The In and Out: Making Games Play Right with Stereoscopic 3D Technologies

Samuel Gateau

Outline

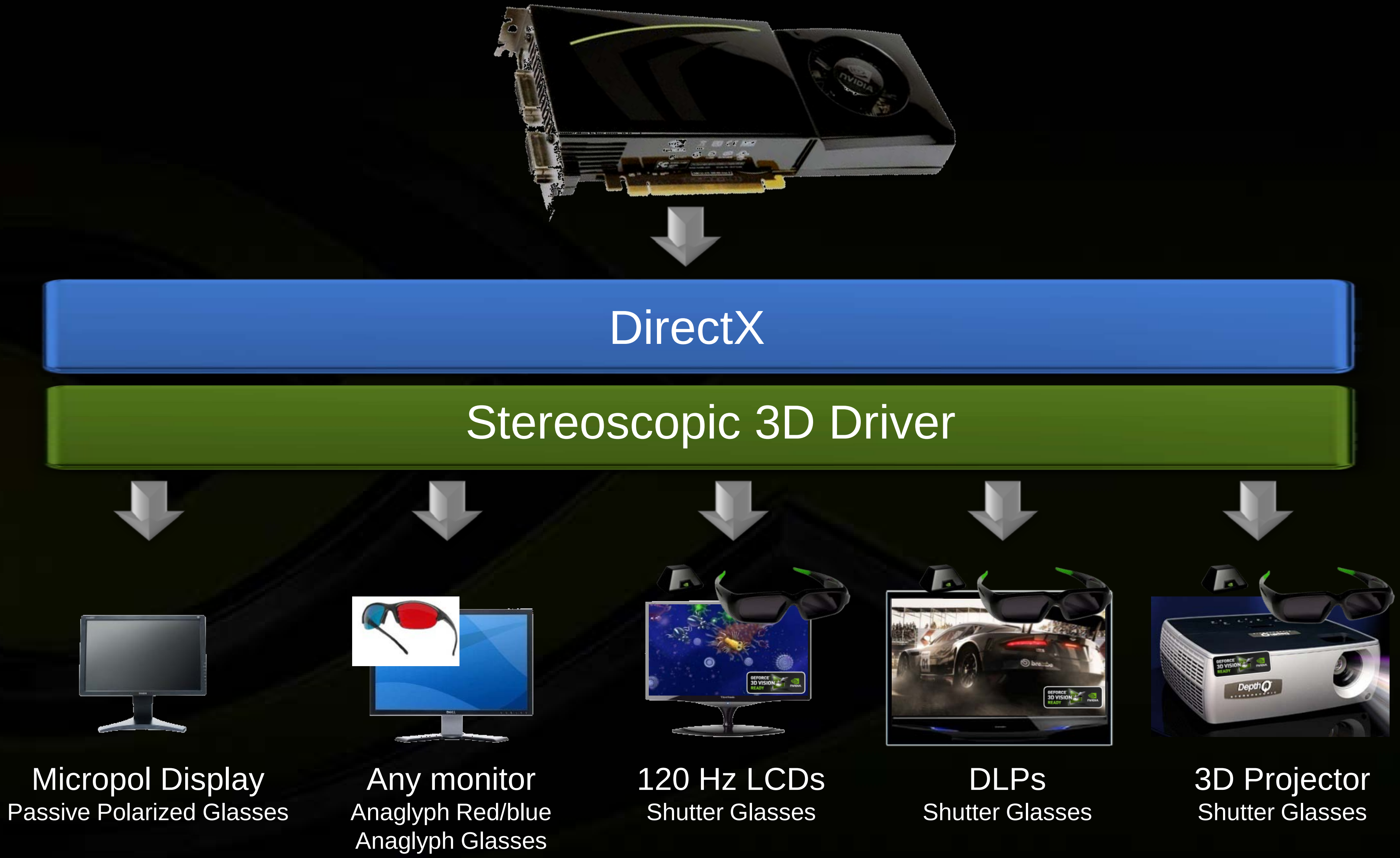
- NVIDIA 3D Vision
  - Stereoscopic driver & HW display solutions
- Stereoscopic basics
  - Definitions and equations
- Rendering techniques
  - Practical recommendations
- Advanced stereo effects
  - Selection Marquee
  - 3D Video

How does it work ?

NVIDIA 3D VISION

NVIDIA 3D Vision

Stereoscopic 3D Driver

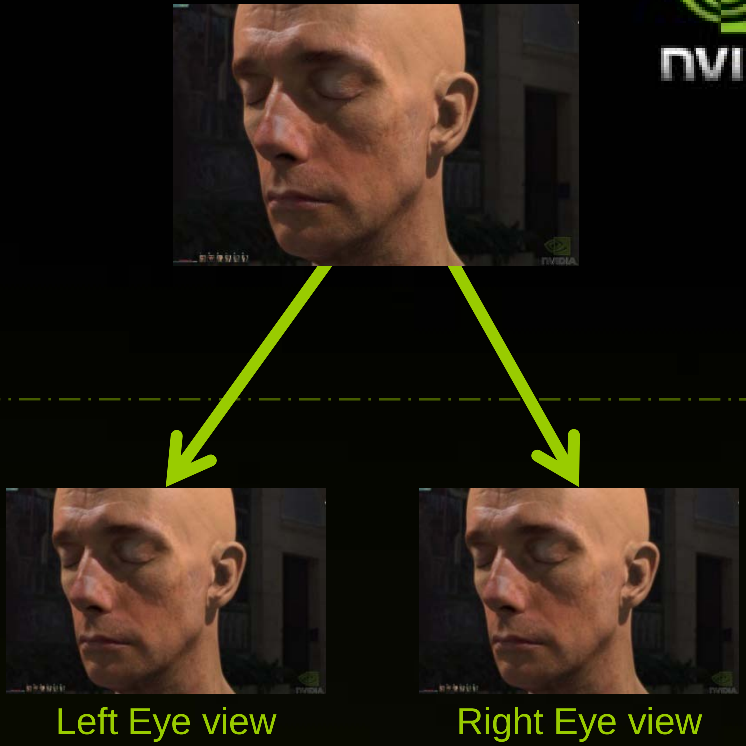


How It Works

3D game data is sent to stereoscopic driver

The driver takes the 3D game data and renders each scene twice – once for the left eye and once for the right eye.

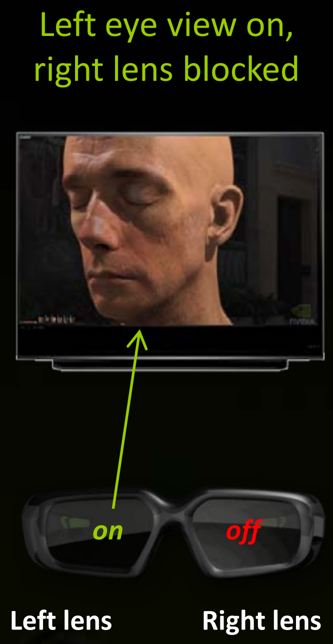
A Stereoscopic display then shows the left eye view for even frames (0, 2, 4, etc) and the right eye view for odd frames (1, 3, 5, etc).

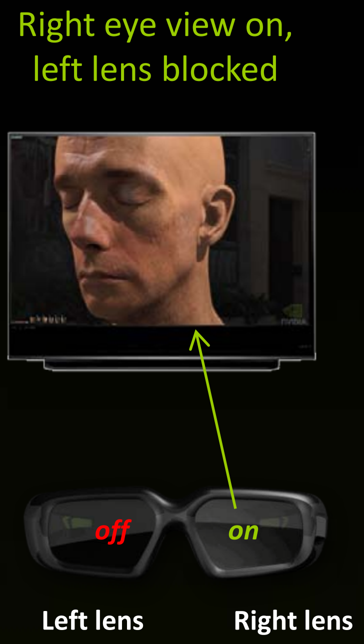


How It Works

In this example active shutter glasses black-out the right lens when the left eye view is shown on the display and black-out the left lens when the right eye view is shown on the display.

This means that the refresh rate of the display is effectively cut in half for each eye. (e.g. a display running at 120 Hz is 60 Hz per eye)





The resulting image for the end user is a combined image that appears to have depth in front of and behind the stereoscopic 3D Display.

NVIDIA 3D Vision

NVAPI Stereoscopic Module

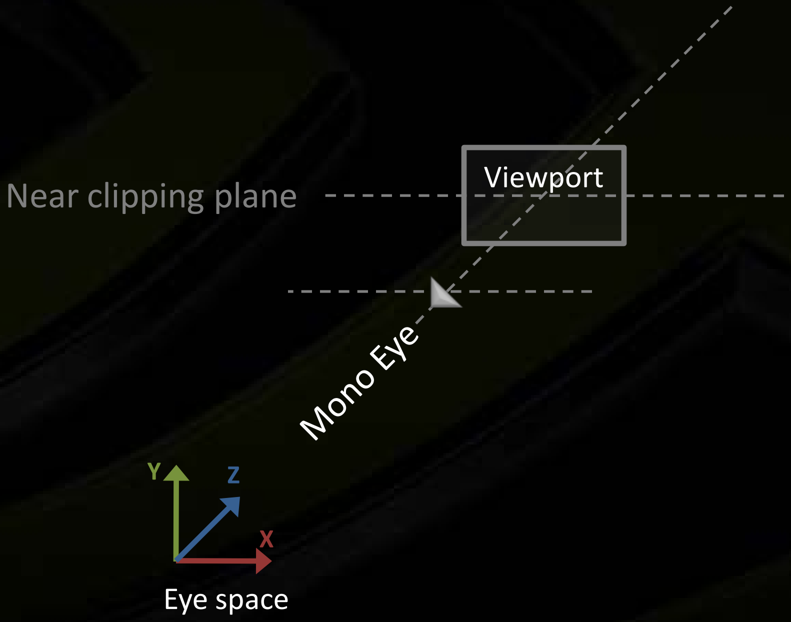
- NVAPI is NVIDIA's core software development kit that allows direct access to NVIDIA GPUs and drivers
- NVAPI now expose a Stereoscopic Module providing access to developer to the Stereoscopic driver settings
  - Detect if the system is 3D Vision capable
  - Manage the stereo profile settings for the game
  - Control dynamically the stereo parameters from within the game engine for a better stereo experience
- For download and documentation http://developer.nvidia.com/object/nvapi.html

Under the hood

STEREOSCOPIC BASICS

Standard Mono

Scene is viewed from one mono eye and projected on Near Clipping plane in Viewport

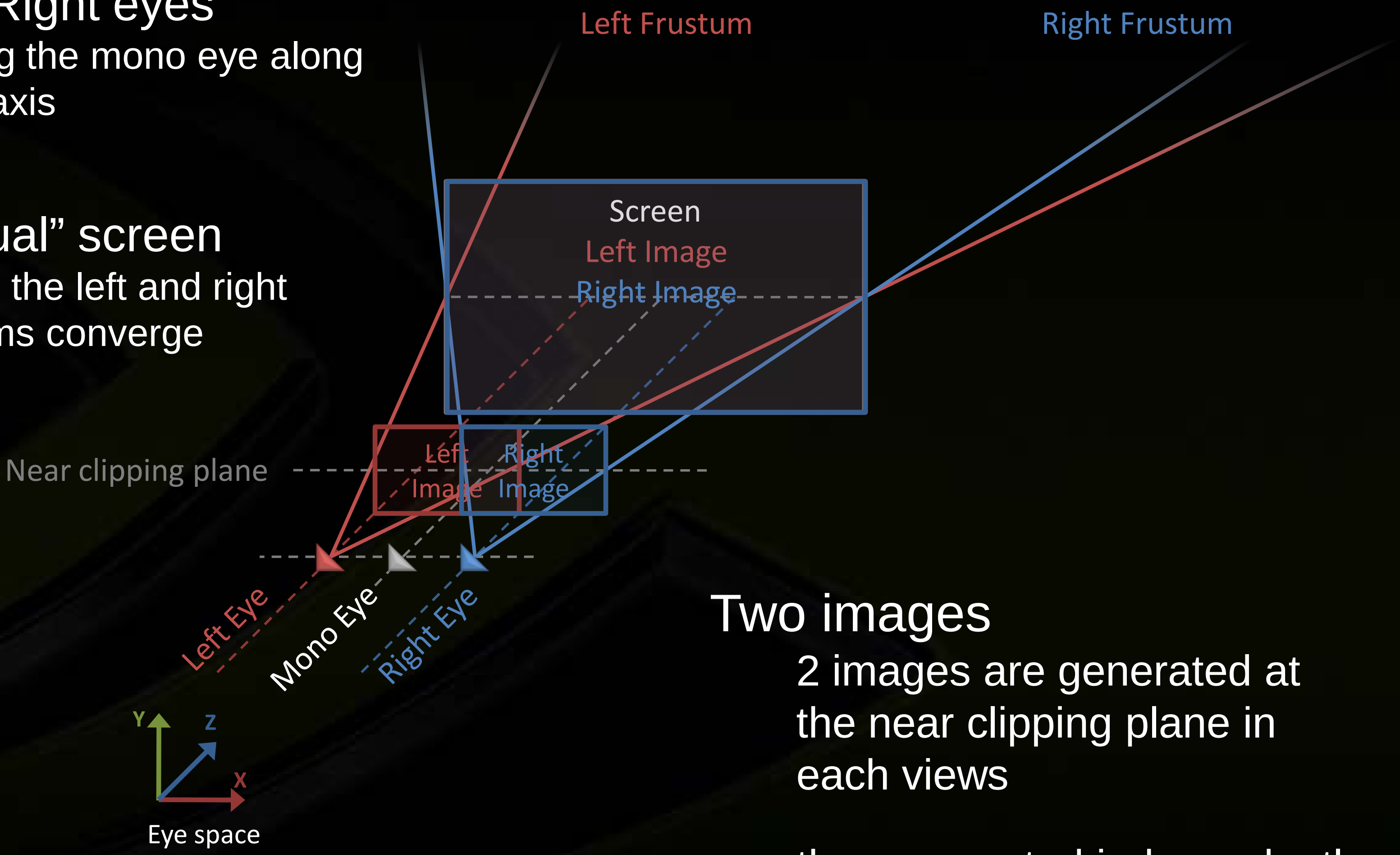


Stereoscopic Basics

Two eyes, one screen, two images

Left and Right eyes
- Shifting the mono eye along the X axis

One “virtual” screen
- Where the left and right frustums converge

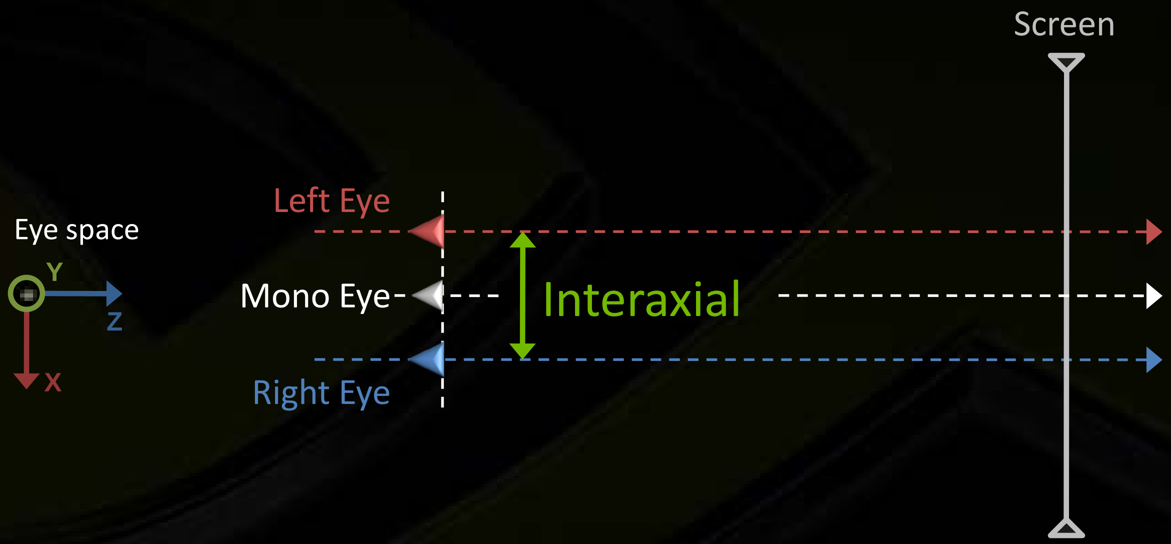


Two images
- 2 images are generated at the near clipping plane in each views
- then presented independently to each eyes of the user on the real screen

Stereoscopic Basics

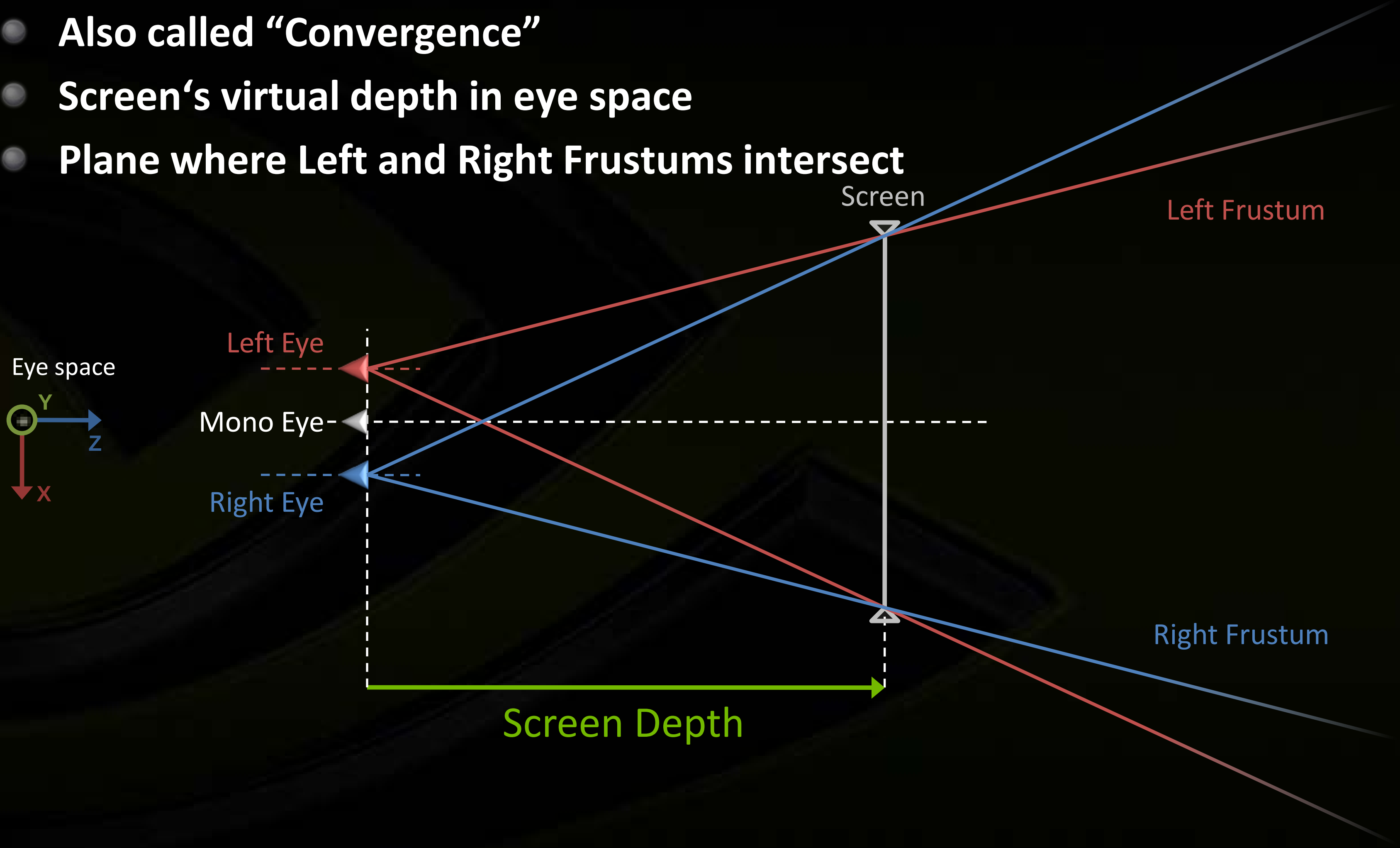
Interaxial

- Distance between the 2 virtual eyes in eye space
- The mono, left & right eyes directions are all parallels



Screen Depth

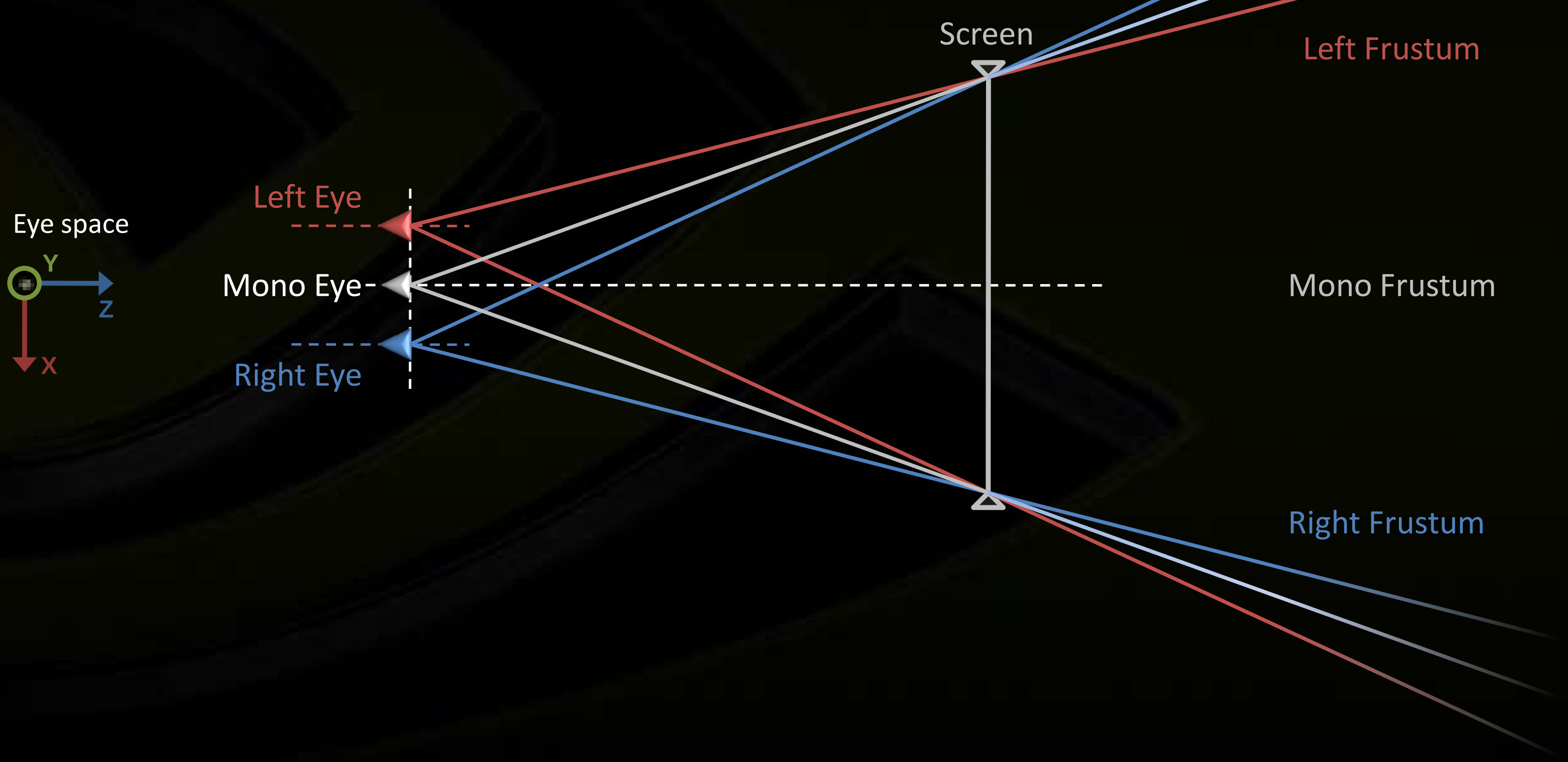
- Also called “Convergence”
- Screen‘s virtual depth in eye space
- Plane where Left and Right Frustums intersect



Stereoscopic Basics

Left / Right Projection

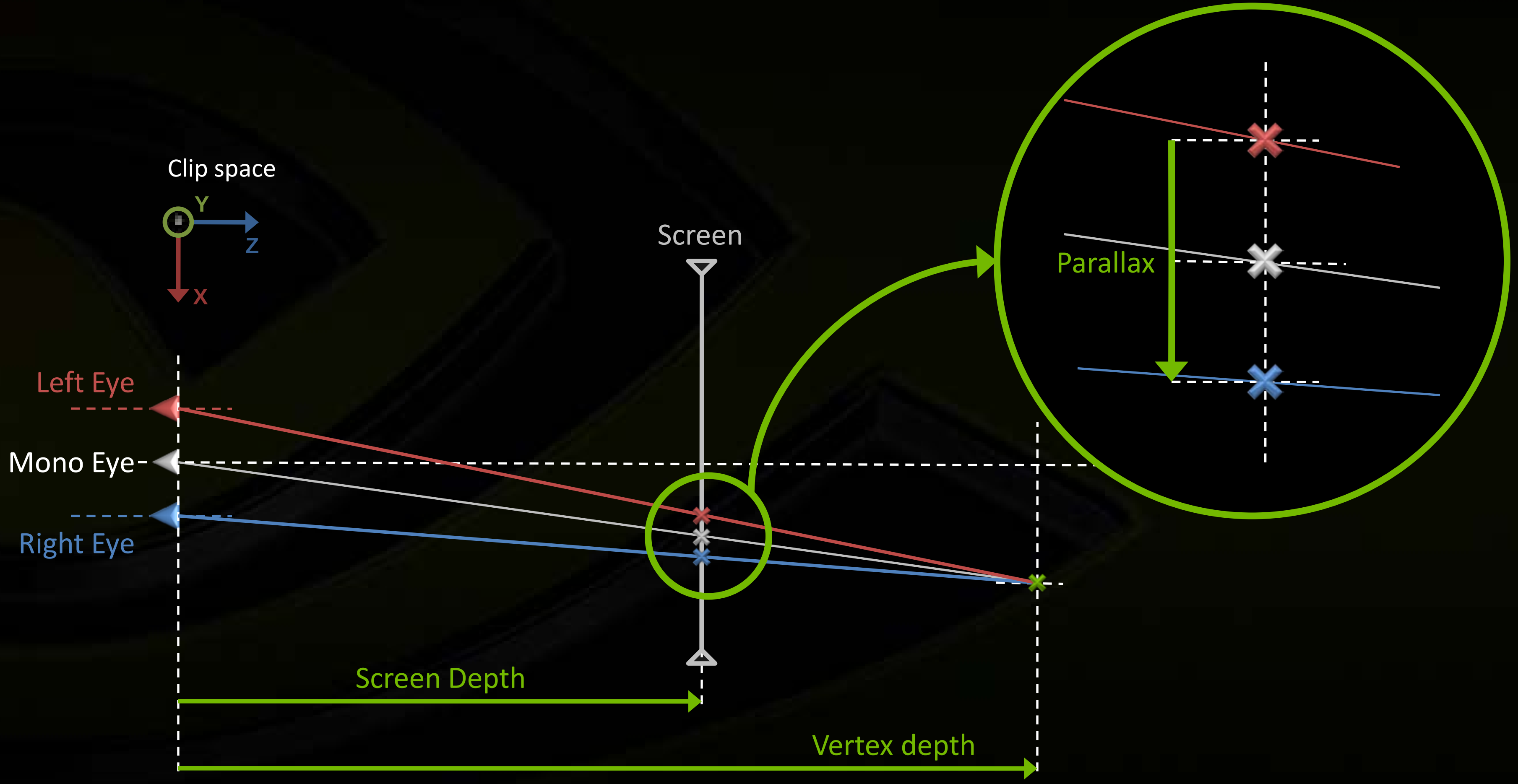
- **Projection matrix for each eyes is a horizontally modified version of regular mono projection matrix**
  - **Shifting X coordinate left or right**



Stereoscopic Basics

Parallax

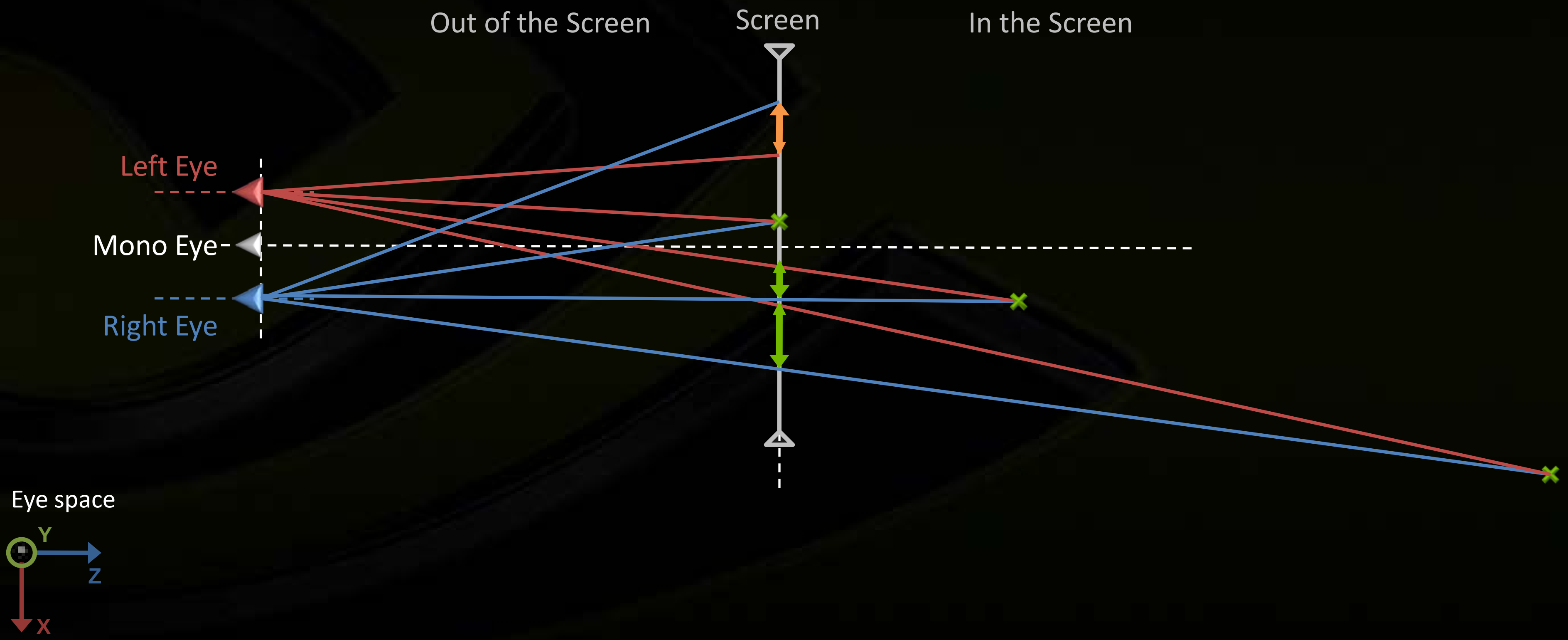
- **Signed Distance on the screen between the projected positions of one vertex in left and right image**
  - **Parallax is function of the depth of the vertex in eye space**



Stereoscopic Basics

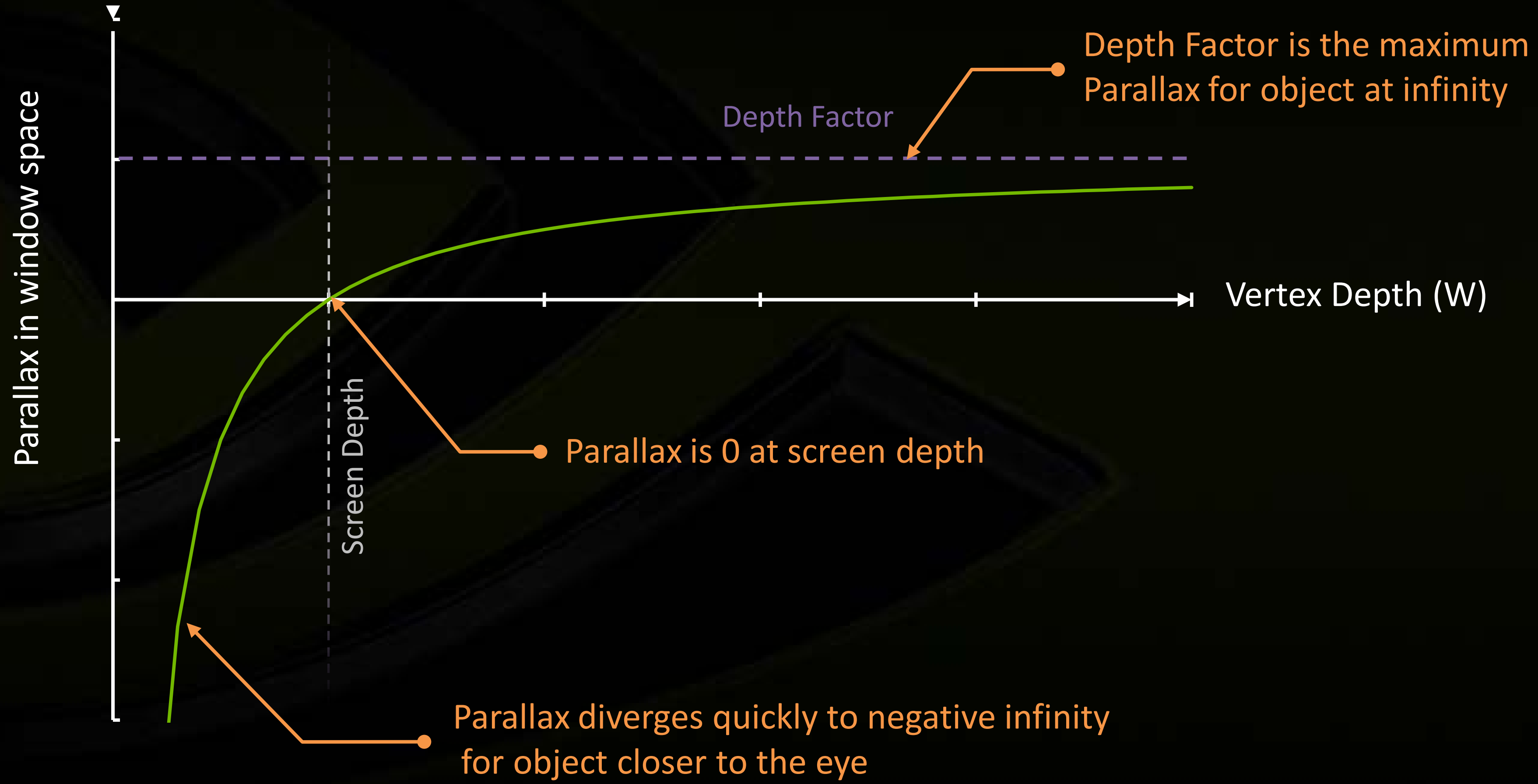
In / Out of the Screen

- Parallax creates the depth perception relative to the screen
- When Parallax is negative, vertex appears Out of the screen



Parallax in equation

- **In normalized window space, after perspective division by W**
  - **Parallax = DepthFactor * (1 – ScreenDepth / W)**



Left / Right surfaces

- The back buffer is duplicated
- Intermediate full screen render targets used to process the final image should also be duplicated
  - High dynamic range, Blur, Bloom
  - Screen Space Ambient Occlusion
- Some render targets DON'T need to be duplicated
  - Shadow map
  - view independent render targets

Stereoscopic Surfaces & Separation

- Stereoscopic rendering is the result of 2 independent actions
- Stereoscopic surfaces
  - Rendering surfaces are duplicated
- Stereoscopic separation
  - Parallax shift is applied to the geometry vertices

Stereoscopic Surfaces in NVIDIA driver

- Automatic duplication is based on the surface size
  - Surfaces equal or larger than back buffer size are duplicated
  - Square surfaces are NOT duplicated
  - Small surfaces are NOT duplicated
  - Heuristic defined by driver profile setting
    - Consult documentation for fine tuning
- Explicit duplication will be available programmatically soon

Stereoscopic Separation in NVIIDA driver

- Every Draw call
  - Issued twice in left & right surfaces
  - Parallax shift is applied in the vertex shader to the vertex’s clip position output
- When separation is not required, render geometry at Screen depth
  - Full screen quad to do image filtering
- No separation if the surface is mono

Recommendations…

RENDERING TECHNIQUES

3D Objects

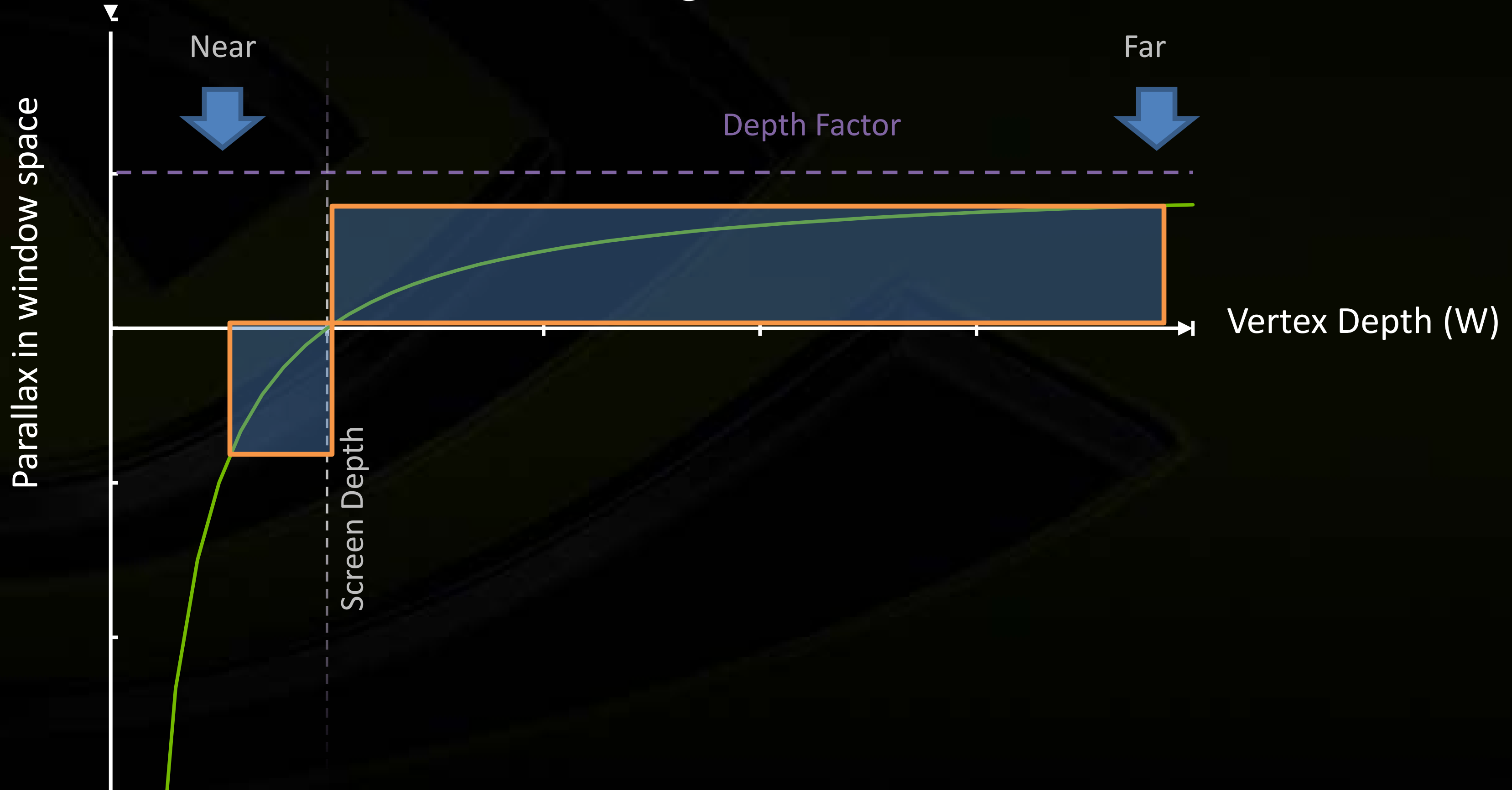
- **All the 3D objects should be rendered using a unique Perspective Projection in a given frame**
- **The sky box should be drawn with a valid depth further than the regular scene**
  - **Best is at the Far distance**

- **All the 3D objects must have a coherent depth relative to the scene**

Rendering Techniques

Parallax Budget

- **Define Screen Depth for best usage of the parallax budget in screen and out of the screen**
  - **Just like when dealing with Near and Far**



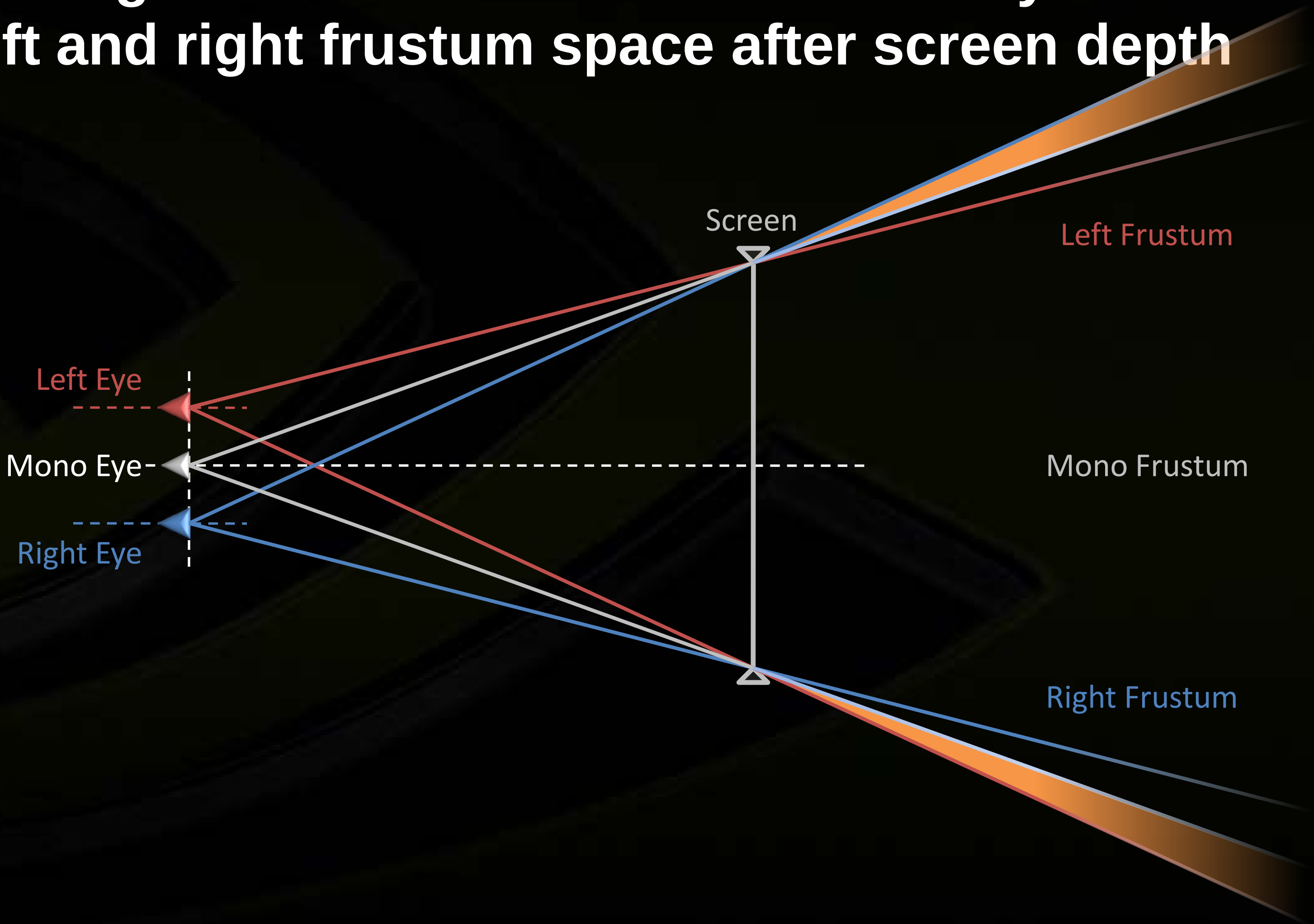
Defining Screen Depth

- Maximizing the usage of the depth range to see the variation of separation along the depth
  - Scene Depth should range between Screen Depth and 100*Screen Depth

- With NVIDIA 3D Vision, Screen depth can be defined from NVAPI

3D Objects Culling

- Culling should be extended horizontally to see extra left and right frustum space after screen depth



2D Objects

- 2D Overlay elements (defined in window space) must be drawn at a valid Depth
  - At the screen depth to look mono
    - Head Up Display interface
    - UI elements
  - At the correct depth when interacting with the 3D scene
    - Mouse Cursor at the pointed object's depth
      Can not use the HW cursor
    - Crosshair
    - Labels or billboards in the scene
  - The depth is provided by the game engine
- Needs to modify the projection to take into account depth

2D Objects hybrid projection

Proposed vertex shader

```
float4 2DObjects_VertexShader(
    in float2 posClip : POSITION, // Input position in clip space
    uniform float depth           // Depth where to draw the 2D object
    ) : POSITION                  // Output the position in clip space
{
    return float4(
        posClip.xy * depth,   // Simply scale the posClip by the depth
                              // to compensate for the division by W
                              // performed before rasterization

        0,       // Z is not used if the depth buffer is not used
                 // If needed Z = ( depth * f – nf )/(f – n);

        depth ); // W is the Z in eye space
}
```

Out of the screen objects

- The user’s brain is fighting against the perception of hovering objects out of the screen
  - Extra care must be taken to achieve a convincing effect
- Make sure object is not clipped by the edges of the window
  - Be aware of the extra guard bands
- Move object slowly from inside the screen to the outside area to give eyes time to adapt
  - Make smooth visibility transitions
  - No blinking
- Realistic rendering helps

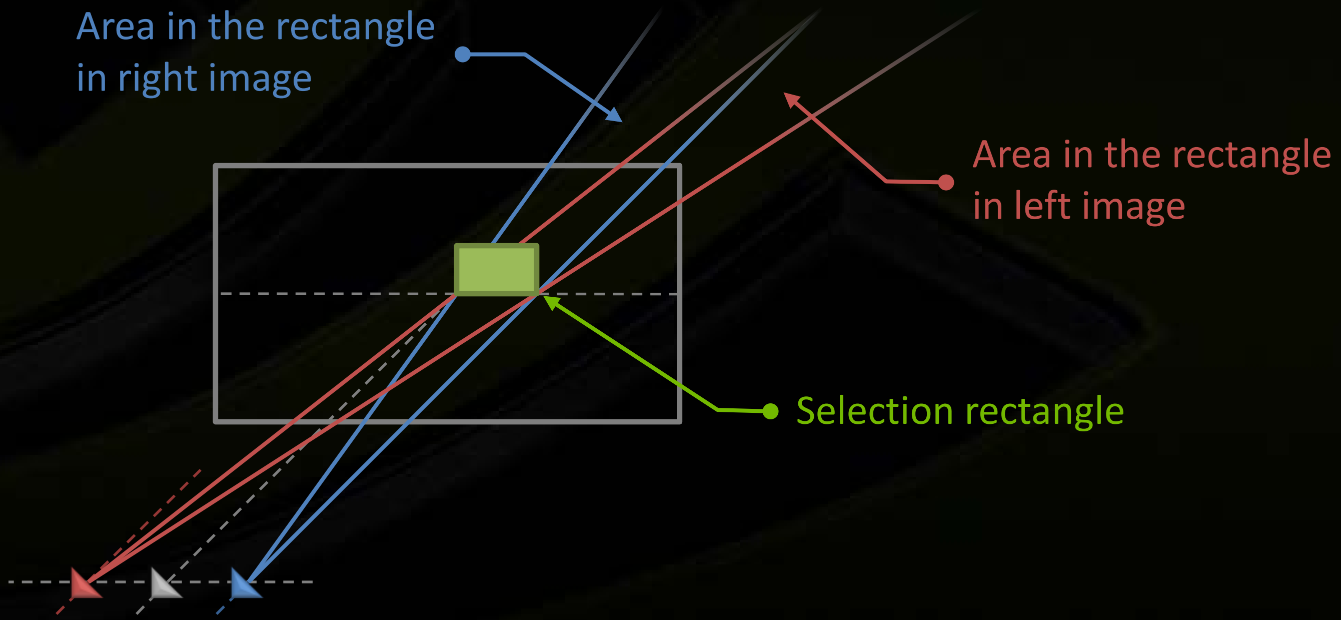
The cool part

ADVANCED STEREOSCOPIC EFFECTS

Selection marquee

Something is broken

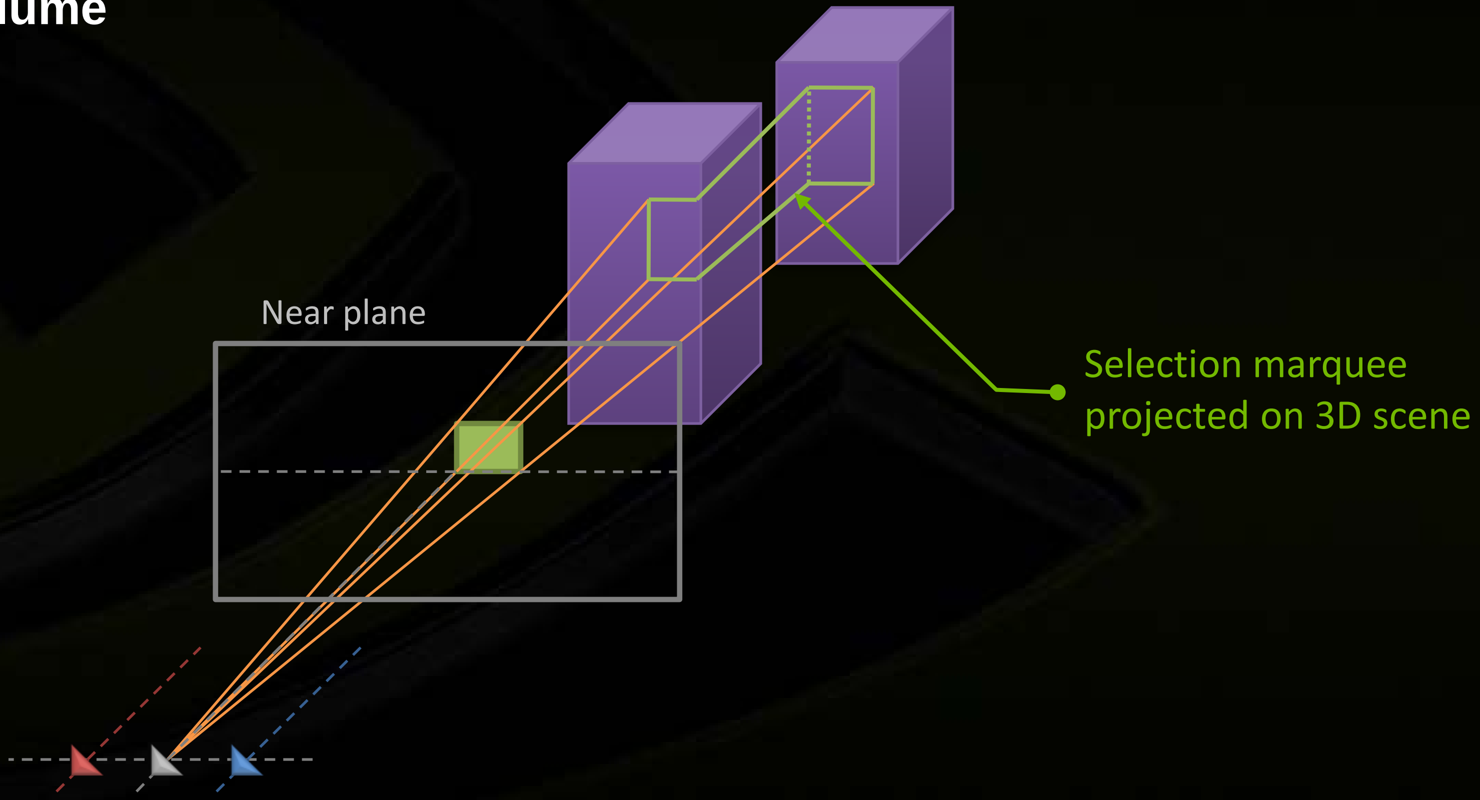
- **In mono, the selection marquee is naturally defined in the window space as a 2D shape**
  - **It can be simply drawn as a rectangle in 2D in window space**
- **In stereo, the same solution does not work**
  - **Each view defines its own selection rectangle in its clipping space**
  - **The vertical edges of the rectangles don't match**


Area in the rectangle in right image
Area in the rectangle in left image
Selection rectangle

Selection marquee

In fact it's a volume

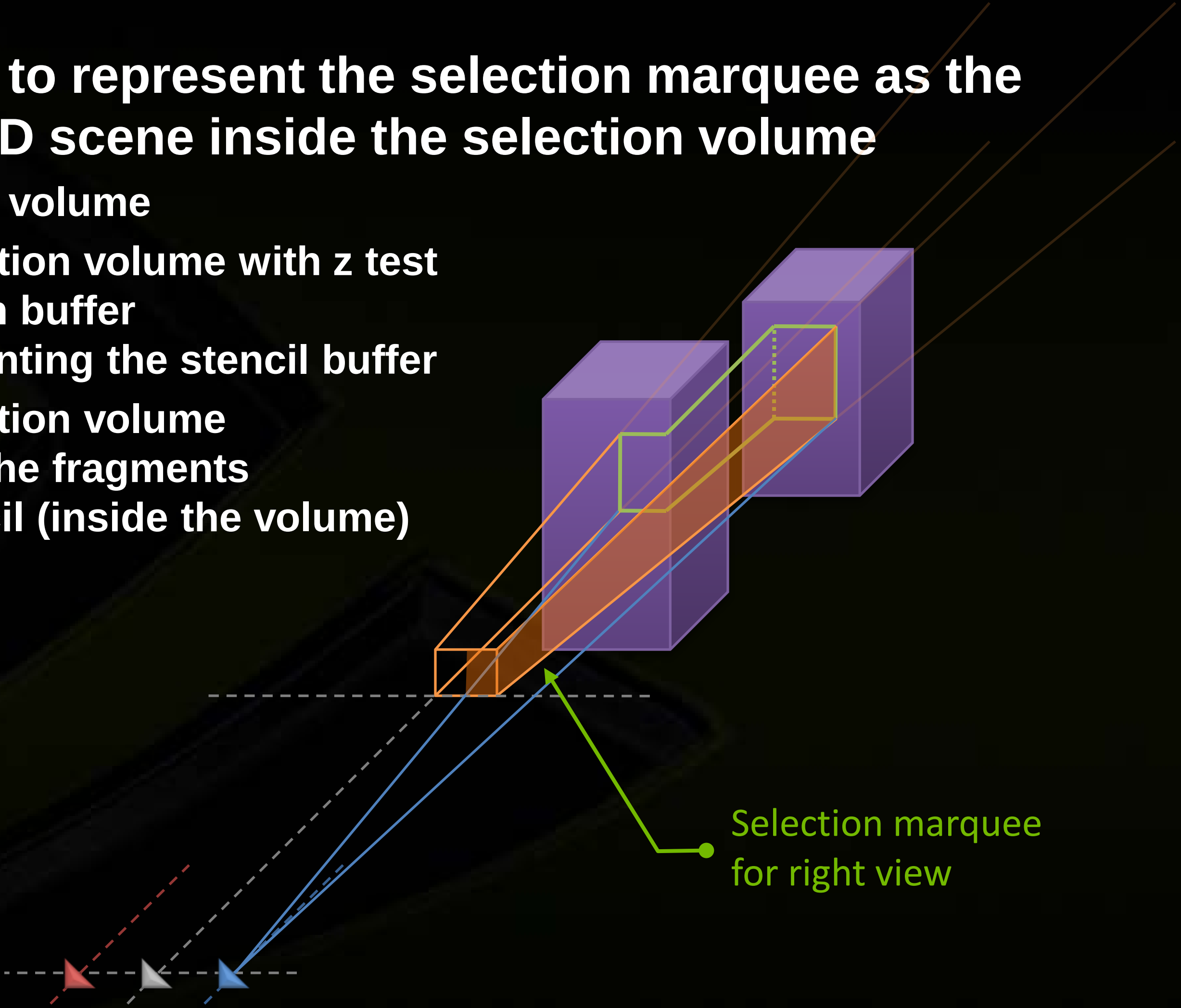
- The selection marquee seen in mono is in fact a 2D shape projected on the 3D scene and intersecting with the scene surface
- the 2D shape defined in near clip plane is generating a selection volume



Selection marquee

Representing the selection surface

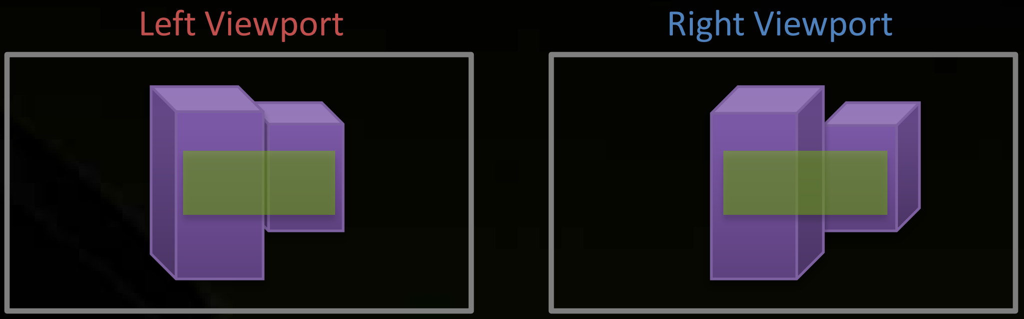
- In stereo, we want to represent the selection marquee as the fragments of the 3D scene inside the selection volume
  - Just like shadow volume
  - Render the selection volume with z test against the depth buffer
    and just incrementing the stencil buffer
  - Render the selection volume
    displaying only the fragments
    with a odd stencil (inside the volume)

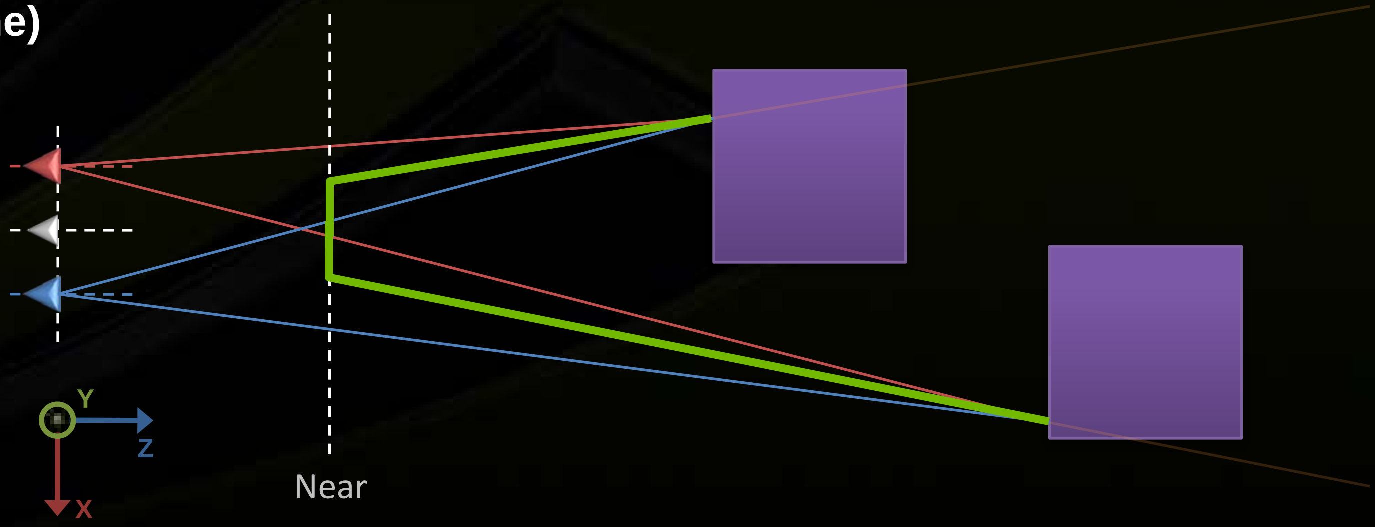


Selection marquee

Representing the selection surface

- Render the selection volume with z test against the depth buffer and just incrementing the stencil buffer
- Render the selection volume
  displaying only the fragments
  with a odd stencil
  (inside the volume)

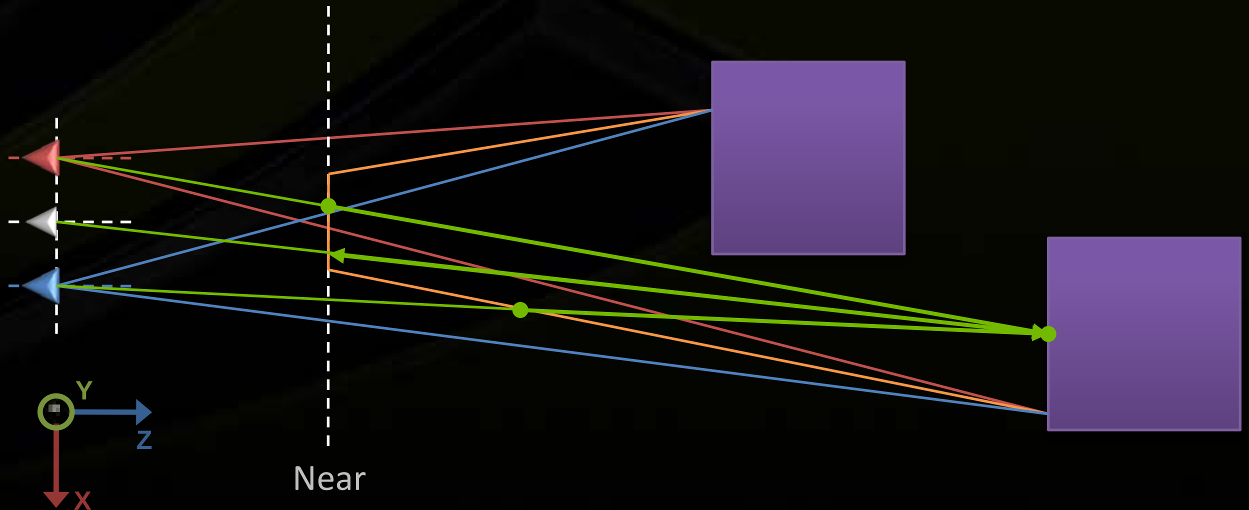




Selection marquee

Going back to the 2D space

- For shading purpose, we need to know the 2D coordinate of the fragment expressed in the original 2D selection space
  - The fragment is on the selection volume
  - At that fragment, fetch the view space position of the 3D scene from a previously rendered position buffer
  - Transform back that position to the mono 2D window space



Advanced Stereoscopic Effects

Selection marquee

Conclusion

- It works with any 2D selection marquee shape, just need to generate the corresponding selection volume
  - Lasso !
- An accurate solution to represent the selection marquee in stereo
- Requires an extra buffer containing the position in view space of 3D scene fragments

3D Video

Displaying a stereo frame

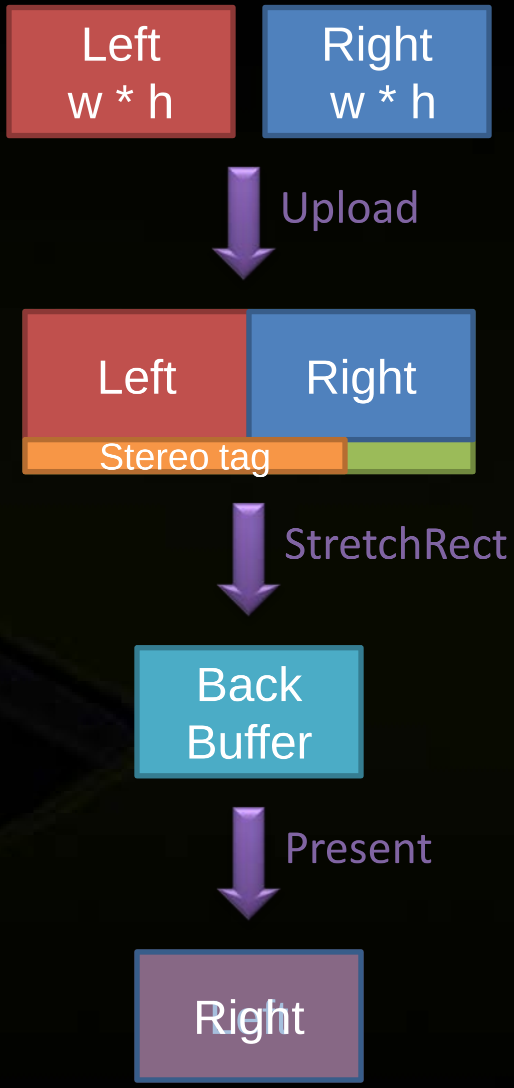
- How to display existing stereo content in Stereo ?
  - Replay 3D video
  - Display 3D photos
  - Prerecorded cut scenes

Advanced Stereoscopic Effects

3D Video

Presenting a stereo frame from 2 images

- On a 3D vision driver enabled platform under DirectX9
- Upload new frame (left and right) in a video surface
  - 2 * Width
  - Height + 1
  - Left image on the left, Right image on the right
  - Edit stereo tag in the last extra raw
- StretchRect the video surface into the back buffer
- Presenting the back buffer will display the stereo frame



Advanced Stereoscopic Effects

3D Video

Presenting a stereo frame: Creating the source image

```
IDirect3DSurface9* gImageSrcLeft;  // Left Source image surface in video memory
IDirect3DSurface9* gImageSrcRight; // Right Source image Surface in video memory

int gImageWidth = 1680;     // Source image width
int gImageHeight = 1050;    // Source image height

IDirect3DSurface9* gImageSrc = NULL; // Source stereo image beeing created

D3DDev->CreateOffscreenPlainSurface(
    gImageWidth * 2,                // Stereo width is twice the source width
    gImageHeight + 1,               // Stereo height add one raw to encode signature
    D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT, // Surface is in video memory
    &gImageSrc, NULL);

// Blit left src image to left side of stereo
RECT srcRect = { 0, 0, gImageWidth, gImageHeight };
RECT dstRect = { 0, 0, gImageWidth, gImageHeight };
D3DDev->StretchRect(gImageSrcLeft, &srcRect, gImageSrc, &destRect, D3DTEXF_LINEAR);

// Blit left src image to left side of stereo
dstRect = {gImageWidth, 0, 2*gImageWidth, gImageHeight };
D3DDev->StretchRect(gImageSrcRight, &srcRect, gImageSrc, &destRect, D3DTEXF_LINEAR);
```

3D Video

Stereo signature

```
// Stereo Blit defines
#define NVSTEREO_IMAGE_SIGNATURE 0x4433564e //NV3D

typedef struct  _Nv_Stereo_Image_Header
{
    unsigned int    dwSignature;
    unsigned int    dwWidth;
    unsigned int    dwHeight;
    unsigned int    dwBPP;
    unsigned int    dwFlags;
} NVSTEREOIMAGEHEADER, *LPNVSTEREOIMAGEHEADER;

// ORed flags in the dwFlags fiels of the _Nv_Stereo_Image_Header structure above
#define     SIH_SWAP_EYES                0x00000001
#define     SIH_SCALE_TO_FIT             0x00000002
```

Advanced Stereoscopic Effects

3D Video

Tagging stereo image with Stereoscopic signature

```
// Lock the stereo image
D3DLOCKED_RECT lr;
gImageSrc->LockRect(&lr,NULL,0);

// write stereo signature in the last raw of the stereo image
LPNVSTEREOIMAGEHEADER pSIH =
    (LPNVSTEREOIMAGEHEADER)(((unsigned char *) lr.pBits) + (lr.Pitch * gImageHeight));

// Update the signature header values
pSIH->dwSignature = NVSTEREO_IMAGE_SIGNATURE;
pSIH->dwBPP = 32;
pSIH->dwFlags = SIH_SWAP_EYES; // Src image has left on left and right on right
pSIH->dwWidth = gImageWidth*2;
pSIH->dwHeight = gImageHeight;

// Unlock surface
gImageSrc->UnlockRect();
```

3D Video

Recording a stereo frame

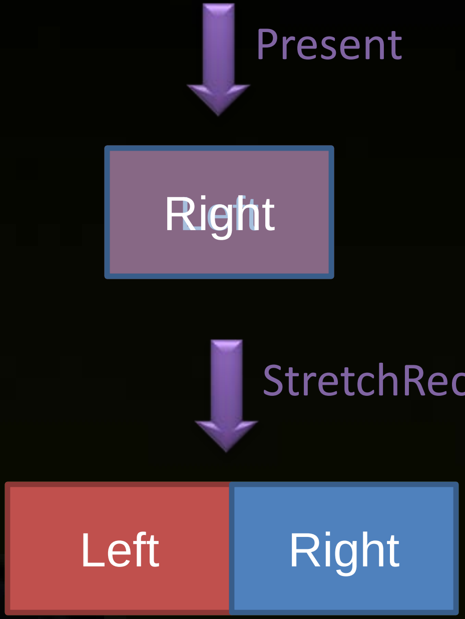
- Saving the full stereoscopic frame (left and right) currently in the back buffer in system memory
  - Reverse operation, it’s called reverse stereo blit
- Generate stereoscopic videos

Advanced Stereoscopic Effects

3D Video

Saving a stereo frame

- On a NVIDIA driver enabled platform under DirectX9
  - Allocate a video image of the stereo image size
    - 2 * Width
    - Height
  - Enable the reverse stereo blit mode with a specific call to NVAPI
  - StrechRect the back buffer into the video image
    - Left image on the left, Right image on the right
  - Disable the reverse stereo blit mode



Advanced Stereoscopic Effects

3D Video

Reverse stereo blitting the backbuffer to a stereo image

```
// Create reverse blit destination double surface
IDirect3DSurface9* gStereoImage = NULL;
D3DDev->CreateOffscreenPlainSurface(
    IMAGE_WIDTH * 2,                  // Stereo width is twice the source width
    IMAGE_HEIGHT,                     // Stereo height is backBuffer height
    D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT, // Surface is in video memory
    &gStereoImage, NULL);

// Turn on reverse blit
NvAPI_Stereo_ReverseStereoBlitControl(gStereoHandle, true);

// Set destination rectangle as double backbuffer's one
RECT backBufferRect = {0, 0, IMAGE_WIDTH, IMAGE_HEIGHT};
RECT stereoImageRect = {0, 0, 2*IMAGE_WIDTH, IMAGE_HEIGHT};

// Reverse blit
D3DDev->StretchRect(BackBuf, &backBufferRect, gStereoImage, &stereoImageRect,
    D3DTEXF_LINEAR);

// Turn off reverse blit
NvAPI_Stereo_ReverseStereoBlitControl(gStereoHandle, false);
```

QUESTIONS ?

Acknowledgements

- Every one in the Stereo driver team !
- The usual suspects in demo and devtech team

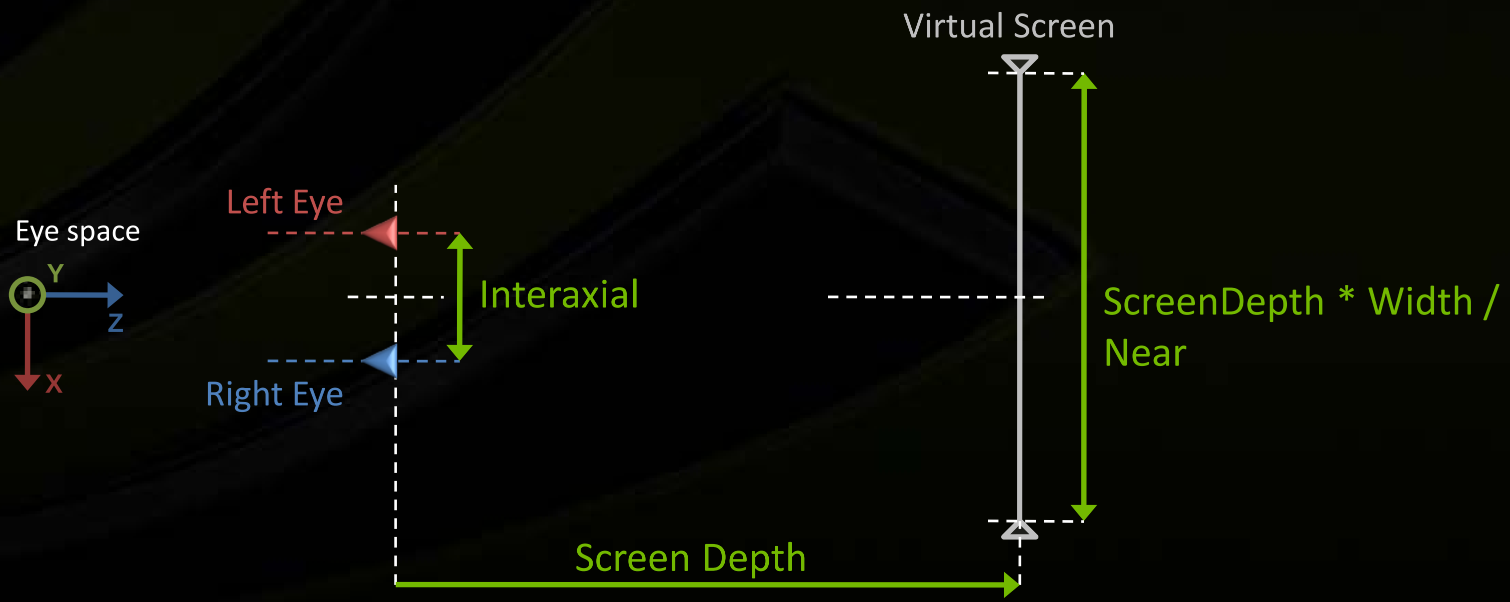
How To Reach Us

- During GDC
  - Expo Suite 656, West Hall
  - Developer Tool Open Chat, 1:30 to 2:30 pm (25th-27th)
- Online
  - Twitter: nvidiadeveloper
  - Website: http://developer.nvidia.com
  - Forums: http://developer.nvidia.com/forums

Stereo Basics

Depth Factor

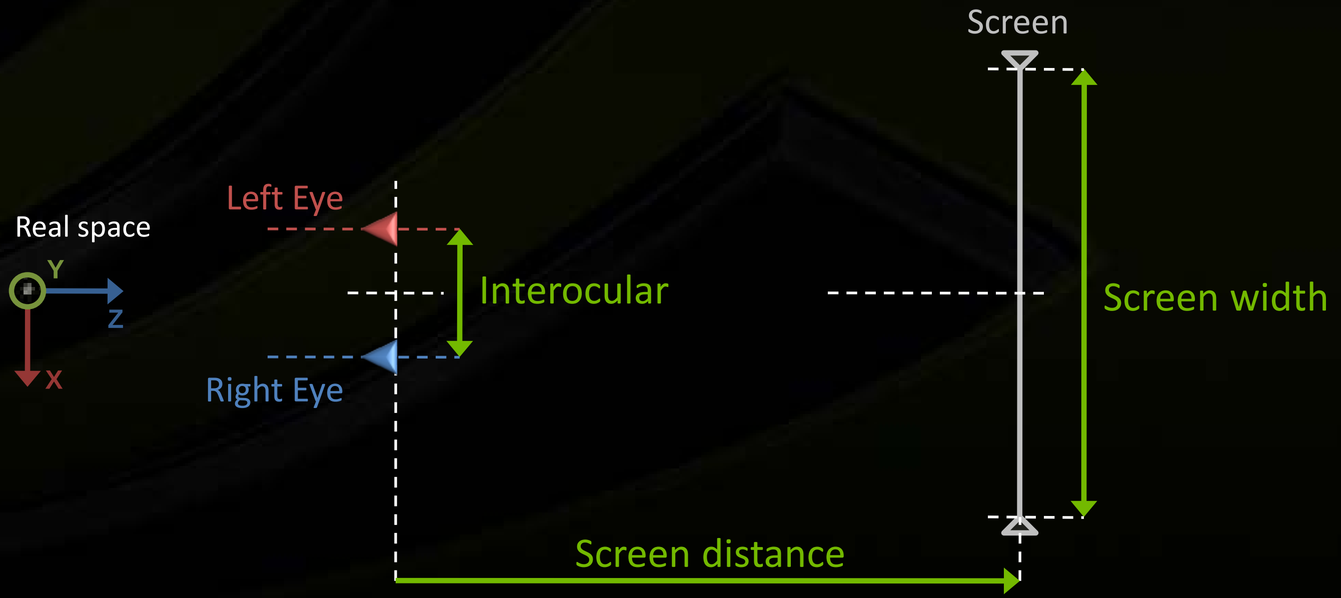
- In virtual space the stereo separation depends on the ratio
  Interaxial / VirtualScreenWidth
- VirtualScreenWidth = ScreenDepth * Width / Near
- User can adjust the depth perception with a scaling factor (DepthScale)
- DepthFactor = DepthScale * Interaxial * Near / (Width * ScreenDepth)



Stereo Basics

Depth Factor in reality

- The reality of the visualization system
  - Interaxial ⇔ Distance between the 2 eyes of the viewer (Interocular)
    - On average, the interocular distance is 2.5'' (6.4 cm)
  - Screen width is fixed
  - Screen distance to the viewer is recommended for a comfortable usage
    - The bigger the screen the further away is the viewer
- Depth Factor is function of **Interocular / ScreenWidth**


Screen
Real space
Y
Z
X
Left Eye
Interocular
Right Eye
Screen width
Screen distance