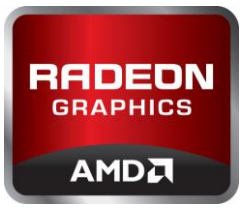

AGS Library Documentation

AMD Graphics Developer Relations Team

Revision 7 – April 28 2015

AGS Library Overview

This document provides an overview of the AGS (AMD GPU Services) library, including a presentation of available functionality and related entry points. The AGS library provides software developers with the ability to query AMD GPU software and hardware state information that is not normally available through standard operating system or graphic APIs. Version 1.0 of the library includes support for querying graphics driver version info, Crossfire (AMD's multi-GPU rendering technology) configuration info, as well as Eyefinity (AMD's multi-display rendering technology) configuration info. AGS is available in dynamic and static library form for 32 and 64 bit versions of Windows XP, Vista, Windows7 and Windows8.

This paper only presents AGS library APIs and associated functionality. Additional information on Catalyst drivers, as well as on Crossfire and Eyefinity technologies is available at www.amd.com. Graphics programming recommendations are detailed in the *Harnessing the Performance of CrossfireX* and in the *Gaming under Eyefinity* whitepapers, both available at developer.amd.com.

Using the AGS library

AGS functionality is accessed through the amd_ags.h header file: this file includes inline functions that abstract loading and unloading of the AGS dll (if it's used), as well as provides functionality that any software developer using the AGS lib or dll would otherwise need to implement. AGS features are built into the dll or static lib files (amd_ags.dll, amd_ags64.dll, amd_ags.lib, or amd_ags64.lib).

To add AGS support to an existing project, follow these steps:

- Determine if AGS functionality will be accessed through a dll or static lib. If the dll option is chosen, make sure to define `_AMD_AGS_USE_DLL` in your project properties. If the static lib option is chosen, no special token needs to be defined.
- Add to your project the appropriate amd_ags library file (amd_ags.dll, amd_gs64.dll, amd_ags.lib, or amd_ags64.lib).
- Add the amd_ags.h file to your project and include it from the source file that will call into the AGS library.

Initializing the API

The AGS library must be initialised before making any subesquent calls to the API. This can be performed before the device is created. The API is cleaned up using `AGSDeInit()`.

AGSReturnCode **AGSInit** ()	
Return Code	AGS_SUCCESS, AGS_FAILURE, AGS_ERROR_LEGACY_DRIVER, AGS_ERROR_MISSING_DLL

AGSReturnCode **AGSDeInit** ()	
Return Code	AGS_SUCCESS, AGS_FAILURE

Querying Graphics Driver Version

The `AGSDriverGetVersionInfo()` function enables developers to query the currently installed version of AMD graphics drivers. The returned structure contains two strings: the first refers to the build id of the graphics driver package ("8.723-100406a-098769C-ATI" is returned for Catalyst 10.4 for example), and the second contains the public release id ("10.4" is returned for Catalyst 10.4 for example). The call to this function must be made from the same thread that `AGSInit()` was called from.

AGSReturnCode **AGSDriverGetVersionInfo** (AGSDriverVersionInfoStruct *lpDriverVersionInfo)		
Output Param	lpDriverVersionInfo	Pointer to an AGSDriverVersionInfoStruct structure that contains driver version information.
Return Code	AGS_SUCCESS, AGS_FAILURE.	
Notes	Available with Catalyst 10.4 and later driver releases.	

AGSDriverVersionInfoStruct structure	
strDriverVersion[256]	Contains the build id of the currently installed graphics driver package. "8.723-100406a-098769C-ATI" is returned for Catalyst 10.4 for example
strCatalystVersion[256]	Contains the public release id of the currently installed graphics driver package. "10.4" is returned for Catalyst 10.4 for example

Querying Latest Graphics Driver Version

The `AGSDriverGetLatestVersionInfo()` function enables developers to query the latest version of AMD graphics drivers for the currently installed AMD hardware from the Internet. The returned structure contains two strings: the first refers to the public release id of the graphics driver package ("10.4" is returned for Catalyst 10.4 for example), and the second contains the URL of the latest graphics driver package ("http://game.amd.com/us-en/drivers_catalyst.aspx?p=win7/windows-7-32bit" is returned for example). The call to this function must be made from the same thread that `AGSInit()` was called from.

AGSReturnCode **AGSDriverGetLatestVersionInfo** (AGSLatestDriverVersionInfoStruct *lpDriverVersionInfo)		
Output Param	lpDriverVersionInfo	Pointer to an AGSLatestDriverVersionInfoStruct structure that contains driver version information.
Return Code	AGS_SUCCESS or AGS_FAILURE.	
Notes	Available with Catalyst 10.4 and later driver releases.	

AGSLatestDriverVersionInfoStruct structure	
strDriverVersion[256]	Contains the public release id of the graphics driver package for the currently installed AMD graphics hardware. "10.4" is returned for Catalyst 10.4 for example
strDriverWebLink[256]	Contains the URL of the latest graphics driver package for the currently installed AMD graphics hardware. "http://game.amd.com/us-en/drivers_catalyst.aspx?p=win7/windows-7-32bit" is returned for example

Querying Crossfire State

The `AGSCrossfireGetGPUCount()` function returns the number of AMD GPUs that operate in parallel to accelerate 3D rendering for the specified display.

AGSReturnCode **AGSCrossfireGetGPUCount** (		int iOSDisplayIndex, int *lpNumGPUs)
Input Param	iOSDisplayIndex	This is an operating system specific display index identifier. The value used should be the index of the display used for rendering operations.
Output Param	lpNumGPUs	This is a pointer to an integer that contains the number of GPUs used to accelerate 3D rendering for the provided display index.
Return Code	AGS_SUCCESS or AGS_FAILURE.	
Notes	Available with Catalyst 9.1 and later driver releases.	

Querying Eyefinity State

Querying Eyefinity configuration state information can be accomplished with the `AGSEyefinityGetConfigInfo()` function which returns the following information:

- Whether Eyefinity is enabled or not;
- The SLS grid configuration of displays used (3x1 layout, 3x2 layout, etc);
- The SLS size of the surface that spans the displays;
- Whether bezel compensation is enabled or not;
- The SLS grid coordinate for each display;
- The total rendering area for each display;
- The visible rendering area for each display;
- The preferred display (to properly position UI elements in games for example).

Remark: This function supports testing Eyefinity support without needing to enable Eyefinity in AMD's Catalyst Control Center application. This is useful for testing Eyefinity support for a windowed application or if the necessary hardware is not available, when testing support for a 6 display setup when using a graphic card that only supports up to 3 displays for example.

This is accomplished by pre-loading the AGSEyefinityInfoStruct structure passed into this function call. The iSLSWidth, iSLSHeight, iSLSGridWidth, and iSLSGridHeight must *all* contain non-zero values in order for this functionality to be triggered. When this is set, the amd_ags library is not used, and the values loaded into the pDisplaysInfo array of structures are simply computed based on the values passed into through the AGSEyefinityInfoStruct structure.

This functionality is only enabled in the amd_ags.h file if _DEBUG is defined to avoid having shipping applications use this debug feature.

In the example below, a 3 wide by 2 tall 2400x1200 is simulated, with each hypothetical display resolution being 800x600. Ideally, the developer using these kinds of values would then create a 2400x1200 window for rendering and display.

Example:

```
eyefinityInfo.iSLSWidth = 2400;

eyefinityInfo.iSLSHeight = 1200;

eyefinityInfo.iSLSGridWidth = 3;
```

eyefinityInfo.iSLSGridHeight = 2;

eyefinityInfo.iBezelCompensatedDisplay = TRUE;

AGSReturnCode **AGSEyefinityGetConfigInfo** (		int iOSDisplayIndex, AGSEyefinityInfoStruct *lpEyefinityInfo, int *lpNumDisplaysInfo, AGSDisplayInfoStruct **lppDisplaysInfo)
Input Param	iOSDisplayIndex	This is an operating system specific display index identifier. The value used should be the index of the display used for rendering operations.
In/Out Param	lpEyefinitiInfo	This is a pointer to an AGSEyefinitiInfoStruct structure that contains system Eyefinity configuration information. Under normal circumstances, it is only used as an output parameter. For testing and debugging purposes, it can be used to fake Eyefinity modes by setting the iSLSWidth, iSLSHeight, iSLSGridWidth, and iSLSGridHeight to valid, non-zero values. These entries are then used to compute the values of the lppDisplayInfoStruct array of structures. In this case, this parameter serves as both an input and output parameter. This enables developers to simulate and test Eyefinity support for various multi-monitor configurations without needing access to a full Eyefinity setup. This functionality is only available if the _DEBUG flag is defined.
Output Param	lpNumDisplaysInfo	Pointer to the number of AGSDisplayInfoStruct structures stored in the returned lppDisplayInfoStruct array. The value returned is equal to the number of displays used for the Eyefinity setup.
Output Param	lppDisplayInfoStruct	Pointer to an array of AGSDisplayInfoStruct structures that contains per display Eyefinity configuration information.
Return Code	AGS_SUCCESS or AGS_FAILURE.	
Notes	Available with Catalyst 10.3 and later driver releases.	

The `AGSEyefinityReleaseConfigInfo()` function is used to release memory allocated in the `AGSEyefinityGetConfigInfo()` call. The calls to these functions must be made from the same thread that `AGSInit()` was called from.

AGSReturnCode **AGSEyefinityReleaseConfigInfo** (AGSDisplayInfoStruct **lppDisplaysInfo)		
Input Param	lppDisplaysInfo	Pointer to an array of AGSDisplayInfoStruct structures that needs to be freed.
Output Param	None	
Return Code	AGS_SUCCESS or AGS_FAILURE.	
Notes	Available with Catalyst 10.3 and later driver releases.	

AGSEyefinityInfoStruct structure	
iSLSActive	Indicates if Eyefinity is active for the operating system display index passed into AGSEyefinityGetConfigInfo (). 1 if enabled and 0 if disabled.
iSLSGridWidth	Contains width of the multi-monitor grid that makes up the Eyefinity Single Large Surface. For example, a 3 display wide by 2 high Eyefinity setup will return 3 for this entry.
iSLSGridHeight	Contains height of the multi-monitor grid that makes up the Eyefinity Single Large Surface. For example, a 3 display wide by 2 high Eyefinity setup will return 2 for this entry.
iSLSWidth	Contains width in pixels of the multi-monitor SLS. The value returned is a function of the width of the SLS grid, of the horizontal resolution of each display, and of whether or not bezel compensation is enabled.
iSLSHeight	Contains height in pixels of the multi-monitor SLS. The value returned is a function of the height of the SLS grid, of the vertical resolution of each display, and of whether or not bezel compensation is enabled.
iBezelCompensatedDisplay	Indicates if bezel compensation is used for the current SLS display area. 1 if enabled and 0 if disabled.

AGSDisplayInfoStruct structure		
int	iGridCoordX	Contains horizontal SLS grid coordinate of the display. The value is zero based with increasing values from left to right of the overall SLS grid. For example, the left-most display of a 3x2 Eyefinity setup will have the value 0, and the right-most will have the value 2.
int	iGridCoordY	Contains vertical SLS grid coordinate of the display. The value is zero based with increasing values from top to bottom of the overall SLS grid. For example, the top display of a 3x2 Eyefinity setup will have the value 0, and the bottom will have the value 1.
AGSSimpleRectStruct	displayRect	Contains the base offset and dimensions in pixels of the SLS rendering area associated with this display. If bezel compensation is enabled, this area will be larger than what the display can natively present to account for bezel area. If bezel compensation is disabled, this area will be equal to what the display can support natively.
AGSSimpleRectStruct	displayRectVisible	Contains the base offset and dimensions in pixels of the SLS rendering area associated with this display that is visible to the end user. If bezel compensation is enabled, this area will be equal to what the display can natively, but smaller than the area described in the displayRect entry. If bezel compensation is disabled, this area will be equal to what the display can support natively and equal to the area described in the displayRect entry. ***Developers wishing to place UI, HUD, or other assets on a given display so that it is visible and accessible to end users need to locate them inside of the region defined by this rect.***
int	iPreferredDisplay	Indicates whether or not this display is the preferred one for rendering of game HUD and UI elements. Only one display out of the whole SLS grid will have this be true if it is the preferred display and 0 otherwise. Developers wishing to place specific UI, HUD, or other game assets on a given display so that it is visible and accessible to end users need to locate them inside of the region defined by this rect. If no display is marked as preferred, then it may be either down to the game to determine where to position the HUD or assume the HUD should cover the entire SLS such as in the case of 2x1 4k resolutions.

Querying Display ID State

The `AGSGetDefaultDisplayIndex()` function is simply a helper function used to query the id of the main display. Its output value can be used as input for AGS functions whose input requires a display index for developers who only target rendering at the main display. The call to this function must be made from the same thread that `AGSInit()` was called from.

AGSReturnCode **AGSGetDefaultDisplayIndex** (int *lpOSDisplayIndex)		
Input Params	None	
Output Params	lpOSDisplayIndex	Operating system index identifier for default display.
Return Code	AGS_SUCCESS or AGS_FAILURE.	

Querying GPU memory size

The `AGSGPUGetDeviceMemorySize()` function returns the memory size of a GPU. The call to this function must be made from the same thread that `AGSInit()` was called from.

AGSReturnCode **AGSGPUGetDeviceMemorySize** (int device, long long *lpSizeInBytes)		
Input Params	device	GPU device number, must be in the range returned by `AGSGPUGetDeviceCount()`
Output Params	lpSizeInBytes	Size of the GPU memory in bytes.
Return Code	AGS_SUCCESS or AGS_FAILURE.	