Active Quad Buffer Stereo on DirectX® 10 and 11

Technical Reference Manual

Part Number: 137-41903-10

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to discontinue or make changes to products, specifications, product descriptions, and documentation at any time without notice. The information contained herein may be of a preliminary or advance nature and is subject to change without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right. AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.
Reproduction of this manual, or parts thereof, in any form, without the express written permission of Advanced Micro Devices, Inc. is strictly prohibited.

USE OF THIS PRODUCT IN ANY MANNER THAT COMPLIES WITH THE MPEG-2 STANDARD IS EXPRESSLY PROHIBITED WITHOUT A LICENSE UNDER APPLICABLE PATENTS IN THE MPEG-2 PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, L.L.C., 6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111.

Trademarks

AMD, the AMD Arrow logo, ATI, the ATI logo, AMD Athlon, AMD LIVE!, AMD Opteron, AMD Phenom, AMD Sempron, AMD Turion, AMD64, All-in-Wonder, Avivo, Catalyst, CrossFireX, FirePro, FireStream, HyperMemory, OverDrive, PowerPlay, PowerXpress, Radeon, Remote Wonder, Stream, SurroundView, Theater, TV Wonder, The Ultimate Visual Experience, and combinations thereof are trademarks of Advanced Micro Devices, Inc.
HyperTransport is a licensed trademark of the HyperTransport Technology Consortium.
Blu-ray Disc is a licensed trademark of the Blu-ray Disc Association.
HDMI is a licensed trademark of HDMI Licensing, LLC.
DisplayPort is a licensed trademark of Video Electronic Standards Association.
Microsoft, Windows, and Vista are registered trademarks of the Microsoft Corporation in the United States and/or other jurisdictions.
Other names are for informational purposes only and may be trademarks of their respective owners.

Dolby Laboratories, Inc.

Manufactured under license from Dolby Laboratories. Dolby and the double-D symbol are trademarks of Dolby Laboratories.

© 1992-1997 Dolby Laboratories, Inc. All rights reserved.

Rovi Corporation

This device is protected by U.S. patents and other intellectual property rights. The use of Rovi Corporation's copy protection technology in the device must be authorized by Rovi Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by Rovi Corporation. Reverse engineering or disassembly is prohibited.

Revision History

Table 1–1 Revision History

Date	Revision	Description
6/20/2011	1.3	Updated Requirements section
7/23/2010	1.2	Added detail about DirectX® 11
7/19/2010	1.1	`GetDisplayModeList` details
4/1/2010	1.0	Document Creation

Contents

Figures

Chapter 1 Getting Started

Tables

Chapter 1 Getting Started

1

Getting Started

1.1 Background

Human vision perceives depth by using two eyes that are horizontally offset, whose images are then recombined in the brain as a 3D image. The more horizontally offset the two images are, the closer the object appears (you can see this effect by holding a finger directly in front of one eye and then shutting that eye while looking at your finger with the other eye).

3D stereo-capable (shutter) glasses simulate the same effect by sequentially displaying an offset image to one eye and then a differently-offset image to the other eye, using the shutter mechanism in the glasses (which is synchronized to the monitor's refresh rate), and toggling between them rapidly. AMD's hardware handles this by switching the display controller's displayable start pointer to quickly cycle between the two images.

Unlike OpenGL, there is no native support for this feature in the DirectX® API, so AMD has extended the API to enable the application to "virtually" render to the left eye or to the right eye. This extension works for DirectX 10 and beyond.

For stereo support to work, the concept of a swap chain is extended to allow for separate left and right eye channels. This stereo support is achieved by extending the size of the back buffer vertically. This is handled internally by the driver, which renders an image to a specific eye in the back buffer by simply modifying the viewport and then allowing the whole primary surface to be represented by one handle.

The middleware sits between the application and uses AMD's API to handle the details from the application side. An application that intends to natively support stereo can also use the AMD extension directly.

1.2 Requirements

The requirements are as follows:

- An AMD Radeon™ 5000 series or newer graphics card
- Display device that accepts the following input format:
  - Frame sequential via DVI (some legacy 3D monitors)
  - Frame sequential via VGA (limited projectors)
  - Frame sequential via DisplayPort (new 3D monitors in 2011)
  - Frame Packing format via HDMI™ (3D TVs and monitors that support HDMI 1.4a Frame Packing)

    Note: Glasses, either passive or shutter, should be provided by the display manufacturer or third-party. The emitter for shutter glasses must be controlled by the display device.
- AMD drivers with Active Stereo Direct 3D support
- Windows Vista® or Windows® 7
- Use of Direct 3D 10 or Direct 3D 11 API
- A single large Z buffer is created to accommodate for both left and right buffers and the hardware offset in-between (described in more detail in the rest of this document).
- Appropriate processes for enabling stereo must be followed
- Include header files `AmdDxExtApi.h` and `AmdDxExtQbStereo.h`

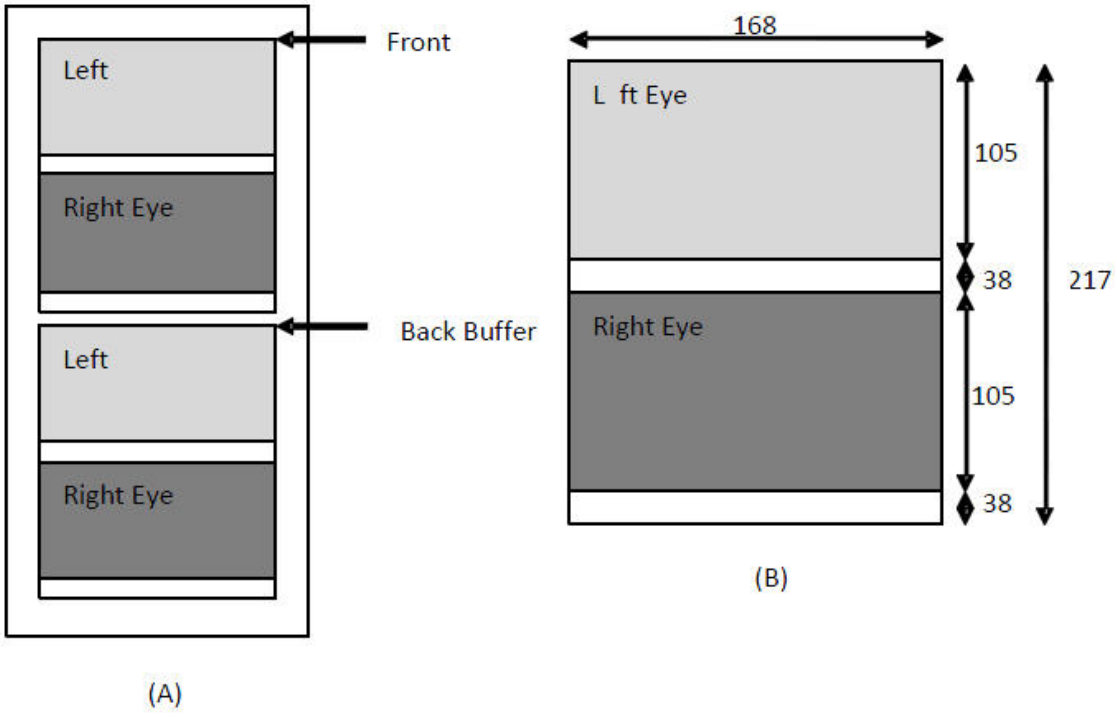
1.3 Mechanism of Operation

The following process should be implemented in order for stereo support to be active:

- AMD Driver adds a new quad buffer stereo extension
- Application creates a Direct 3D 10, 10.1, or 11 device in windowed mode or without a swap chain
- Application creates the extension library by calling `AmdDxExtCreate` DLL function in atidxx32.dll or atidxx64.dll
- Application calls `GetExtInterface` with the newly created `AmdDxExtStereoID` token, and stores stereo interface pointer
- Application enables stereo interface by calling `EnableStereo(TRUE)` with the stereo extension
- Application calls extension's `GetDisplayModeList` list to select a valid stereo display mode for the fullscreen swap chain

- Application enables fullscreen mode and the driver will allocate enough space for the left and right image in each buffer of the swap chain
- Application calls `GetLineOffset(&lineOffset)` to retrieve the offset. This returns the vertical line offset to the primary surface that represents the right eye. If there is an error or fullscreen is not enabled, this function returns 0.
- When rendering to the right buffer the application needs to set the viewport Y offset equal to the `lineOffset` value passed back when the `GetLineOffset` method was called. The left buffer will start at line 0. The following figure illustrates this:

Figure 2–1 3D Stereo Swap Chain



- Application Presents once both left and right have been rendered.
- Display will cycle between presented left and right buffers until next present when it will switch to the next set of left and right buffers.
- If the application wishes to copy the right eye back buffer data to another resource, it may do so by calling `CopySubresourceRegion` with the source box dimensions set to start at the right eye offset. This may produce a debug message warning that the copy is outside the extent, but the driver correctly receives the original coordinates passed in from the caller. The runtime does not clamp here.

1.4 The Stereo Extension Interface

In order to send stereo commands to the driver and to receive data back, the DXX extension interface must be used. The communication interface is described below:

Table 2–1 Stereo Extension Interface

Method Signature	Description
`HRESULT EnableStereo(BOOL enable);`	When set to `TRUE`, all future primary surfaces are created with Stereo support. The application is responsible for creating or recreating swap chains when this state is toggled.
`HRESULT GetDisplayModeList( DXGI_FORMAT EnumFormat, UINT Flags, UINT *pNumModes, DXGI_MODE_DESC *pDesc);`	Similar to the DXGI method, except this returns a list of supported modes in Stereo. To get the count for a specified format and flags, pass in `NULL` for `pDesc`.
`UINT GetLineOffset( IDXGISwapChain* pSwapChain );`	This returns the vertical line offset of the Stereo swap chain created. Returns 0 if the surface was created without enabling stereo or if the application is in windowed mode.

1.5 Miscellaneous

User scissor: Setting a user scissor is not necessary in order to use stereo but if it is used, ensure that it is set the same way the viewport is set. That is, if rendering to the right eye set the start co-ordinates to (0, lineOffset).

1.6 Example Code

The following code snippet initializes the extension interface from the application side.

Note: For DirectX 11, replace `AmdDxExtCreate` with `AmdDxExtCreate11` and `PFNAmdDxExtCreate` with `PFNAmdDxExtCreate11`.

```
/*****************************************************************************
* OpenStereoInterface
*
* @brief - Check for extension support and get the extension object
* - Get the stereo extension interface
*****************************************************************************/
HRESULT FTF_CLASS_NAME::OpenStereoInterface()
{
    PFNAmdDxExtCreate AmdDxExtCreate;
    HMODULE hDLL;
    HRESULT hr = S_OK;

#if defined _WIN64
    hDLL = GetModuleHandle("atidxx64.dll");
#else
    hDLL = GetModuleHandle("atidxx32.dll");
#endif

    // Find the DLL entry point
    AmdDxExtCreate = reinterpret_cast<PFNAmdDxExtCreate>(
                     GetProcAddress(hDLL, "AmdDxExtCreate"));
    if (AmdDxExtCreate == NULL)
    {
        hr = E_FAIL;
    }

    // Create the extension object
    if (hr == S_OK)
    {
        hr = AmdDxExtCreate(m_pD3DDevice, &m_pExt);
    }

    // Get the extension version information
    if (hr == S_OK)
    {
        AmdDxExtVersion extVersion;
        hr = m_pExt->GetVersion(&extVersion);
        if (hr == S_OK)
        {
        char buf[256];
        sprintf(str, "Extension Version: %d.%d\n",
              extVersion.majorVersion, extVersion.minorVersion);
        OutputDebugString(str);
        }
    }

    if (hr != S_OK)
    {
        CloseStereoInterface();
    }
}
/*****************************************************************************
* CloseStereoInterface
*
* @brief - Release the stereo interface
* - Release the extension object
*****************************************************************************/
VOID FTF_CLASS_NAME::CloseStereoInterface()
{
    if (m_pStereo != NULL)
    {
        m_pStereo->Release();
        m_pStereo = NULL;
    }

if (m_pExt != NULL)
```

```
    {
        m_pExt->Release();
        m_pExt = NULL;
    }
}
```

Before we call `OpenStereoInterface()` to enable the extension, we initialize quad buffer stereo is by separating the device and swap chain create calls separately and enabling quad buffer stereo support in the middle. Quad buffer stereo support only works with primary surfaces. Therefore, the swap chain description should be created for fullscreen use with the `DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH` flag set.

```
//
// Create the device in fullscreen mode
//

HRESULT hr = D3D10CreateDevice(NULL, D3D10_DRIVER_TYPE_HARDWARE,
                               NULL, createFlags, D3D10_SDK_VERSION,
                               &m_pD3DDevice);

//
// Open the stereo interface and store the line offset
//

if (SUCCEEDED(hr))
{
    // Initialize stereo extension
    if (SUCCEEDED(OpenStereoInterface()))
{
    m_pStereo = static_cast<IAmdDxExtQuadBufferStereo*>(
                        m_pExt->GetExtInterface(AmdDxExtQuadBufferStereoID));

    // Enable quad buffer support for swap chain
    if (m_pStereo != NULL)
    {
        hr = m_pStereo->EnableQuadBufferStereo(TRUE);
    }
    else
    {
        hr = E_FAIL;
    }
    // Create quad buffer swap chain
    if (SUCCEEDED(hr))
    {
      IDXGIDevice* pDXGIDevice = NULL;
      IDXGIAdapter* pDXGIAdapter = NULL;
      IDXGIFactory* pIDXGIFactory = NULL;

      m_pD3DDevice->QueryInterface(__uuidof(IDXGIDevice), (void **)&pDXGIDevice);
      pDXGIDevice->GetParent(__uuidof(IDXGIAdapter), (void **)&pDXGIAdapter);
      pDXGIAdapter->GetParent(__uuidof(IDXGIFactory), (void **)&pIDXGIFactory);

      if ((pIDXGIFactory != NULL) && (pDXGIDevice != NULL))
      {
        pIDXGIFactory->CreateSwapChain(m_pD3DDevice, sc, &m_pSwapChain);
        pDXGIDevice->Release();
      }
      else
      {
        hr = E_FAIL;
      }
    }

    // Store line offset to right eye
    if (m_pSwapChain != NULL)
    {
        m_rightEyeHeight = m_pStereo->GetLineOffset(m_pSwapChain);
    }
  }
}
```

1.7 GetDisplayModeList Example Code

```
HRESULT FTF_CLASS_NAME::DumpModes(DXGI_FORMAT displayFormat)
{
    if (!m_pStereo)
    {
        return E_FAIL;
    }

    UINT NumModes = 0;

    HRESULT hr = m_pStereo->GetDisplayModeList(displayFormat, 0, &NumModes, NULL);

    if (SUCCEEDED(hr) && NumModes > 0)
    {
        DXGI_MODE_DESC* pModeDescs = new DXGI_MODE_DESC[NumModes];

        if (!pModeDescs)
        {
            return E_FAIL;
        }

        hr = m_pStereo->GetDisplayModeList(displayFormat, 0, &NumModes, pModeDescs);

        if (SUCCEEDED(hr))
        {
            for (UINT i=0; i < NumModes; ++i)
            {
                const DXGI_MODE_DESC& ModeDesc = pModeDescs[i];

                printf("Width: %d Height: %d Format: %d RefreshRate: (%d/%d) = %f\n",
                    ModeDesc.Width,
                    ModeDesc.Height,
                    ModeDesc.Format,
                    ModeDesc.RefreshRate.Numerator,
                    ModeDesc.RefreshRate.Denominator,
                    (float)ModeDesc.RefreshRate.Numerator /
                    (float)ModeDesc.RefreshRate.Denominator);
            }
        }

        delete[] pModeDescs;
    }

    return hr;
}
```

Index