Stereo Unprojection

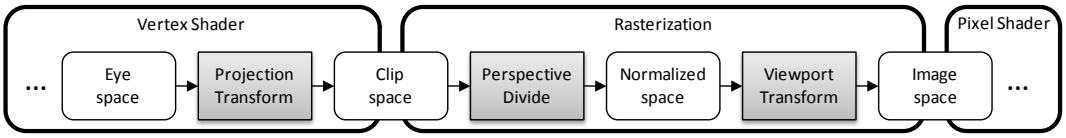
This document exposes the technical problems faced in stereo with nvidia driver automatic mode when the fragment position must be unprotected in mono space in the pixel shader.

Problem

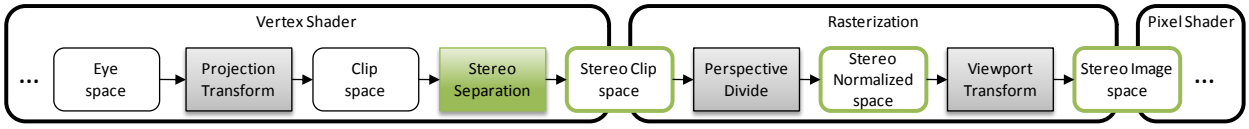
Transform pipeline

Here is the standard transform pipeline from Eye space to Image space for the POSITION attribute of a primitive vertex (in the vertex shader) and the generated fragments (in Rasterization and pixel shader)

In mono



In stereo



In stereo, the 3 differences are

- The stereo separation is applied at the end of the vertex shader on the POSITION attribute of the vertex output. In stereo the POSITION is expressed in the stereo Clip space
- The rasterization stage is operated in the stereo space and not in the mono space
- The fragment POSITION attribute input in the pixel shader is expressed in the Stereo Image space and not in the mono space.

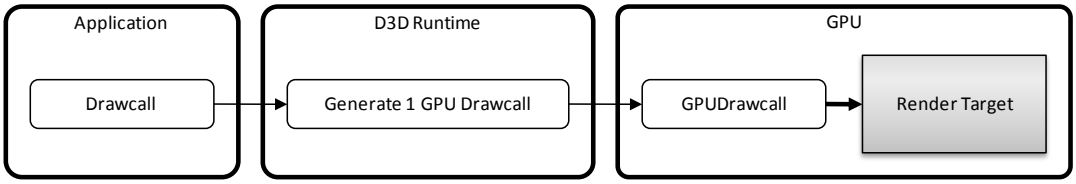
The stereo separation applied to the clip space position is

$$Pos_{s-clip}.yzw = Pos_{clip}.yzw$$

$$Pos_{s-clip}^{right/left}.x = Pos_{clip}.x \pm Separation * (Pos_{clip}.w - Convergence)$$

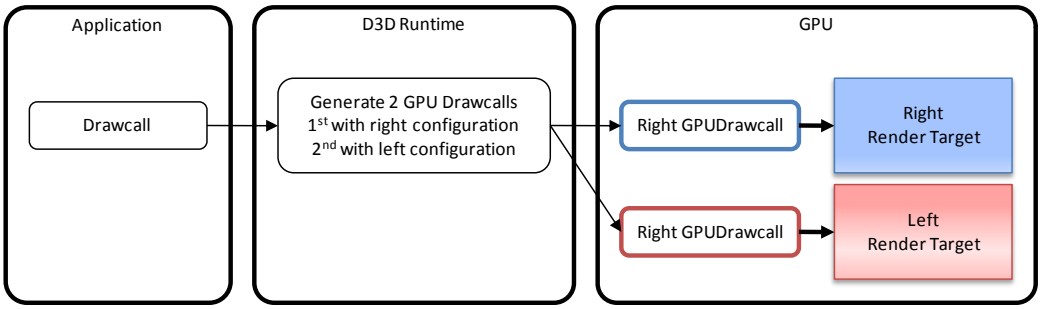
Stereo Drawcall

When the application is issuing a draw call through D3D, a standard mono draw call is executed as follow.



If the render target is stereo then the stereo driver duplicates it, issuing 2 drawcalls, one rendering in the right eye render target, the second rendering in the left eye render target. The parameters

configuring the stereo separation in the vertex shader are updated before each of the 2 draw calls. The stereo draw call execution looks as follow.

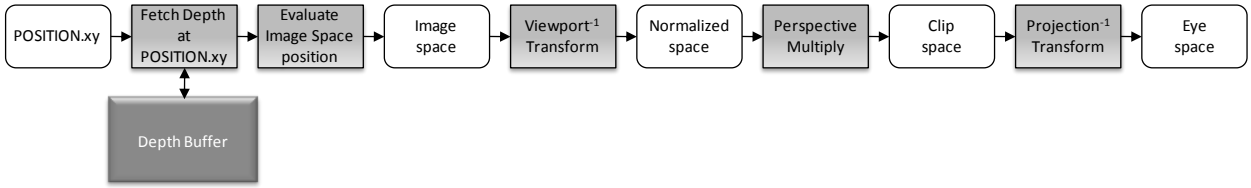


In Stereo, there is no way for the application to know which drawcall side is issued (left or right) when running the shader.

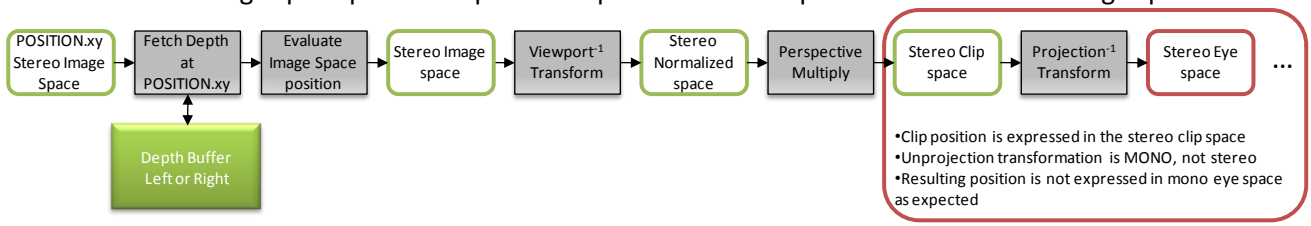
Unprojection

In a deferred shading scenario, it's standard for an application to recreate in the pixel shader a 3D position of the fragment from the input 2D position of the pixel and the depth information fetched from the depth buffer.

In such a case, the shader code execute the following transformation



In stereo the 2D image space position input in the pixel shader is expressed in the stereo image space.

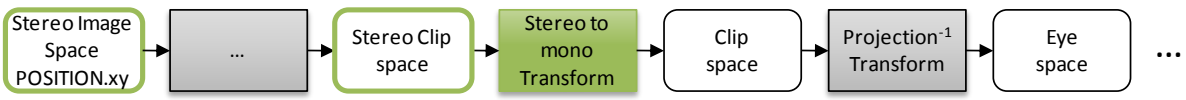


By default the transformation chain stays in stereo space and never goes back to the expected mono space because the application cannot differentiate between the left or right drawcall.

The Unprojection matrix used is the inverse of the mono projection. Hence the unprojection transformation of the stereo clip position to the mono eye space is wrong.

Solution

To fix the unprojection issue, the app needs to transform the position expressed in stereo space back to mono space. In the pixel shader, the position (in stereo clip space) should be applied the reverse stereo separation before applying the unprojection in the mono space.



The stereo to mono transformation needed is

$$Pos_{clip}.yzw = Pos_{s-clip}.yzw$$

$$Pos_{clip}.x = Pos_{s-clip}^{r/l}.x \mp Separation * \left(Pos_{s-clip}.w - Convergence\right)$$

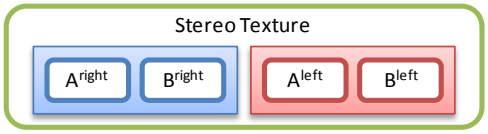
It can be expressed as

$$Pos_{clip}.x = Pos_{s-clip}^{r/l}.x + A^{r/l} * Pos_{s-clip}.w + B^{r/l}$$

$$A^{\text{right}} = -Separation \quad B^{\text{right}} = Separation * Convergence$$
$$A^{\text{left}} = Separation \quad B^{\text{left}} = -Separation * Convergence$$

The sign of $A^{r/l}$ and $B^{r/l}$ depends on the side of the drawcall. The correct value for these 2 stereo parameters can be provided to the pixel shader through a stereo texture. A stereo texture has 2 versions (a left and a right) and the correct binding is automatically managed by the stereo driver.

We can create a stereo texture containing the appropriate values and sign for each side.



The 2 stereo parameters are then fetched from that texture to apply the stereo un-separation to the clip position. The HLSL sm4 pixel shader function doing that transformation would look like

```
Texture2D g StereoParamMap;

float4 stereoToMonoCPOS( float4 cposStereo )
{
    float2 stereoParam = float2( g_StereoParamMap.Load( int3( 0, 0, 0) ).x
                                 g_StereoParamMap.Load( int3( 1, 0, 0) ).x );
    float4 cposMono = cposStereo;
    cposMono.x = cposStereo.x + stereoParam.x * cposStereo.w + stereoParam.y;
    return cposMono;
}
```