## GPU Debugger

The CodeXL GPU Debugging module traces application activity that makes use of OpenCL and OpenGL to provide application behavior information necessary for finding bugs and optimizing application performance.

With CodeXL, you can look inside your OpenCL and OpenGL API usage to see the effect individual commands have on application behavior.

CodeXL also lets you debug your OpenCL kernels at runtime, inspect variable values across different work item and work groups, inspect the kernel call stack, and more.

There are different ways to use the analytic capabilities of the CodeXL GPU Debugging Module: from locating bugs through removing redundant calls and errors to performing regression tests.

Whether you want to shorten debugging time, improve application quality, or optimize application performance, CodeXL displays the information you want.

The GPU debugging controls include:

* [**Toolbars**](#_topic_GPUDebuggingToolbars)
* [**Views**](#_topic_GPUDebuggingViews)
* [**Dialogs**](#_topic_GPUDebuggingDialogs)

### GPU Debugging Toolbars

There are two types of GPU debugging toolbars:

* [**Images and Buffers**](#_topic_ImagesandBufferstoolbar)
* [**Current Work Item**](#_topic_CurrentWorkItemtoolbar)

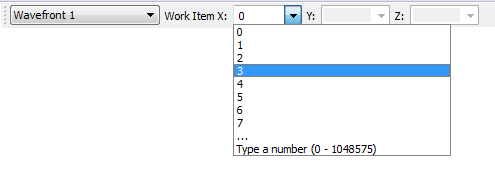
#### Images and Buffers Toolbar

****

The Images and Buffers toolbar provides control over the CodeXL [**object views**](#_topic_Objectviews).

|  |  |  |
| --- | --- | --- |
|  | Select / pan | Change the mouse click operation from selecting pixels to panning the image view. |
|  | Zoom controls | Control the zoom level of the image view. |
|  | Rotation controls | Rotate the image and data views clockwise or counterclockwise. |
|  | Channel Selection controls | For multi-channeled images and textures, select which of the RGBA channels to show in the image and data views. |
|  | Invert | Invert the image view's displayed colors. |
|  | Grayscale | Desaturate the image view's displayed image. |

#### Current Work Item toolbar

****

The Current Work Item toolbar is shown during debugging. Use this toolbar to select the active thread.

When N-Dimensional kernel debugging is taking place (clEnqueueNDRangeKernel), use this toolbar to select the current work item.

* When looking at variables (watch view, locals view, hovering over the variable name), the values shown will be those relevant to this work item.
* When stepping through code, code locations where this work item is not valid will be skipped.
* For coordinates that have more than 8 work items, the last item in the drop down list will show the range of available work items (in the above example, there are 1048576 work items, with indexes ranging between 0 and 1048575).
* To reach a work item whose index is larger than 7, you need to type the index manually and press enter (or switch to another combo box). For example, if you want the X coordinate to be set on the index is 8192, you should type the string 8192 in the X combo box and press enter:



Thread / Wavefront Selection Combo-box

This combo-box contains a list of the host threads currently active in the debugged application. Selecting a thread shows its call stack in the Call Stack view.

During kernel debugging, the active wavefronts are also shown in this combo-box. Selecting a wavefront displays its kernel source call stack.

Note that when debugging with breakpoint emulation, all work-items are gathered in a single virtual wavefront, which does not represent the actual work-item distribution on the hardware.

X, Y, Z Combo-boxes

These combo-boxes are filled with valid values for the work-item coordinates (based on the global work offset and size). If the work dimension is too low, the combo-box is disabled (for example: Z in a 2-dimensional work).

### GPU Debugging Views

There are six GPU debugging views.

* [**API Function Calls History**](#_topic_APIFunctionCallsHistoryview)
* [**Memory view**](#_topic_Memoryview)
* [**OpenCL Multi-Watch**](#_topic_OpenCLMultiWatchviews)
* [**Visual Studio native debugging views**](#_topic_VisualStudionativedebuggingviews)
* [**Statistics view**](#_topic_Statisticsview)
* [**Object views**](#_topic_Objectviews)

#### API Function Calls History View

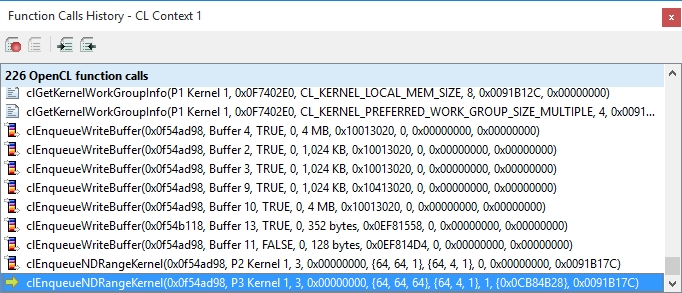
This view displays a log of OpenCL, OpenGL, OpenGL extensions, as well as WGL and glX function calls executed in each context.

Function Calls List

The function calls are displayed as a list, ordered by the time in which they were called. Each list line represents a single function call; it contains:

* the called function name,
* its arguments values, and
* the function type (as an icon).

The function types are marked as shown in the following screenshot.



The icons have the following meanings

|  |  |
| --- | --- |
|  | OpenCL function. |
|  | OpenCL buffer and image function. |
|  | OpenCL queue function. |
|  | OpenGL function. |
|  | OpenGL extension function. |
|  | A WGL function. |
|  | A GLX function. |
|  | OpenGL program and shader function. |
|  | OpenGL texture function. |
|  | OpenGL buffer function. |
|  | GL\_string\_marker\_GREMEDY function. |

The Viewed Render Context

The list title bar shows the viewed context and the number of functions executed in it. Use the CodeXL Debugging Objects Explorer tree view to change the displayed context.

Breaked-On Function

A yellow arrow indicates the function called when the debugged process was suspended. Note that the process is suspended by CodeXL before the suspending function is executed. This lets you use the Step (F10) command to observe the effect its execution has on the API and objects.

Frame Terminators

When frame terminators are specified in the project settings, then during Frame Terminator execution the list in the context is cleared, and the function calls count is reset to 0.

Displaying Function Call Properties

Selecting a list line presents the appropriate function call in the Properties view.

HTML Log File

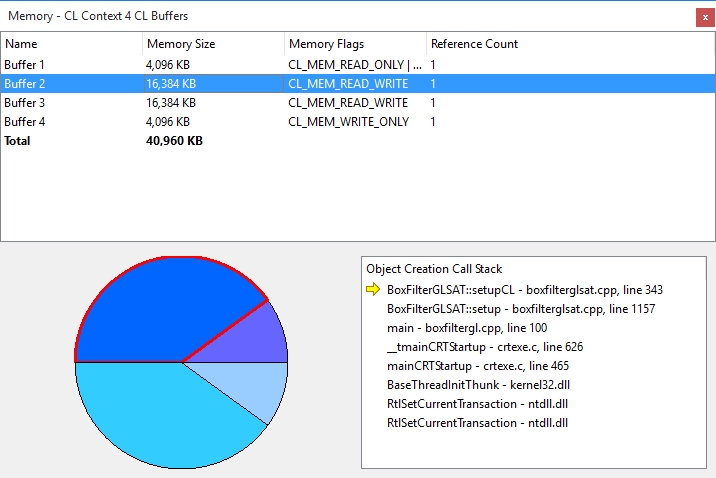
You can save the OpenCL / OpenGL function calls history in an HTML log file using the Record button. After pressing the Record button, a log file is created for each active context. This log file contains the details of each API function call, program and shader sources and image files for image and texture objects (if enabled in the Options dialog). To stop recording, press the record button again. To view the log file gathered so far, press the Open Current HTML Log File button.

#### Memory view

This view gives you information about your compute and graphic memory consumption and usage, as well as detected graphic or compute memory leaks.

Graphic Object Details View

The Graphic Object Details view lists all the graphic memory allocated objects of the selected type in the appropriate context (for example, Context 1 textures). The columns in the list vary depending on the selected item type, displaying information relevant to the current items' memory size.



Object Creation Calls Stack View

This view displays the calls stack when the currently selected item was created. Select a stack frame to view its information. If your application has the appropriate debug information and source code, double-clicking a stack frame opens the Visual Studio Source Code editor, highlighting the appropriate line.

Graph View

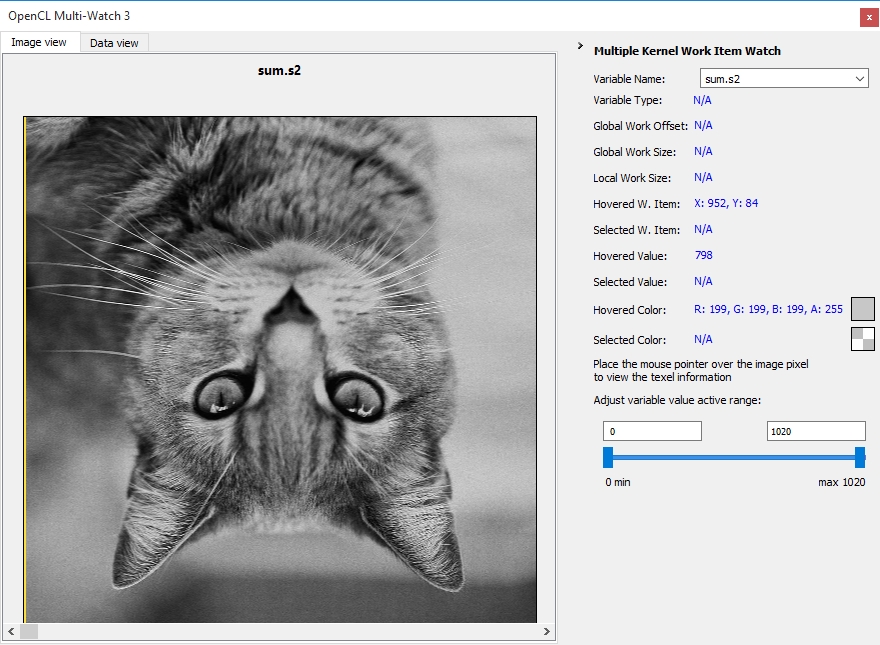
This view contains details of the graphic memory consumed by the objects displayed in the Graphic Objects Details view.

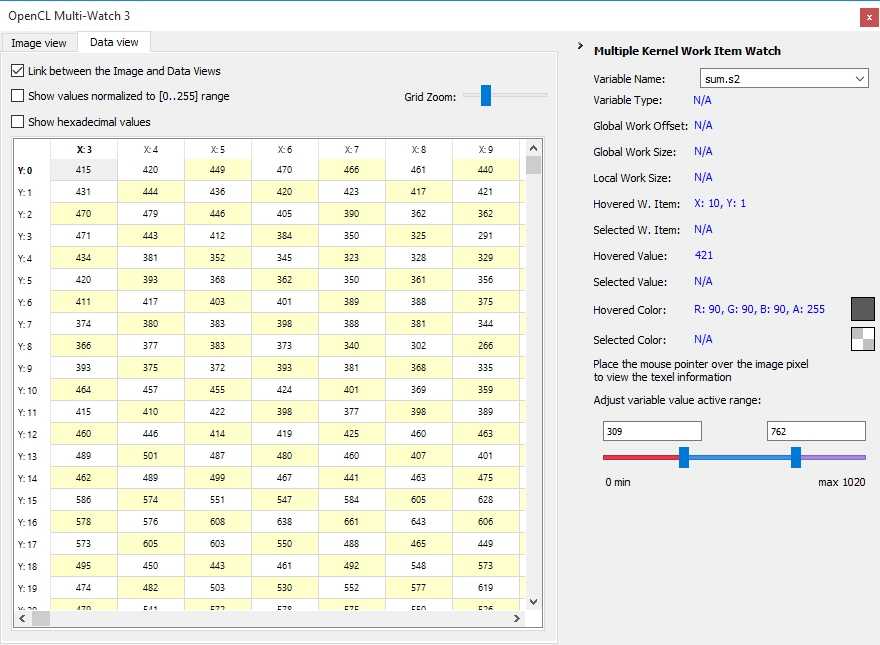
#### OpenCL Multi-Watch Views

This view lets you compare the values of an OpenCL Kernel variable across work-items and work-groups.

Variable Values

The Multiwatch main window displays either a graphical (image view) or spreadsheet (data view) visualization of the selected kernel variable across the various work-items and work-groups. This view operates very similarly to the Object views. The two following screenshots display a multi watch view for the kernel variable sum.s2. The upper screenshot shows an image view, the lower shows a data view.





Variable Name Combo-box

Select a variable name from the list, or type a watch expression. If the expression can be parse and evaluated, the values are updated in the main view.

Kernel Work Geometry and Selected / Hovered Details

Selecting, or hovering with the cursor over, work-items displays the kernel work geometry (local and global work size and offset) for the current N-dimensional kernel execution, as well as their location and value.

Value Range Slider

Shows the value range, and allows marking the valid value range. Values above and below these lines are colored to show they are not in the selected range.

#### Visual Studio Native Debugging Views

Many Visual Studio native views have information from the CodeXL GPU Debugging Engine.

Source Code Window

* During API debugging, CodeXL displays the C/C++ source that led to the API function call in the source code window.
* During OpenCL kernel debugging, CodeXL displays the kernel source code in the source code window. This is done from a project file, if available, or from a temporary file for applications that generate the kernel source at run time.
* Double-clicking an OpenCL kernel or program, or an OpenGL shader in the CodeXL Debugging Object Explorer, displays its source code in the source code window.
* Double-clicking an object allocation call stack frame in the Memory view displays the source location associated with that stack frame in the source code window.

Breakpoints View

* CodeXL API function breakpoints are displayed as C/C++ function breakpoints.
* CodeXL kernel function name breakpoints are displayed as function breakpoints with the prefix Kernel: .
* CodeXL Error / Warning breakpoints appear as function breakpoints, with their descriptive string as the breakpoint.
* Kernel source code breakpoints are displayed as breakpoints in the kernel source file. Note that if a temporary kernel source file is created at runtime, the breakpoints set in it do not work in future runs of the application. To associate a kernel with a source file, include the source file in the debugged project, with a .cl file extension.

Watch View

During OpenCL kernel debugging, enter variable names in the Watch view to see them update during debugging or when switching between work-items in the Work-Item toolbar.

Autos View

During OpenCL kernel debugging, the Autos view displays the values of variables near the program counter, if possible.

Locals View

During OpenCL kernel debugging, the Locals view displays all the available kernel variables in the current scope. An additional variable, KernelDispatchDetails, which indicates the kernel’s work size and the selected work item ID and workgroup, is also displayed.

Call Stack View

During API debugging, the current API function's call stack is displayed in the Call Stack view. During OpenCL kernel debugging, the kernel's call stack is displayed in the Call Stack view. Note that since the kernel is debugged at execution time, the call stack does not contain the clEnqueueNDRangeKernel or clEnqueueTask API function call that started the debugging.

#### Statistics View

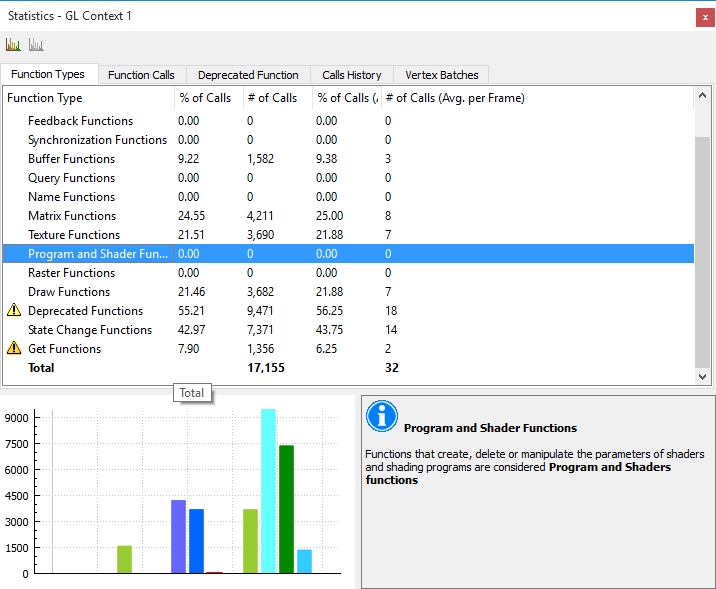
This view lets you view statistical information about your OpenCL and OpenGL APIs usage.

Context Selection

Select an OpenCL or OpenGL context in the CodeXL Debugging Object Explorer to update the statistics to that context's information.

Graph View

This view contains a graphical representation of the information in the Function Types, Function Calls Statistics, State Change Statistics, Deprecated Function Statistics, and Vertex Batch Statistics views.



Properties Box

This box, located in the lower right-hand side of the Statistics view, displays the selected object properties. The properties include information about the object, as well as an explanation of any warnings (represented by exclamation point icons next to the items).

Statistics Tabs

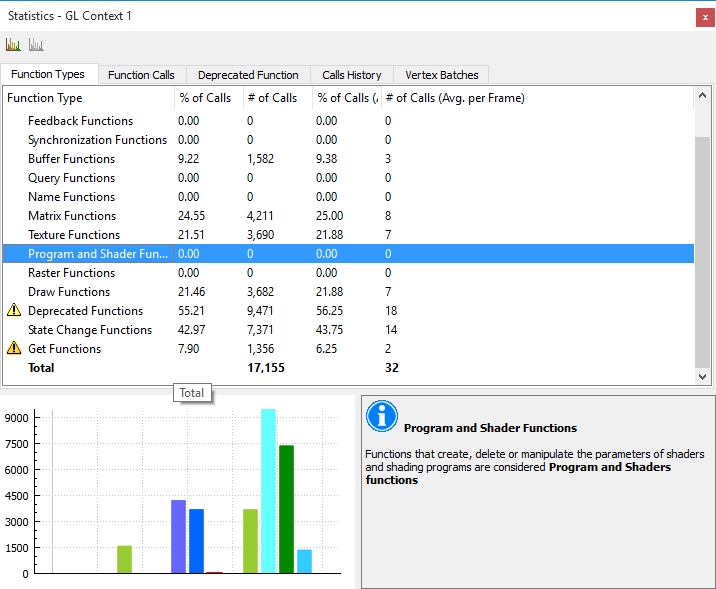
Statistics tabs include:

* [**Function Type Statistics**](#_Function_Type_Statistics) **−** Displays details of OpenCL / OpenGL function calls to categories. Note that a function can belong to multiple or none of the categories. To see which categories a function belongs to, find it in the Function Calls Statistics view.
* [**Function Call Statistics**](#_topic_FunctionCallStatisticsview) **−** Displays a breakdown of all the OpenCL and OpenGL functions used, as well as useful tips and information about unrecommended functions.
* [**Deprecated Function Statistics**](#_topic_DeprecatedFunctionStatisticsview) **−** Displays a breakdown of the usage of functions deprecated by any OpenCL or OpenGL version. In Analyze Mode, this view also displays details about usage of deprecated features in partially deprecated functions.
* [**Calls History**](#_API_Function_Calls) **−** Displays the history of API calls executed for last frame on the current displayed context.
* [**Vertex Batch Statistics**](#_topic_VertexBatchStatisticsview) **−** Displays a breakdown of vertex drawing OpenGL function calls (or vertex batches) by the number of vertices drawn in each.

##### Function Type Statistics view

The Function Types Statistics view displays a breakdown of OpenCL / OpenGL function calls to categories.

Note that a function can belong to multiple or none of the categories. If you wish to know which categories does a function belong to, find it in the Function Calls Statistics view.



Graph and Properties Views

Graph View

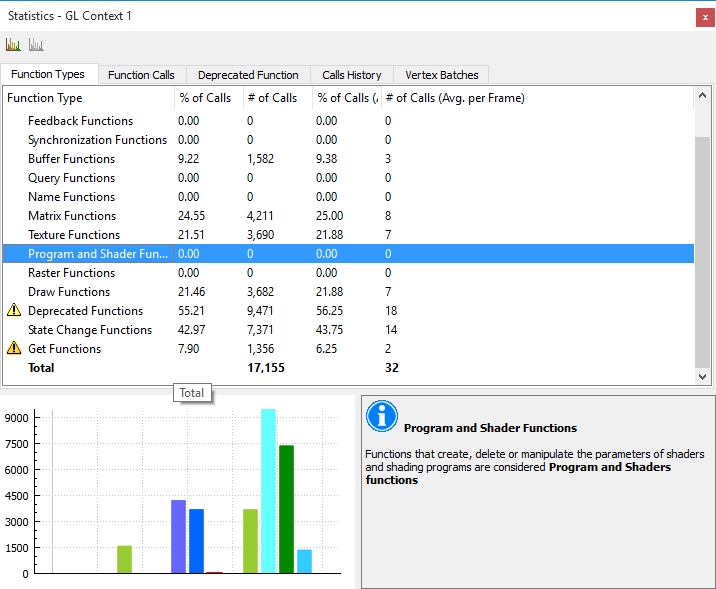
When in the Function Types Statistics view, the Graph view displays a bar graph showing the part each function type takes. The grid lines represent 25, 50, and 75 percent of all function calls. Selecting a function type in the list highlights it in the graph. Redundant State Change functions (when available) are marked in red; the Get functions are marked in orange.

Properties Box

When in the Function Types Statistics view, the Properties box shows information about the currently selected function type.

##### Function Call Statistics View

The Function Calls Statistics view shows the number of times each OpenCL / OpenGL function call was executed in the previously rendered frame, as well as its percentage of the total functions execution.



Detailed Function Calls

For most of the functions, the calls to that function are displayed using a single entry regardless of the arguments passed to the function.

For the following functions, CodeXL provides a separation based on one of the arguments used in the function call to provide more precise information, i.e. they are listed by their enumerators.

* glBegin
* glBindBuffer
* glBindBufferARB
* glBindTexture
* glDisable
* glDisableClientState
* glDrawArrays
* glDrawArraysIndirect
* glDrawArraysInstanced
* glDrawArraysInstancedARB
* glDrawArraysInstancedBaseInstance
* glDrawArraysInstancedEXT
* glDrawElements
* glDrawElementsIndirect
* glDrawElementsInstanced
* glDrawElementsInstancedARB
* glDrawElementsInstancedBaseInstance
* glDrawElementsInstancedBaseVertexBaseInstance
* glDrawElementsInstancedEXT
* glDrawRangeElements
* glDrawTransformFeedback
* glDrawTransformFeedbackInstanced
* glDrawTransformFeedbackStream
* glDrawTransformFeedbackStreamInstanced
* glEnable
* glEnableClientState
* glIsEnabled
* glMultiDrawArrays
* glMultiDrawArraysEXT
* glMultiDrawArraysIndirect
* glMultiDrawElements
* glMultiDrawElementsEXT
* glMultiDrawElementsIndirect
* glTexParameter\*

Exporting Function Calls Statistics Data Into a file

The Function Calls Statistics data can be exported to a file (.csv) using the right-click context menu "Export Function Calls Statistics" command. Exporting the function calls statistics data can help you compare the function calls statistics of different frames. It also allows you to perform regression tests by comparing the function calls statistics data of two versions of your application.

Functions Not Recommended

The Usage of certain OpenCL and OpenGL functions is unrecommended, mostly for performance taxing reasons. These functions are noted as such in this view in varying degrees - mildly unrecommended (yellow warning sign), intermediately unrecommended (orange warning sign) and highly unrecommended (red warning sign). Click on an unrecommended function to display information about it and a better alternative to using it in the Properties view.

Graph and Properties Views

Graph View

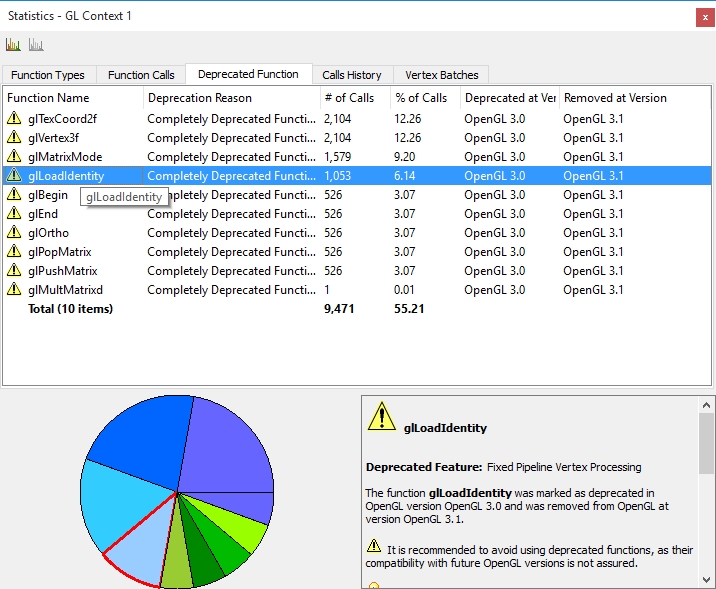
When in the Function Calls Statistics view, the Statistics view Graph view will display a pie chart of the OpenCL / OpenGL function calls. Each pie "slice" is one API function (or in some cases, a combination of an OpenGL function and enumerator). Selecting a function or functions in the list causes their respective slice in the graph to be highlighted. The graph can be rotated by clicking on it and dragging.

Properties Box

When in the Function Calls Statistics view, the Statistics view Properties Box will display the function name. If the function is an unrecommended one, the Properties Box will also display an explanation of why using this function is unrecommended, as well as a recommendation for an alternative to using this function.

##### Deprecated Function Statistics view

The Deprecated Function Statistics view displays information about your usage of functions deprecated by any OpenCL or OpenGL version.



Each of the lines in the list displays the usage of one Deprecated function or a combination of a partially deprecated function and a deprecated feature. The number of calls to this function or deprecated uses of this function and feature combination is displayed (as well as a percentage of the total API function calls). Finally, the OpenCL / OpenGL version when this feature was deprecated and the version it was removed (if any) are displayed.

Graph and Properties Views

Graph View

When in the Deprecated Function Statistics view, the Statistics view Graph view displays a pie chart, representing a breakdown of the deprecated function calls per combination of deprecated function and deprecated feature.

Properties View

When in the Deprecated Function Statistics view, the Statistics view Properties view displays the currently selected deprecated function's name, as well as information about the deprecated feature it is a part of, and other functions and behaviors belonging to the same deprecated feature. A forward-compatible alternative to this feature is supplied in bold font.

##### API Function Calls History View

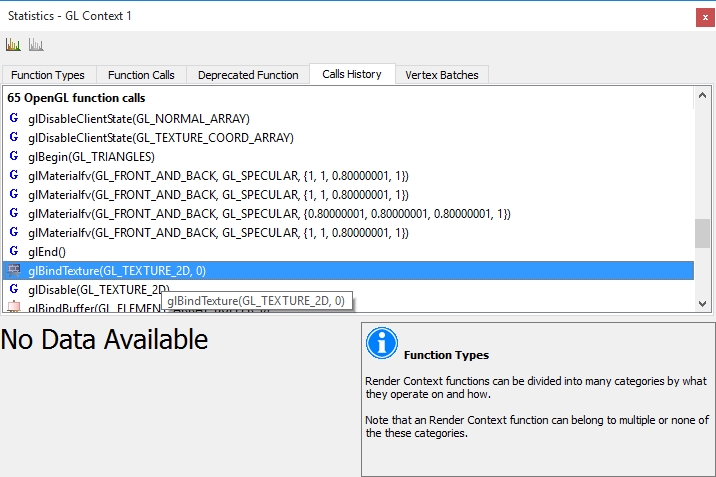
This view displays a log of OpenCL, OpenGL, OpenGL extensions, as well as WGL and glX function calls executed in each context.

Function Calls List

The function calls are displayed as a list, ordered by the time in which they were called. Each list line represents a single function call; it contains:

* the called function name,
* its arguments values, and
* the function type (as an icon).

The function types are marked as shown in the following screenshot.

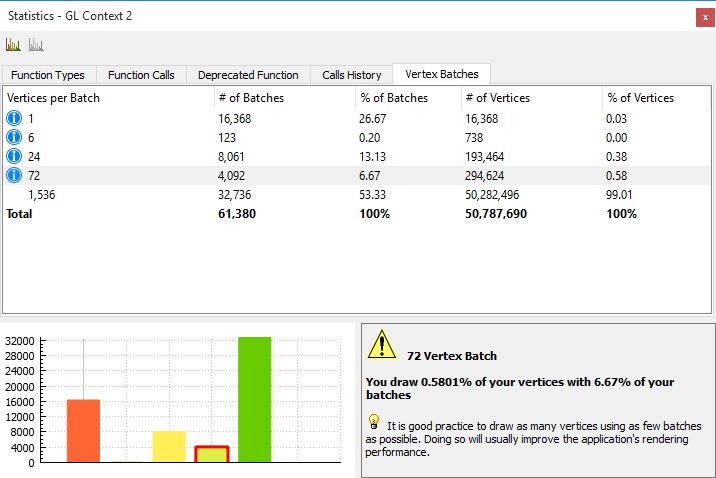


The icons have the following meanings

|  |  |
| --- | --- |
|  | OpenCL function. |
|  | OpenCL buffer and image function. |
|  | OpenCL queue function. |
|  | OpenGL function. |
|  | OpenGL extension function. |
|  | A WGL function. |
|  | A GLX function. |
|  | OpenGL program and shader function. |
|  | OpenGL texture function. |
|  | OpenGL buffer function. |
|  | GL\_string\_marker\_GREMEDY function. |

##### Vertex Batch Statistics view

The Vertex Batch Statistics view displays information about your usage of OpenGL vertex drawing functions or vertex batches, divided by the batch size (i.e. how many vertices were drawn with a single OpenGL function call).



Vertex Batches

Each of the lines in the view displays statistics for a range of vertex batch sizes (normal view) or a specific batch size (detailed view). The number and percentage of batches of this size represents the "cost" of drawing with this batch size. The number and percentage of vertices represent the "benefit" gained from drawing with this batch size. Batches which are small in relation to the application (have a higher percentage of batches than is expected from their percentage of vertices) are noted with a warning icon.

Show Detailed Batch Statistics

Toggles between the Vertex Batch Statistics view's normal (gather batch sizes into ranges by amount) and detailed (show each vertex batch size separately) views.

Graph and Properties Views

Graph View

When in the Vertex Batch Statistics view, the Statistics view Graph view displays a bar chart, in which each bar is a range of batch sizes. The height of the bar is the number of batches drawn in this range, and the bar's color represents the range's position in the application's distribution (Red bars are the smallest batches and green bars are the largest ones). The grid lines represent 25, 50 and 75 percent of vertex drawing function calls. When the Vertex Batch Statistics shows the detailed view, this graph becomes a histogram of the vertex batch sizes.

Properties View

When in the Vertex Batch Statistics view, the Statistics view Properties view displays the currently selected batch size range, the percentage statistics for this range and a short explanation about vertex batches.

#### Object Views

The Object views - Image view and Data view - allow you to view all OpenCL buffers and images, and OpenGL textures, static buffers, VBOs (vertex buffer objects), FBOs (frame buffer objects), render buffers and pbuffers (pixel buffers).

Each object can be viewed both as an image in the Image view (except OpenCL Buffers and OpenGL VBOs) and as a spreadsheet containing the object raw data in the Data view. The object properties and parameters will appear in the properties view.

[**Image View**](#_topic_ObjectImageview)

[**Data View**](#_topic_ObjectDataview)

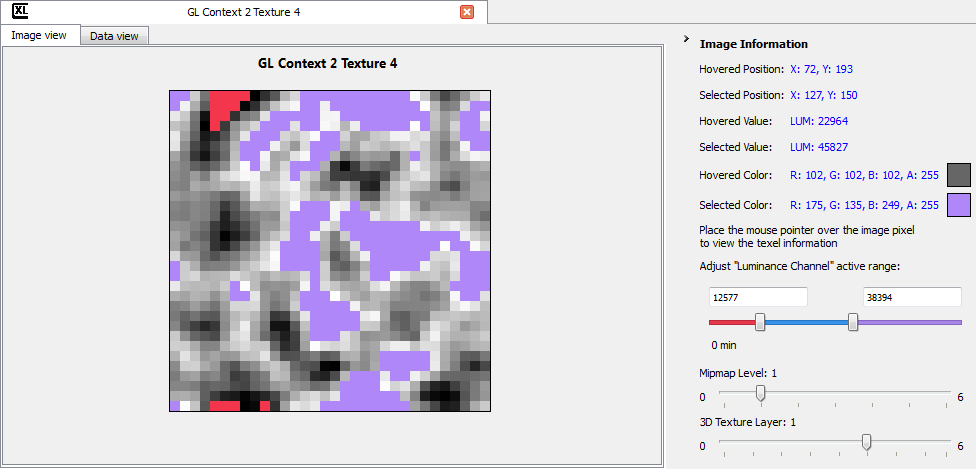
Information Panel

The information panel helps you make fast inquiries and adjustments to the currently viewed object pixels/texels.

The information panel is composed of a few main elements (which are only shown when relevant:

Current and Previous Pixel

When hovering with the mouse over the Image view, the currently highlighted pixel data is displayed as the current pixel; Both RGBA values and raw data value are displayed. If you wish to compare between two different pixels, you can save a specific pixel value by clicking on it with the left mouse button. The pixel data will be stored as the "Selected Pixel", allowing you to make fast comparison to another pixel.



Single Channel Range Adjustment Slider

When viewing an object that has a single component data format ("Depth" / "Luminance" / "Intensity" / etc.) the image view translates these values to grayscale values. Use the single channel range adjustment slider to adjust the data translation to grayscale values.

3D Layer Slider

When viewing 3D images or textures, you can scroll through the 2D images that make up the 3D image using the 3D layer slider located in the information panel.

Texture Array Layer Slider

When viewing texture arrays, you can scroll through the textures that make up the array using the Texture array layer slider located in the information panel.

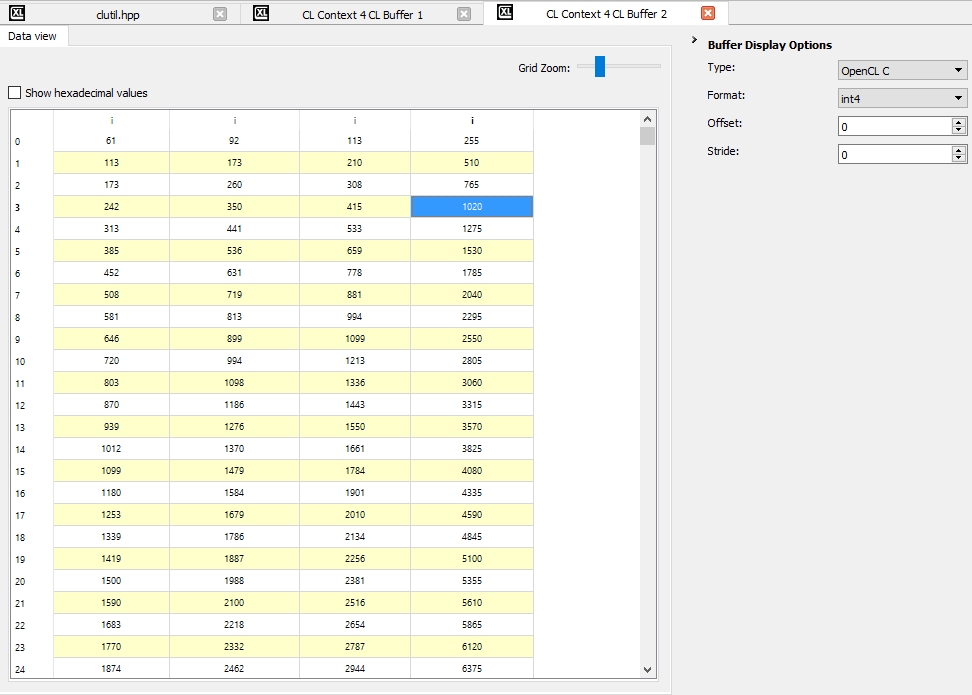
Texture Mipmap Slider

When viewing a texture which has automatic or manual mipmaps defined, you can scroll through the different texture levels using the Mipmap slider located inside the information panel.

OpenCL buffer / OpenGL VBO Format Controls

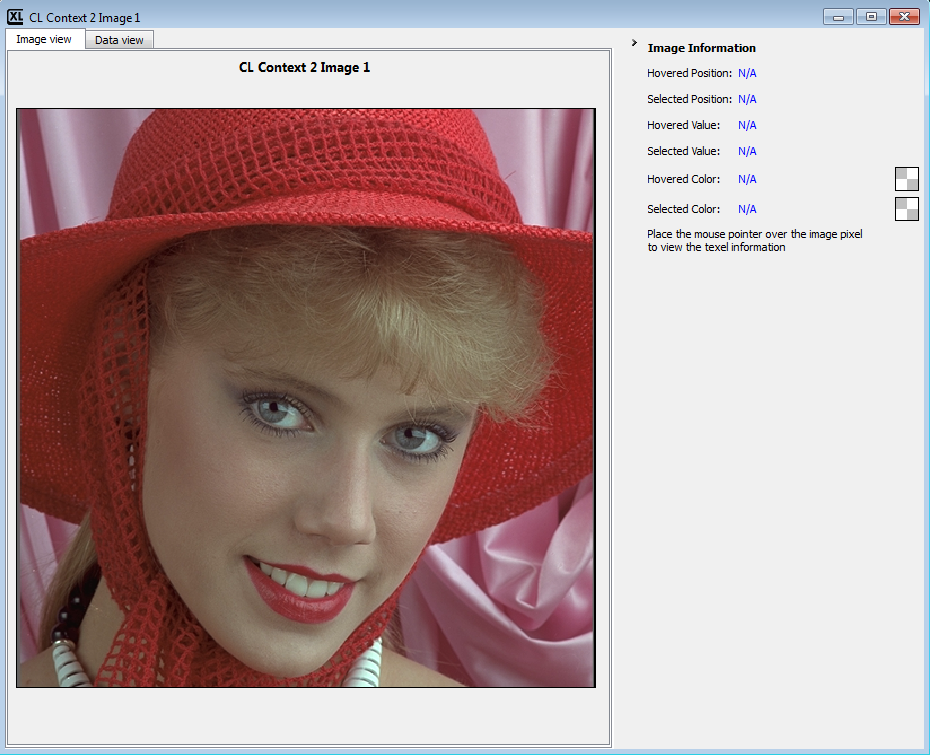
When viewing an OpenCL buffer or an OpenGL VBO, the information panel will contain several items that allow choosing how the VBO data is displayed:

* Data combo-box - choose which kind of data the buffer contains. Each option in this combo-box matches either the OpenCL C types or an OpenGL gl\*Pointer function ("Interleaved" matches glInterleavedArrays). Choose "All" to display all formats. An exception to this is the "Index" option which also matches the glDrawElements (and variants) function's indices parameter and not only glIndexPointer.
* Format combo-box - choose the format the data is stored in, matching the OpenCL C type the kernel uses (\_\_global XXXX\*) size and type parameters of the gl\*Pointer function or the format parameter of glInterleavedArrays. The formats are named as data-components-format, for example C4UB stands for color data stored in 4 components of unsigned byte type.
* Offset spin control - choose the offset from the beginning of the VBO to the start of the data. This should match the pointer parameter of the gl\*Pointer function.
* Stride spin control - choose the stride (space in bytes between the end of one vertex's data and the next one's) to match the stride parameter of the gl\*Pointer function.



##### Object Image view

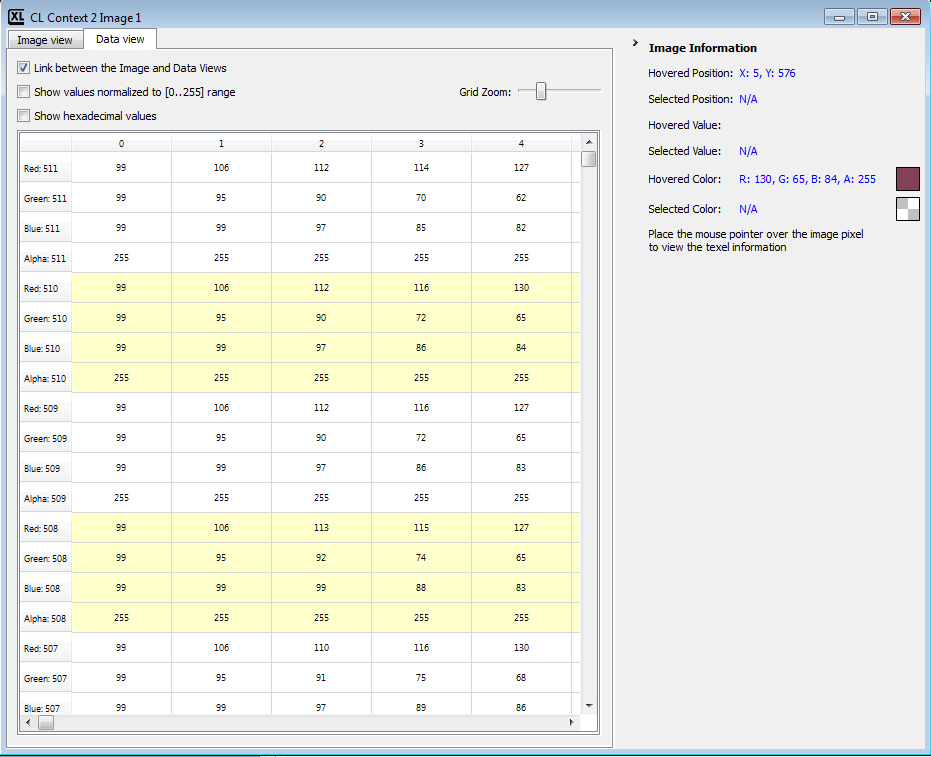
You can view an image, a buffer or a texture object loaded image using the Image view.



Images and Buffers toolbar



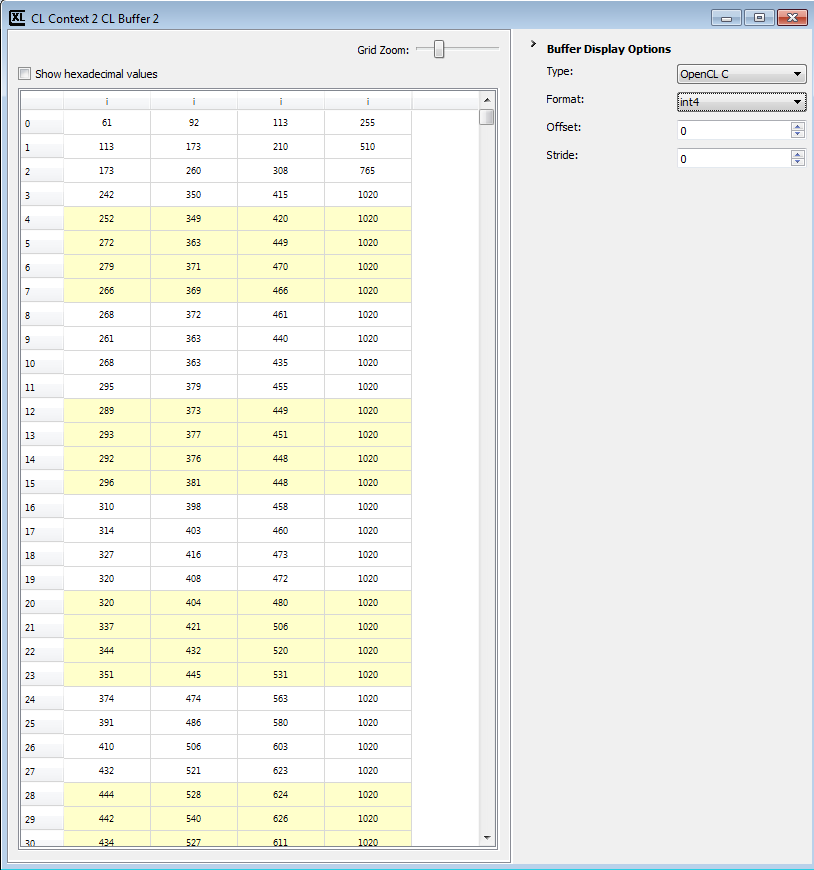
Use the Images and Buffers toolbar to control the Image view



Click on the data tab to see the image data

##### Object Data view

You can view a buffer, an image or a texture object raw data using the Data view.



Grid Zoom

The grid zoom slider allows you to set the zoom level of the displayed raw data cells.

Link Image and Data Views

When enabled, clicking on a pixel at the image view will select this pixel at the data view grid.

Show Normalized Values

The values showed in the grid are the original data values of the object as held by OpenCL and OpenGL. These values may be in various data formats such as OpenCL C types, or GL\_FLOAT, GL\_INT, GL\_SHORT and GL\_BYTE. When enabled, the data view will display the values in the grid normalized to the GL\_BYTE type ([0..255] range), regardless of the original data type.

Viewing OpenCL buffers and OpenGL Vertex Buffer Objects (VBOs)

These buffers can only be shown in the Data View. After choosing the VBO's data, format, offset and stride; each vertex's data will be shown as a separate line in the view. The column headers represent the meaning of each value - X, Y, Z, W for vertex position; Nx, Ny, Nz for normal direction; R, G, B, A for vertex color and S, T, R, Q for texture coordinates.

### GPU Debugging Dialogs

This section provides information on:

* [**Adding / removing breakpoints**](#_topic_AddRemoveCodeXLBreakpointsDialog)
* [**GPU debugging project settings**](#_topic_GPUDebuggingProjectsettings)
* [**GPU debugging global settings**](#_topic_GPUDebuggingGlobalsettings)

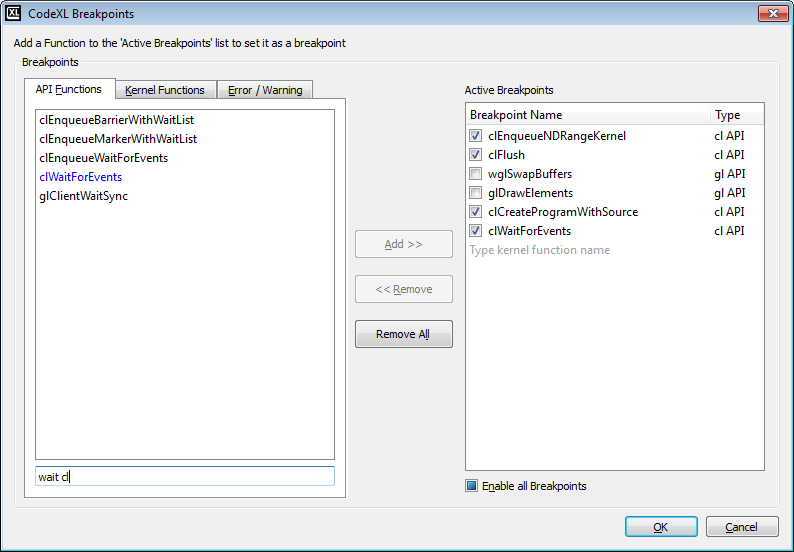
#### Add / Remove CodeXL Breakpoints Dialog

The Breakpoint dialog lets you choose OpenCL and OpenGL API function breakpoints, as well as kernel function name breakpoints.

Breakpoints added this way are added as C/C++ function breakpoints to the Visual Studio Breakpoints view. Adding an OpenCL or OpenGL function as a Visual Studio breakpoint (Debug > Breakpoints > New Breakpoint...) adds it to this dialog.

The debugged process execution is suspended when a debugged process thread calls an API function marked as a breakpoint. The process run is suspended before the function is executed; this lets you observe the effect the breakpoint function has on the application behavior.

The debugged process execution is suspended when a kernel with a function name marked as a breakpoint starts executing. This lets you start kernel debugging without having to use the clEnqueueNDRangeKernel function call.



Breakpoints

|  |  |
| --- | --- |
| API Functions | Contains OpenCL and OpenGL API functions, as well as platform-specific binding functions (WGL, CGL, GLX, EAGL, or EGL) supported by CodeXL. |
| Kernel Functions | When an OpenCL application is debugged, contains the names of all kernel functions that are used by the application. |
| Error / Warning | These are special breakpoints that can be set on the occurrence of special events, such as:   * Break on OpenGL error - This is hit whenever an OpenGL function generates an error. The output window displays the error code. Note that if your application uses glGetError for validation, its flow might change with this option on. * Break on OpenCL error – This is hit whenever an OpenCL function returns an error value. The output window displays the error code. * Break on Detected error - The CodeXL OpenCL and OpenGL servers sometimes detect errors that are not mentioned in the specifications. This breakpoint is hit when such an error is detected. The output window displays the error details. * Break on Deprecated function - OpenGL 3.0 and up and OpenCL 1.1 and up include functions and behaviors that are marked as deprecated and must be removed to maintain forward compatibility. This breakpoint is hit when such a function is called. |
| Text Filter | Enables filtering the Breakpoints list. Entering multiple values separated by spaces or commas searches for function names containing all the strings. |
| Active Breakpoints | Contains the currently selected API and kernel function breakpoints. You can Enable/Disable a specific breakpoint by checking / unchecking the box next to the function name. You also can type in kernel function names to add kernels that were not created yet in the application. |
| Add Breakpoint | To add a breakpoint, select one or more functions from the API Functions or Kernel Functions list, and add it to the Active Breakpoints list by double-clicking on it, or by pressing the **Add** button. |
| Remove Breakpoint | To remove a breakpoint, select the function from the Active Breakpoints list, and remove it by double-clicking on it or by pressing the **Remove** button. |
| Remove all Breakpoints | Press this button to remove all of the active breakpoints. |

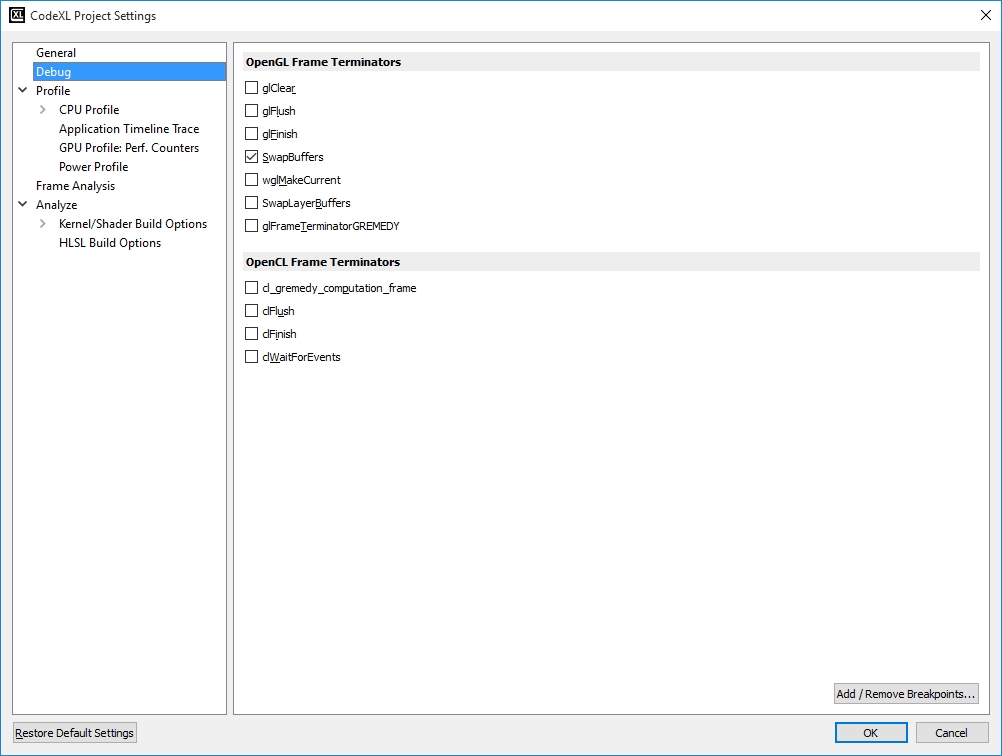
Settings

Enable /Disable all Function Breakpoints

This section describes how to change the enabled status of all active breakpoints.

#### GPU Debugging Project settings

Use the File > Project Settings menu (Ctrl-P) to open the project settings dialog. Navigate to the “Debug” node in the project settings tree displayed in the left panel of the dialog.



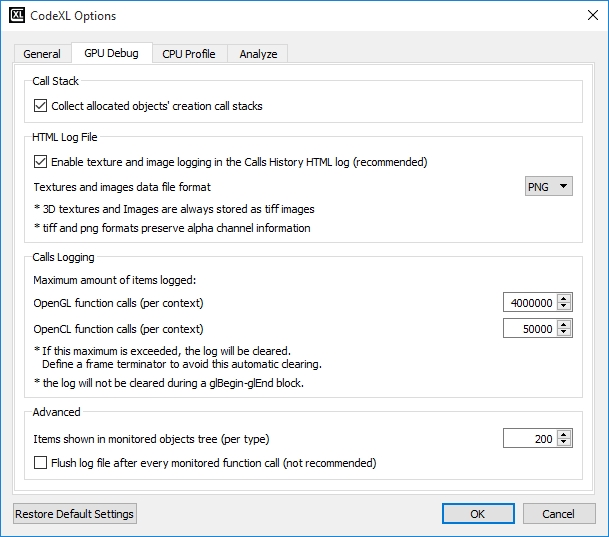
|  |  |
| --- | --- |
| OpenCL Frame Terminators | OpenCL frame terminators are the functions that end your application computation frame. They let you view the log of a single computation frame, not the entire calls log. Available Frame Terminators are: cl\_gremedy\_computation\_frame, clFlush, clFinish, and clWaitForEvents. |
| OpenGL Frame Terminators | OpenGL frame terminators are the functions that end your application render frame. They let you view the log of a single render frame, not the entire calls log. Available Frame Terminators are: glClear, glFlush, glFinish, wglSwapBuffers, wglMakeCurrent, wglSwapLayerBuffers, and glFrameTerminatorGREMEDY.  For example:  glFlush is usually chosen for single-buffered applications.  wglSwapBuffers is usually chosen for double-buffered applications.  You must select at least one OpenGL Frame Terminator. See **Frames and Frame Terminators** for more details. |
| Add / Remove Breakpoints | Click this button to open the [Add / Remove CodeXL Breakpoints dialog](#_topic_AddRemoveCodeXLBreakpointsDialog). |

#### GPU Debugging Global Settings

The GPU Debug global settings are set when the project is initially created, and affect every session.

To set the options:

1. In the CodeXL menu, click **CodeXL Options**.   
   The Edit CodeXL Global Settings dialog box is displayed.
2. Select the GPU Debug tab. See the descriptions below for each of the global debug settings.
3. Click **OK** to activate the new options and close the dialog box, or click **Restore Default Settings** to reset the dialog box selections to the system default, or click **Cancel** to close without changes.



Call Stack

|  |  |
| --- | --- |
| Additional Source Code Directories | When specifying additional source code directories, If the source code file is not found at its debug information path, it will searched, by file name only, in the additional source code directories. The search is performed according to the order in which the directories are specified. The additional source code directories are semicolon separated. |
| Source Code Root Location | When specifying a source code root location, the root location is prefixed to each source code file path. Only one source code root location can be specified. Example: If the source code root is "D:\Dir1\" and the debug information source code path is "\Dir2\Dir3\MyFile.cpp", the Source Code editor input path will be "D:\Dir1\Dir2\Dir3\MyFile.cpp". |
| Collect Allocated Objects' Creation Calls Stacks | Deselect this checkbox if you don't want CodeXL to collect the graphic memory allocated objects creation calls stacks (which are displayed in the Memory View). This can improve Debug Mode performance in some cases when many graphic memory objects are allocated. |

HTML Log File

|  |  |
| --- | --- |
| Enable Texture Images Logging in the Calls History HTML Log | When this box is checked, textures data is saved and displayed in the calls log file. |
| Textures Data File Format | You can select the format by choosing the appropriate radio button. Available formats are: JPG, BMP, PNG, and TIFF. 3D textures are stored as tiff images. |

Calls Logging

Set the logging limit for OpenGL API function calls and OpenCL API function calls. If this limit is exceeded, the log is cleared. Define frame terminators in the project settings to avoid this automatic clearing.

Advanced

|  |  |
| --- | --- |
| Floating-Point Precision | The maximum number of significant digits that will be displayed in the Object and Multi-Watch views. |
| Flush Log File After Every Monitored Function Call | When the Flush log file after every monitored function call check box is checked, CodeXL will flush the OpenCL / OpenGL calls history log file after every API function call instead of not use memory cached batches. This feature can help tracking the function call that led to a debugged application crash. Using this feature dramatically slows down the debugged application performance and therefore it is not recommended for regular use. |
| Restore Default Settings | Click this button to restore all settings on all pages to their default values. |

### GPU Debugging API Support

CodeXL GPU debugging supports OpenCL up to and including version 2.0, OpenGL up to and including version 4.5 (including compatibility profiles) and a large variety of OpenCL and OpenGL extensions.

Kernel debugging is not supported for OpenCL 2.0 kernels.

The supported extensions include:

OpenCL Extensions

|  |  |
| --- | --- |
| **Extension name** | **Support level** |
| cl\_khr\_3d\_image\_writes | Full |
| cl\_khr\_byte\_addressable\_store | Full |
| cl\_khr\_dx9\_media\_sharing | Full |
| cl\_khr\_d3d10\_sharing | Full |
| cl\_khr\_d3d11\_sharing | Full |
| cl\_khr\_fp64 | Full |
| cl\_khr\_gl\_sharing | Full |
| cl\_khr\_icd | Full |
| cl\_khr\_global\_int32\_base\_atomics | Standard \* |
| cl\_khr\_global\_int32\_extended\_atomics | Standard \* |
| cl\_khr\_local\_int32\_base\_atomics | Standard \* |
| cl\_khr\_local\_int32\_extended\_atomics | Standard \* |
| cl\_ext\_atomic\_counters\_32 | Standard \* |
| cl\_ext\_cl20\_atomics | Standard \* |
| cl\_ext\_cl20\_svm | Standard \* |
| cl\_ext\_device\_fission | Standard |
| cl\_amd\_c11\_atomics | Standard \* |
| cl\_amd\_device\_attribute\_query | Standard |
| cl\_amd\_fp64 | Full |
| cl\_amd\_media\_ops | Standard |
| cl\_amd\_popcnt | Standard \* |
| cl\_amd\_printf | Standard \* |
| cl\_amd\_svm | Standard \* |
| cl\_amd\_vec3 | Standard |
| cl\_nv\_d3d9\_sharing | Standard |
| cl\_nv\_d3d10\_sharing | Standard |
| cl\_nv\_d3d11\_sharing | Standard |
| \* Using this extension in a kernel might prevent the kernel from being debuggable. If kernel debugging is attempted, a message notifying the user that the kernel uses unsupported features will be displayed. | |

OpenGL extensions

|  |  |
| --- | --- |
| **Extension name** | **Support level** |
| GL\_ARB\_arrays\_of\_arrays | Standard |
| GL\_ARB\_base\_instance | Full |
| GL\_ARB\_blend\_func\_extended | Standard |
| GL\_ARB\_clear\_buffer\_object | Standard |
| GL\_ARB\_color\_buffer\_float | Standard |
| GL\_ARB\_compatibility | Full |
| GL\_ARB\_compressed\_texture\_pixel\_storage | Standard |
| GL\_ARB\_compute\_shader | Full |
| GL\_ARB\_conservative\_depth | Standard |
| GL\_ARB\_copy\_buffer | Full |
| GL\_ARB\_copy\_image | Standard |
| GL\_ARB\_debug\_output | Full |
| GL\_ARB\_debug\_output | Full |
| GL\_ARB\_depth\_buffer\_float | Standard |
| GL\_ARB\_depth\_clamp | Full |
| GL\_ARB\_depth\_clamp | Standard |
| GL\_ARB\_depth\_texture | Standard |
| GL\_ARB\_draw\_buffers | Standard |
| GL\_ARB\_draw\_buffers\_blend | Standard |
| GL\_ARB\_draw\_elements\_base\_vertex | Full |
| GL\_ARB\_draw\_elemnts\_base\_vertex | Full |
| GL\_ARB\_draw\_indirect | Full |
| GL\_ARB\_draw\_instanced | Full |
| GL\_ARB\_ES2\_compatibility | Standard |
| GL\_ARB\_ES3\_compatibility | Standard |
| GL\_ARB\_explicit\_attrib\_location | Standard |
| GL\_ARB\_explicit\_uniform\_location | Standard |
| GL\_ARB\_fragment\_coord\_conventions | Full |
| GL\_ARB\_fragment\_coord\_conventions | Standard |
| GL\_ARB\_fragment\_layer\_viewport | Standard |
| GL\_ARB\_fragment\_program | Standard |
| GL\_ARB\_fragment\_program\_shadow | Standard |
| GL\_ARB\_fragment\_shader | Full |
| GL\_ARB\_framebuffer\_no\_attachments | Standard |
| GL\_ARB\_framebuffer\_object | Full |
| GL\_ARB\_framebuffer\_sRGB | Full |
| GL\_ARB\_geometry\_shader4 | Full |
| GL\_ARB\_geometry\_shader4 | Full |
| GL\_ARB\_get\_program\_binary | Standard |
| GL\_ARB\_gpu\_shader\_fp64 | Full |
| GL\_ARB\_gpu\_shader5 | Full |
| GL\_ARB\_half\_float\_pixel | Standard |
| GL\_ARB\_half\_float\_vertex | Standard |
| GL\_ARB\_instanced\_arrays | Full |
| GL\_ARB\_instanced\_arrays | Full |
| GL\_ARB\_internalformat\_query | Standard |
| GL\_ARB\_internalformat\_query2 | Standard |
| GL\_ARB\_invalidate\_subdata | Standard |
| GL\_ARB\_map\_buffer\_alignment | Standard |
| GL\_ARB\_map\_buffer\_range | Standard |
| GL\_ARB\_matrix\_palette | Standard |
| GL\_ARB\_multi\_draw\_indirect | Full |
| GL\_ARB\_multisample | Standard |
| GL\_ARB\_multitexture | Full |
| GL\_ARB\_occlusion\_query | Standard |
| GL\_ARB\_occlusion\_query2 | Standard |
| GL\_ARB\_pixel\_buffer\_object | Standard |
| GL\_ARB\_point\_parameters | Full |
| GL\_ARB\_point\_sprite | Standard |
| GL\_ARB\_program\_interface\_query | Standard |
| GL\_ARB\_provoking\_vertex | Full |
| GL\_ARB\_provoking\_vertex | Standard |
| GL\_ARB\_robust\_buffer\_access\_behavior | Standard |
| GL\_ARB\_sample\_shading | Standard |
| GL\_ARB\_sampler\_objects | Standard |
| GL\_ARB\_seamless\_cube\_map | Full |
| GL\_ARB\_seamless\_cube\_map | Standard |
| GL\_ARB\_separate\_shader\_objects | Standard |
| GL\_ARB\_shader\_atomic\_counters | Full |
| GL\_ARB\_shader\_bit\_encoding | Standard |
| GL\_ARB\_shader\_image\_load\_store | Standard |
| GL\_ARB\_shader\_image\_size | Standard |
| GL\_ARB\_shader\_objects | Full |
| GL\_ARB\_shader\_precision | Standard |
| GL\_ARB\_shader\_storage\_buffer\_object | Standard |
| GL\_ARB\_shader\_subroutine | Standard |
| GL\_ARB\_shading\_language\_100 | Full |
| GL\_ARB\_shading\_language\_420pack | Standard |
| GL\_ARB\_shading\_language\_packing | Standard |
| GL\_ARB\_shadow | Full |
| GL\_ARB\_shadow\_ambient | Standard |
| GL\_ARB\_stencil\_texturing | Standard |
| GL\_ARB\_sync | Full |
| GL\_ARB\_sync | Full |
| GL\_ARB\_tessellation\_shader | Full |
| GL\_ARB\_texture\_border\_clamp | Full |
| GL\_ARB\_texture\_buffer\_object | Full |
| GL\_ARB\_texture\_buffer\_object\_rgb32 | Standard |
| GL\_ARB\_texture\_buffer\_range | Standard |
| GL\_ARB\_texture\_compression | Full |
| GL\_ARB\_texture\_compression\_rgtc | Standard |
| GL\_ARB\_texture\_cube\_map | Full |
| GL\_ARB\_texture\_cube\_map\_array | Full |
| GL\_ARB\_texture\_env\_add | Full |
| GL\_ARB\_texture\_env\_combine | Full |
| GL\_ARB\_texture\_env\_crossbar | Full |
| GL\_ARB\_texture\_env\_dot3 | Full |
| GL\_ARB\_texture\_float | Standard |
| GL\_ARB\_texture\_gather | Standard |
| GL\_ARB\_texture\_mirrored\_repeat | Full |
| GL\_ARB\_texture\_multisample | Full |
| GL\_ARB\_texture\_multisample | Standard |
| GL\_ARB\_texture\_non\_power\_of\_two | Full |
| GL\_ARB\_texture\_query\_levels | Standard |
| GL\_ARB\_texture\_rectangle | Full |
| GL\_ARB\_texture\_rg | Standard |
| GL\_ARB\_texture\_rgb10\_a2ui | Standard |
| GL\_ARB\_texture\_storage | Standard |
| GL\_ARB\_texture\_storage\_multisample | Standard |
| GL\_ARB\_texture\_swizzle | Standard |
| GL\_ARB\_texture\_view | Standard |
| GL\_ARB\_timer\_query | Standard |
| GL\_ARB\_transform\_feedback\_instanced | Standard |
| GL\_ARB\_transform\_feedback2 | Standard |
| GL\_ARB\_transform\_feedback3 | Standard |
| GL\_ARB\_transpose\_matrix | Full |
| GL\_ARB\_uniform\_buffer\_object | Full |
| GL\_ARB\_vertex\_array\_bgra | Full |
| GL\_ARB\_vertex\_array\_bgra | Standard |
| GL\_ARB\_vertex\_array\_object | Full |
| GL\_ARB\_vertex\_attrib\_64bit | Standard |
| GL\_ARB\_vertex\_attrib\_binding | Standard |
| GL\_ARB\_vertex\_blend | Full |
| GL\_ARB\_vertex\_buffer\_object | Standard |
| GL\_ARB\_vertex\_program | Standard |
| GL\_ARB\_vertex\_shader | Full |
| GL\_ARB\_vertex\_type\_2\_10\_10\_10\_rev | Standard |
| GL\_ARB\_viewport\_array | Standard |
| GL\_ARB\_window\_pos | Full |
| GL\_EXT\_bgra | Full |
| GL\_EXT\_bindable\_uniform | Full |
| GL\_EXT\_blend\_logic\_op | Full |
| GL\_EXT\_blend\_minmax | Full |
| GL\_EXT\_blend\_subtract | Full |
| GL\_EXT\_compiled\_vertex\_array | Full |
| GL\_EXT\_direct\_state\_access | Full |
| GL\_EXT\_draw\_instanced | Full |
| GL\_EXT\_framebuffer\_blit | Full |
| GL\_EXT\_framebuffer\_multisample | Full |
| GL\_EXT\_framebuffer\_object | Full |
| GL\_EXT\_geometry\_shader4 | Full |
| GL\_EXT\_multi\_draw\_arrays | Full |
| GL\_EXT\_packed\_pixels | Full |
| GL\_EXT\_stencil\_two\_side | Full |
| GL\_EXT\_texture | Full |
| GL\_EXT\_texture\_array | Full |
| GL\_EXT\_texture\_buffer\_object | Full |
| GL\_EXT\_texture\_integer | Full |
| GL\_EXT\_texture\_rectangle | Full |
| GL\_EXT\_texture\_shared\_exponent | Full |
| GL\_EXT\_texture3D | Full |
| GL\_EXT\_vertex\_shader | Standard |
| GL\_AMD\_debug\_output | Full |
| GL\_AMDX\_debug\_output | Full |
| GL\_APPLE\_aux\_depth\_stencil | Full |
| GL\_APPLE\_client\_storage | Full |
| GL\_APPLE\_element\_array | Full |
| GL\_APPLE\_fence | Standard |
| GL\_APPLE\_float\_pixels | Full |
| GL\_APPLE\_flush\_buffer\_range | Standard |
| GL\_APPLE\_flush\_render | Standard |
| GL\_APPLE\_object\_purgeable | Standard |
| GL\_APPLE\_packed\_pixels | Full |
| GL\_APPLE\_pixel\_buffer | Standard |
| GL\_APPLE\_specular\_vector | Full |
| GL\_APPLE\_texture\_range | Standard |
| GL\_APPLE\_transform\_hint | Full |
| GL\_APPLE\_vertex\_array\_object | Standard |
| GL\_APPLE\_vertex\_array\_range | Standard |
| GL\_APPLE\_vertex\_program\_evaluators | Full |
| GL\_APPLE\_ycbcr\_422 | Full |
| GL\_ATI\_draw\_buffers | Standard |
| GL\_ATI\_fragment\_shader | Standard |
| GL\_ATI\_text\_fragment\_shader | Standard |
| GL\_GREMEDY\_frame\_terminator | Full |
| GL\_GREMEDY\_string\_marker | Full |
| GL\_HP\_occlusion\_test | Standard |
| GL\_KHR\_debug | Standard |
| GL\_NV\_fragment\_program | Standard |
| GL\_NV\_fragment\_program\_option | Standard |
| GL\_NV\_fragment\_program2 | Standard |
| GL\_NV\_geometry\_shader4 | Full |
| GL\_NV\_occlusion\_query | Standard |
| GL\_NV\_primitive\_restart | Full |
| GL\_NV\_texgen\_reflection | Full |
| GL\_NV\_texture\_rectangle | Full |
| GL\_NV\_texture\_shader | Standard |
| GL\_NV\_texture\_shader3 | Standard |
| GL\_NV\_vertex\_program | Standard |
| GL\_NV\_vertex\_program1\_1 | Standard |
| GL\_NV\_vertex\_program2 | Standard |
| GL\_NV\_vertex\_program2\_option | Standard |
| GL\_NV\_vertex\_program3 | Standard |
| GL\_OES\_draw\_texture | Standard |
| GL\_SGIS\_generate\_mipmap | Full |
| GL\_SGIS\_texture\_border\_clamp | Full |
| GL\_SGIS\_texture\_edge\_clamp | Full |
| GL\_SGIS\_texture\_lod | Full |
| GL\_SGIS\_texture\_select | Full |
| GL\_SGIX\_depth\_texture | Full |
| GL\_SGIX\_interlace | Full |
| GL\_SGIX\_shadow | Full |
| GL\_SGIX\_shadow\_ambient | Full |
| GLX\_ARB\_create\_context | Full |
| GLX\_ARB\_create\_context\_profile | Full |
| GLX\_ARB\_fbconfig\_float | Standard |
| GLX\_ARB\_framebuffer\_sRGB | Full |
| GLX\_ARB\_get\_proc\_address | Full |
| GLX\_ARB\_multisample | Standard |
| GLX\_SGI\_video\_sync | Standard |
| GLX\_SGIX\_fbconfig | Standard |
| WGL\_AMD\_gpu\_association | Standard |
| WGL\_ARB\_buffer\_region | Standard |
| WGL\_ARB\_create\_context | Full |
| WGL\_ARB\_create\_context\_profile | Full |
| WGL\_ARB\_extensions\_string | Full |
| WGL\_ARB\_framebuffer\_sRGB | Full |
| WGL\_ARB\_make\_current\_read | Standard |
| WGL\_ARB\_multisample | Standard |
| WGL\_ARB\_pbuffer | Standard |
| WGL\_ARB\_pixel\_format | Standard |
| WGL\_ARB\_pixel\_format\_float | Standard |
| WGL\_ARB\_render\_texture | Standard |
| WGL\_I3D\_genlock | Standard |
| WGL\_NV\_gpu\_affinity | Standard |
| WGL\_NV\_present\_video | Standard |
| WGL\_NV\_swap\_group | Standard |
| WGL\_NV\_video\_out | Standard |

Standard Extension Support Level

The Standard Extension Support Level enables one to log the calls and arguments of the extension functions, set breakpoints at the extension functions, and watch the values of the extension states variables.

Full Extension Support Level

In addition to supporting the Standard Extension Support Level features, the Full Extension Support Level enables one to view the extension-related data in the corresponding CodeXL views.

### GPU Kernel Debugging Support

CodeXL GPU debugging supports OpenCL kernel debugging. In order to step into a kernel, and start debugging, use one of the following methods:

1. Breakpoint in clEnqueueNDRangeKernel
2. Break in a kernel

Breakpoint in clEnqueueNDRangeKernel

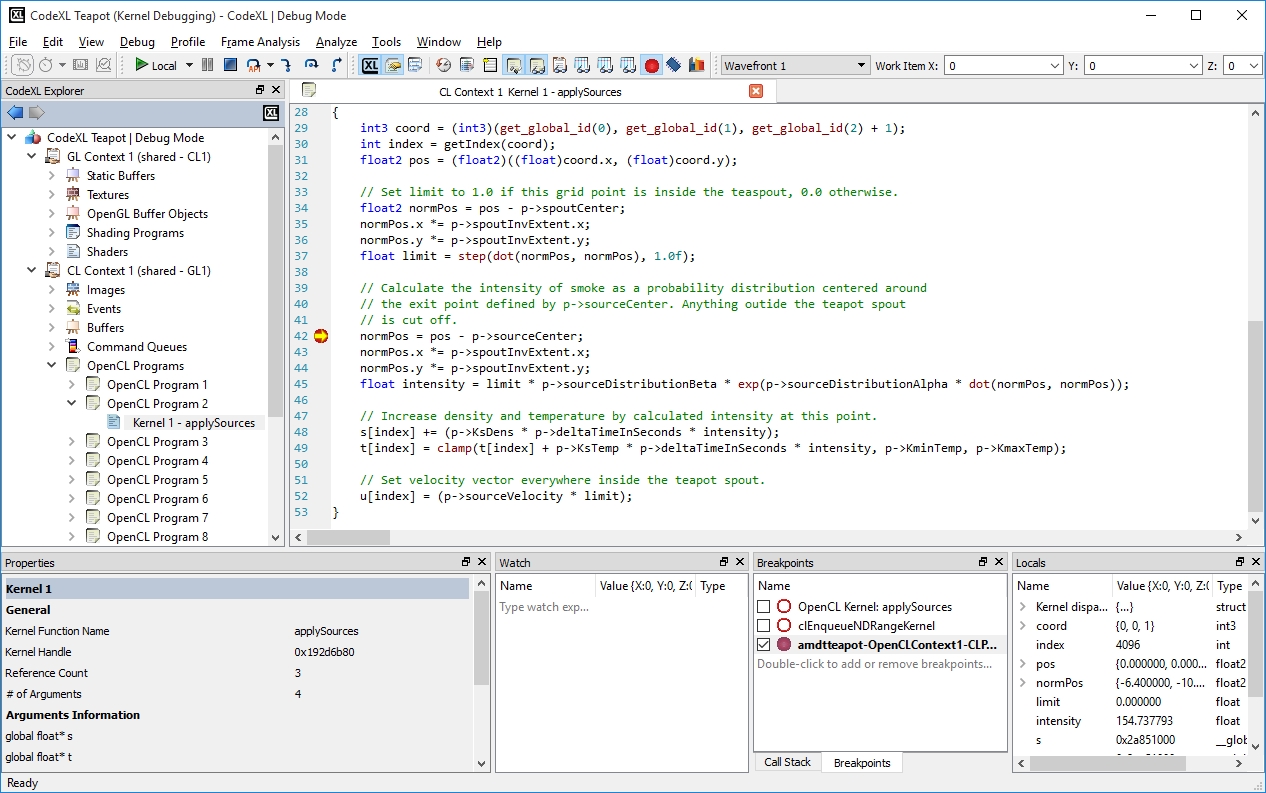
1. Put a breakpoint in clEnqueueNDRangeKernel using the [breakpoints](#_topic_AddRemoveCodeXLBreakpointsDialog) dialog.
2. Once the breakpoint hits, click F11, or step-in button 
3. The kernel source file is opened, and you can use step-over, step-in and step-out buttons.

Break in a kernel

1. Run your application.
2. After the kernel is created, break into the debugger.
3. Open the [breakpoints](#_topic_AddRemoveCodeXLBreakpointsDialog) dialog.
4. Navigate to the “Kernel Functions” tab
5. Select the kernel in which you want to break, and click the “Add button”.
6. Click Ok.
7. Resume the debugged application.
8. Next time this kernel is executed,the kernel source file will open, and you will be able to use step-over, step-in and step-out buttons.

Kernel Debugging

Once CodeXL goes into kernel debugging mode, the kernel source code is opened. If the kernel was created with a string, a temporary file with the kernel source is created.

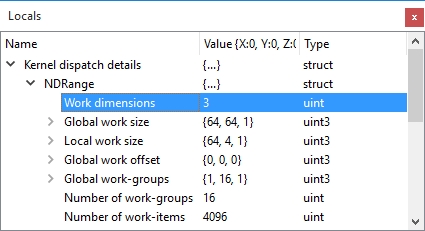


While debugging the kernel you can:

1. Put a breakpoint in any of the kernel source code lines.
2. Step-in, Step-out and Step-over the kernel source code lines.
3. See the values of variables in various views:

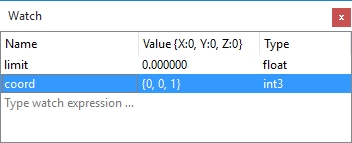
* Locals view

The locals view display a tree of the current local variables.



* Watch view

Type an expression, or drag an expression from the source code view to see the current variable value.



Use the work item toolbar in order to change the currently viewed work item both in Watch view and Locals view.



* Multi-watch view

In order to see multiple work item view of a variable, open the [Multi-watch view](#_OpenCL_Multi-Watch_Views) and use the visual and numeric data representation to see the variable’s values.

* Use the mouse to hover a variable and watch its value in a tooltip

As seen in the attached screenshot, once CodeXL is in Kernel Debugging mode, when the mouse cursor hovers over the variable name, a tooltip with the variable value is displayed.

