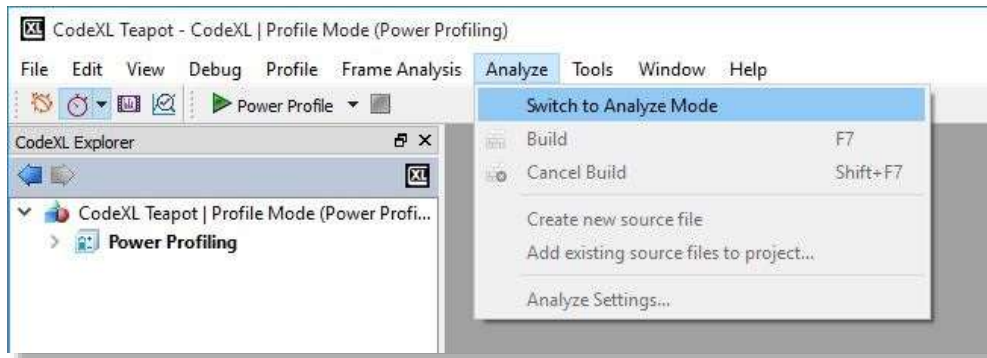
#### Option 1:

Click on the Analyze Mode button in the CodeXL Mode toolbar.



#### Option 2:

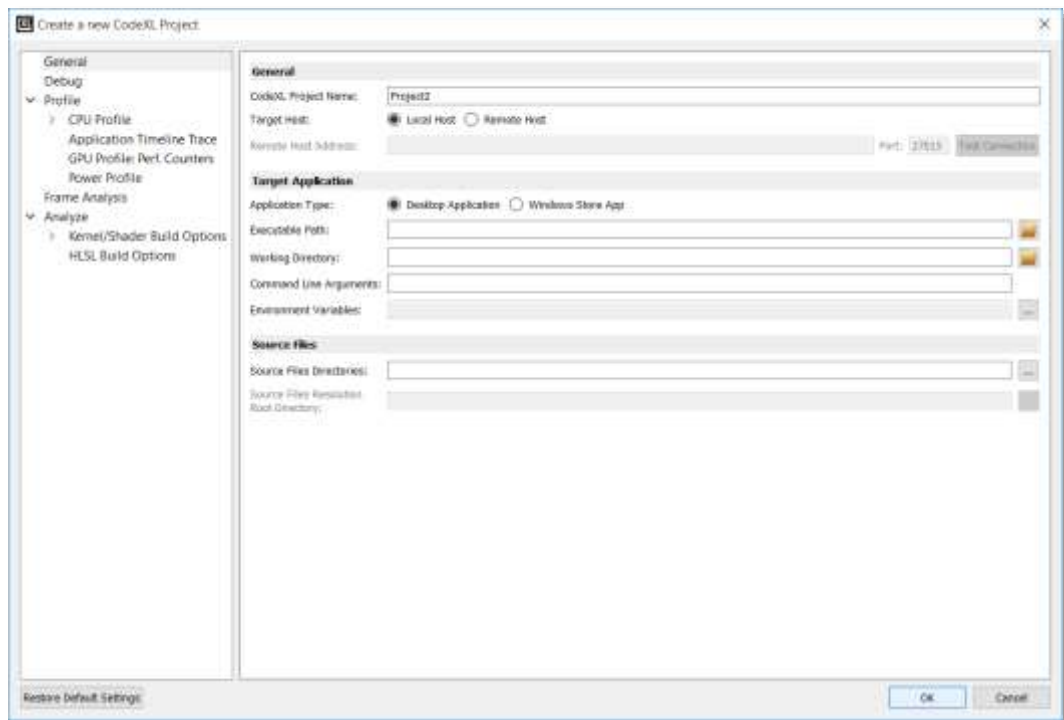
Click Analyze in the main menu.



Once you switch to Analyze mode, you can create a new project, open a previously saved project or load the Teapot sample.

### Creating a new project for Analysis

Click on the “File->Create Project”, or use the Ctrl+N shortcut. The following CodeXL Project Settings dialog will appear:



Rename the project, and click on the OK.

After the new project has been created, the CodeXL Analyzer Explorer Tree should appear in the left pane:

### Working with the new CodeXL Analyzer Explorer Tree

If you are familiar with the former versions of the Analyzer, you probably noticed that the tree has a different structure than the one used in previous versions. Let's examine the structure of the new CodeXL Analyzer Explorer:



1. **Programs and Folders:** before describing how to technically create Programs and Folders, let's first discuss what those objects are, and why they can be useful.

- a. **Programs (OpenGL, Vulkan):**

As of version 2.0, CodeXL can compile and link together multiple source files for OpenGL and Vulkan. This is especially important when different shaders have mutual impact on one another's ISA and performance statistics. To provide that type of support, CodeXL Analyzer introduced the concept of a Program. There are two types of Programs in CodeXL 2.0:

- Rendering Programs
- Compute Programs

A Rendering Program represents a graphics pipeline, and can have a single shader attached to each of its stages:

- Vertex
- Tessellation Control
- Tessellation Evaluation
- Geometry
- Fragment

A Compute Program represents a compute pipeline, and can have a single compute shader attached to its single stage.

When you build a program that has multiple shaders attached to it, all shaders are being compiled and linked together. This way, you get more accurate ISA and performance statistics than those generated using previous versions of CodeXL.

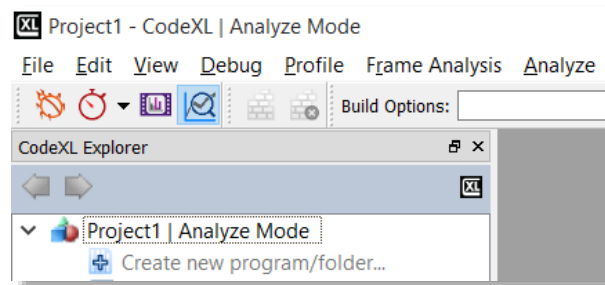
- b. **Folders (OpenCL, DirectX):**

Folders are logical containers of source files. When you build a folder that has multiple source files attached to it, the source files are simply being built one after the other. Unlike programs, there is no kind of interdependency between the source files in a given folder: when a folder is being built, each source file is being compiled independently. Folders can be used to organize the project, by serving as a logical separator. They can also be used to ease the process of comparing build results, since now the build results are being maintained per-folder: you can create two different Folders, each containing the same source files, but have a different configuration (for example, create two DirectX Folders, each with a different shader model). After building the two Folders, you can toggle between the performance statistics of the two Folders to see the differences.

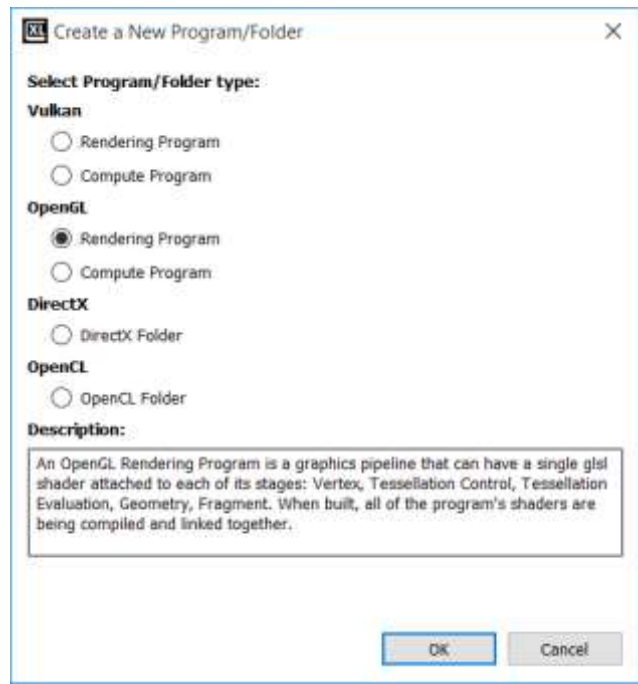
You may ask yourself why CodeXL does not support the concept of DirectX Programs, just like it does for OpenGL and Vulkan. This is a good point. Supporting DirectX Programs is at a high priority in the Analyzer's roadmap, and we will do our best to add that feature in the upcoming versions of the product.

### Creating a new Program or Folder

To create a new Program or a Folder, double-click on the "Create new program/folder" item in CodeXL Analyzer Explorer Tree:

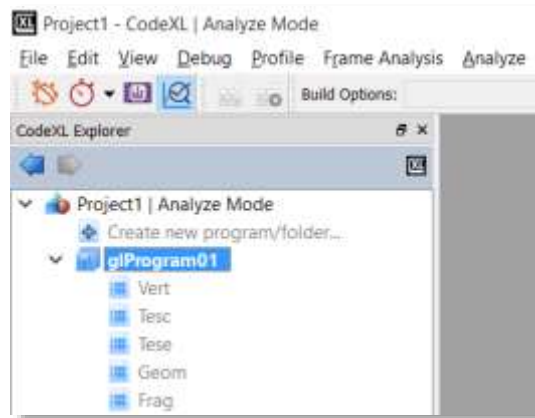


Then, the following dialog would pop-up:



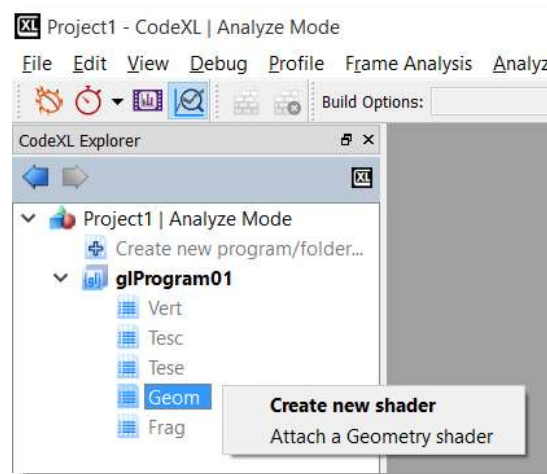
Select the Program/Folder type of choice, and click OK.

Then, the empty Program/Folder would appear in the Explorer Tree. For Example, if you choose an OpenGL Rendering Program, you will see an empty OpenGL Rendering Program created:



### Working with Programs

After creating a new program, you will see that it contains an empty placeholder for every pipeline stage. Right-click on any stage to add an existing shader or create a new one:

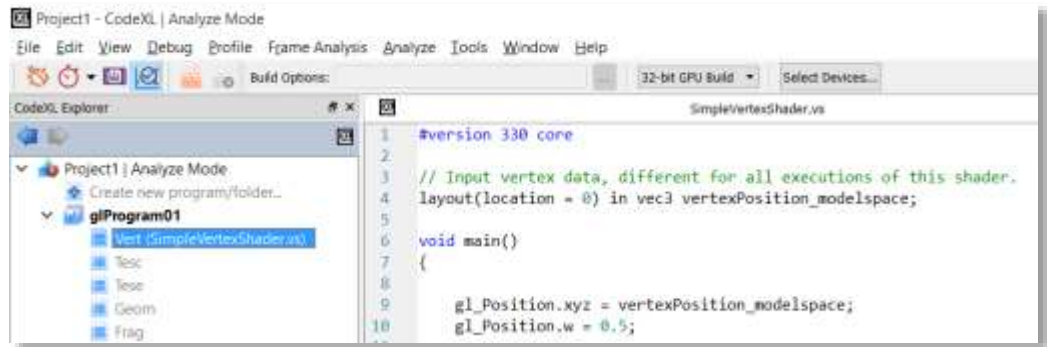


**Note:** You can also double-click on a stage to create a new shader and automatically attach it to that Program's stage.

Whether you chose to create a new shader or to attach an existing one, after the shader is added to the program, it will also be listed under the Source Files pool for future use. This will enable you to attach that specific shader to other, future or existing, programs as well. There is no limit for the number of programs that can reference the same shader.

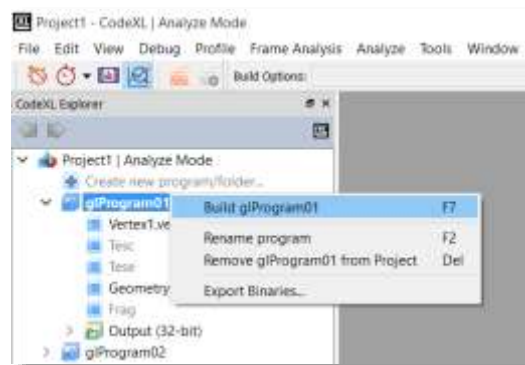


## Getting Started with CodeXL

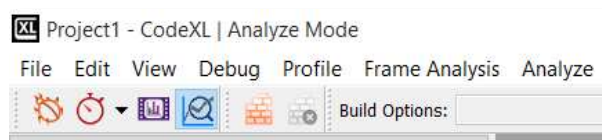


As you can see in the above screenshot, we attached SimpleVertexShader.vs as the vertex shader to our OpenGL Rendering Program, and it was also automatically added to the Source Files pool. We can now drag SimpleVertexShader.vs from the Source Files pool and drop it on the stage node of any Program that we may add to the project, to reuse SimpleVertexShader.vs (there is no dependency in the build process between different Programs).

To build the program, right-click on it and select the Build option, or use the F7 shortcut:



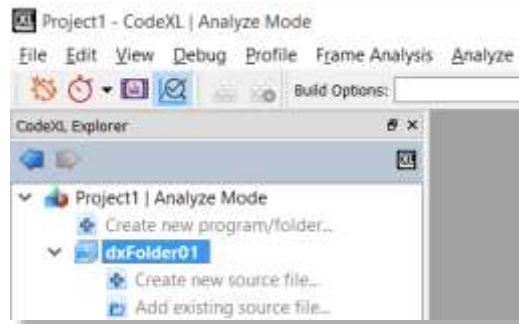
You can also select the Program and manually click on the Build button in the Analyzer toolbar:



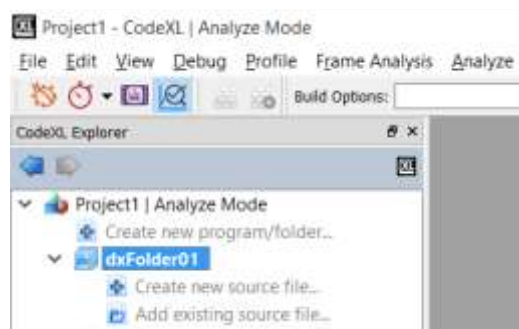
### Working with Folders

After creating a new OpenCL or DirectX Folder, an empty Folder would be listed in the Explorer Tree:

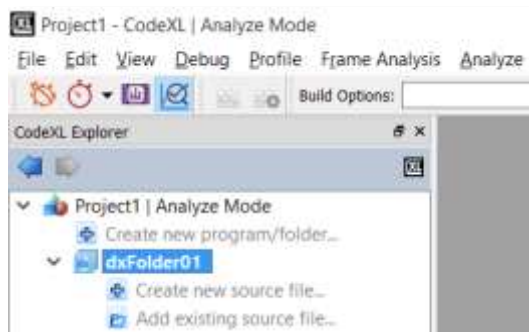




To create a new source file, and automatically add it to the Folder, double-click on the “Create new source file item...” item of the folder:



To add an existing source file, and automatically add it to the Folder, double-click on the “Add existing source file item...” item of the folder:

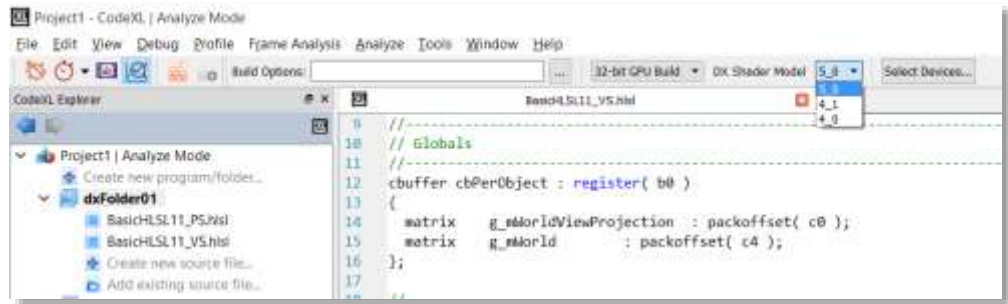


To configure the build properties of a source file under a specific Folder, click on that source file and use the Analyzer toolbar’s Type and Entry point drop-down lists. The first sets the type of the shader and the latter specifies the specific target shader (among the shaders in the source file). This configuration is Folder-specific. That is, the same source file can be set with different properties under different Folders. CodeXL will remember those configurations for you.

To configure the build properties of the Folder, click on the Folder and adjust the enabled items in the Analyzer toolbar. For CodeXL 2.0, this is only relevant to the DX Shader Model property of DX Folders:

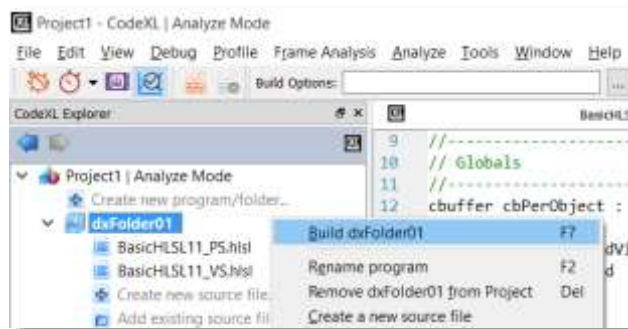


## Getting Started with CodeXL



Once set, the DX Shader Model value will hold for all the shaders in the selected Folder. For example, if you choose 5\_0 as the DX Shader Model, any D3D vertex shader in that Folder will be compiled using shader model vs\_5\_0.

To build the whole Folder, right-click on it and select the Build item:



Unlike the case with Programs, Folders are more flexible as they allow you to build selected source files, without being required to build the whole Folder. To build selected source files, click on the selected source files under the program, while holding the Ctrl key. Then, right-click on one of the selected files and select the build option:

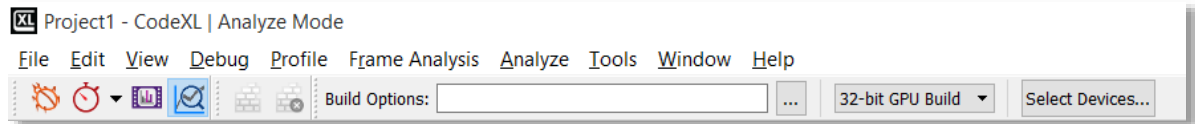


### Selecting target devices

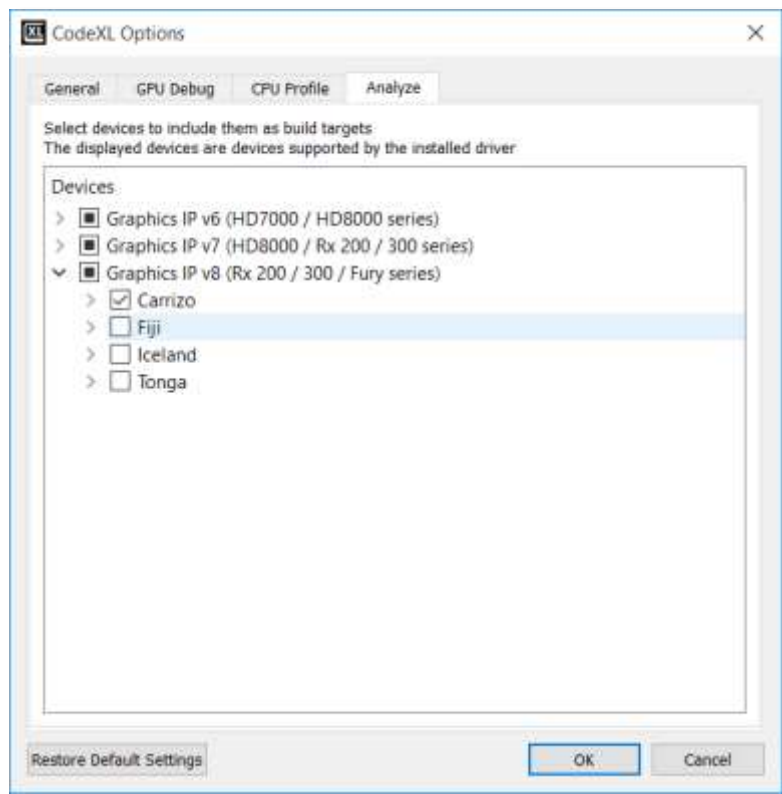
CodeXL Analyzer can target a variety of devices, independent of the device that is physically installed on your system. To select the target devices, for which the build would be performed, first click on the Select Devices button in the Analyzer toolbar:



## Getting Started with CodeXL

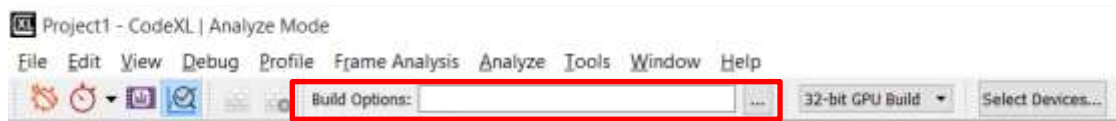


Then, the CodeXL Options dialog would pop-up with its Analyze tab activated. The devices are grouped by generations. You can use the check boxes to select and remove devices:



### Build Options - defining kernel/shader compilation options

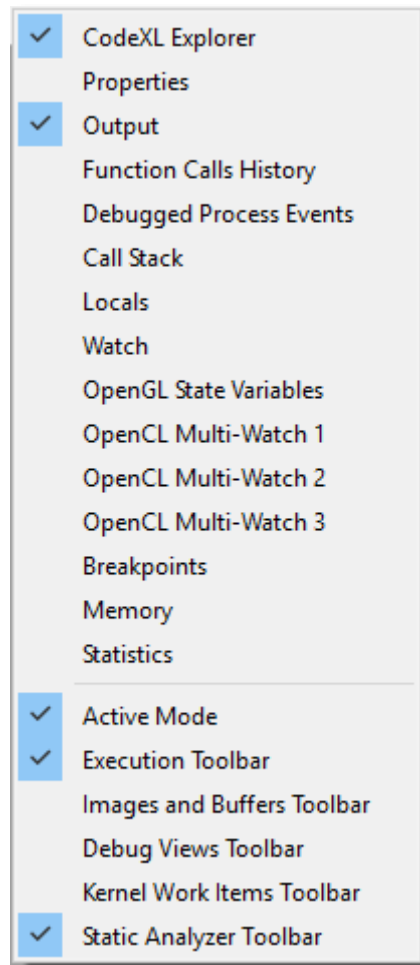
In the Static Analyze toolbar, there is a text box where you can manually define specific OpenCL/HLSL build options (there is no support for GLSL build options):



DirectX (Direct3D) shaders written on high-level shader language (HLSL) are supported on MS Windows only.




**Note:** The display of this toolbar is dynamic; you can set it from the right-click menu in the main CodeXL frame:



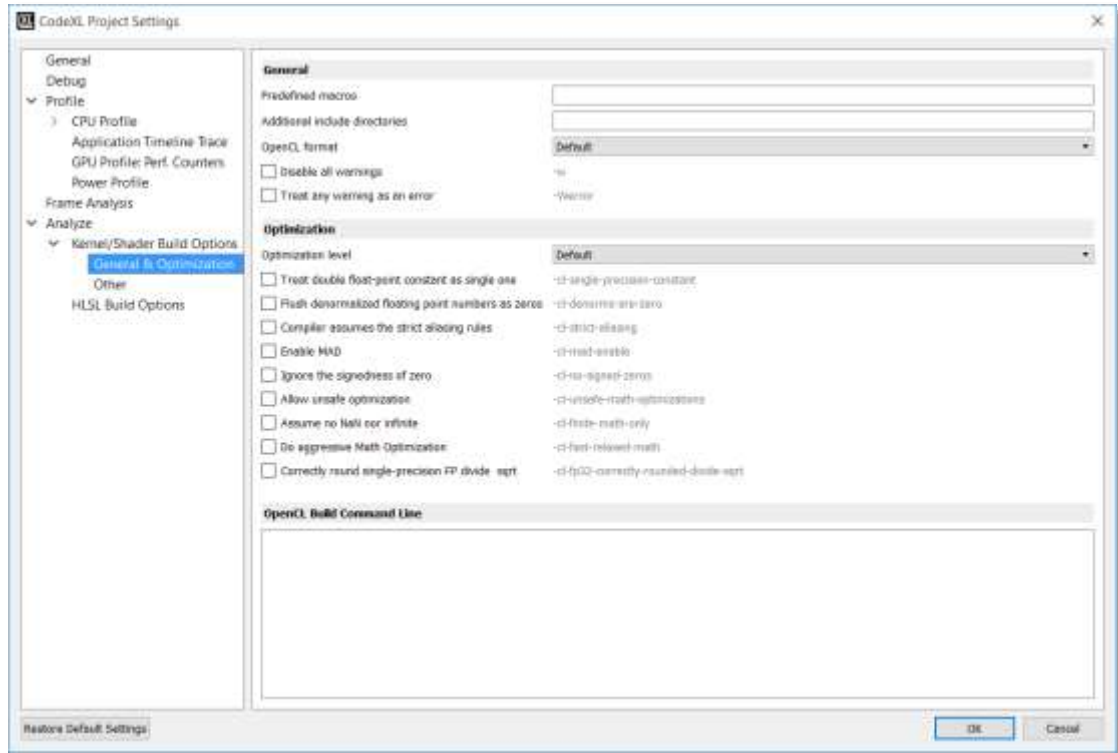
The Build Options box is a place to set compiler build flags such as `-x clc++` or `-o3`. Any compiler build flag can be placed in this box.

### OpenCL Build Options Dialog

This dialog will help you choose the correct OpenCL build options for you and hopefully will prevent you from making spelling mistakes while typing the options manually.

To open the **OpenCL Build Options** dialog, press the  Button. You can browse between the 'General & Optimization' tab and the 'Other' tab to view all the available options. Once you choose an option, the option text will appear in the text box below marked as 'OpenCL Build Command Line'. This string will also appear in the menu bar after you click the **OK** button.

Typing the command line in the text box will also mark the corresponding check boxes in the dialog.



## Examples of using build options

For building the **tpAdvectFieldScalar.cl** kernel from CodeXL TeaPot sample project, enter the following options:

```
-D GRID_NUM_CELLS_X=64 -D GRID_NUM_CELLS_Y=64 -D GRID_NUM_CELLS_Z=64  
-D GRID_INV_SPACING=1.000000f -D GRID_SPACING=1.000000f -D  
GRID_SHIFT_X=6 -D GRID_SHIFT_Y=6 -D GRID_SHIFT_Z=6 -D  
GRID_STRIDE_Y=64 -D GRID_STRIDE_SHIFT_Y=6 -D GRID_STRIDE_Z=4096 -D  
GRID_STRIDE_SHIFT_Z=12 -I path_to_example_src
```

On windows, *path\_to\_example\_src* should be:

**C:\Users\Public\Public Documents\CodeXL\Examples\Teapot\res**

On Linux, *path\_to\_example\_src* should be:

**/opt/CodeXL\_X.X/examples/Teapot/TeaPotLib/**

Adding the option '-h' will dump the list of OpenCL compiler available options in the output tab.

## HLSL Build Options Dialog

This dialog will help you choose the correct DirectX build options for you and hopefully will prevent making spelling mistakes while typing the options manually.



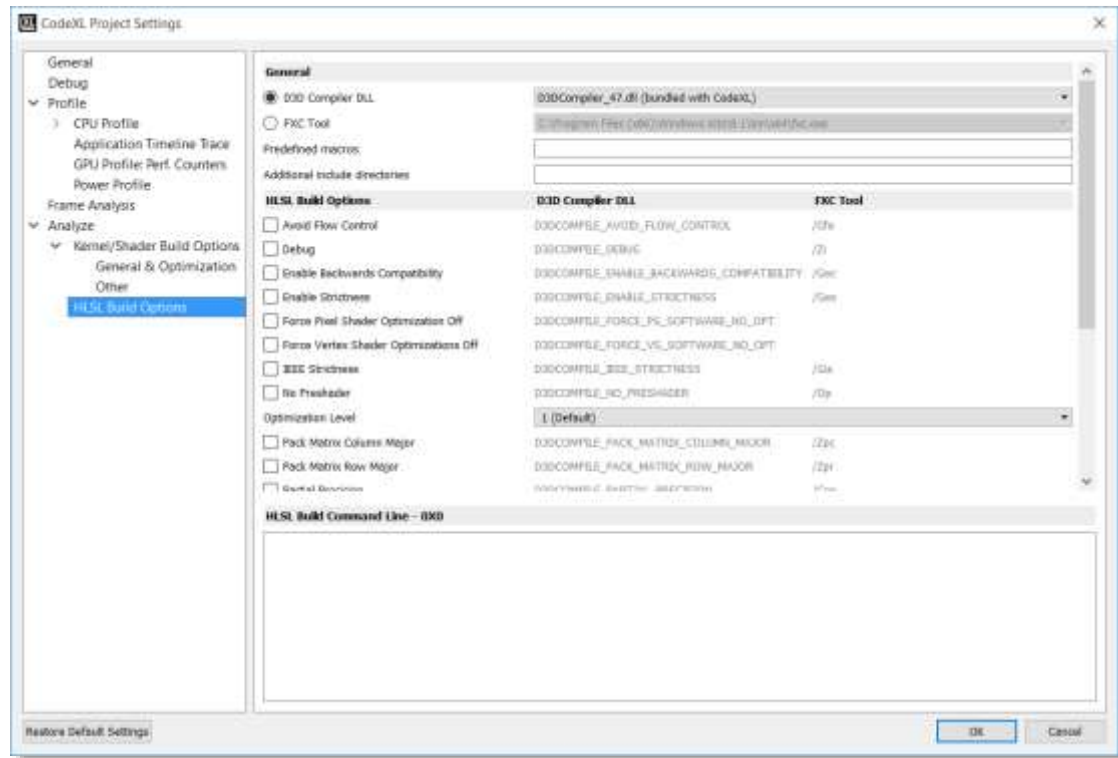
To open the dialog, press The  Button. The dialog will be opened. Click the "HLSL Build Options" node to view the available options.

Once you choose an option, the option text is displayed in the "HLSL Build Command Line" text box that appears below.



This build option string will also appear in the toolbar's build options box after you click the OK button.

As an alternative to selecting options through the radio buttons, it is possible to type a command in the "HLSL Build Command Line" text box. Build options types in the text box will automatically be translated to update of the relevant controls accordingly. For example, typing "D3DCOMPILER\_DEBUG" in the lower text box automatically updates the "Debug" check box to be checked.



### HLSL Compiler Tools

First select the build tool of choice for compiling the shader – D3d compiler / FXC compiler.

The CodeXL installation includes a copy of the Microsoft DirectX compiler DLL: d3dcompiler\_47.dll. You may specify a different path if you want CodeXL to use a different d3dcompiler module. If you select the FXC compiler tool you must specify a path to the location of this tool.

To select the path of the compiler module, click the 'Browse...' option from the combo-box. When selecting Browse, a dialog box will open for selecting the compiler file.

- For D3D compiler – any file called d3compiler\_\*.dll can be selected
- For FXC compiler – only FXC.exe file can be selected.

Note: for D3D compiler the bundled file is selected by default.



### Output Tab

The compiler output appears in the Output tab. The example below shows successful builds (no warnings or errors) for 4 devices.

If errors occur, the output will display the error and the line in which the error occurred.

```
Build started: Building C:\Program Files (x86)\AMD\CodeXL\examples\Teapot\res\tpApplyVorticity.cl for 4 devices:
Build started: Building C:\Program Files (x86)\AMD\CodeXL\examples\Teapot\res\tpApplyVorticity.cl for 4 devices:
Building for Device0: succeeded.
Building for Device1: succeeded.
Building for Device2: succeeded.
Building for Device3: succeeded.
Build completed for 4 devices: 4 succeeded, 0 failed.
Build started: Building C:\Program Files (x86)\AMD\CodeXL\examples\Teapot\res\tpApplyVorticity.cl for 4 devices:
Building for Device0: succeeded.
Building for Device1: succeeded.
Building for Device2: succeeded.
Building for Device3: succeeded.
Build completed for 4 devices: 4 succeeded, 0 failed.
```

If there were errors, the output pane will display the error and the line where the error occurred:

```
Output
Frontend phase failed compilation.

-----
Compiling for WinterPark... ..Failed!
OpenCL Compile Error: clBuildProgram failed (CL_BUILD_PROGRAM_FAILURE).
C:\Program Files (x86)\AMD\CodeXL\examples\Teapot\res\tpApplyVorticity.cl, line 25: error:
expression must have struct or union type
float3 ccU = curlU[index].x.yz;
                        ^
```

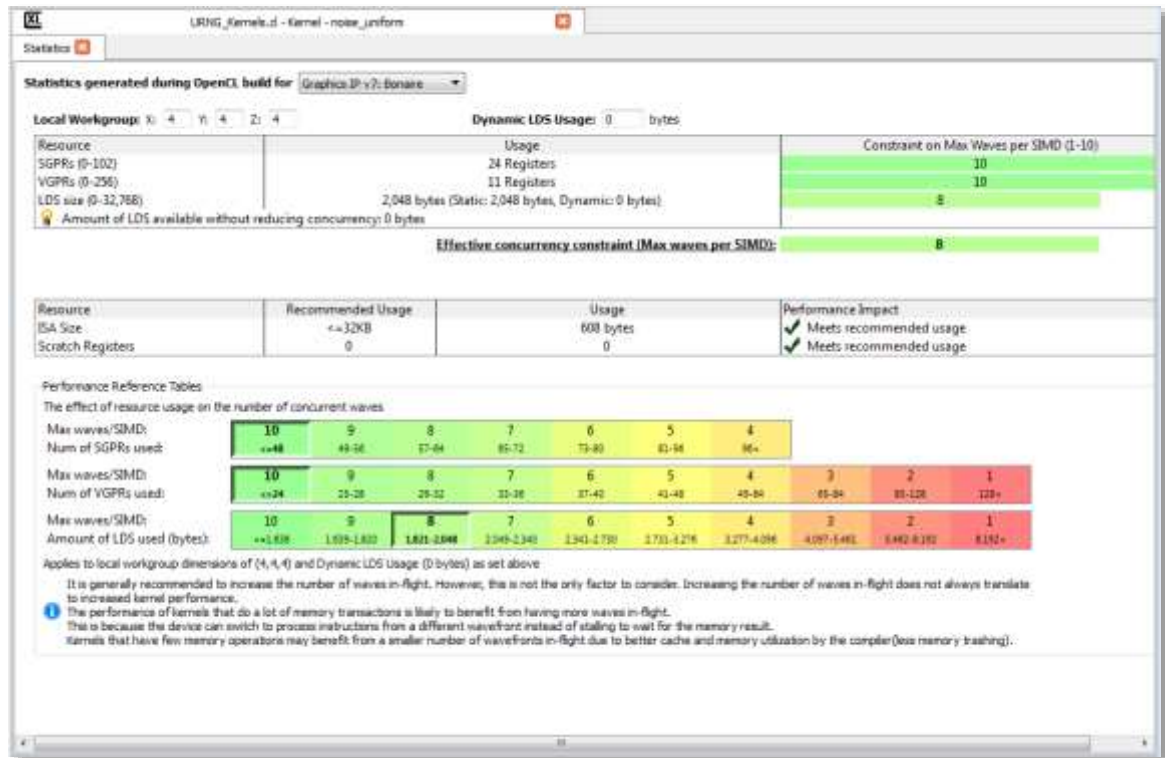
Double clicking on an error navigates the user to the Source Code view, displaying the kernel/shader source code:

```
tpApplyVorticity.cl
18  __global float4* curlU,    // Contains curlU
19  __constant SmokeSimConstants* p)
20  {
21      int3 coord = (int3)(get_global_id(0), get_global_id(1),
22      int index = getIndex(coord);
23
24      // Read in the vorticity vector at this location
25      float3 ccU = curlU[index].x.yz;
26
27      float3 Neu;
28      float forw;
```

### Statistics Tab

The Statistics tab gives detailed statistics for the selected kernel/shader for each target device. To open the Statistics tab, expand the desired kernel in the project tree, and double-click the Statistics node:

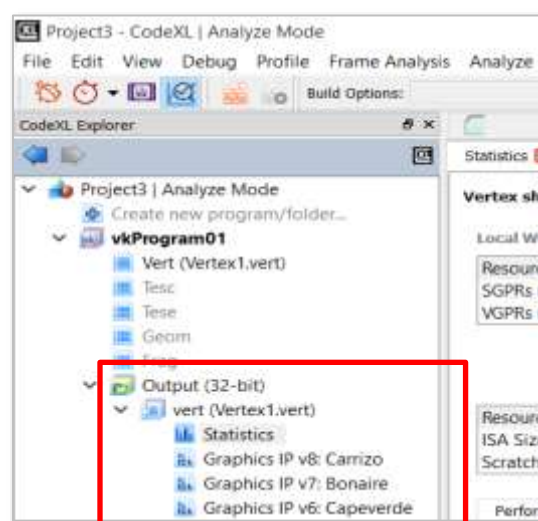




Note: the statistics tab for pre-GCN devices (v4 & v5 generations) is a bit different. For more information, see the complete help manual document.

### Viewing compilation output: ISA and IL

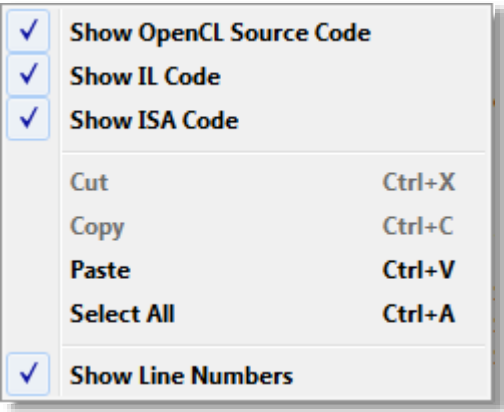
The performance statistics tab will be opened automatically when the build process is over. To view the compilation output, double click the node of the desired ASIC in the explorer tree, under the Program/Folder and configuration (32-bit or 64-bit):



This will open a tab containing the source code, the IL and the ISA. The program source code and the AMD IL code will be presented as standard text documents.



The context menu enables to display/hide line numbers for each source code/IL/ISA tab.



Using this view, you can inspect the ISA code of GCN devices and see the estimation for instruction cost in clock cycle. The view contains 5 columns:

- **Address:** the instruction's offset within the program (in bytes)
- **Opcode:** the operation to be performed
- **Operands:** the data for the operation
- **Cycles:** the number of clock cycles which are required by a Compute Unit in order to process the instruction for a 64-thread Wavefront, while neglecting the system load and any other runtime-related factor.
- **Instruction Type:** the category of instructions to which the instruction belongs
- **Hex:** binary representation of the instruction, in hexadecimal format

1. Note that code labels which appear in the Operands column are clickable. By clicking on a label link, you can navigate to the label's spot in the code.



2. Note that this view is only available for GCN devices. For pre-GCN devices, the plain textual ISA view will be displayed.

ISA Code - calPriceVega					
0x001C28	V_CVT_F64_U32	v[72:73] v120	32	Vector Arithmetics	7E902D78
0x001C2C	V_MOV_B32	v74 0	4	Vector Arithmetics	7E940280
0x001C38	V_MOV_B32	v76 0	4	Vector Arithmetics	7E980280
0x001C54	V_RCP_F64	v[80:81] v[74:75]	16	Vector Arithmetics	7EA04B4A
0x001C80	V_CVT_F64_U32	v[78:79] v121	32	Vector Arithmetics	7E9C2D79
0x001CC4	V_RCP_F64	v[84:85] v[80:81]	16	Vector Arithmetics	7EA84B50
0x001D10	V_CVT_F64_U32	v[80:81] v122	32	Vector Arithmetics	7EA02D7A
0x001D24	V_RCP_F64	v[86:87] v[82:83]	16	Vector Arithmetics	7EAC4B52
0x001D70	V_CVT_F64_U32	v[82:83] v123	32	Vector Arithmetics	7EA42D7B
0x001D84	V_RCP_F64	v[88:89] v[84:85]	16	Vector Arithmetics	7EB84B54
0x001DD0	V_MOV_B32	v76 0	4	Vector Arithmetics	7E980280
0x001DE4	S_AND_SAVEEXEC_B64	s[16:17] s[16:17]	4	Scalar Arithmetics	7E902010
0x001DF8	S_CBRANCH_EXECZ	label_07BA	4/16	Branch	BF88003B
0x001DFC	S_NOP	0x0000	1	Flow Control	BF800000
0x001E00	V_MOV_B32	v82 0	4	Vector Arithmetics	7EA40280
0x001E14	V_MOV_B32	v82 0	4	Vector Arithmetics	7EA40280
0x001EE0	V_CVT_I32_F64	v76 v[76:77]	32	Vector Arithmetics	7E98074C
0x001EE4	V_AND_B32	v99 3 v76	4	Vector Arithmetics	26C69883
label_07BA:					
0x001EE8	S_ANDN2_B64	exec s[16:17] exec	4	Scalar Arithmetics	89FE7E10
0x001EF4	S_CBRANCH_EXECZ	label_0857	4/16	Branch	BF880099
0x001EF8	V_MOV_B32	v82 0			7EA40280
0x001F14	V_CNDMASK_B32	v82 0 v82 vcc			00A4A480
0x001F50	V_FRACT_F64	v[84:85] v[84:85]			7EA86554
0x001F64	V_CMP_GT_F64	vcc 0 v[92:93]			7CC8B880
0x001F70	V_CNDMASK_B32	v93 0 v92 vcc			00B8B880
0x001F74	V_MOV_B32	v92 0			7EB80280
0x001F88	V_CVT_I32_F64	v92 v[92:93]	32	Vector Arithmetics	7EB8075C
0x001F8C	V_CVT_F64_I32	v[93:94] v92	32	Vector Arithmetics	7EBA095C
0x002000	V_CVT_F64_I32	v[110:111] v99	32	Vector Arithmetics	7EDC0963
0x002004	S_NOP	0x0000	1	Flow Control	BF800000
0x002150	V_AND_B32	v84 3 v84	4	Vector Arithmetics	26A8A883
label_0857:					
0x00215C	S_MOV_B64	exec s[16:17]	4	Scalar Arithmetics	BEFE0110
0x002160	V_AND_B32	v85 1 v99	4	Vector Arithmetics	26AAC681
0x002194	S_NOP	0x0000	1	Flow Control	BF800000
0x0022F0	V_CMP_NE_I32	vcc 0 v86	4	Vector Arithmetics	7D8AAC80
0x0022F4	V_CNDMASK_B32	v77 v82 v89 vcc	4	Vector Arithmetics	009A8352

Branch-not-taken takes 4 clock cycles. Branch-taken takes 16 clock cycles (assuming that the branch address is found in the instruction cache)

## Known Issues

For a list of known CodeXL issues, review the release notes on the CodeXL web page and the AMD Developer Tools CodeXL forum:

<https://github.com/GPUOpen-Tools/CodeXL/issues>

## Support

AMD general developer support page:

<http://developer.amd.com/support/>

Tools & SDKs section in AMD Developer Tools website:

<http://developer.amd.com/tools-and-sdks/>

OpenCL Zone in AMD Developer Tools website:

<http://developer.amd.com/tools-and-sdks/opencl-zone/>



AMD Accelerated Parallel Processing OpenCL Programming Guide:

[http://developer.amd.com/wordpress/media/2013/07/AMD Accelerated Parallel Processing OpenCL Programming Guide-rev-2.7.pdf](http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf)

For GPU development issues relating to other AMD tools, see the AMD GPU Developer Tools Forum:

[community.amd.com/community/devgurus/gpu\\_developer\\_tools](http://community.amd.com/community/devgurus/gpu_developer_tools)

To report a specific problem or request help with CodeXL, visit the CodeXL Forum at:

<https://github.com/GPUOpen-Tools/CodeXL/issues>

## DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of non-infringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, AMD Radeon, AMD FirePro, gDEBugger, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL is a trademark of Apple Inc. used by permission by Khronos. OpenGL is a registered trademark of Silicon Graphics, Inc. in the United States and/or other countries worldwide. Microsoft, Windows, DirectX and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other jurisdictions. Vulkan is a registered trademark of Khronos Group Inc. in the United States and/or other jurisdictions. Linux is the registered trademark of Linus Torvalds in the United States and/or other jurisdictions. PCIe and PCI Express are registered trademarks of PCI-SIG Corporation. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.