Active Stereo on DirectX® 9

Technical Reference Manual

Part Number: 137-41904-10

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	1.07	Updated Requirements section
7/22/2010	1.06	Added `ATI_STEREO_GETDISPLAYMODES` support
2/04/2010	1.05	Added `PrimaryAA` command, support for non-Direct 3D 9Ex devices, several code samples
1/14/2010	1.04	Added `PerSurfAA` command, removed fixed bugs from outstanding issues
12/01/2009	1.03	Updated outstanding issues, added version control structure
12/01/2009	1.02	Added blt control commands
11/13/2009	1.01	Updated `ATI_STEREO_GETLINEOFFSET` description
9/11/2009	1.0	Document creation

Contents

Figures

Chapter 1 Getting Started

Tables

Chapter 1 Getting Started

1

Getting Started

1.1 Background

In order for stereo support to work, extra surfaces need to be created for the right-side front and back buffers. Just like the front buffer in a regular swap chain, the front-left and front-right need to be available to the driver to present to the screen so as to keep both right and left surfaces visible. As there is no native support for this within the API, AMD has implemented a backdoor mechanism which allows an application to send commands to the driver in order to set stereo modes or receive data from the driver. These include commands to turn stereo on (to display both left and right eyes or just one for debugging), get the supported stereo display modes, or get the line offset from the end of the left eye the right eye portion of the surface.

1.2 Requirements

The main requirements are:

- 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, will 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 9 or Direct 3D 9Ex API
- A single large Z buffer must be 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 the provided header file atid3dstereo.h

1.3 Mechanism of Operation

The following process should be followed by an application to access stereo support:

- AMD Driver exposes special fourCC, AQBS, to advertise stereo support
- Application creates an `IDirect3D9` interface by calling `Direct3DCreate9`
- Application creates an `IDirect3DDevice` device by calling `IDirect3D9::CreateDevice`
- The device should be created in windowed mode (Windowed = TRUE in the present parameters structure) even if the application creates a fullscreen device (the same resolution can be used)
- The application creates an off-screen surface with the format set to the stereo 3D fourCC type AQBS. This should be done immediately after creating the device. This will be used as a communication medium between the driver and the application
- The application locks the surface and fills in the appropriate fields of the returned communication structure: `dwSignature='STER'` and `dwCommand=ATI_STEREO_ENABLESTEREO` (a full set of commands is described later in this document)
- In `Unlock` the AMD driver will check the `dwSignature` and if it is valid check `dwCommand` to see which command is sent. For `dwCommand=ATI_STEREO_ENABLESTEREO`, it will double allocate the flip chain (2× the height + hardware offset) once the device is set to fullscreen mode and set the system to display both left and right buffers.
- If the application wants to enable Anti-Aliasing on the front/back buffers, then the command `ATI_STEREO_ENABLEPRIMARYAA` must be sent.
- At this point, all resources created with `D3DPOOL_DEFAULT` must be freed to ensure Reset completes successfully. This includes the AQBS surface.
- The application sends the command `ATI_STEREO_GETDISPLAYMODES`, once to get the number of display modes and again to populate an array of display modes (as allocated by the application). It should then select a valid fullscreen stereo display mode for initializing the present parameters.
- The application should then call `IDirect3DDevice9::Reset()` with the present parameters set to enable fullscreen mode and `MultiSampleType` set to a valid hardware value greater than 1 (for example: 2 or 3). A valid hardware value greater than 1 is required for stereo to work correctly and is ignored for multi-sampling purposes unless the `ATI_STEREO_ENABLEPRIMARYAA` command was sent prior to calling Reset. Setting this value to 0 or 1 may appear to work in some cases, but will likely cause corruption and/or the hardware to hang.
- The AMD driver then receives the create for the flip chain and it will allocate the proper size for each buffer

- The AQBS surface must be recreated and the application should send another communication command the driver to request the line offset: the application locks the fourCC surface, sets `dwSignature='STER'`, `dwCommand=ATI_STEREO_GETLINEOFFSET`, `dwOutBufferSize = sizeof(DWORD)` and `pOutBuffer = &dwLineOffset`. The line offset will be written to `dwLineOffset` after the application calls `Unlock`.
- When rendering to the right buffer the application needs to set the viewport Y offset equal to the `dwLineOffset` value passed back when the `ATI_STEREO_GETLINEOFFSET` command was sent. The left buffer will start at line 0. The following figure illustrates this.

Figure 2–1 3D Swap Chain Including Above-and-Below Tiled Display Buffers with Padding to Ensure the Boundary Between the Left and Right Eye is Some Multiple of Lines as Required by the Hardware

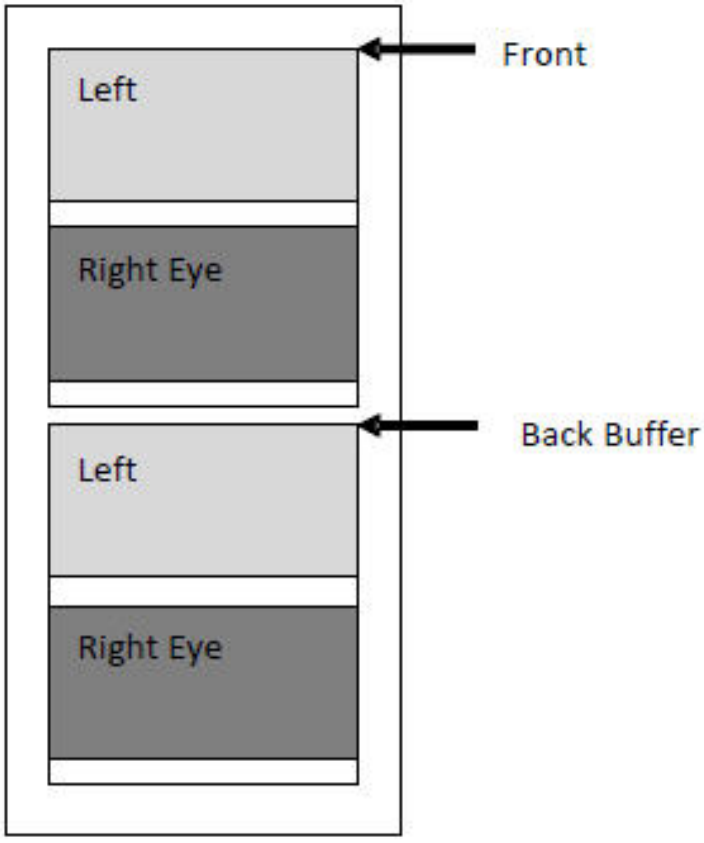
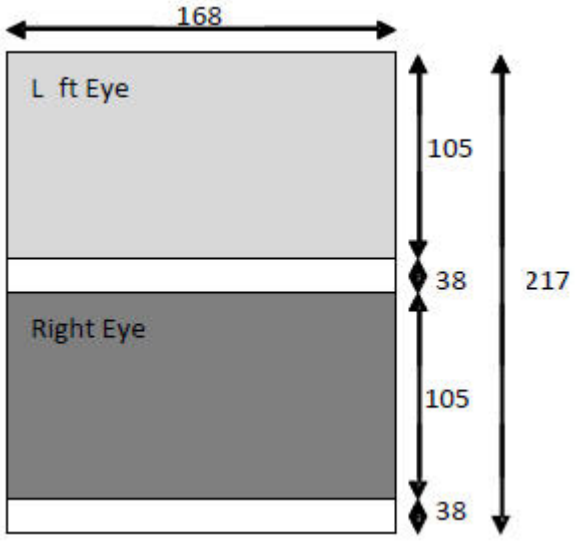


Figure 2–2 3D Swap Chain Depicting a 1680 × 1050 Resolution Display with Required Hardware Padding (38 Lines in this Case)



- The application presents once both left and right have been rendered and the `Present` function is called.
- The display cycles between presented left and right buffers until next present when it will switch to the next set of left and right buffers.

1.4 The Driver Communication Surface

In order to send commands to the driver and to receive data from the driver, a communication surface must be used. The process for using this is as follows:

- Create an offscreen plain surface of format fourCC AQBS. The width and height should be 10×10.
- Lock the surface. On lock, the driver will allocate and return a pointer to a `ATIDX9STEREOCOMMPACKET` structure. This structure is the communication surface.
- Assign and cast the `pBits` pointer to a locally created `ATIDX9STEREOCOMMPACKET` pointer.

The elements of the communication surface are described below:

Table 2–1 Driver Communication Surface

Field	Type	Description
`dwSignature`	DWORD	Indicates to the driver that the app is sending a command. Should be set to char sequence 'STER'.
`dwSize`	DWORD	Size of this structure. Passed to the app when the communication surface is locked.
`stereoCommand`	ATIDX9STEREOCOMMAND	Command given to the driver chosen from the `ATIDX9STEREOCOMMAND` enum. (Described below)
`pResult`	HRESULT *	Pointer to a buffer where the error code for the stereoCommand will be written to. D3D_OK is returned when successful. (Optional)

Field	Type	Description
dwOutBufferSize	DWORD	Size in bytes of optional buffer to place outgoing data into. Must be specified if data is to be returned by stereoCommand (i.e. get display modes or get line offset). If no pointer is specified when data is to be returned, an error will be written to pResult.
pOutBuffer	BYTE *	Pointer to buffer where the outgoing data will be placed. Must be specified if data is to be returned by stereoCommand (i.e. get display modes or get line offset). If no pointer is specified when data is to be returned, an error will be written to pResult.
dwInBufferSize	DWORD	Size in bytes of optional buffer containing input data. Must be specified for SETSRCEYE and SETDSTEYE commands.
pInBuffer	BYTE *	Pointer to a buffer where the input data is stored. Must be specified for SETSRCEYE and SETDSTEYE commands for which it is used to select the left or right eye.

The stereo commands are:

Command	Description
ATI_STEREO_GETVERSIONDATA	Returns a copy of ATIDX9STEREOVERSION
ATI_STEREO_ENABLESTEREO	Enable stereo
ATI_STEREO_ENABLELEFTONLY	Enable stereo but only display to the left eye.
ATI_STEREO_ENABLERIGHTONLY	Enable stereo but only display to the right eye.
ATI_STEREO_GETLINEOFFSET	Return the line offset to the beginning of the right eye. This command is only available once the device is in fullscreen mode.
ATI_STEREO_GETDISPLAYMODES	Returns the stereo modes available. The application will use the ATIDX9GETDISPLAYMODES structure to get a list of stereo modes. To retrieve the number of stereo modes available: • Set pStereoModes to NULL • Send the stereo command. The mode count will be returned in dwNumModes To retrieve the list of stereo modes: • Allocate dwNumModes of D3DDISPLAYMODE structure size memory and assign it to pStereoModes • Send the stereo command. The number of modes written will be returned in dwNumModes. The number of modes returned may be less than the dwNumModes returned in the first step but will never be greater The application is responsible to free the memory allocated in the second step. See the example for usage.

Command	Description
ATI_STEREO_SETSRCEYE	Sets the source eye for blts and surface copy API commands. Left/Right eye selection must be passed in using pInBuffer and dwInBufferSize. **Note:** Draws are *not* affected by this command. A list of affected API calls follows.
ATI_STEREO_SETDSTEYE	Sets the dest eye for blts and surface copy API commands. Left/Right eye selection must be passed in using pInBuffer and dwInBufferSize. **Note:** Draws are *not* affected by this command. A list of affected API calls follows.
ATI_STEREO_ENABLEPERSURFAA	Enables per surface anti-aliasing. This ensures that each non-primary render target will have its own AA buffer associated with it. This will likely improve performance for AA enabled stereo applications at the cost of extra memory usage.
ATI_STEREO_ENABLEPRIMARYAA	Enables AA for front and back buffers. This must be set along with MultiSampleType in D3DPRESENT_PARAMETERS when calling Reset or ResetEx. **Note:** This will *not* create independent AA buffers for primaries (i.e. they are still be shared).

For ATI_STEREO_SETSRCEYE and ATI_STEREO_SETDSTEYE, following API calls are affected if the source or destination is a surface representing the swapchain:

- StretchRect
- UpdateSurface
- GetFrontBufferData
- GetRenderTargetData

For ATI_STEREO_SETSRCEYE and ATI_STEREO_SETDSTEYE, dwInBufferSize must be set to sizeof(DWORD) and pInBuffer must be a pointer to a DWORD containing one of the following:

- ATI_STEREO_LEFTEYE
- ATI_STEREO_RIGHTEYE

The ATI_STEREO_GETVERSIONDATA command can be used to ensure developers are using the correct version. The major and minor version returned by this command should match the values in the header file.

1.5 Known Limitations/Issues

1. **Using AA on Front and Back Buffers:** To enable this when stereo is enabled, the field MultiSampleType in D3DPRESENT_PARAMETERS must be set when calling Reset/ResetEx *and* the command ATI_STEREO_ENABLEPRIMARYAA must be sent. This applies to both Direct 3D 9 and Direct 3D 9Ex devices. If this command is not sent, the MultiSampleType field in D3DPRESENT_PARAMETERS will not be used for AA on the primary surfaces.
2. **Using GDI:** At present, GDI will not work with stereo enabled because there is no shared primary.

3. **Locking the Backbuffer:** With stereo enabled, backbuffers cannot be locked.
4. **Right Buffer Data Access:** With the DirectX® 9 API, there is no way to lock and view the right eye back or front buffers during normal operation. This is because the API functions `GetFrontBufferData` and `GetBackBuffer` perform blts and create surface interfaces using the width and height of the swap chain specified with the presentation parameters so only the left buffer will ever get copied or be accessible when these functions are used. Additionally, blts/ surface copies to the right eye using `StretchRect` and `UpdateTexture` will not work because the DirectX runtime filters out rectangular co-ordinates that exceed the declared boundaries of the surface. Workaround: The commands `ATI_STEREO_SETSRCEYE` and `ATI_STEREO_SETDSTEYE` inform the driver that any subsequent blt or surface copy commands from/to the swap chain will come from and/or go to the specified eye(s). When set for the right eye source/dest, rects passed in will be offset appropriately in the driver.
5. **`ATI_STEREO_GETDISPLAYMODES`:** The command `ATI_STEREO_GETDISPLAYMODES` has not been implemented yet. At the moment, most standard resolutions will be supported on ordinary CRT/LCD monitors. In the future this may be restricted to fewer modes.
6. **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)`
7. **`ATI_STEREO_ENABLERIGHTONLY`:** This command is not currently supported. In some cases, this command may result in a lost device and windows indicating that the driver has stopped responding. AMD is working on a fix for this issue.

2

Examples

2.1 Creating a Direct 3D 9 Device with Stereo

```
void APPCLASS::CreateDevice()
{
   LPDIRECT3D9 m_pD3D;
   LPDIRECT3DDEVICE9 m_pD3DDev;
   D3DPRESENT_PARAMETERS d3dpp;
   D3DDISPLAYMODE d3dMode;
   D3DLOCKED_RECT lockedRect;
   DWORD flags;
   HRESULT hResult;
   DWORD modeSelect;

//Create the D3D interface and device
hr = Direct3DCreate9(D3D_SDK_VERSION);
GetWindowedPresentParameters(&d3dpp);
GetBehaviourFlags(&flags);
hr = pD3D->CreateDevice(0, D3DDEVTYPE_HAL, Hwnd(),flags, &d3dpp, &pD3DDev);
//Create resources including AQBS surface to be used to communicate with the driver
CreateDefaultPoolResources();
//Send the command to the driver using the temporary surface
hResult = SendStereoCommand(ATI_STEREO_ENABLESTEREO, NULL, 0, 0, 0);

if(hResult != D3D_OK)
{
    DisplayError("Stereo driver command EnableStereo Failed");
}

//Select a stereo mode for display
ATIDX9GETDISPLAYMODES displayModeParams;
displayModeParams.dwNumModes = 0;
displayModeParams.pStereoModes = NULL;

//Send stereo command to get the number of available stereo modes.
hResult = SendStereoCommand(ATI_STEREO_GETDISPLAYMODES, (BYTE *)(&displayModeParams),
                      sizeof(ATIDX9GETDISPLAYMODES), 0, 0);
if(hResult != D3D_OK)
{
    DisplayError("Stereo command GetDisplayMode Failed");
}

if(displayModeParams.dwNumModes != 0)
{
  //Allocating memory to get the list of modes.
  displayModeParams.pStereoModes = new D3DDISPLAYMODE[displayModeParams.dwNumModes];

  //Send stereo command to get the list of stereo modes
  hResult = SendStereoCommand(ATI_STEREO_GETDISPLAYMODES, (BYTE *)(&displayModeParams),
                          sizeof(ATIDX9GETDISPLAYMODES), 0, 0);
}

if(hResult != D3D_OK)
{
    DisplayError("Stereo command GetDisplayMode Failed");
}

GetFullscreenPresentParameters(&d3dpp);

If(displayModeParams.pStereoModes != NULL)
{
    //Select a display mode from the list.
    //"SelectDisplayMode()" is mode selection function to be implemented
    //by the application as per requirement.
    modeSelect = SelectDisplayMode(displayModeParams);
    D3DDISPLAYMODE mode = displayModeParams.pStereoModes[modeSelect];
```

```
    d3dpp.BackBufferWidth = mode.Width;
    d3dpp.BackBufferHeight = mode.Height;
    d3dpp.BackBufferFormat = (D3DFORMAT)mode.Format;
    d3dpp.FullScreen_RefreshRateInHz = mode.RefreshRate;

    //Free the memory allocated to store the mode list.
    delete[] displayModeParams.pStereoModes;
}


if (bIsAAEnabled())
{
    SendStereoCommand(ATI_STEREO_ENABLEPRIMARYAA, NULL, 0, 0, 0);
    d3dpp.MultiSampleType = GetAASamples();
}
else
{
    //A valid multisample value other then 0 or 1 must be set for stereo. (ex 2)
    d3dpp.MultiSampleType = D3DMULTISAMPLE_2_SAMPLES;
}

    //Resources allocated in the default pool must be freed before calling Reset
    FreeDefaultPoolResources();
    pD3DDev->Reset(&d3dpp);

    //Re-create resources including comm. surface
    CreateDefaultPoolResources();

    //Retrieve the line offset
    hResult = SendStereoCommand(ATI_STEREO_GETLINEOFFSET, (BYTE *)(&m_lineOffset),
     sizeof(DWORD), 0, 0);
}
```

2.2 Basic Render Code

```
void APPCLASS::Render()
{
   D3DVIEWPORT9 viewPort;

   BeginScene();
   // Draw Left Eye Scene
   viewPort.X = 0;
   viewPort.Y = 0;
   viewPort.Width = WinWidth();
   viewPort.Height = WinHeight();

   pD3DDev->SetViewPort(&viewPort);
   DrawLeft();

   // Draw Right Eye Scene
   viewPort.X = 0;
   viewPort.Y = m_lineOffset;
   viewPort.Width = WinWidth();
   viewPort.Height = WinHeight();
   pD3DDev->SetViewPort(&viewPort);
   DrawRight();

   EndScene();

   // Present both left and right buffers to the driver which will continuously
   // alternate between them until the next present
   pD3DDev->Present(NULL, NULL, NULL, NULL);
}

void APPCLASS::CopyToBackBuffer()
{
   D3DVIEWPORT9 viewPort;
   HRESULT hResult;
   IDirect3DSurface9 *pBackBuffer;
   RECT srcRect, dstRect;
   DWORD dwEye;

   m_pD3DDev->GetBackBuffer(0, 0, D3DBACKBUFFER_TYPE_MONO, &pBackBuffer);

   memset(&viewPort, 0, sizeof(D3DVIEWPORT9));
   viewPort.X = 0;
   viewPort.Y = 0;
   viewPort.Width = WinWidth();
   viewPort.Height = m_lineOffset + WinHeight();

   hResult = m_pD3DDev->SetViewport(&viewPort);

   srcRect.top = 0;
   srcRect.bottom = WinHeight();
   srcRect.left = 0;
   srcRect.right = WinWidth();

   dstRect.top = 0;
   dstRect.bottom = WinHeight();
   dstRect.left = 0;
   dstRect.right = WinWidth();

   // Draw Left Eye image
   dwEye = ATI_STEREO_LEFTEYE;
   hResult = SendStereoCommand(ATI_STEREO_SETDSTEYE, NULL, 0, (BYTE *)&dwEye,
      sizeof(dwEye) );
   hResult = m_pD3DDev->StretchRect(m_pLeftSurface, 0, pBackBuffer, &dstRect,
      D3DTEXF_LINEAR);
```

```
   // Draw Right Eye image
   dwEye = ATI_STEREO_RIGHTEYE;
   hResult = SendStereoCommand(ATI_STEREO_SETDSTEYE, NULL, 0, (BYTE *)&dwEye,
      sizeof(dwEye));
   hResult = m_pD3DDev->StretchRect(m_pRightSurface, 0, pBackBuffer, &dstRect,
      D3DTEXF_LINEAR);
}
```

2.3 Helper Functions (`SendStereoCommand`)

```
HRESULT APPCLASS::SendStereoCommand(
   ATIDX9STEREOCOMMAND stereoCommand,
   BYTE *pOutBuffer,
   DWORD dwOutBufferSize,
   BYTE *pInBuffer,
   DWORD dwInBufferSize)
{
   HRESULT hr;
   ATIDX9STEREOCOMMPACKET *pCommPacket;
   D3DLOCKED_RECT lockedRect;

   hr = m_pCommSurface->LockRect(&lockedRect, 0, 0);

   if(FAILED(hr))
   {
   SetErrorMsg("Failure in Stereo9L::SendStereoCommand(): LockRect");
   }

   pCommPacket = (ATIDX9STEREOCOMMPACKET *)(lockedRect.pBits);
   pCommPacket->dwSignature = 'STER';
   pCommPacket->pResult = &hr;
   pCommPacket->stereoCommand = stereoCommand;

   if (pOutBuffer && !dwOutBufferSize)
   {
   SetErrorMsg("Failure in Stereo9L::SendStereoCommand(): No outbuffer size specified");
   }

   pCommPacket->pOutBuffer = pOutBuffer;
   pCommPacket->dwOutBufferSize = dwOutBufferSize;

   if (pInBuffer && !dwInBufferSize)
   {
   SetErrorMsg("Failure in Stereo9L::SendStereoCommand(): No outbuffer size specified");
   }
   pCommPacket->pInBuffer = pInBuffer;
   pCommPacket->dwInBufferSize = dwInBufferSize;

   m_pCommSurface->UnlockRect();

   return hr;
}
```

Index