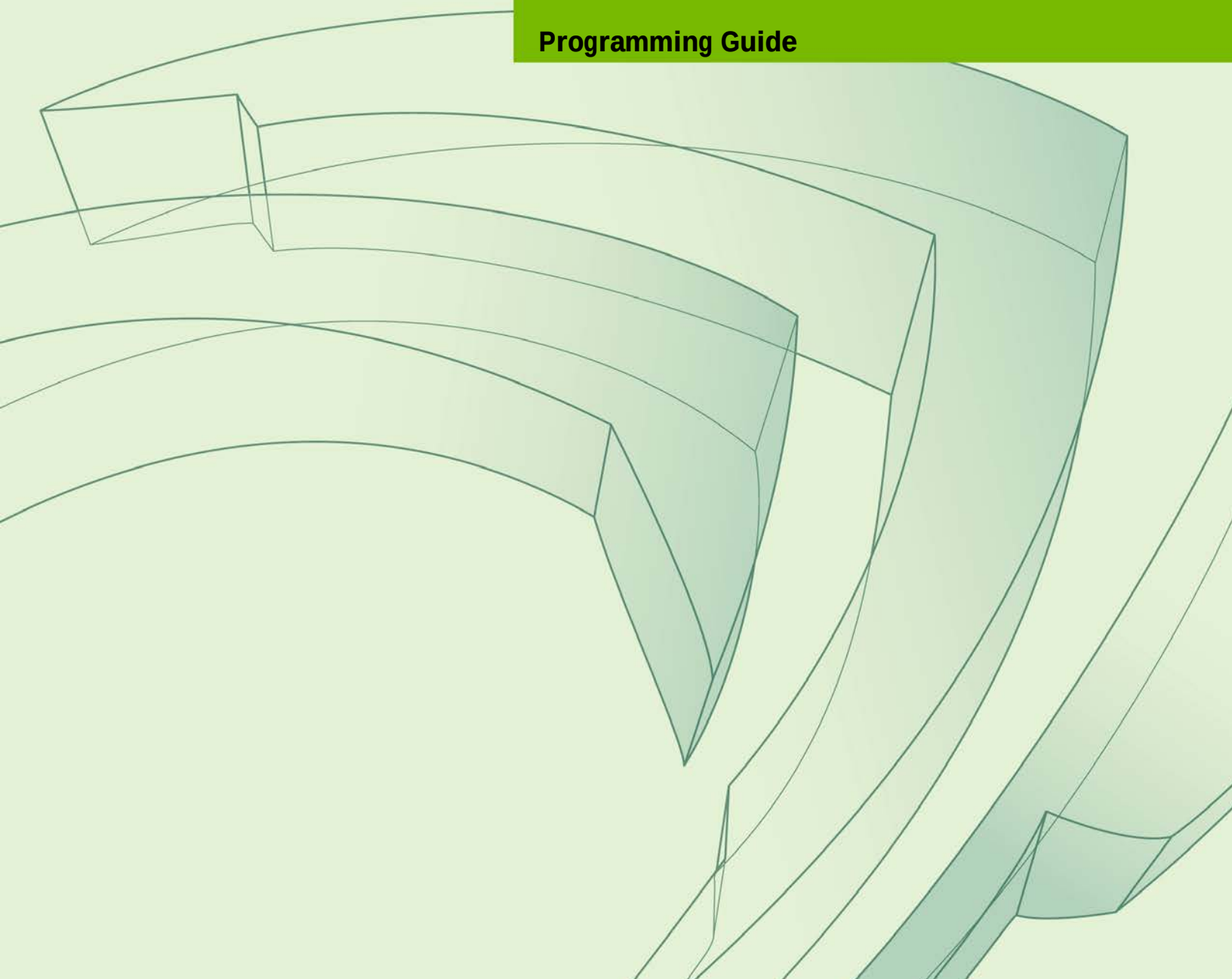
NVIDIA LENS MATCHED SHADING

PG-08039-001_v03 | November 2016

Programming Guide



DOCUMENT CHANGE HISTORY

PG-08039-001_v03

Version	Date	Authors	Description of Change
01	2016-05-18	GK, SB	Initial release
02	2016-09-06	GK	Correct typo in scissor rectangles comments
03	2016-11-07	SB	Added documentation for DirectX 12 support

TABLE OF CONTENTS

1 INTRODUCTION TO LENS MATCHED SHADING

The NVAPI programming interfaces for the Lens Matched Shading feature enable rendering in which the w component in the GPU hardware is modified according to the following equation:

$$w' = w + Ax + By$$

1.1 DIRECTX VERSIONS SUPPORTED

The Lens Matched Shading feature is supported on DirectX 11 and DirectX 12.

1.2 DRIVER VERSIONS SUPPORTED

The Lens Matched Shading feature for DirectX 11 is available starting with Release 367.

The Lens Matched Shading feature for DirectX 12 is available starting with Release 375.

2 DIRECTX 11 RENDERING WITH LENS MATCHED SHADING

Use the NVAPI programming interfaces for the Lens Matched Shading feature as explained in the sections that follow.

2.1 QUERYING THE FEATURE CAPABILITY

To query the Lens Matched Shading feature capability, call the `NvAPI_D3D_QueryModifiedWSupport` NVAPI.

```
NV_QUERY_MODIFIED_W_SUPPORT_PARAMS ModifiedW = {0};

ModifiedW.version = NV_QUERY_MODIFIED_W_SUPPORT_PARAMS_VER;

NvAPI_D3D_QueryModifiedWSupport (pDevice, &ModifiedW);

// ModifiedW.bModifiedWSupported will be TRUE if supported
```

2.2 SETTING LENS MATCHED SHADING COEFFICIENTS

The Lens Matched Shading coefficients are the terms A and B in the equation on page 1. You must specify these coefficients for each viewport separately.

To set the Lens Matched Shading coefficients:

1. Create a set of Lens Matched Shading parameters that specifies the coefficients for all viewports.
2. Call the `NvAPI_D3D_SetModifiedWMode` NVAPI, passing the set of Lens Matched Shading parameters as a parameter to the API.

The following example sets the Lens Matched Shading coefficients for four viewports.

```
NV_MODIFIED_W_PARAMS ModifiedWParams = {0};

ModifiedWParams.version      = NV_MODIFIED_W_PARAMS_VER;
ModifiedWParams.numEntries   = 4;

ModifiedWParams.modifiedWCoefficients[0].fA = -0.2f;
ModifiedWParams.modifiedWCoefficients[0].fB = 0.2f;

ModifiedWParams.modifiedWCoefficients[1].fA = 0.2f;
ModifiedWParams.modifiedWCoefficients[1].fB = 0.2f;

ModifiedWParams.modifiedWCoefficients[2].fA = -0.2f;
ModifiedWParams.modifiedWCoefficients[2].fB = -0.2f;

ModifiedWParams.modifiedWCoefficients[3].fA = 0.2f;
ModifiedWParams.modifiedWCoefficients[3].fB = -0.2f;


NvAPI_D3D_SetModifiedWMode (pDeviceorContext, &ModifiedWParams);
```

You must specify the viewports and the scissor rectangles for each viewport as shown in the following example.

```
D3D11_VIEWPORT            Viewports[4];
ZeroMemory(&Viewports, sizeof(D3D11_VIEWPORT)*_countof(Viewports));

Viewports[0].Height = ...; // Scene height
Viewports[1].Height = ...; // Scene height
Viewports[2].Height = ...; // Scene height
Viewports[3].Height = ...; // Scene height
Viewports[0].Width  = ...; // Scene width
Viewports[1].Width  = ...; // Scene width
Viewports[2].Width  = ...; // Scene width
Viewports[3].Width  = ...; // Scene width

pDeviceContext->RSSetViewports(4, Viewports);

D3D11_RECT ScissorRects[4];
ZeroMemory(&ScissorRects, sizeof(D3D11_RECT)*_countof(ScissorRects));
```

```
ScissorRects[0].left   = 0;
ScissorRects[0].top    = 0;
ScissorRects[0].right  = ...; // Scene width / 2
ScissorRects[0].bottom = ...; // Scene height / 2

ScissorRects[1].left   = ...; // Scene width / 2
ScissorRects[1].top    = 0;
ScissorRects[1].right  = ...; // Scene width
ScissorRects[1].bottom = ...; // Scene height / 2

ScissorRects[2].left   = 0;
ScissorRects[2].top    = ...; // Scene height / 2
ScissorRects[2].right  = ...; // Scene width / 2
ScissorRects[2].bottom = ...; // Scene height

ScissorRects[3].left   = ...; // Scene width / 2
ScissorRects[3].top    = ...; // Scene height / 2
ScissorRects[3].right  = ...; // Scene width
ScissorRects[3].bottom = ...; // Scene height

pDeviceContext->RSSetScissorRects(4, &ScissorRects[0]);
```

2.3 DISABLING LENS MATCHED SHADING

To disable Lens Matched Shading, call the `NvAPI_D3D_SetModifiedWMode` NVAPI with zero entries.

```
NV_QUERY_MODIFIED_W_SUPPORT_PARAMS ModifiedWParams = {0};

ModifiedWParams.version      = NV_MODIFIED_W_PARAMS_VER;
ModifiedWParams.numEntries   = 0;

NvAPI_D3D_SetModifiedWMode(pDeviceOrContext, &ModifiedWParams);
```

2.4 SETTING THE VIEWPORT MASK

The viewport mask specifies the viewports on which the geometry is rendered. To specify that the geometry is rendered on a viewport, set the bit for the viewport in the viewport mask.

For example, if you need the geometry to be rendered on four viewports, set the viewport mask to 0xF. It is recommended to set the bit for a viewport only if you require individual geometry to be rendered on the viewport.

You must specify the viewport mask in the HLSL shader.

The following example uses a Geometry shader for setting the viewport mask.

```
struct GSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;

     uint4 ViewportMask : NV_VIEWPORT_MASK;
}

[maxvertexcount(3)]
void GSMain(triangle VSOutput Input[3],
            inout TriangleStream<GSOutput> TriStream)
{
     GSOutput OutVtx;

     for (int v = 0; v < 3; ++v)
     {
         OutVtx.Pos    = Input[v].Pos;
         OutVtx.Color  = Input[v].Color;
         OutVtx.Tex    = Input[v].Tex;

         // 4 bits set indicate broadcast rendering to 4 viewports
         // recommended to set only required bits
         OutVtx.ViewportMask = 0xf;
         ...
         ...
         TriStream.Append(OutVtx);
     }
}
```

Note: You must declare the custom semantic for Viewport Mask as `uint4`.

When the shaders are compiled to Direct3D shader bytecode, use the `NvAPI_D3D11_CreateGeometryShaderEx_2` NVAPI to create the Geometry shader on the Direct3D 11 device.

```
ID3D11GeometryShader *pGeometryShader = NULL;
NvAPI_D3D11_CREATE_GEOMETRY_SHADER_EX GSExArgs = {0};

// create geometry shader with NVAPI
GSExArgs.version = NVAPI_D3D11_CREATEGEOMETRYSHADEREX_2_VERSION;
GSExArgs.NumCustomSemantics = 1;
GSExArgs.pCustomSemantics = (NV_CUSTOM_SEMANTIC*) malloc
```

```
  ((sizeof(NV_CUSTOM_SEMANTIC))*GSExArgs.NumCustomSemantics);
memset(GSExArgs.pCustomSemantics, 0,
  (sizeof(NV_CUSTOM_SEMANTIC))*GSExArgs.NumCustomSemantics);

GSExArgs.pCustomSemantics[0].version = NV_CUSTOM_SEMANTIC_VERSION;
GSExArgs.pCustomSemantics[0].NVCustomSemanticType =
   NV_VIEWPORT_MASK_SEMANTIC;
strcpy_s(&(GSExArgs.pCustomSemantics[1].NVCustomSemanticNameString[0]),
   NVAPI_LONG_STRING_MAX, "NV_VIEWPORT_MASK");

GSExArgs.UseViewportMask = false;

NvAPI_D3D11_CreateGeometryShaderEx_2(pDevice, pBytecode, BytecodeSize,
                                     NULL, &GSExArgs, &pGeometryShader)
...
free(GSExArgs.pCustomSemantics);
```

2.5 USING LENS MATCHED SHADING WITH FASTGS

Fast Geometry (FastGS) shader enables you to write Geometry shaders with certain restrictions to make them run more efficiently on the GPU.

Note: FastGS is not supported with the optimization disabled (/Od) HLSL compilation option.

You can use FastGS with the Lens Matched Shading feature.

The following example HLSL code shows how to use Lens Matched Shading with FastGS.

```
struct VSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;
}

struct GSOutput
{
    VSOutput Passthrough;

    uint4 ViewportMask : NV_VIEWPORT_MASK;
};

VSOutput VSMain (float4 position : POSITION,
                 float4 color : COLOR,
                 float2 Tex : TEXCOORD0)
```

```
{
     VSOutput OutVtx;

     OutVtx.Pos = position;
     ...
     ...
     return OutVtx;
}

[maxvertexcount(1)]
void FastGSMain(triangle VSOutput In[3],
                inout TriangleStream<GSOutput>
                 TriStream)
{
    GSOutput output;

    output.Passthrough = In[0];
    output.ViewportMask = 0xf;

    TriStream.Append(output);
}
```

To create the Geometry shader, use the `NvAPI_D3D11_CreateGeometryShaderEx_2` NVAPI with the fast GS option as shown in the following example.

```
ID3D11GeometryShader *pGeometryShader = NULL;

// create geometry shader with NVAPI
NvAPI_D3D11_CREATE_GEOMETRY_SHADER_EX GSExArgs = {0};
...
// specify custom semantics
...
GSExArgs.ForceFastGS = true;
GSExArgs.UseViewportMask = false;

NvAPI_D3D11_CreateGeometryShaderEx_2(pDevice, pBytecode, BytecodeSize,
                                     NULL, &GSExArgs, &pGeometryShader)
```

2.6 USING LENS MATCHED SHADING WITH SINGLE PASS STEREO

You can use Single Pass Stereo and Lens Matched Shading together to generate two eye views in which the Lens Matched Shading coefficients are the same for both eyes or are different for each eye.

2.6.1 Same Lens Matched Shading Coefficients for Both Eyes

If you need two eye views in which the Lens Matched Shading coefficients are the same for both eyes, generate both eye views in single draw call with the Lens Matched Shading coefficients applied to both eye views.

Create the Vertex, Hull, Domain, and Geometry shaders by using the NVAPI calls described in *NVIDIA Single Pass Stereo Programming Guide*. In these calls, use custom semantic variables for Single Pass Stereo (for example, `NV_X_RIGHT`) and for the viewport mask (for example `NV_VIEWPORT_MASK`).

The following example HLSL code shows how to use Single Pass Stereo and Lens Matched Shading together to generate two eye views in which the Lens Matched Shading coefficients are the same for both eyes.

```
struct VSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;

     float4 X_Right     : NV_X_RIGHT;       // Single Pass Stereo
}

struct GSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;

     float4 X_Right     : NV_X_RIGHT;       // Single Pass Stereo
     uint4 ViewportMask : NV_VIEWPORT_MASK; // viewport mask

     ...
}


VSOutput VSMain (float4 position : POSITION,
                 float4 color : COLOR,
                 float2 Tex : TEXCOORD0)
{
     VSOutput OutVtx;

     OutVtx.Pos = position;
     ...
     ...
     OutVtx.X_Right = (...); // right eye X value computation
     ...
     ...
```

```
    return OutVtx;
}

[maxvertexcount(3)]
void GSMain(triangle VSOutput Input[3],
            inout TriangleStream<GSOutput> TriStream)
{
    GSOutput OutVtx;

    for (int v = 0; v < 3; ++v)
    {
        OutVtx.Pos          = Input[v].Pos;
        OutVtx.Color        = Input[v].Color;
        OutVtx.Tex          = Input[v].Tex;
        OutVtx.X_Right      = Input[v].X_Right;
        OutVtx.ViewportMask = 0xf;
        ...
        ...

        TriStream.Append(OutVtx);
    }
}
```

Note: You must declare the custom semantic for Single Pass Stereo as `float4`.

Note: You must declare the custom semantic for Viewport Mask as `uint4`.

After the shaders are compiled to Direct3D shader bytecode, create the Vertex, Hull, Domain, and Geometry shaders on the Direct3D 11 device by using the NVAPI calls described in *NVIDIA Single Pass Stereo Programming Guide*.

The following example creates the Vertex shader and Geometry shader.

```
ID3D11VertexShader *pVertexShader = NULL;
NvAPI_D3D11_CREATE_VERTEX_SHADER_EX VSExArgs = {0};

// Create Vertex Shader with NVAPI
VSExArgs.version = NVAPI_D3D11_CREATEVERTEXSHADEREX_VERSION;
VSExArgs.NumCustomSemantics = 1;
VSExArgs.pCustomSemantics = (NV_CUSTOM_SEMANTIC*)malloc
    ((sizeof(NV_CUSTOM_SEMANTIC)*VSExArgs.NumCustomSemantics);
memset(VSExArgs.pCustomSemantics, 0,
    (sizeof(NV_CUSTOM_SEMANTIC))*VSExArgs.NumCustomSemantics);
VSExArgs.pCustomSemantics[0].version = NV_CUSTOM_SEMANTIC_VERSION;
VSExArgs.pCustomSemantics[0].NVCustomSemanticType =
    NV_X_RIGHT_SEMANTIC;
```

```
strcpy_s(&(VSExArgs.pCustomSemantics[0].NVCustomSemanti
    cNameString[0]), NVAPI_LONG_STRING_MAX, "NV_X_RIGHT");

NvAPI_D3D11_CreateVertexShaderEx(pDevice, pBytecode, BytecodeSize,
                                 NULL, &VSExArgs, &pVertexShader);
...
free(VSExArgs.pCustomSemantics);
...
...
ID3D11GeometryShader *pGeometryShader = NULL;
NvAPI_D3D11_CREATE_GEOMETRY_SHADER_EX GSExArgs = {0};

// create geometry shader with NVAPI
GSExArgs.version = NVAPI_D3D11_CREATEGEOMETRYSHADEREX_2_VERSION;
GSExArgs.NumCustomSemantics = 2;
GSExArgs.pCustomSemantics = (NV_CUSTOM_SEMANTIC*) malloc
  ((sizeof(NV_CUSTOM_SEMANTIC))*GSExArgs.NumCustomSemantics);
memset(GSExArgs.pCustomSemantics, 0,
  (sizeof(NV_CUSTOM_SEMANTIC))*GSExArgs.NumCustomSemantics);
GSExArgs.pCustomSemantics[0].version = NV_CUSTOM_SEMANTIC_VERSION;
GSExArgs.pCustomSemantics[0].NVCustomSemanticType =
   NV_X_RIGHT_SEMANTIC;
strcpy_s(&(GSExArgs.pCustomSemantics[0].NVCustomSemanticNameString[0]),
   NVAPI_LONG_STRING_MAX, "NV_X_RIGHT");

GSExArgs.pCustomSemantics[1].version = NV_CUSTOM_SEMANTIC_VERSION;
GSExArgs.pCustomSemantics[1].NVCustomSemanticType =
   NV_VIEWPORT_MASK_SEMANTIC;
strcpy_s(&(GSExArgs.pCustomSemantics[1].NVCustomSemanticNameString[0]),
   NVAPI_LONG_STRING_MAX, "NV_VIEWPORT_MASK");

GSExArgs.UseViewportMask = false;

NvAPI_D3D11_CreateGeometryShaderEx_2(pDevice, pBytecode, BytecodeSize,
                                 NULL, &GSExArgs, &pGeometryShader)
...
free(GSExArgs.pCustomSemantics);
```

2.6.2 Different Lens Matched Shading Coefficients for Each Eye

If you need two eye views in which the Lens Matched Shading coefficients are different for each eye, you must define two sets of each of the following items:

- A and B coefficients
- Viewports
- Scissor rectangles

One set of these items is for left eye view and the other set of these items is for right eye view.

```
NV_MODIFIED_W_PARAMS ModifiedWParams = {0};

ModifiedWParams.version      = NV_MODIFIED_W_PARAMS_VER;
ModifiedWParams.numEntries   = 8;

// ModifiedWParams.modifiedWCoefficients[0 .. 3] for left eye
// ModifiedWParams.modifiedWCoefficients[4 .. 7] for right eye

NvAPI_D3D_SetModifiedWMode(pDeviceOrContext, &ModifiedWParams);

...

D3D11_VIEWPORT          Viewports[8];

// Viewports[0 .. 3] for left eye
// Viewports[4 .. 7] for right eye

pDeviceContext->RSSetViewports(8, Viewports);

...

D3D11_RECT ScissorRects[8];

// ScissorRects[0 .. 3] for left eye
// ScissorRects[4 .. 7] for right eye

pDeviceContext ->RSSetScissorRects(8, &ScissorRects[0]);

...
```

You must specify independent viewport mask mode by calling the `NvAPI_D3D_SetSinglePassStereoMode` NVAPI with the independent viewport mask parameter set to `true`.

```
NvAPI_D3D_SetSinglePassStereoMode(pDeviceOrContext, 2, 1, true);
```

Additionally, in the custom semantic for the viewport mask of the HLSL shader, the lower 16 bits represent viewports for left eye and the upper 16 bits represent viewports for the right eye.

```
[maxvertexcount(3)]
void GSMain(triangle VSOutput Input[3],
            inout TriangleStream<GSOutput> TriStream)
{
     GSOutput OutVtx;
```

```
    for (int v = 0; v < 3; ++v)
    {
        ...
        // independent viewport mask for left eye and right eye
        // lower 16 bits for left eye
        // upper 16 bits for right eye
        OutVtx.ViewportMask = 0x00f0000f; // 0x00f0_000f
        ...
        ...
    }
}
```

3 DIRECTX 11 API REFERENCE

3.1 STRUCTURES

3.1.1 NV_QUERY_MODIFIED_W_SUPPORT_PARAMS

```
typedef struct _NV_QUERY_MODIFIED_W_SUPPORT_PARAMS
{
    NvU32 version;
    NvU32 bModifiedWSupported;
} NV_QUERY_MODIFIED_W_SUPPORT_PARAMS;
```

3.1.1.1 Members

`version`

Type: `NvU32`

The version of the `NV_QUERY_ MODIFIED_W_SUPPORT_PARAMS` structure

`bModifiedWSupported`

Type: `NvU32`

Indicates whether Modified W is supported on the current setup

3.1.1.2 Remarks

The `NV_QUERY_MODIFIED_W_SUPPORT_PARAMS` structure provides information about the Modified W capability on the current setup. This structure is used in the `NvAPI_D3D_QueryModifiedWSupport()` function call.

3.1.2 NV_MODIFIED_W_COEFFICIENTS

```
typedef struct _NV_MODIFIED_W_COEFFICIENTS
{
    float fA;
    float fB;
    float fAReserved;
    float fBReserved;

    float fReserved[2];
} NV_MODIFIED_W_COEFFICIENTS;
```

3.1.2.1 Members

`fA`

Type: `float`

The value of the A coefficient in the following equation:

$$w' = w + Ax + By$$

`fB`

Type: `float`

The value of the B coefficient in the following equation:

$$w' = w + Ax + By$$

`fAReserved`

Type: `float`

Reserved

`fBReserved`

Type: `float`

Reserved

`fReserved`

Type: `float[2]`

Reserved

3.1.2.2 Remarks

This structure is used to specify Modified W coefficient values. It is used in the function `NvAPI_D3D_SetModifiedWMode()`.

3.1.3 NV_MODIFIED_W_PARAMS

```
typedef struct _NV_MODIFIED_W_PARAMS
{
    NvU32 version;
    NvU32 numEntries;
    NV_MODIFIED_W_COEFFICIENTS
             modifiedWCoefficients[NV_MODIFIED_W_MAX_VIEWPORTS];
    NvU32 id;
    NvU32 reserved[NV_MODIFIED_W_MAX_VIEWPORTS];
} NV_MODIFIED_W_PARAMS;
```

3.1.3.1 Members

`version`

Type: `NvU32`

The version of the `NV_MODIFIED_W_PARAMS` structure

`numEntries`

Type: `NvU32`

The number of valid `NV_MODIFIED_W_COEFFICIENTS` structures in the array

`modifiedWCoefficients`

Type: `NV_MODIFIED_W_COEFFICIENTS`

Specifies Modified W coefficient values

`id`

Type: `NvU32`

Reserved

`reserved`

Type: `NvU32 [NV_MODIFIED_W_MAX_VIEWPORTS]`

Reserved

3.1.3.2 Remarks

This structure is used to configure Modified W mode. It is used in the function `NvAPI_D3D_SetModifiedWMode()`.

3.2 FUNCTIONS

3.2.1 NvAPI_D3D_QueryModifiedWSupport

```
NVAPI_INTERFACE NvAPI_D3D_QueryModifiedWSupport(
  [in]     IUnknown *pDevice,
  [inout]  NV_QUERY_MODIFIED_W_SUPPORT_PARAMS
           *pQueryModifiedWSupportedParams
);
```

3.2.1.1 Parameters

`pDevice [in]`

Type: `IUnknown*`

Pointer to the D3D11 `device ID3D11Device*`, which inherits `IUnknown*`

`pQueryModifiedWSupportedParams [inout]`

Type: `NV_QUERY_MODIFIED_W_SUPPORT_PARAMS*`

Pointer to the `NV_QUERY_MODIFIED_W_SUPPORT_PARAMS` structure

3.2.1.2 Return Value

Returns `NVAPI_OK` on success.

`bModifiedWSupported` becomes `TRUE` if Modified W is supported.

3.2.1.3 Remarks

This function determines whether the hardware supports the Modified W feature.

3.2.2 NvAPI_D3D_SetModifiedWMode

```
NVAPI_INTERFACE NvAPI_D3D_SetModifiedWMode(
     [in] IUnknown                 *pDevOrContext,
     [in] NV_MODIFIED_W_PARAMS     *psModifiedWParams
);
```

3.2.2.1 Parameters

`pDevOrContext [in]`

Type: `IUnknown *`

Pointer to the D3D11 device `ID3D11Device*` or `ID3D11DeviceContext*`, which inherits `IUnknown*`

`psModifiedWParams [in]`

Type: `NV_MODIFIED_W_PARAMS*`

Used to specify Modified W parameters

3.2.2.2 Return Value

Returns `NVAPI_OK` on success.

3.2.2.3 Remarks

This function sets the mode of Modified W.

Note that this function is an asynchronous function and returns `NVAPI_OK` if all arguments are valid. The returned value `NVAPI_OK` does not reflect that Modified W is supported or is set in hardware. You must call `NvAPI_D3D_QueryModifiedWSupport()` to confirm that the current setup supports Modified W before calling this set-function.

4 DIRECTX 12 RENDERING WITH LENS MATCHED SHADING

Use the NVAPI programming interfaces for the Lens Matched Shading feature as explained in the sections that follow.

4.1 QUERYING THE FEATURE CAPABILITY

To query the Lens Matched Shading feature capability, call the `NvAPI_D3D12_QueryModifiedWSupport` NVAPI.

```
ComPtr<ID3D12Device> pDevice;

NV_QUERY_MODIFIED_W_SUPPORT_PARAMS ModifiedW = {0};

ModifiedW.version = NV_QUERY_MODIFIED_W_SUPPORT_PARAMS_VER;

NvAPI_D3D12_QueryModifiedWSupport (pDevice.Get(), &ModifiedW);

// ModifiedW.bModifiedWSupported will be TRUE if supported
```

4.2 SETTING LENS MATCHED SHADING COEFFICIENTS

The Lens Matched Shading coefficients are the terms A and B in the equation on page 1. You must specify these coefficients for each viewport separately.

To set the Lens Matched Shading coefficients:

1. Create a set of Lens Matched Shading parameters that specifies the coefficients for all viewports.
2. Call the `NvAPI_D3D12_SetModifiedWMode` NVAPI, passing the set of Lens Matched Shading parameters as a parameter to the API.

   You must set Lens Matched Shading parameters on a `D3D12_COMMAND_LIST_TYPE_DIRECT` Graphics Command List. The state of Lens Matched Shading persists till the Command List is closed. As soon as `pCommandList->Close()` is called, the Lens Matched Shading state is reset back to disabled and parameters are reset back to zero.

   Lens Matched Shading state is restricted only to the Command List on which `NvAPI_D3D12_SetModifiedWMode` is called. Therefore, all render calls made only to this specific Command List are affected. Any other Command List for which Lens Matched Shading parameters are not set will function normally.

The following example sets the Lens Matched Shading coefficients for four viewports.

```
ComPtr<ID3D12GraphicsCommandList> pCommandList;

NV_MODIFIED_W_PARAMS ModifiedWParams = {0};

ModifiedWParams.version      = NV_MODIFIED_W_PARAMS_VER;
ModifiedWParams.numEntries   = 4;

ModifiedWParams.modifiedWCoefficients[0].fA = -0.2f;
ModifiedWParams.modifiedWCoefficients[0].fB = 0.2f;

ModifiedWParams.modifiedWCoefficients[1].fA = 0.2f;
ModifiedWParams.modifiedWCoefficients[1].fB = 0.2f;

ModifiedWParams.modifiedWCoefficients[2].fA = -0.2f;
ModifiedWParams.modifiedWCoefficients[2].fB = -0.2f;

ModifiedWParams.modifiedWCoefficients[3].fA = 0.2f;
ModifiedWParams.modifiedWCoefficients[3].fB = -0.2f;

NvAPI_D3D12_SetModifiedWMode(pCommandList.Get(), &ModifiedWParams);
```

You must specify the viewports and the scissor rectangles for each viewport as shown in the following example.

```
D3D12_VIEWPORT viewports[4];
memset(viewports, 0, sizeof(viewports));

viewports[0].TopLeftX = 0;
```

```
viewports[0].TopLeftY = 0;
viewports[0].MinDepth = 0.0f; // Total screen height
viewports[0].MaxDepth = 1.0f;

...

viewports[0].Height = ...;   // Scene height
viewports[1].Height = ...;   // Scene height
viewports[2].Height = ...;   // Scene height
viewports[3].Height = ...;   // Scene height
viewports[0].Width  = ...;   // Scene width
viewports[1].Width  = ...;   // Scene width
viewports[2].Width  = ...;   // Scene width
viewports[3].Width  = ...;    // Scene width

pCommandList->RSSetViewports(4, &viewports[0]);


D3D12_RECT scissorRects[4];
memset(scissorRect, 0, sizeof(scissorRects));

scissorRects[0].left   = 0;
scissorRects[0].top    = 0;
scissorRects[0].right  = ...; // Scene width / 2
scissorRects[0].bottom = ...; // Scene height / 2

scissorRects[1].left   = ...; // Scene width / 2
scissorRects[1].top    = 0;
scissorRects[1].right  = ...; // Scene width
scissorRects[1].bottom = ...; // Scene height / 2

scissorRects[2].left   = 0;
scissorRects[2].top    = ...; // Scene height / 2
scissorRects[2].right  = ...; // Scene width / 2
scissorRects[2].bottom = ...; // Scene height

scissorRects[3].left   = ...; // Scene width / 2
scissorRects[3].top    = ...; // Scene height / 2
scissorRects[3].right  = ...; // Scene width
scissorRects[3].bottom = ...; // Scene height

pCommandList->RSSetScissorRects(4, &scissorRects[0]);
```

4.3 DISABLING LENS MATCHED SHADING

To disable Lens Matched Shading, call the `NvAPI_D3D12_SetModifiedWMode` NVAPI with zero entries.

```
NV_QUERY_MODIFIED_W_SUPPORT_PARAMS ModifiedWParams = {0};

ModifiedWParams.version      = NV_MODIFIED_W_PARAMS_VER;
ModifiedWParams.numEntries   = 0;

NvAPI_D3D12_SetModifiedWMode(pCommandList.Get(), &ModifiedWParams);
```

4.4 SETTING THE VIEWPORT MASK

The viewport mask specifies the viewports on which the geometry is rendered. To specify that the geometry is rendered on a viewport, set the bit for the viewport in the viewport mask.

For example, if you need the geometry to be rendered on four viewports, set the viewport mask to 0xF. It is recommended to set the bit for a viewport only if you require individual geometry to be rendered on the viewport.

You must specify the viewport mask in the HLSL shader.

The following example uses a Geometry shader for setting the viewport mask.

```
struct GSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;

     uint4 ViewportMask : NV_VIEWPORT_MASK;
}

[maxvertexcount(3)]
void GSMain(triangle VSOutput Input[3],
            inout TriangleStream<GSOutput> TriStream)
{
     GSOutput OutVtx;

     for (int v = 0; v < 3; ++v)
     {
         OutVtx.Pos    = Input[v].Pos;
         OutVtx.Color  = Input[v].Color;
         OutVtx.Tex    = Input[v].Tex;

         // 4 bits set indicate broadcast rendering to 4 viewports
         // recommended to set only required bits
         OutVtx.ViewportMask = 0xf;
         ...
```

```
        ...
        TriStream.Append(OutVtx);
    }
}
```

Note: You must declare the custom semantic for Viewport Mask as `uint4`.

When the shaders are compiled to Direct3D shader bytecode, use the `NvAPI_D3D12_CreateGraphicsPipelineState` NVAPI to create the Graphics Pipeline State Object (PSO) which incorporates this Geometry shader. You need to provide additional information for PSO extension corresponding to the Geometry shader.

```
ComPtr<ID3D12PipelineState> pPSO;
ComPtr<ID3D12Device> pDevice;
UINT numExtensions;

const NVAPI_D3D12_PSO_EXTENSION_DESC* ppPSOExtensionsDesc[1];
NVAPI_D3D12_PSO_GEOMETRY_SHADER_DESC gsExDesc;

...

// Describe the graphics pipeline state object (PSO) using regular PSO
descriptor
D3D12_GRAPHICS_PIPELINE_STATE_DESC pPSODesc = ...;

// Custom Semantics for Geometry Shader
memset(&gsExDesc, 0, sizeof(gsExDesc));
gsExDesc.ForceFastGS = false;   // True, if using FastGS
gsExDesc.psoExtension = NV_PSO_GEOMETRY_SHADER_EXTENSION;
gsExDesc.version = NV_GEOMETRY_SHADER_PSO_EXTENSION_DESC_VER;
gsExDesc.baseVersion = NV_PSO_EXTENSION_DESC_VER;
gsExDesc.NumCustomSemantics = 1;
gsExDesc.pCustomSemantics = (NV_CUSTOM_SEMANTIC *)malloc(1 *
sizeof(NV_CUSTOM_SEMANTIC));
memset(gsExDesc.pCustomSemantics, 0, (1 * sizeof(NV_CUSTOM_SEMANTIC)));

gsExDesc.pCustomSemantics[0].version = NV_CUSTOM_SEMANTIC_VERSION;
gsExDesc.pCustomSemantics[0].NVCustomSemanticType =
NV_VIEWPORT_MASK_SEMANTIC;
strcpy_s(&(gsExDesc.pCustomSemantics[0].NVCustomSemanticNameString[0]),
NVAPI_LONG_STRING_MAX, "NV_VIEWPORT_MASK");

ppPSOExtensionsDesc[0] = &gsExDesc;
numExtensions = 1;

NvAPI_D3D12_CreateGraphicsPipelineState(pDevice.Get(), &psoDesc,
numExtensions, ppPSOExtensionsDesc, &pPSO);
```

```
free(gsExDesc.pCustomSemantics);
```

4.5 USING LENS MATCHED SHADING WITH FASTGS

Fast Geometry (FastGS) shader enables you to write Geometry shaders with certain restrictions to make them run more efficiently on the GPU.

Note: FastGS is not supported with the optimization disabled (/Od) HLSL compilation option.

You can use FastGS with the Lens Matched Shading feature.

The following example HLSL code shows how to use Lens Matched Shading with FastGS.

```
struct VSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;
}

struct GSOutput
{
    VSOutput Passthrough;

    uint4 ViewportMask : NV_VIEWPORT_MASK;
};

VSOutput VSMain (float4 position : POSITION,
                 float4 color : COLOR,
                 float2 Tex : TEXCOORD0)
{
     VSOutput OutVtx;

     OutVtx.Pos = position;
     ...
     ...
     return OutVtx;
}

[maxvertexcount(1)]
void FastGSMain(triangle VSOutput In[3],
                inout TriangleStream<GSOutput>
                 TriStream)
{
```

```
    GSOutput output;

    output.Passthrough = In[0];
    output.ViewportMask= 0xf;

    TriStream.Append(output);
}
```

To create the PSO that has the Fast Geometry shader, fill the `NVAPI_D3D12_PSO_GEOMETRY_SHADER_DESC` with `ForceFastGS = true` as shown in the following example.

```
NVAPI_D3D12_PSO_GEOMETRY_SHADER_DESC gsExDesc;

// Describe the graphics pipeline state object (PSO) using regular PSO
descriptor
...
// Specify custom semantics for Geometry Shaders in gsExDesc
...
gsExDesc.ForceFastGS = true;
...
gsExDesc.UseViewportMask = false;

NvAPI_D3D12_CreateGraphicsPipelineState(pDevice.Get(), &psoDesc,
numExtensions, ppPSOExtensionsDesc, &pPSO);
```

4.6 USING LENS MATCHED SHADING WITH SINGLE PASS STEREO

You can use Single Pass Stereo and Lens Matched Shading together to generate two eye views in which the Lens Matched Shading coefficients are the same for both eyes or are different for each eye.

4.6.1 Same Lens Matched Shading Coefficients for Both Eyes

If you need two eye views in which the Lens Matched Shading coefficients are the same for both eyes, generate both eye views in single draw call with the Lens Matched Shading coefficients applied to both eye views.

Create the PSO using NVAPI with PSO extensions for the Vertex, Hull, Domain, and Geometry shaders as described in *NVIDIA Single Pass Stereo Programming Guide*. When

creating these shaders, use custom semantic variables for Single Pass Stereo (for example, NV_X_RIGHT) and for the viewport mask (for example NV_VIEWPORT_MASK).

The following example HLSL code shows how to use Single Pass Stereo and Lens Matched Shading together to generate two eye views in which the Lens Matched Shading coefficients are the same for both eyes.

```
struct VSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;

     float4 X_Right     : NV_X_RIGHT;       // Single Pass Stereo
}

struct GSOutput
{
     float4 Pos : SV_POSITION;
     float4 Color : COLOR;
     float2 Tex : TEXCOORD0;

     float4 X_Right     : NV_X_RIGHT;       // Single Pass Stereo
     uint4 ViewportMask : NV_VIEWPORT_MASK; // viewport mask

     ...
}


VSOutput VSMain (float4 position : POSITION,
                 float4 color : COLOR,
                 float2 Tex : TEXCOORD0)
{
     VSOutput OutVtx;

     OutVtx.Pos = position;
     ...
     ...
     OutVtx.X_Right = (...); // right eye X value computation
     ...
     ...
     return OutVtx;
}

[maxvertexcount(3)]
void GSMain(triangle VSOutput Input[3],
            inout TriangleStream<GSOutput> TriStream)
{
     GSOutput OutVtx;

     for (int v = 0; v < 3; ++v)
     {
```

```
        OutVtx.Pos          = Input[v].Pos;
        OutVtx.Color        = Input[v].Color;
        OutVtx.Tex          = Input[v].Tex;
        OutVtx.X_Right      = Input[v].X_Right;
        OutVtx.ViewportMask = 0xf;
        ...
        ...

        TriStream.Append(OutVtx);
    }
}
```

Note: You must declare the custom semantic for Single Pass Stereo as `float4`.

Note: You must declare the custom semantic for Viewport Mask as `uint4`.

After the shaders are compiled to Direct3D shader bytecode, create the PSO using NVAPI with PSO extensions for the Vertex, Hull, Domain, and Geometry shaders as described in *NVIDIA Single Pass Stereo Programming Guide*.

The following example creates the PSO using NVAPI with PSO extensions for Vertex shader and Geometry shader.

```
ComPtr<ID3D12PipelineState> pPSO;
ComPtr<ID3D12Device> pDevice;
UINT numExtensions;

const NVAPI_D3D12_PSO_EXTENSION_DESC* ppPSOExtensionsDesc[2]; // To
incorporate the two PSO extensions: Vertex Shader and Geometry Shader
NVAPI_D3D12_PSO_GEOMETRY_SHADER_DESC gsExDesc;

...

// Describe the graphics pipeline state object (PSO) using regular PSO
descriptor
D3D12_GRAPHICS_PIPELINE_STATE_DESC pPSODesc = ...;


// Custom Semantics for Vertex Shader
memset(&vsExDesc, 0, sizeof(vsExDesc));
vsExDesc.psoExtension = NV_PSO_VERTEX_SHADER_EXTENSION;
vsExDesc.version = NV_VERTEX_SHADER_PSO_EXTENSION_DESC_VER;
vsExDesc.baseVersion = NV_PSO_EXTENSION_DESC_VER;
vsExDesc.NumCustomSemantics = 2;
vsExDesc.pCustomSemantics = (NV_CUSTOM_SEMANTIC *)malloc(2 *
sizeof(NV_CUSTOM_SEMANTIC));
```

```
memset(vsExDesc.pCustomSemantics, 0, (2 * sizeof(NV_CUSTOM_SEMANTIC)));

vsExDesc.pCustomSemantics[0].version = NV_CUSTOM_SEMANTIC_VERSION;
vsExDesc.pCustomSemantics[0].NVCustomSemanticType =
NV_X_RIGHT_SEMANTIC;
strcpy_s(&(vsExDesc.pCustomSemantics[0].NVCustomSemanticNameString[0]),
NVAPI_LONG_STRING_MAX, "NV_X_RIGHT");

vsExDesc.pCustomSemantics[1].version = NV_CUSTOM_SEMANTIC_VERSION;
vsExDesc.pCustomSemantics[1].NVCustomSemanticType =
NV_VIEWPORT_MASK_SEMANTIC;
strcpy_s(&(vsExDesc.pCustomSemantics[1].NVCustomSemanticNameString[0]),
NVAPI_LONG_STRING_MAX, "NV_VIEWPORT_MASK");

ppPSOExtensionsDesc[numExtensions++] = &vsExDesc;

// Custom Semantics for Geometry Shader
memset(&gsExDesc, 0, sizeof(gsExDesc));
gsExDesc.ForceFastGS = false;   // True, if using FastGS
gsExDesc.psoExtension = NV_PSO_GEOMETRY_SHADER_EXTENSION;
gsExDesc.version = NV_GEOMETRY_SHADER_PSO_EXTENSION_DESC_VER;
gsExDesc.baseVersion = NV_PSO_EXTENSION_DESC_VER;
gsExDesc.NumCustomSemantics = 2;
gsExDesc.pCustomSemantics = (NV_CUSTOM_SEMANTIC *)malloc(2 *
sizeof(NV_CUSTOM_SEMANTIC));
memset(gsExDesc.pCustomSemantics, 0, (2 * sizeof(NV_CUSTOM_SEMANTIC)));

gsExDesc.pCustomSemantics[0].version = NV_CUSTOM_SEMANTIC_VERSION;
gsExDesc.pCustomSemantics[0].NVCustomSemanticType =
NV_X_RIGHT_SEMANTIC;
strcpy_s(&(gsExDesc.pCustomSemantics[0].NVCustomSemanticNameString[0]),
NVAPI_LONG_STRING_MAX, "NV_X_RIGHT");

gsExDesc.pCustomSemantics[1].version = NV_CUSTOM_SEMANTIC_VERSION;
gsExDesc.pCustomSemantics[1].NVCustomSemanticType =
NV_VIEWPORT_MASK_SEMANTIC;
strcpy_s(&(gsExDesc.pCustomSemantics[1].NVCustomSemanticNameString[0]),
NVAPI_LONG_STRING_MAX, "NV_VIEWPORT_MASK");

ppPSOExtensionsDesc[numExtensions++] = &gsExDesc;

NvAPI_D3D12_CreateGraphicsPipelineState(pDevice.Get(), &psoDesc,
numExtensions, ppPSOExtensionsDesc, &pPSO);

free(vsExDesc.pCustomSemantics);
free(gsExDesc.pCustomSemantics);
```

4.6.2 Different Lens Matched Shading Coefficients for Each Eye

If you need two eye views in which the Lens Matched Shading coefficients are different for each eye, you must define two sets of each of the following items:

- A and B coefficients
- Viewports
- Scissor rectangles

One set of these items is for left eye view and the other set of these items is for right eye view.

```
NV_MODIFIED_W_PARAMS ModifiedWParams = {0};

ModifiedWParams.version      = NV_MODIFIED_W_PARAMS_VER;
ModifiedWParams.numEntries   = 8;

// ModifiedWParams.modifiedWCoefficients[0 .. 3] for left eye
// ModifiedWParams.modifiedWCoefficients[4 .. 7] for right eye

NvAPI_D3D_SetModifiedWMode(pDeviceOrContext, &ModifiedWParams);

...

D3D11_VIEWPORT          Viewports[8];

// Viewports[0 .. 3] for left eye
// Viewports[4 .. 7] for right eye

pDeviceContext->RSSetViewports(8, Viewports);

...

D3D11_RECT ScissorRects[8];

// ScissorRects[0 .. 3] for left eye
// ScissorRects[4 .. 7] for right eye

pDeviceContext ->RSSetScissorRects(8, &ScissorRects[0]);

...
```

You must specify independent viewport mask mode by calling the `NvAPI_D3D12_SetSinglePassStereoMode` NVAPI with the independent viewport mask parameter set to `true`.

```
NvAPI_D3D12_SetSinglePassStereoMode(pDevice.Get(), 2, 1, true);
```

Additionally, in the custom semantic for the viewport mask of the HLSL shader, the lower 16 bits represent viewports for left eye and the upper 16 bits represent viewports for the right eye.

```
[maxvertexcount(3)]
void GSMain(triangle VSOutput Input[3],
            inout TriangleStream<GSOutput> TriStream)
{
     GSOutput OutVtx;

     for (int v = 0; v < 3; ++v)
     {
         ...
         // independent viewport mask for left eye and right eye
         // lower 16 bits for left eye
         // upper 16 bits for right eye
         OutVtx.ViewportMask = 0x00f0000f; // 0x00f0_000f
         ...
         ...
     }
}
```

5 DIRECTX 12 RESTRICTIONS

5.1 CACHED PSO

A Graphics Pipeline State Object (PSO) created using NVAPI `NvAPI_D3D12_CreateGraphicsPipelineState` does not support the Cached PSO feature of DirectX 12. Do not call `GetCachedBlob()` with such a PSO.

6 DIRECTX 12 API REFERENCE

6.1 STRUCTURES

6.1.1 NV_QUERY_MODIFIED_W_SUPPORT_PARAMS

```
typedef struct _NV_QUERY_MODIFIED_W_SUPPORT_PARAMS
{
    NvU32 version;
    NvU32 bModifiedWSupported;
} NV_QUERY_MODIFIED_W_SUPPORT_PARAMS;
```

6.1.1.1 Members

`version`

Type: `NvU32`

The version of the `NV_QUERY_ MODIFIED_W_SUPPORT_PARAMS` structure

`bModifiedWSupported`

Type: `NvU32`

Indicates whether Modified W is supported on the current setup

6.1.1.2 Remarks

The `NV_QUERY_MODIFIED_W_SUPPORT_PARAMS` structure provides information about the Modified W capability on the current setup. This structure is used in the `NvAPI_D3D12_QueryModifiedWSupport()` function call.

6.1.2 NV_MODIFIED_W_COEFFICIENTS

```
typedef struct _NV_MODIFIED_W_COEFFICIENTS
{
    float fA;
    float fB;
    float fAReserved;
    float fBReserved;

    float fReserved[2];
} NV_MODIFIED_W_COEFFICIENTS;
```

6.1.2.1 Members

`fA`

Type: `float`

The value of the A coefficient in the following equation:

$$w' = w + Ax + By$$

`fB`

Type: `float`

The value of the B coefficient in the following equation:

$$w' = w + Ax + By$$

`fAReserved`

Type: `float`

Reserved

`fBReserved`

Type: `float`

Reserved

`fReserved`

Type: `float[2]`

Reserved

6.1.2.2 Remarks

This structure is used to specify Modified W coefficient values. It is used in the function `NvAPI_D3D12_SetModifiedWMode()`.

6.1.3 NV_MODIFIED_W_PARAMS

```
typedef struct _NV_MODIFIED_W_PARAMS
{
    NvU32 version;
    NvU32 numEntries;
    NV_MODIFIED_W_COEFFICIENTS
             modifiedWCoefficients[NV_MODIFIED_W_MAX_VIEWPORTS];
    NvU32 id;
    NvU32 reserved[NV_MODIFIED_W_MAX_VIEWPORTS];
} NV_MODIFIED_W_PARAMS;
```

6.1.3.1 Members

`version`

Type: `NvU32`

The version of the `NV_MODIFIED_W_PARAMS` structure

`numEntries`

Type: `NvU32`

The number of valid `NV_MODIFIED_W_COEFFICIENTS` structures in the array

`modifiedWCoefficients`

Type: `NV_MODIFIED_W_COEFFICIENTS`

Specifies Modified W coefficient values

`id`

Type: `NvU32`

Reserved

`reserved`

Type: `NvU32 [NV_MODIFIED_W_MAX_VIEWPORTS]`

Reserved

6.1.3.2 Remarks

This structure is used to configure Modified W mode. It is used in the function `NvAPI_D3D12_SetModifiedWMode()`.

6.2 FUNCTIONS

6.2.1 NvAPI_D3D12_QueryModifiedWSupport

```
NVAPI_INTERFACE NvAPI_D3D_QueryModifiedWSupport(
  [in]     ID3D12Device *pDevice,
  [inout]  NV_QUERY_MODIFIED_W_SUPPORT_PARAMS
           *pQueryModifiedWSupportedParams
);
```

6.2.1.1 Parameters

`pDevice [in]`

Type: `ID3D12Device*`

Pointer to the D3D12 device `ID3D12Device*`

`pQueryModifiedWSupportedParams [inout]`

Type: `NV_QUERY_MODIFIED_W_SUPPORT_PARAMS*`

Pointer to the `NV_QUERY_MODIFIED_W_SUPPORT_PARAMS` structure

6.2.1.2 Return Value

Returns `NVAPI_OK` on success.

`bModifiedWSupported` becomes `TRUE` if Modified W is supported.

6.2.1.3 Remarks

This function determines whether the hardware supports the Modified W feature.

6.2.2 NvAPI_D3D12_SetModifiedWMode

```
NVAPI_INTERFACE NvAPI_D3D12_SetModifiedWMode(
     [in] ID3D12GraphicsCommandList *pCommandList,
     [in] NV_MODIFIED_W_PARAMS      *psModifiedWParams
);
```

6.2.2.1 Parameters

`pCommandList [in]`

Type: `ID3D12GraphicsCommandList*`

Pointer to the `D3D12_COMMAND_LIST_TYPE_DIRECT` Graphics Command List that will be used for setting Single Pass Stereo mode and further rendering.

`psModifiedWParams [in]`

Type: `NV_MODIFIED_W_PARAMS*`

Used to specify Modified W parameters

6.2.2.2 Return Value

Returns `NVAPI_OK` on success.

6.2.2.3 Remarks

This function sets the mode of Modified W.

Note that Modified W state persists on a particular Command List until it is closed. The state is reset to default (disabled) for every newly created Command List. See “Setting Lens Matched Shading Coefficients” on page 18.

You must call `NvAPI_D3D12_QueryModifiedWSupport()` to confirm that the current setup supports Modified W before calling this set-function.

Notice

The information provided in this specification is believed to be accurate and reliable as of the date provided. However, NVIDIA Corporation (“NVIDIA”) does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This publication supersedes and replaces all other specifications for the product that may have been previously supplied.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and other changes to this specification, at any time and/or to discontinue any product or service without notice. Customer should obtain the latest relevant specification before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer. NVIDIA hereby expressly objects to applying any customer general terms and conditions with regard to the purchase of the NVIDIA product referenced in this specification.

NVIDIA products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on these specifications will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this specification. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this specification, or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this specification. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this specification is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ROVI Compliance Statement

NVIDIA Products that support Rovi Corporation's Revision 7.1.L1 Anti-Copy Process (ACP) encoding technology can only be sold or distributed to buyers with a valid and existing authorization from ROVI to purchase and incorporate the device into buyer's products.

This device is protected by U.S. patent numbers 6,516,132; 5,583,936; 6,836,549; 7,050,698; and 7,492,896 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.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright