Practical 3D Vision in Games

Principle, Implementation and Optimization

Outline

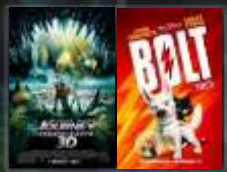
- **NVIDIA 3D Vision™**
  - Stereoscopic driver & HW display solutions
- **Stereoscopy Basics**
  - Definitions and equations
- **Rendering with 3D Vision**
  - What & how to render in stereo mode
- **Issues and Solutions**
  - Issues encountered in real games and our solutions

What does it offer? How it works?

NVIDIA® 3D VISION™

gameworks.nvidia.com

Dimensionalized Experience

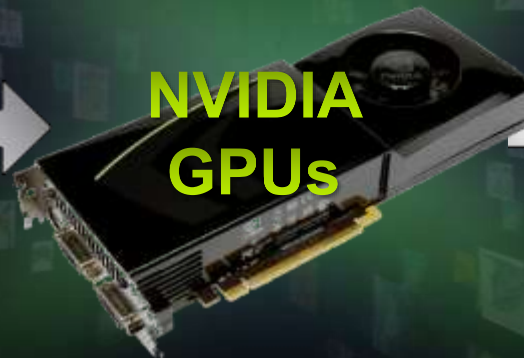


3D Movies

3D Pictures

3D Games

3D Webcast



NVIDIA GPUs

The programmability of the GPU allows NVIDIA to import <u>any</u> 3D data format and decode, convert, or transform the data for viewing on a 3D-Ready displays.

120 Hz LCDs

3D DLP HDTVs

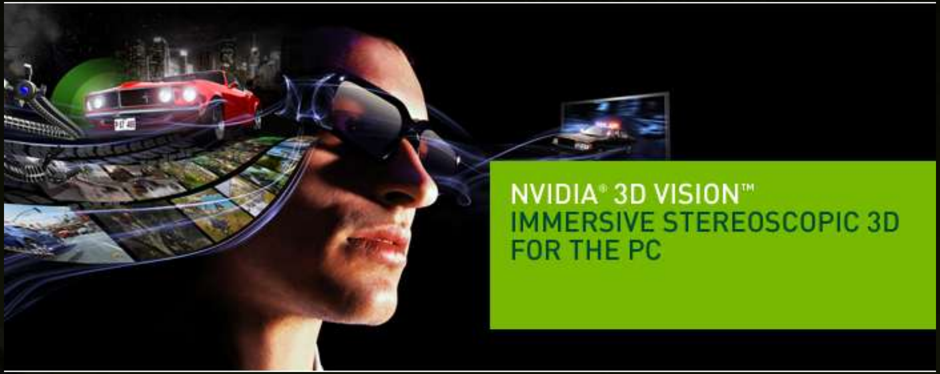
3D Projectors

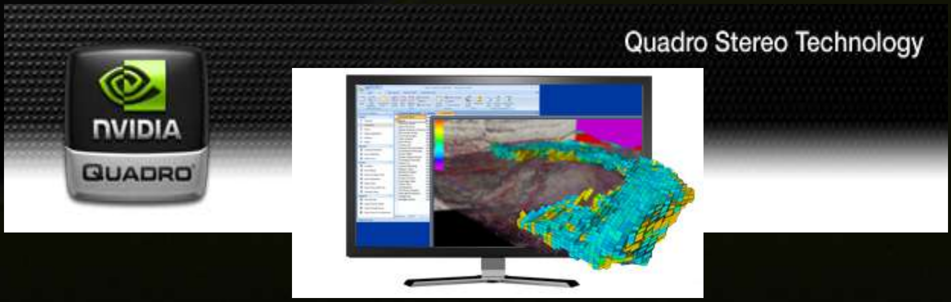
Anaglyph 3D

NVIDIA 3D Vision

Stereo Support

- GeForce
  - Stereo Driver
    - Vista & Win7
    - D3D9 / D3D10
- Quadro
  - GeForce features
  - Professional OpenGL Stereo Quad Buffer
    - Multiple synchronized stereo displays
    - Multi-platform
    - 3D Vision and many other stereo displays





NVIDIA 3D Vision

NVIDIA 3D Vision Solutions

	NVIDIA 3D Vision Discover	NVIDIA 3D Vision
Availability	Bundled with select NVIDIA GPUs for a sneak peak at stereoscopic 3D	Sold as a complete kit for full HD stereoscopic 3D
3D Glasses type	NVIDIA optmized anaglyph (red/cyan)	Wireless Shutter glasses
3D Mode	Anaglyph with optimized color and image processing on the GPU	Page flip 120 Hz & checkerboard pattern 3D
Color Fidelity	Limited Color	Full Color
Display requirements	All desktop LCD and CRT displays	3D-Vision-Ready displays
NVIDIA GeForce GPU	GeForce 8 series and higher	GeForce 8 series and higher
Operating System	Microsoft Windows Vista Microsoft Windows 7	Microsoft Windows Vista Microsoft Windows 7
View 3D pictures	Y	Y
Watch 3D movies	Y	Y
Play real-time 3D games	Y	Y
3D consumer applicaiton	Y	Y

NVIDIA 3D Vision

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.

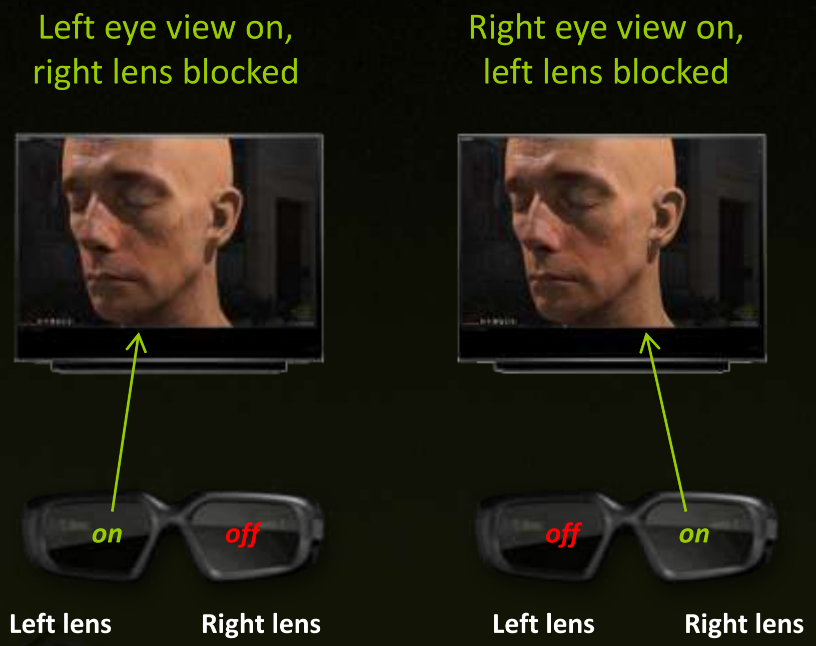
Left Eye view

Right Eye view

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).

NVIDIA 3D Vision

How It Works (cont.)



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.

How It Works (cont. 2)

- 3D Vision designed for transparent game integration
  - Game engine not knowing about stereo driver
  - Driver automatically generate left and right images
  - None to little programming works required
- Full control of stereo parameters also supported
  - For advanced usages and effects
  - Interfaces provided in NVAPI

NVAPI Stereoscopic Module

- NVAPI is NVIDIA's core software development kit that allows direct access to NVIDIA GPUs and drivers
- For advanced 3D Vision programming
  - NVAPI now expose a Stereoscopic Module for control the stereo settings in driver
  - 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

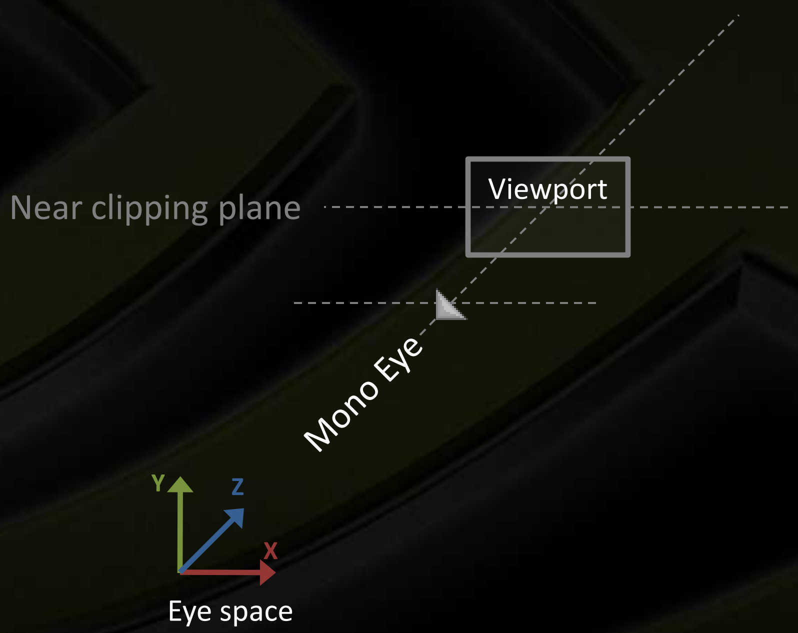
Definitions & Equations

Stereoscopy Basics

gameworks.nvidia.com

Standard Mono Rendering

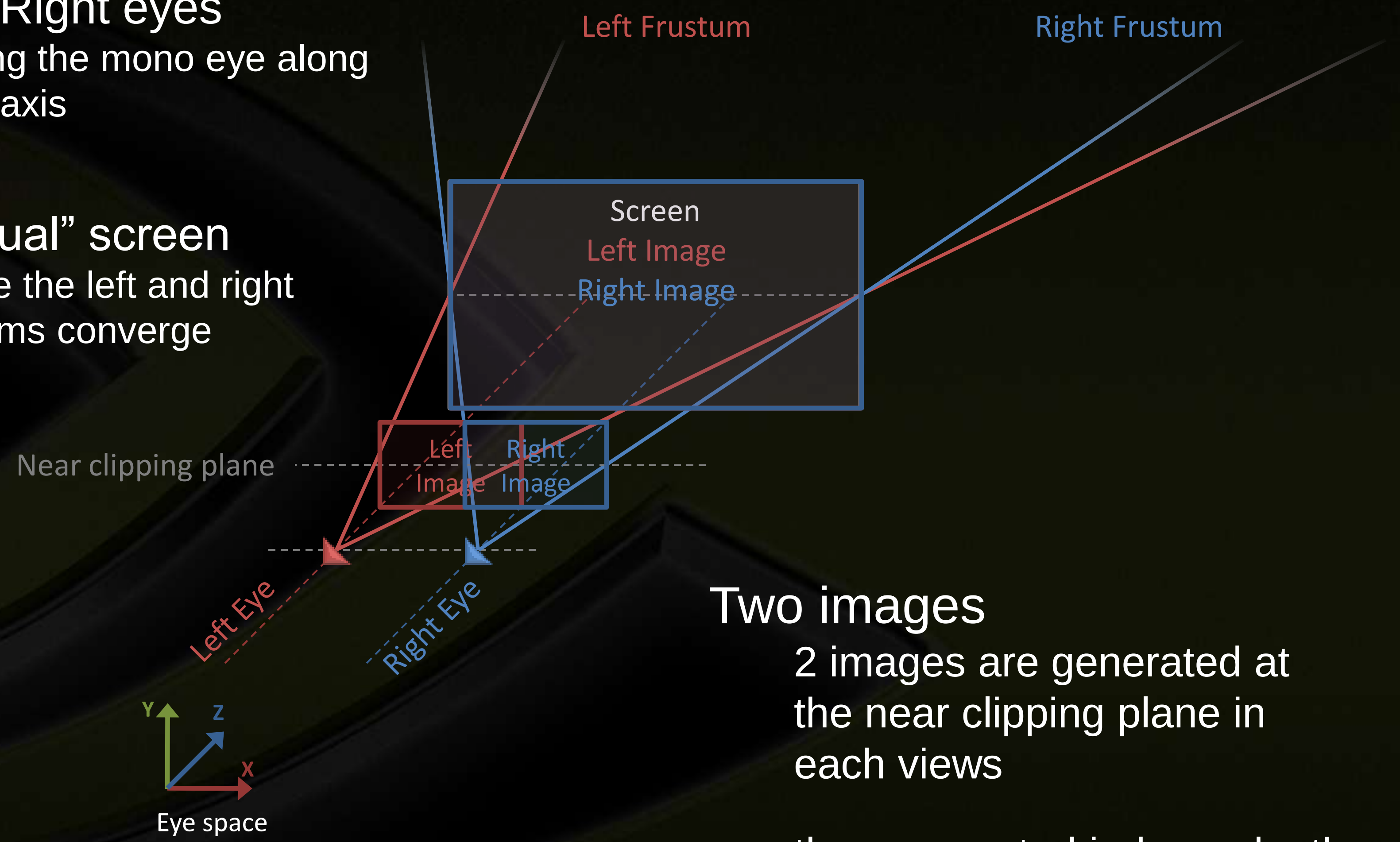
Scene is viewed from one mono eye and projected on Near Clipping plane in Viewport



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

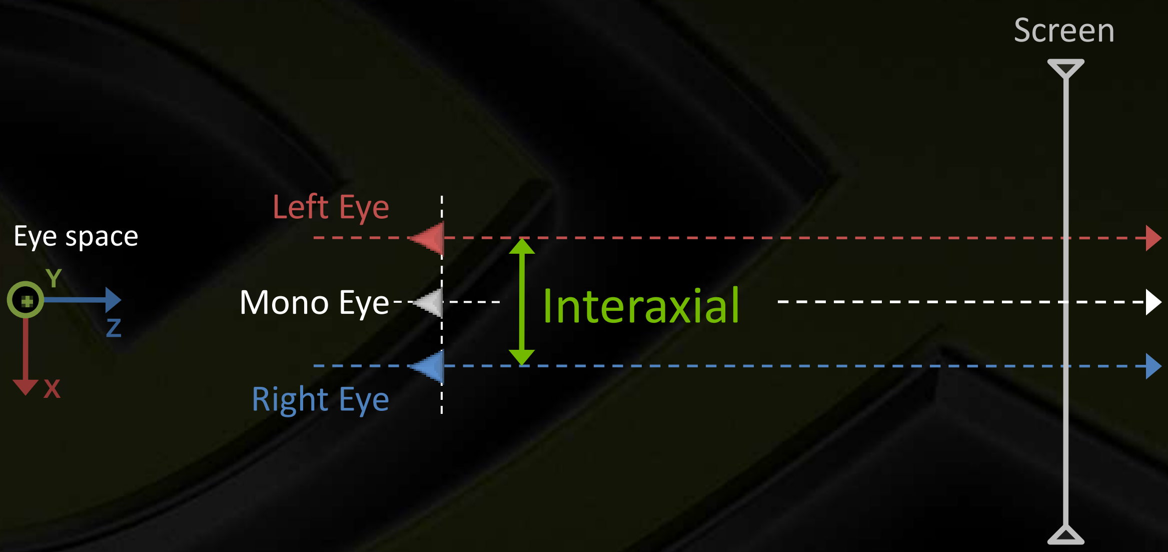
Stereoscopic Rendering

- Render geometry twice from left and right viewpoints into left and right images

- 3 independent modifications to standard pipe
  - Use stereo surfaces
    - Duplicate render surfaces
  - Do stereo drawcalls
    - Duplicate drawcalls
  - Apply stereo separation
    - Modify projection matrix

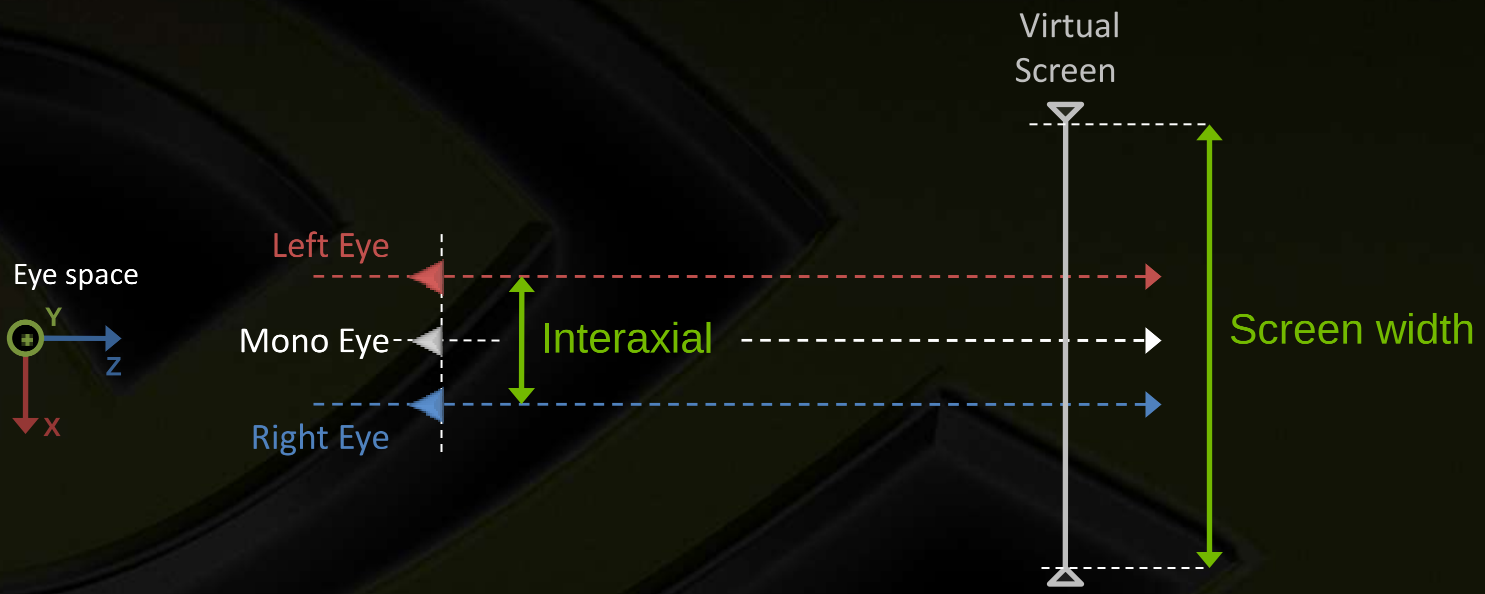
Definition: Interaxial

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



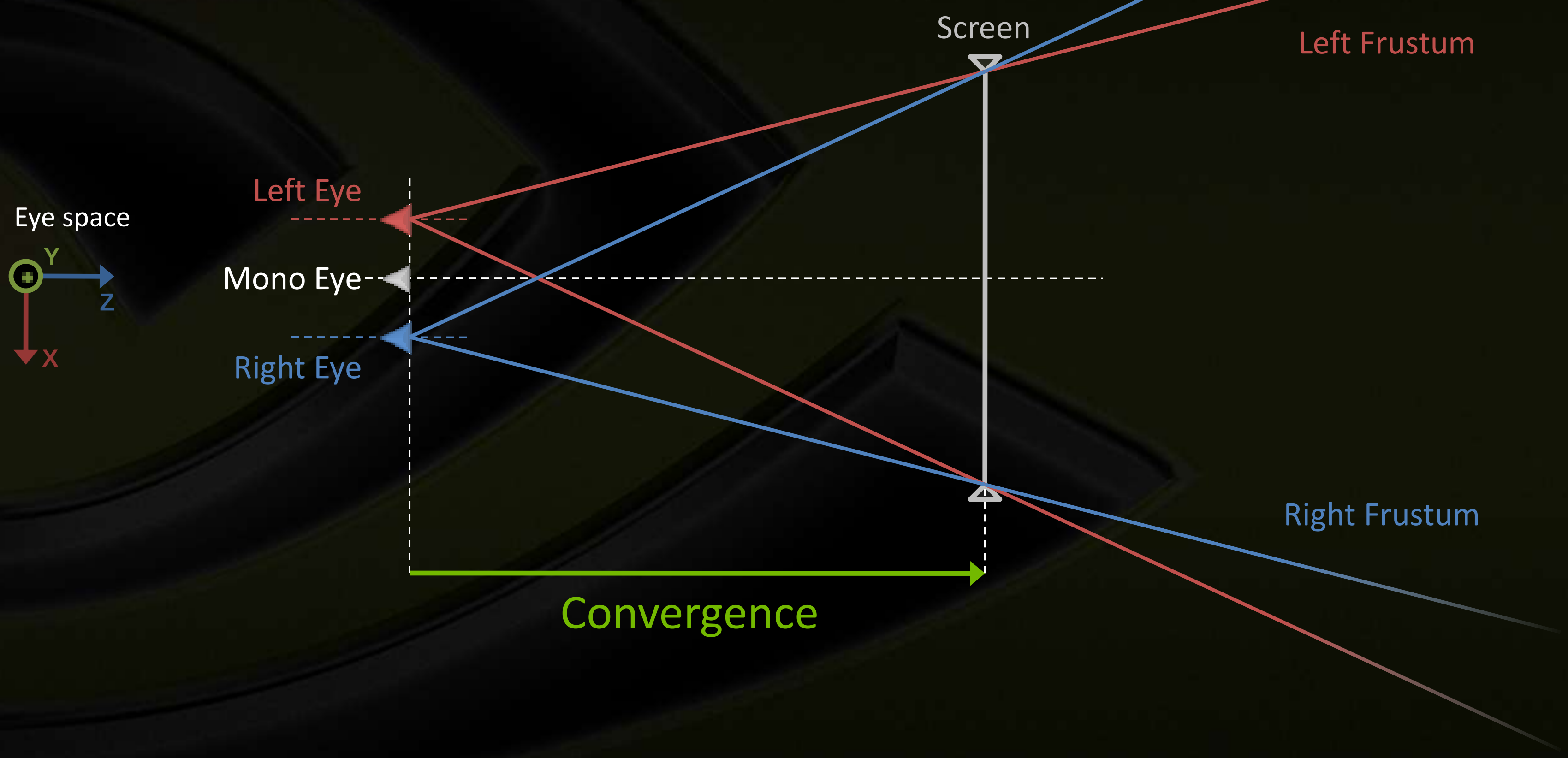
Definition: Separation

- **Normalized version of interaxial by the virtual screen width**
  Separation = Interaxial / ScreenWidth



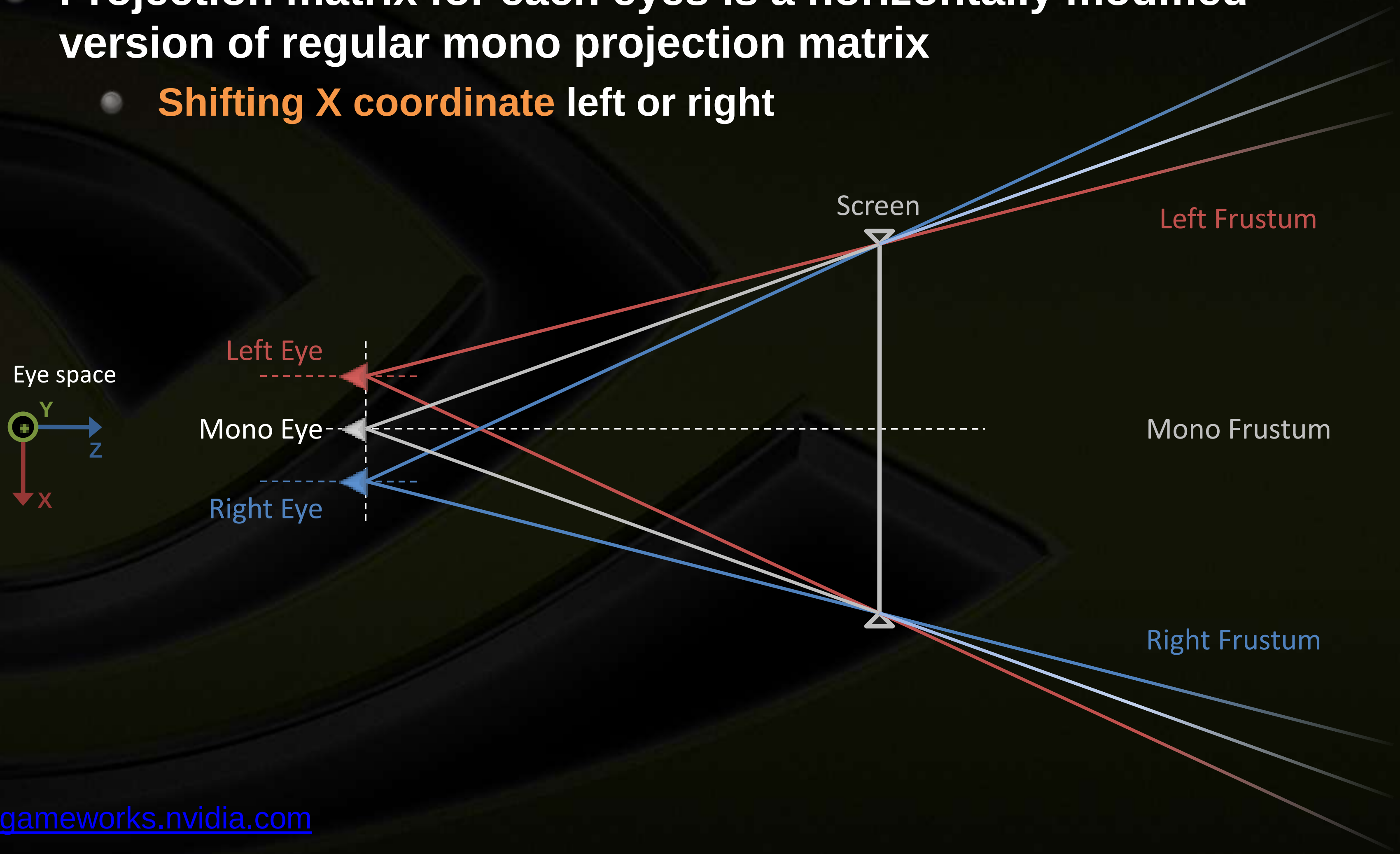
Definition: Convergence

- Also called "Screen Depth"
- Screen's virtual depth in eye space
- Plane where Left and Right Frustums intersect



Left / Right Projection

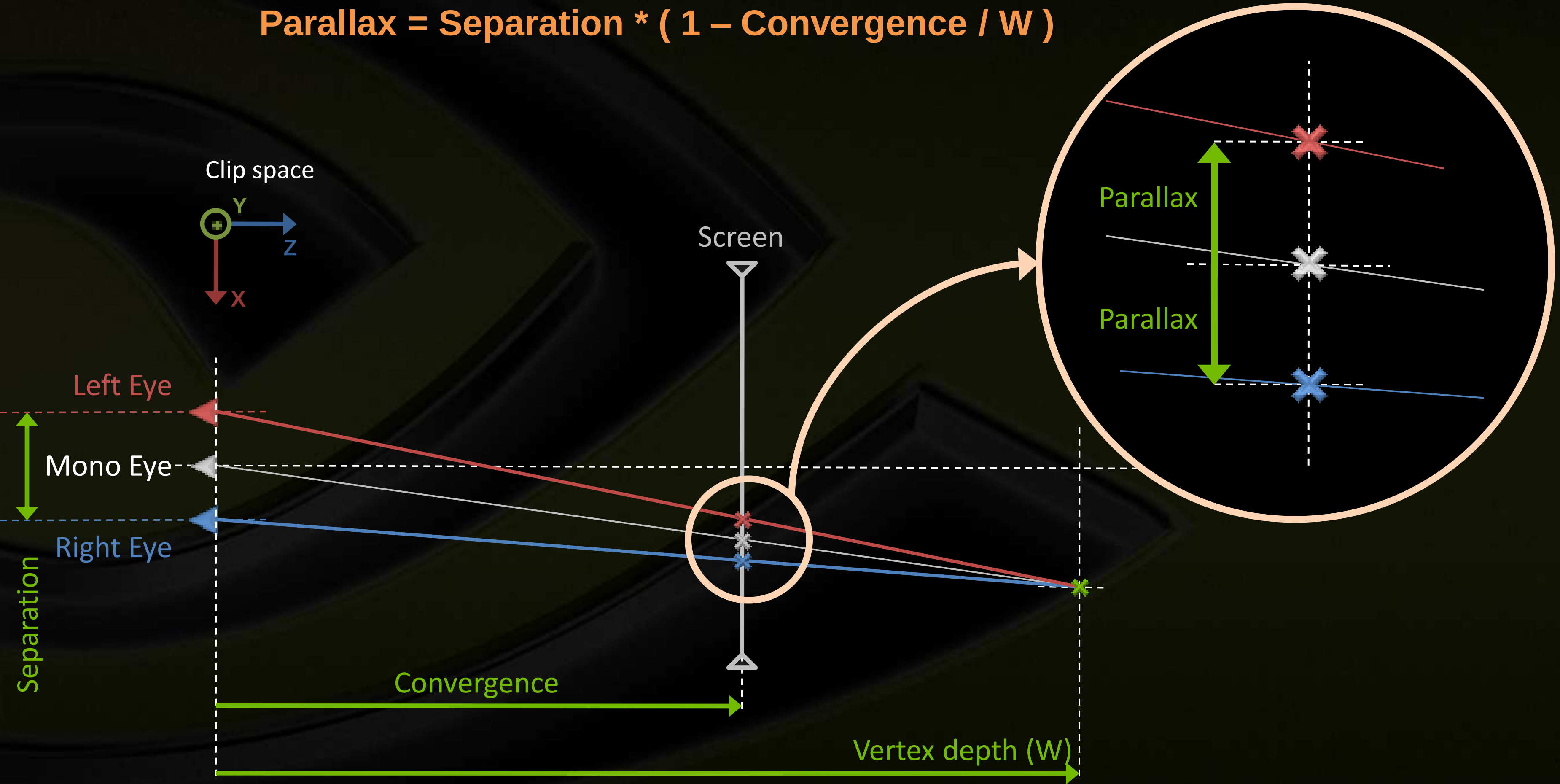
- **Projection matrix for each eyes is a horizontally modified version of regular mono projection matrix**
  - **Shifting X coordinate left or right**



Stereoscopy Basics

Definition: Parallax

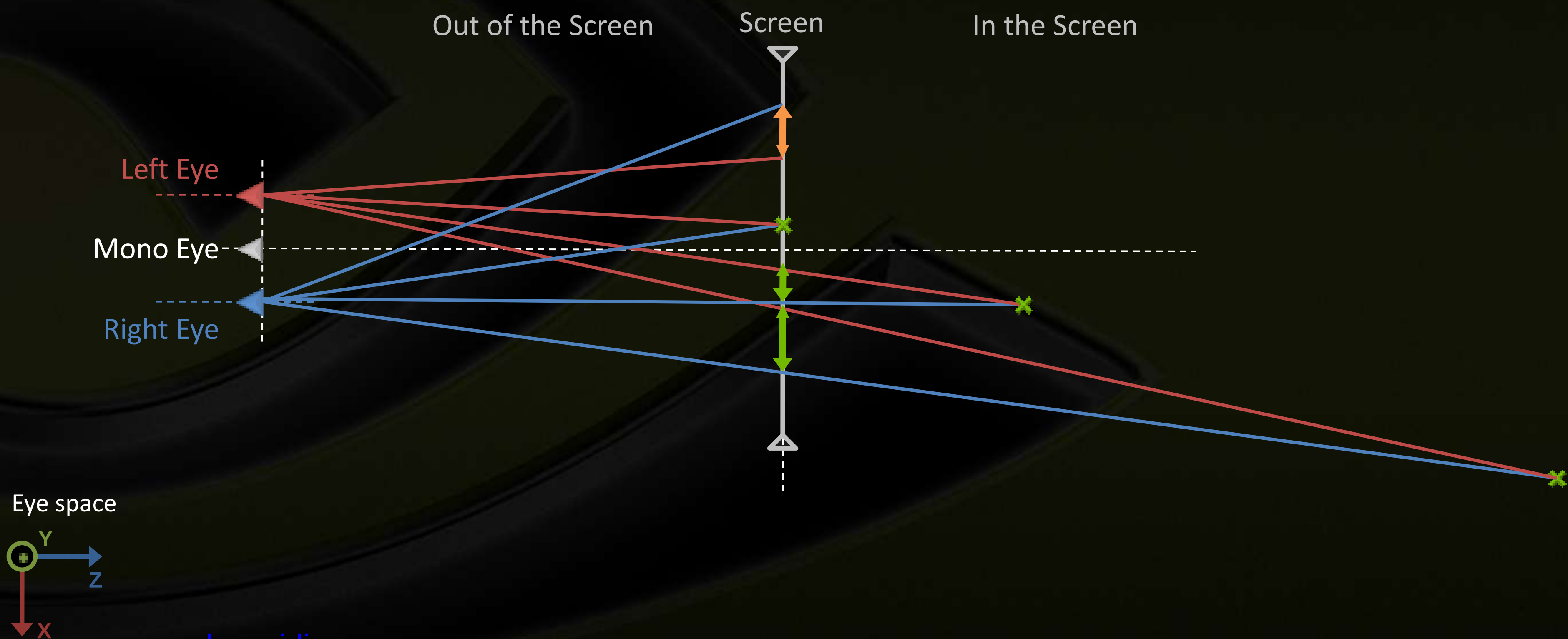
- **Signed distance on the screen between the two projected positions of a vertex**
  - **Parallax is function of the depth of the vertex in eye space**
    Parallax = Separation * (1 – Convergence / W)



Stereoscopy 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**

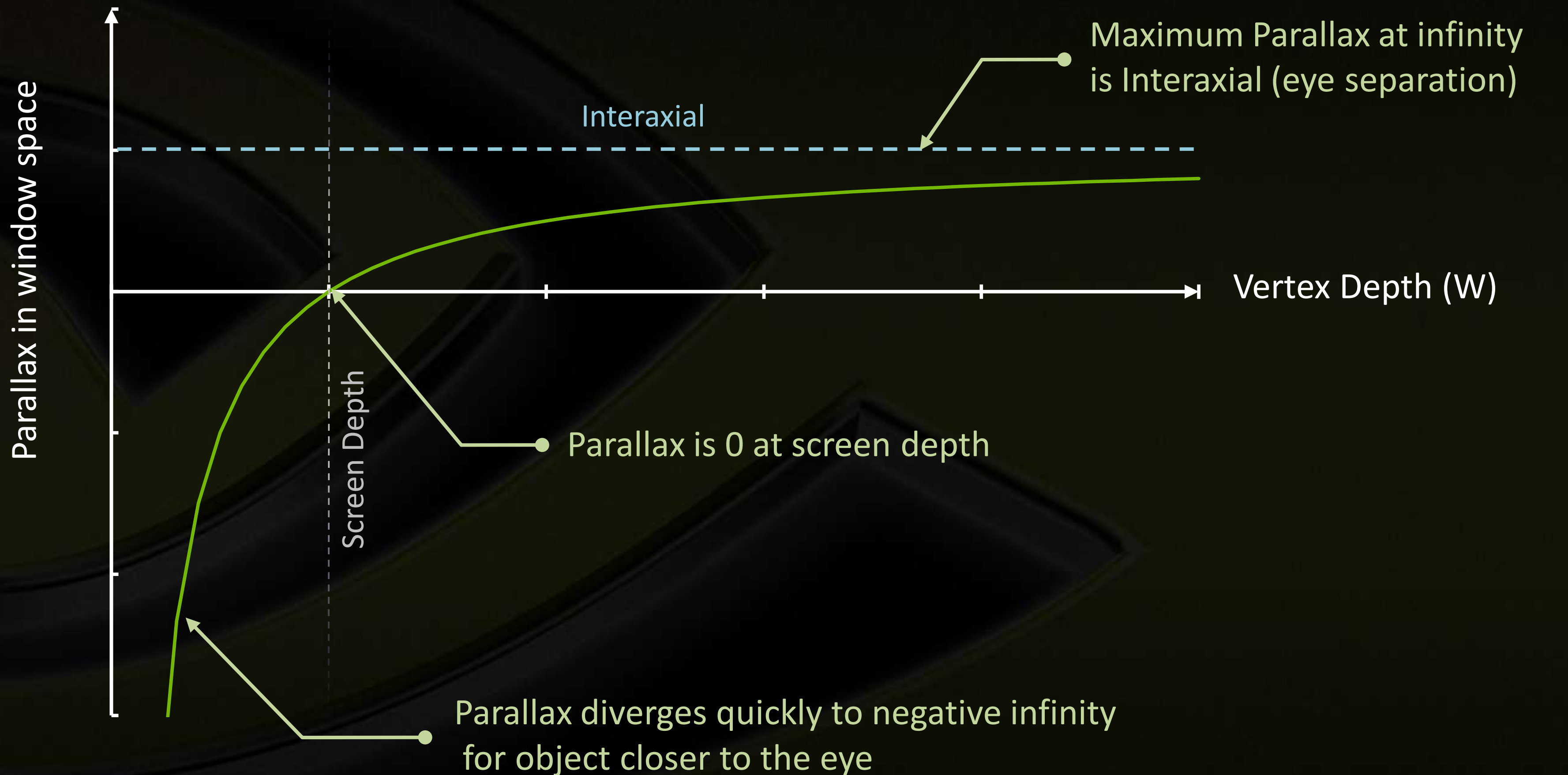


gameworks.nvidia.com

Stereoscopy Basics

Parallax in equation

- **Parallax = Separation * (1 – Convergence / W)**



Checklist for Definitions & Equations

- **Interaxial & Separation**
  - The actual & normalized distance between the two eyes
- **Convergence**
  - The screen depth in eye space
- **Parallax**
  - In eye space, the signed distance between the two projected positions of a vertex
  - Parallax = Separation * (1 – Convergence / W)
- **In Screen and Out of Screen**
  - In screen: between screen and far clip plane
    Parallax > 0, W > 1
  - Out of screen: between eye plane and screen
    Parallax < 0, 0 < W < 1

What's in there? How to make it better?

Rendering with 3D Vision

gameworks.nvidia.com

The Driver Magic

- What happened in the stereo driver?
  - Trying to add stereo effects into game engines transparently

1. Duplicate render targets

   Left & right surfaces for stereo rendering

2. Duplicate draw calls

   Each draw call is executed twice

3. Apply stereo separation

   Shift x coordinate left and right after vertex shader

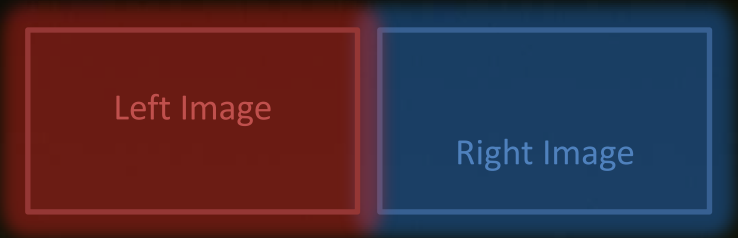
Surface Duplication

- Automatic duplication is based on driver heuristics
  - Transparent to game engine
  - Depending on surface size
    - Surfaces equal or larger than back buffer size are duplicated
    - Square surfaces are NOT duplicated
    - Small surfaces are NOT duplicated
- Explicit duplication is also available
  - Game engine takes full control
  - NVAPI provides the necessary interface
- In the rest of this presentation, we talk mainly automatic duplication

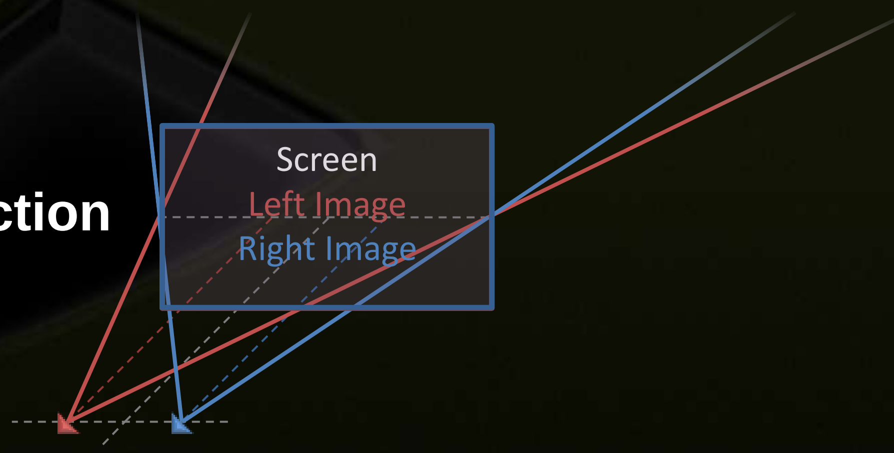
Rendering with 3D Vision

Stereo Rendering Surfaces

- View dependent render targets must be duplicated
  - Back buffer
  - Depth Stencil buffer

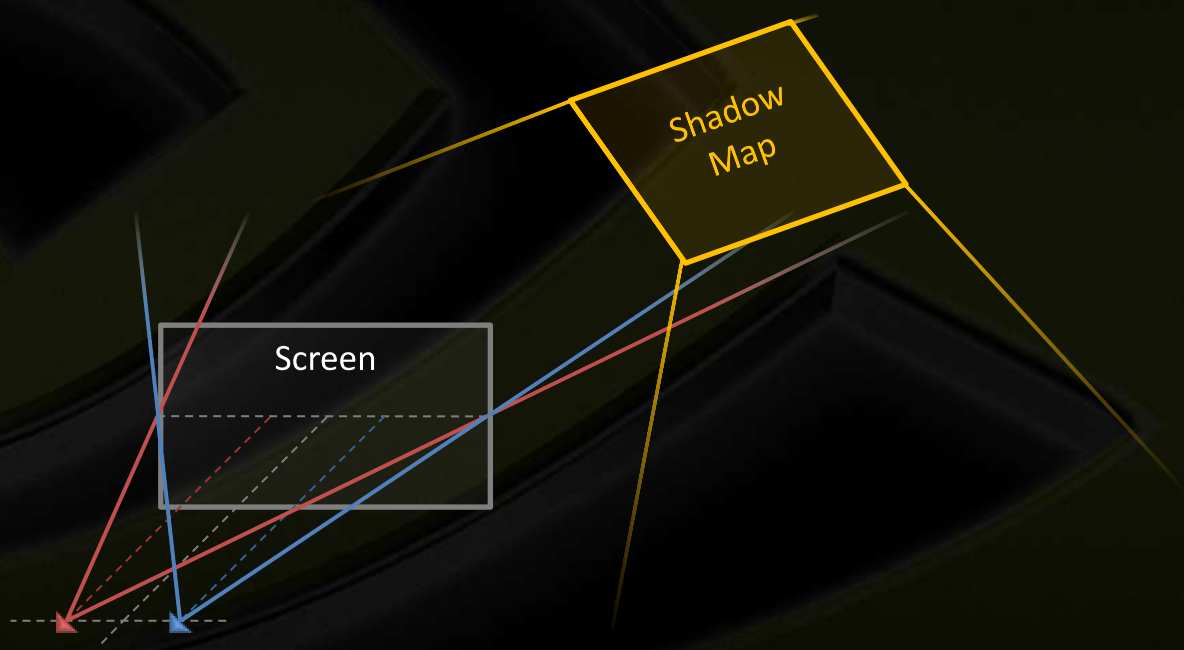


- Intermediate full screen render targets used to process final image
  - HDR, blur, bloom, DOF
  - SSAO
  - Screen space shadow projection



Mono Rendering Surfaces

- View independent render targets DON'T need to be duplicated
  - Shadow map
  - Spot light textures



Stereo or Mono: More Case Studies

Use Case	Surface Type	Stereo Projection	Stereo Drawcalls
Shadow maps	Mono	No Use Shadow projection	Draw once
Main frame Any Forward rendering pass	Stereo	Yes	Draw twice
Reflection maps	Stereo	Yes Generate a stereo reflection projection	Draw twice
Post processing effect (Drawing a full screen quad)	Stereo	No No Projection needed at all	Draw twice
Lighting in deferred shading (Drawing a light sprite)	Stereo G-buffers	Yes Be careful of the Unprojection Should be stereo	Draw twice

Draw Call Duplication

- In NVIDIA stereo driver (automatic mode)
  - For stereo surfaces, every draw call is issued twice for left & right surfaces
  - For mono surfaces, no change
- Pseudo code in the driver

```
HRESULT NVDisplayDriver::draw()
{
    if (StereoSurface)
    {
        VShader = GetStereoShader(curShader);
        SetConstants("-Separation", "Convergence");
        SetBackBuffer(GetStereoBuffer(curBackBuffer, LEFT_EYE));
        reallyDraw();

        SetConstants("+Separation", "Convergence");
        SetBackBuffer(GetStereoBuffer(curBackBuffer, RIGHT_EYE));
        reallyDraw();
    }
    else
        reallyDraw();
}
```

Apply Stereo Seperation

- In automatic mode
  - Driver appends parallax shift to vertex shaders
  - The position output from vertex is modified:

Pos.x += EyeSign * Scale * Separation * (Pos.w - Convergence)

Scale: a parameter controls parallax effect, user adjustable

- In explicit mode
  - Game engine applies separation itself in vertex shader
  - Get the parameters

Scale: NvAPI_Stereo_GetSeparation()

Separation: NvAPI_Stereo_GetEyeSeparation()

Convergence: NvAPI_Stereo_GetConvergence()

EyeSign: -1 for left eye, +1 for right eye

gameworks.nvidia.com

3D Objects

- All the 3D objects should be rendered using a unique perspective projection in a given frame
- All the 3D objects must have a coherent depth relative to the scene
- Most lighting effects requires no changes in shader
  - View-dependent lighting, highlight and specular, are probably fine being evaluated with mono eye
- Reflection and refraction should be rendered in stereo mode

Pseudo 3D Objects: Sky, Billboards…

- Sky box should be drawn with a valid depth further than the regular scene
  - Must be stereo projected
  - Best is at a very far distance so parallax is maximum
  - And cover the full screen
- Billboard elements (particles, leaves) should be rendered in a plane parallel to the viewing plane
  - Doesn’t look perfect
- Relief mapping looks bad
  - Parallax occlusion map, steep map, etc.
  - May requires extra fix-ups in pixel shader (discuss later)

3D Objects in Multiple Viewports

- **Some games may display more than one 3D scenes on screen**
  - **Small viewports portraying selected characters**
  - **Split screen for multiple players**

- **Each viewport may have its own convergence**
  - **Game engine is required to take care of each viewport**
  - **Use NVAPI function NvAPI_Stereo_SetConvergence()**

3D Objects: Out of Screen Effects

- 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
- Objects clipped by the edges of the monitor
  - Look strange
  - 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

2D Objects

- 2D Overlay elements must be drawn at a valid Depth
  - Give each 2D element a valid W value
- Not interacting with 3D scene
  - HUD, UI elements
  - Place at screen depth to look mono W = 1.0
- Interacting with 3D scene
  - Mouse Cursor at the pointed object's depth
    Can not use the HW cursor
  - Crosshair
  - Labels or billboards in the scene
  - Place at scene depth W = scene depth in eye space

2D Objects: Add Correct Depth

```
float  depth;

VS_OUTPUT  Render2D_VS(VS_INPUT Input)
{
   VS_OUTPUT Output;
   … …

   Output.Pos = float4(Input.Pos.xy * depth, 0, depth);

   return Output;
}
```

- **Set depth = 1.0, no parallax**

gameworks.nvidia.com

How can I fix …

Issues and Solutions

gameworks.nvidia.com

Many Issues

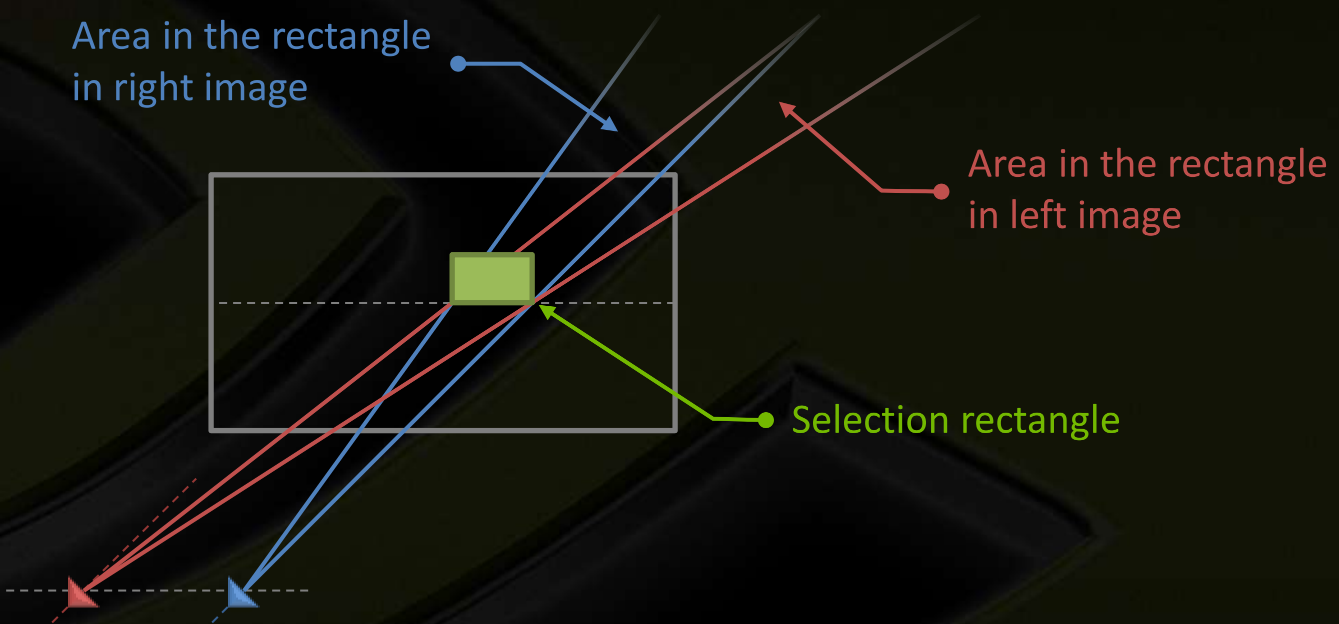
- In the past 3 years, we encountered many stereo issues you may also encounter in the future
  - Crosshair, cursor and selection marquee
  - Replay stereo video/image in game engine
  - Frustum culling
  - Deferred shading alike techniques
  - View-dependent lighting/texture effects
  - IME under full screen mode
  - Eyestrain & motion sickness
- Automatic stereo mode works for most games
  - But game engines need to take care of depth of 2D objects

Crosshair & Cursor

- In real world, people don’t aim with two eyes
- Where to place crosshair in stereo mode?
  - Workaround: place crosshair at scene depth
  - May not feel like reality, but feel “right”
- Hardware cursor does not provide depth
  - Games heavily depend on cursor picking should consider draw cursor manually with correct depth

Selection Marquee

- **In mono, the selection 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**



- **The accurate selection should use the volume cast by rectangle marquee. For details, check out document: http://developer.download.nvidia.com/presentations/2009/GDC/GDC09-3DVision-The_In_and_Out.pdf**

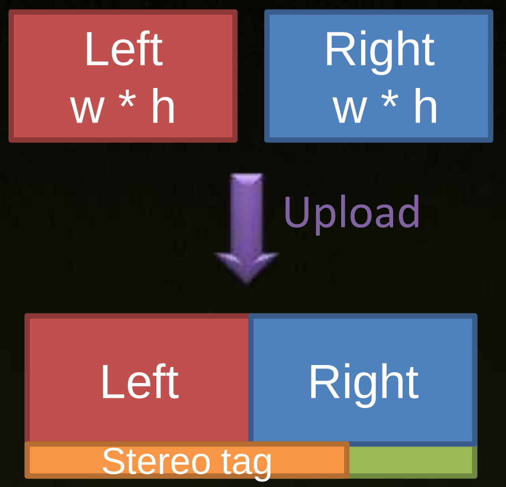
Stereo Video/Image Display

- Many games want to play pre-rendered video clips in stereo mode
- Play existing stereo content in 3D Vision
  - Replay stereo video
  - Display stereo photos as menu background
  - Prerecorded cut scenes

Issues & Solutions

Stereo Video/Image Display (cont.)

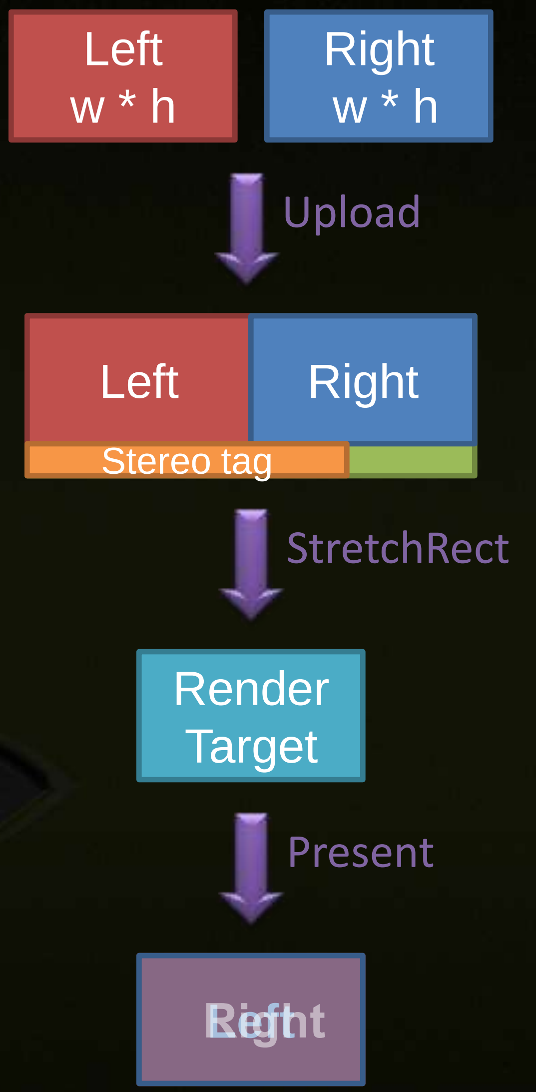
- Introducing to stereo texture
- A D3D texture with
  width * 2
  Height + 1
  Left image on left half
  Right image on right half
  NV3D tag in the extra row
- NV3D tag is edited at creation time



Stereo Video/Image Display (cont. 2)

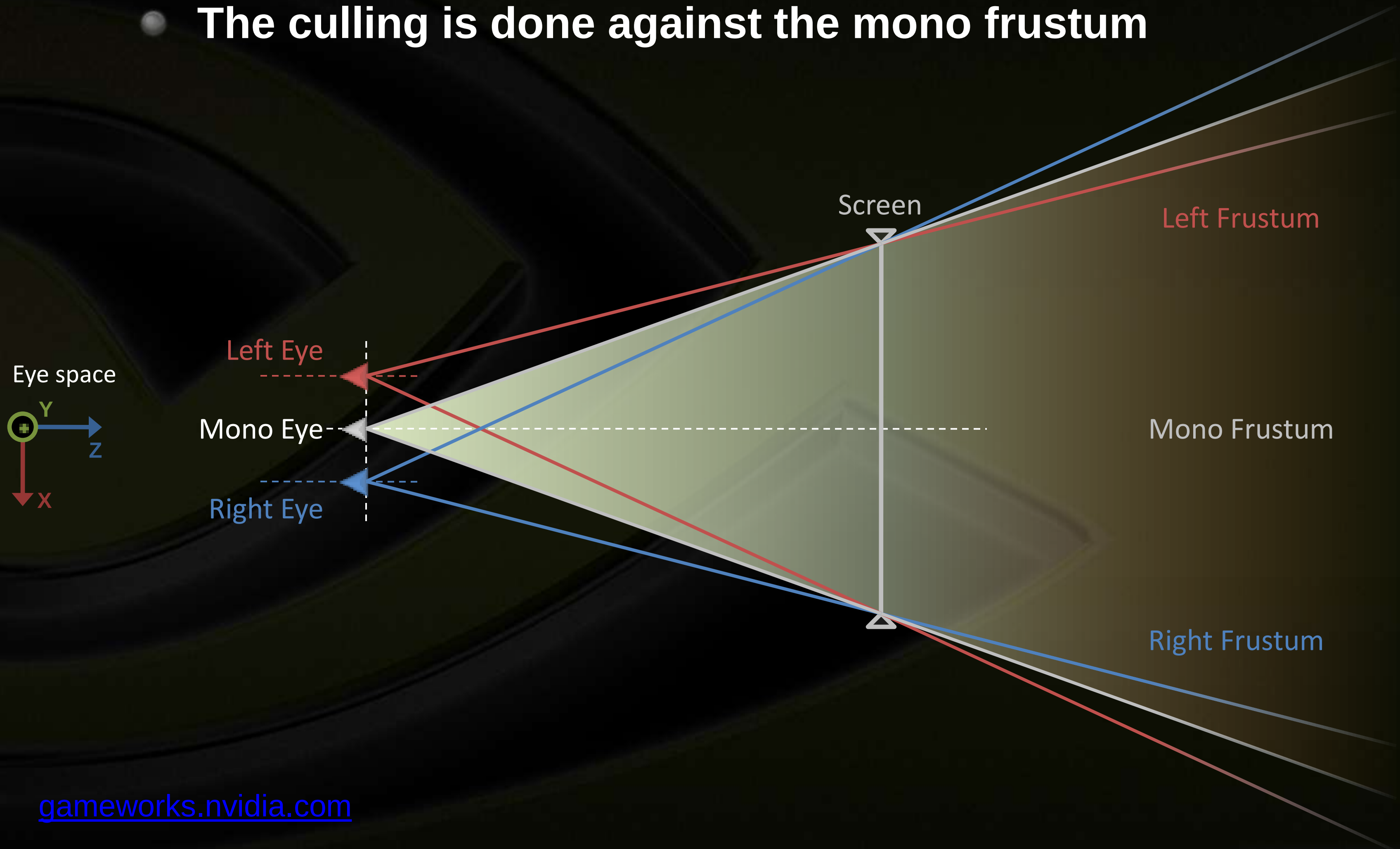
- **Copy stereo texture to a stereo render target**
  - **Driver automatically puts**
    left half -> left surface
    right half -> right surface

- **Use stereo texture in a pixel shader**
  - **Copy this texture into a common texture in each frame**
  - **Driver automatically track left & right**
    copy left half in left draw call
    copy right half in right draw call



Frustum Culling

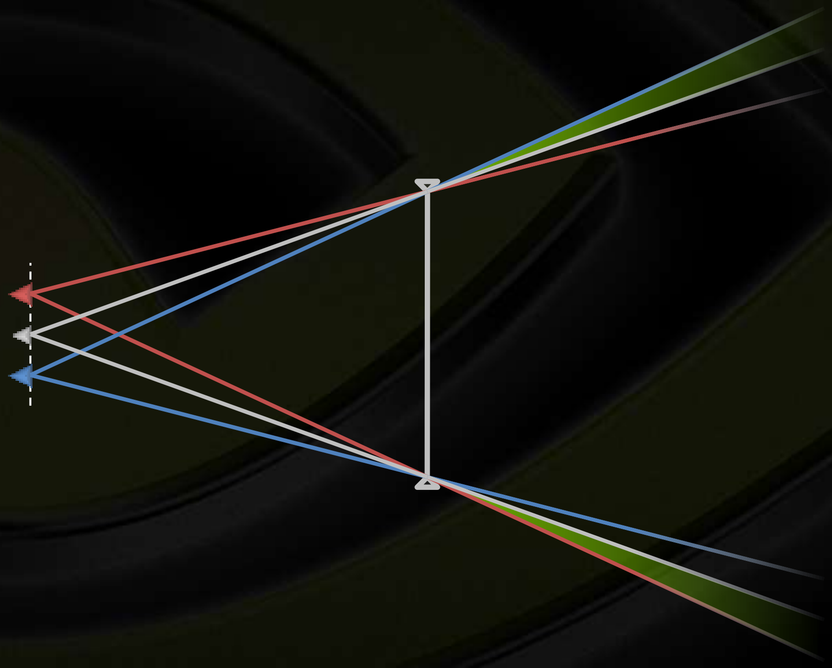
- **Most game engines cull objects out of view frustum**
  - **The culling is done against the mono frustum**



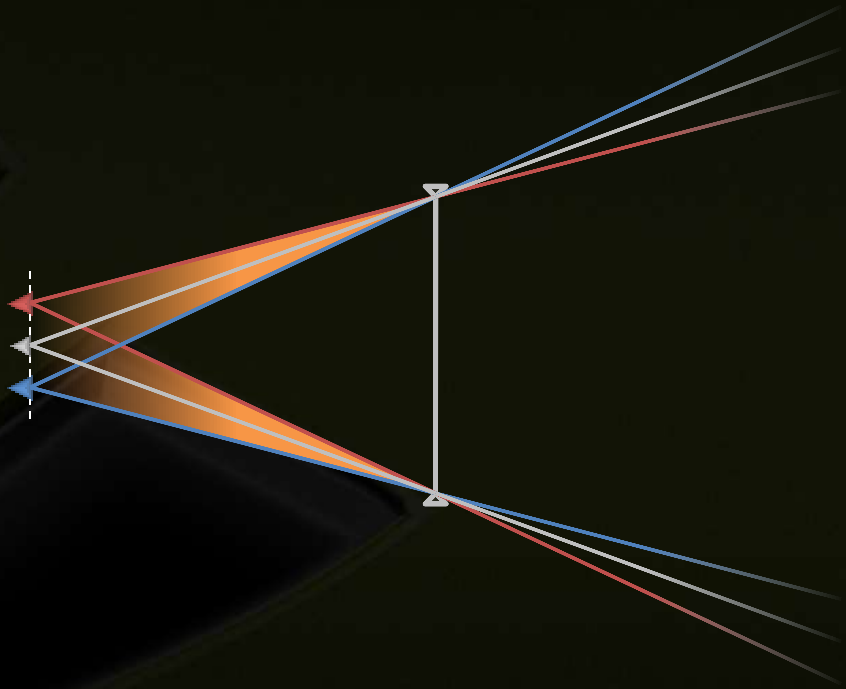
Issues & Solutions

Frustum Culling (cont.)

In screen regions missing



Out of screen region in one eye only

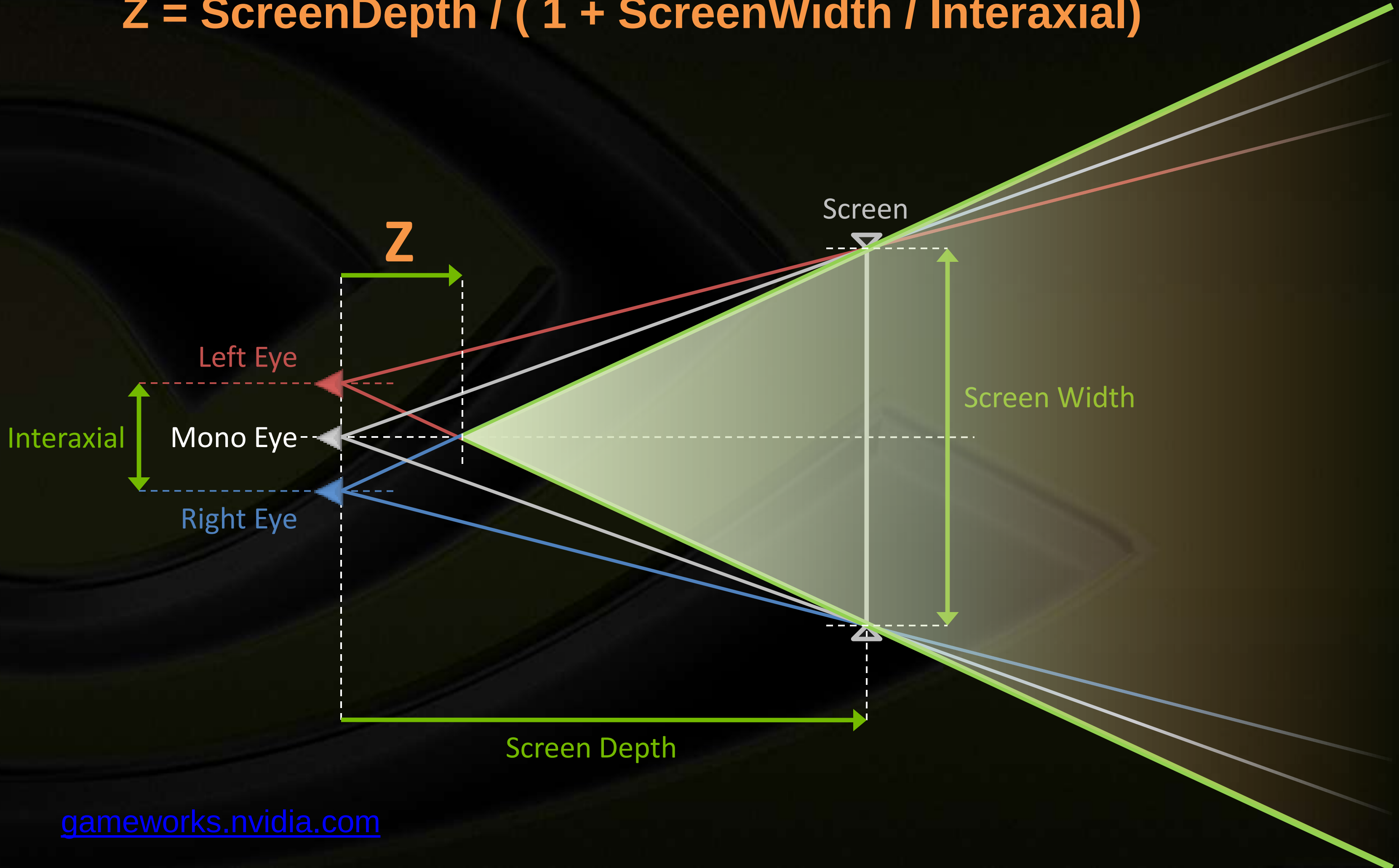


Issues & Solutions

Frustum Culling (cont. 2)

- **The correct frustum for culling: compounding frustum**

  Z = ScreenDepth / (1 + ScreenWidth / Interaxial)

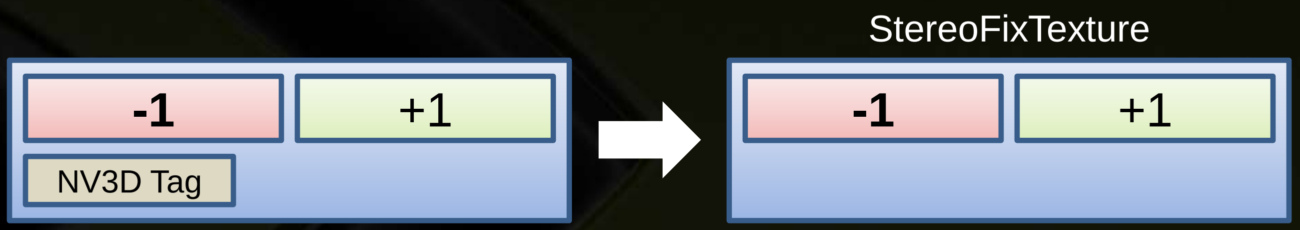


Deferred Shading

- The lighting pass transforms screen position back into world position
  - But the screen position is modified for left & right eyes!
- Explicit modification of pixel shaders required
  - Pixel shaders need to know it’s in left or right draw call
  - Count in parallax shift when doing forward/reverse transform
- How pixel shader knows it’s in left or right draw call?
  - Stereo texture can help

Deferred Shading (cont.)

- Create a small stereo texture
  - -1.0 in left half, 1.0 in right half
  - Pixel shader easily knows left draw call or right draw call



  - In left draw call, Sample(StereoFixTexture) == -1
  - In right draw call, Sample(StereoFixTexture) == 1

Deferred Shading (cont. 2)

- **Math in pixel shader**

Transform light volumes into stereo screen space

1. The parallax of light geometries
   Parallax = Scale * Separation * (Pos_mono.w – Convergence)
2. Apply parallax shift
   Pos_stereo.x = Pos_mono.x + Sample(StereoFixTexture) * Parallax
3. Perform w-division, transform into [-1, 1] space
   Pos_stereo.xy /= Pos.w
4. Use Pos_stereo.xy to read data from G-Buffer including SceneDepth

Compute world space position

1. Reverse w-division of the scene
   Pos_stereo.xy *= SceneDepth
2. The Parallax of the scene
   Parallax = Scale * Separation * (SceneDepth – Convergence)
3. Remove parallax shift
   Pos_mono.x = Pos_stereo.x – Sample(StereoFixTexture) * Parallax
4. Unproject Pos_mono into world space

View-Dependent Shading Effects

- ## Highlight & specular
  - For accurate rendering, reflection vectors should be calculated from left eye and right eye vectors
  - However, using mono eye vector doesn't raise perceptible artifacts in most time.
  - No change in pixel shader

- ## Relief mapping
  - Parallax occlusion map, steep map, cone map, etc.
  - Micro ray marching: the eye ray has to be created from correct eye position (left or right)
  - Use stereo texture to check left or right draw call

IME in Full Screen Mode

- 3D Vision must work in exclusive full screen mode
- Not all IMEs can work in exclusive mode
- Workaround: suggest users using D3D compatible IMEs when playing in stereo mode

Eyestrain & Motion Sickness

- Many things can cause eyestrain even sickness
  - Overly large or small field of view
  - Flickering fluorescent lighting in the environment
  - Lack of proper motion blur
  - Incorrect combination of interaxial, convergence and object placement
  - Too many out of screen objects

Interocular

- Interocular: the distance between two pupils
  - Human have 6.0cm ~ 6.5cm interocular on average
  - Equivalent to visible on-screen parallax of infinite objects
  - Human eyes are not able to overlap two images close to or greater than interocular
- Interaxial-interocular relation

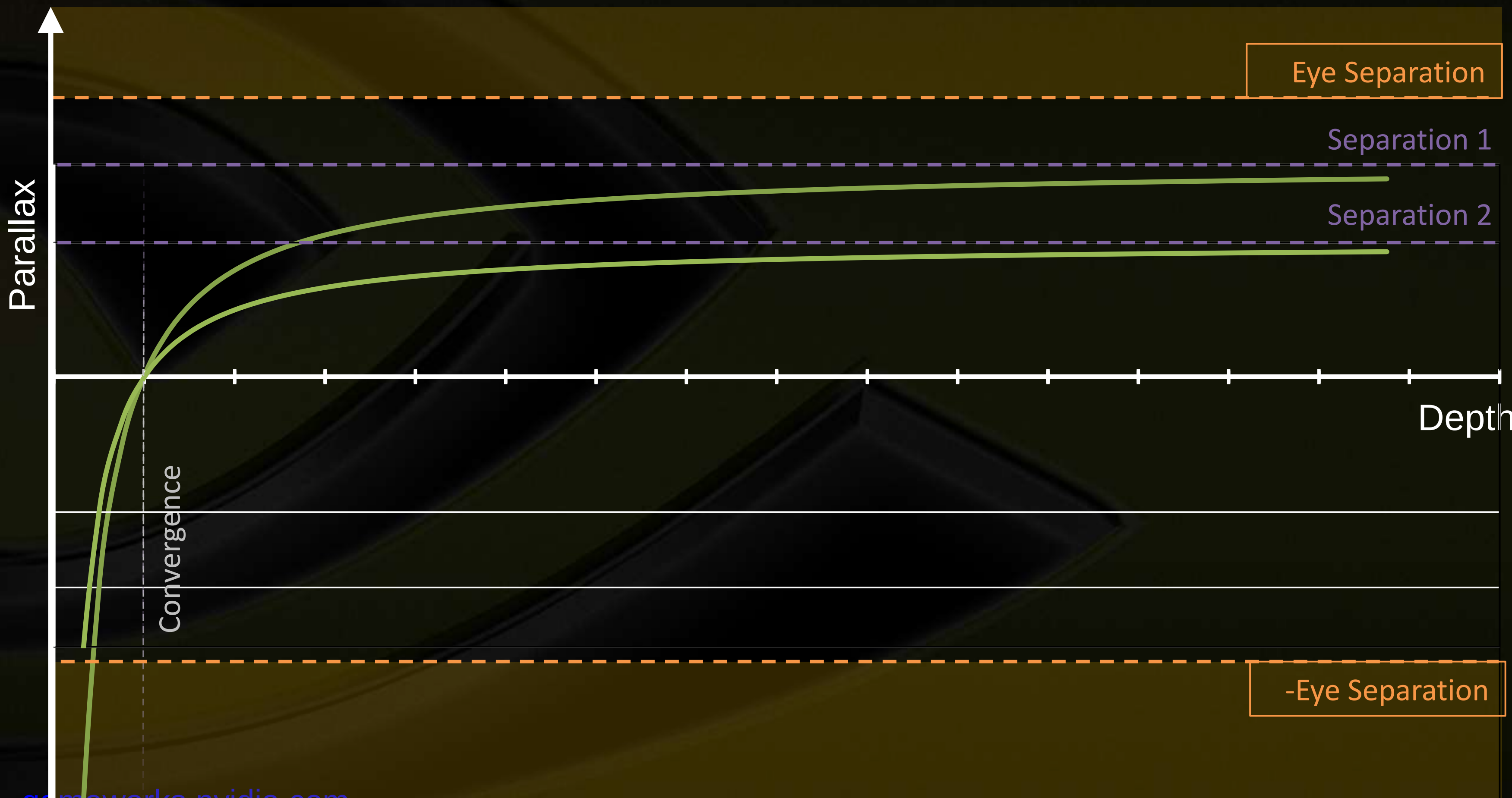
  Interaxial = Interocular / RealScreenWidth

  - Depending on how big the real screen is, interaxial varies from user to user
  - Let users adjust interaxial to a comfortable range

Reduce Eyestrain & Motion Sickness

Safe Parallax Range

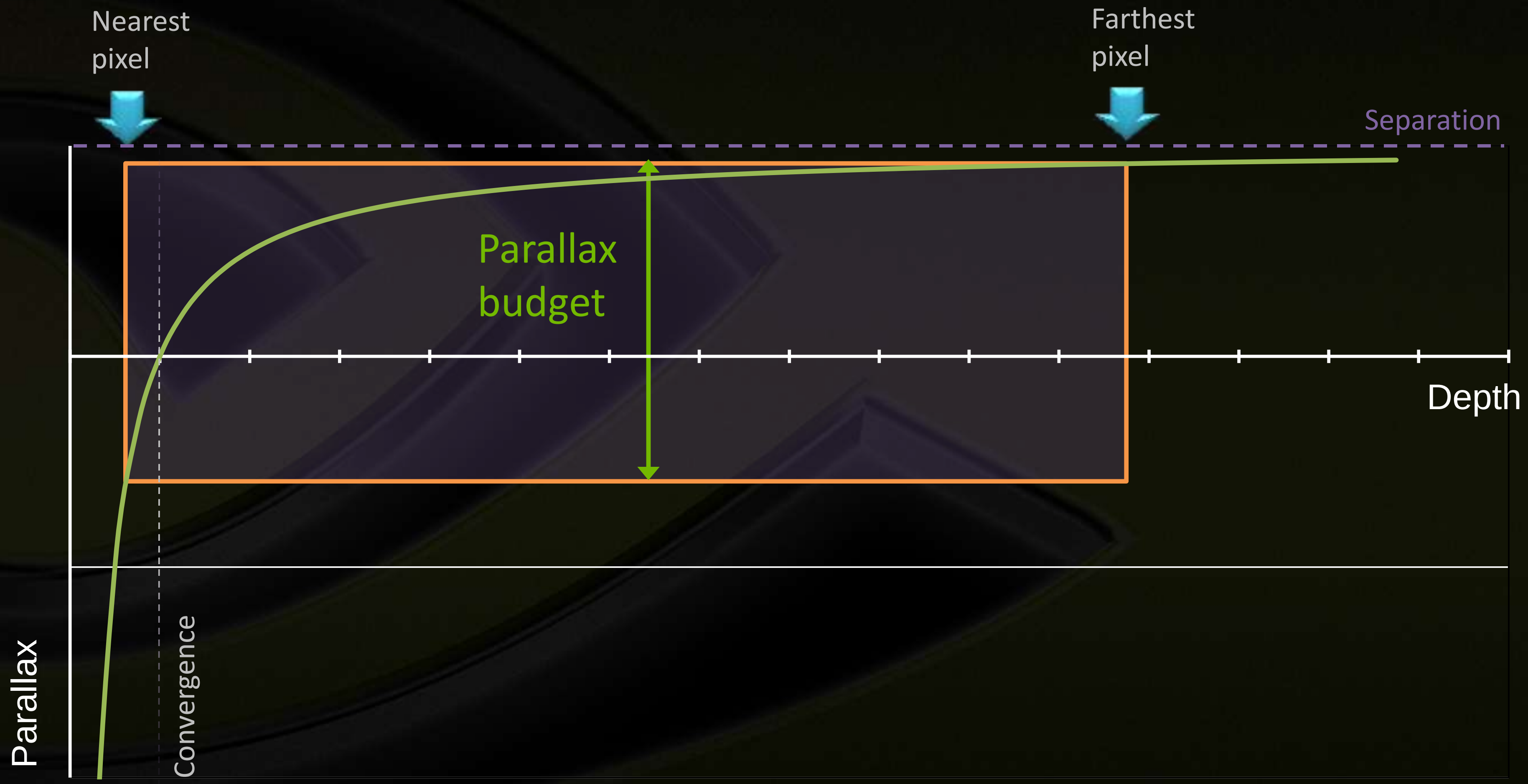
- With a certain interaxial, object parallax has “safe range”



Reduce Eyestrain & Motion Sickness

Parallax Budget

- **How much parallax variation can be used**



gameworks.nvidia.com

Reduce Eyestrain & Motion Sickness

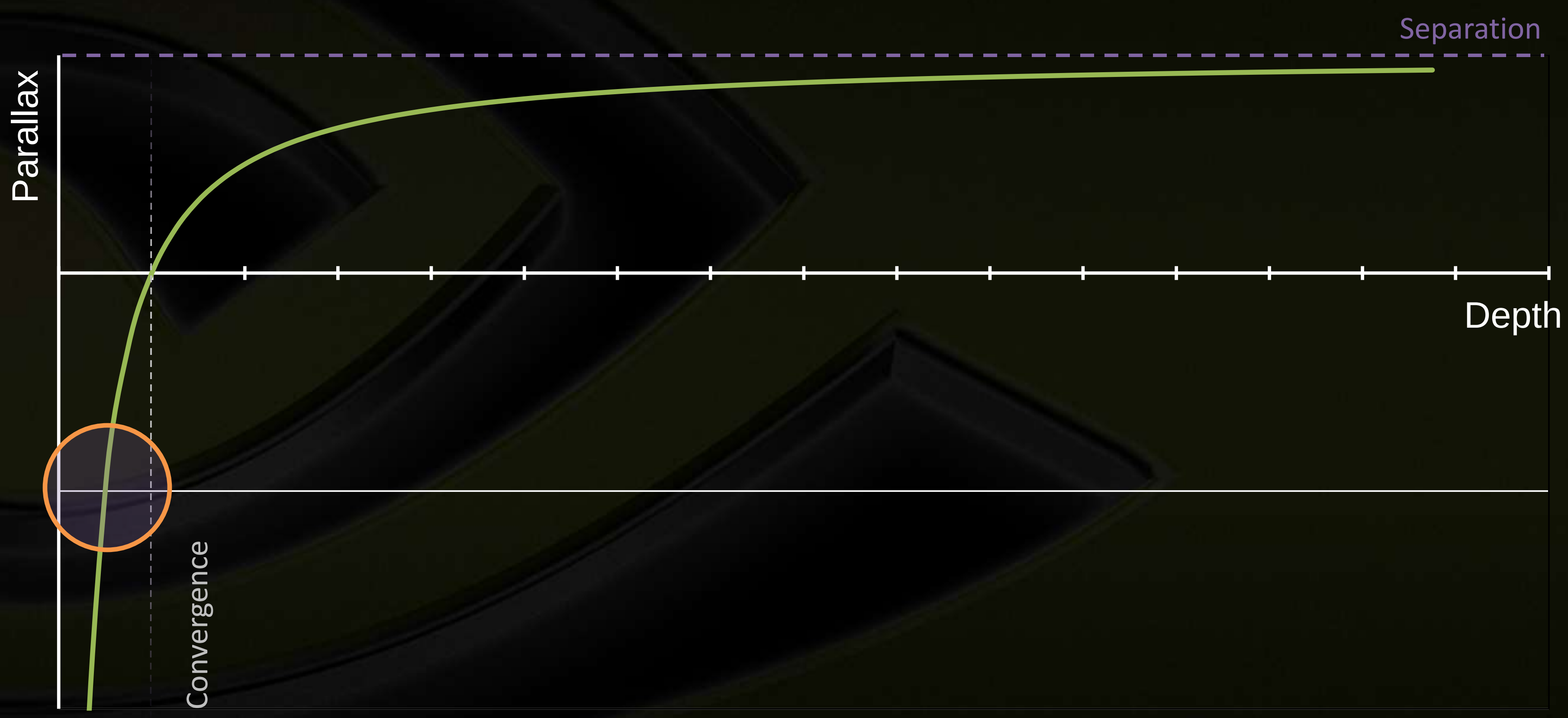
Parallax Budget: Farthest Pixel

- At 100 * ScreenDepth, Parallax is 99% of the Interaxial
  - For pixels further than 100 * ScreenDepth, elements looks flat with little to no depth differentiation
- Between 10 to 100 * ScreenDepth, Parallax vary of only 9%
  - Objects in that range have a subtle depth differentiation

Reduce Eyestrain & Motion Sickness

Parallax Budget: Nearest pixel

- **At Convergence/2, Parallax is equal to –Separation**
  - **Out of the screen and the parallax is very large (> Separation) and can cause eye strains**



gameworks.nvidia.com

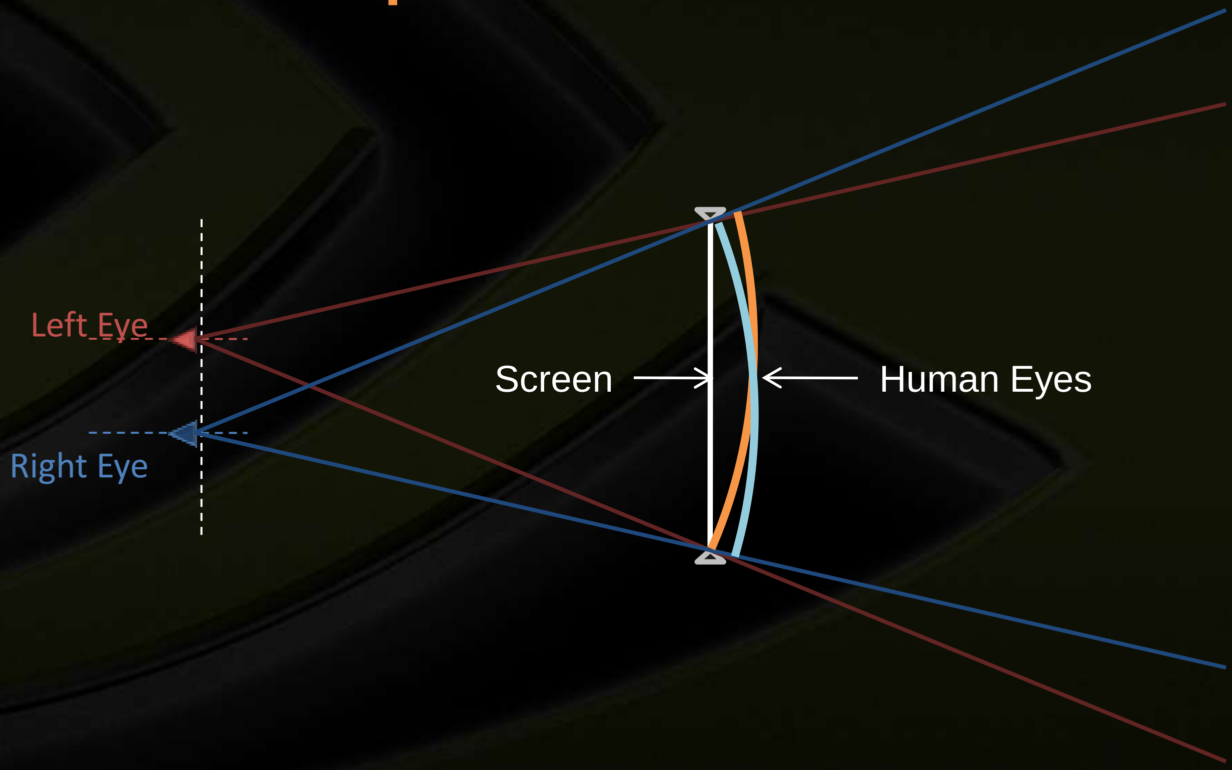
Comfortable Range & Convergence

- Screen depth (convergence) should be defined by application depending on the camera and the scene

- Make sure the scene objects are in the range
  [ScreenDepth / 2, 100 * ScreenDepth]

Projection Surface

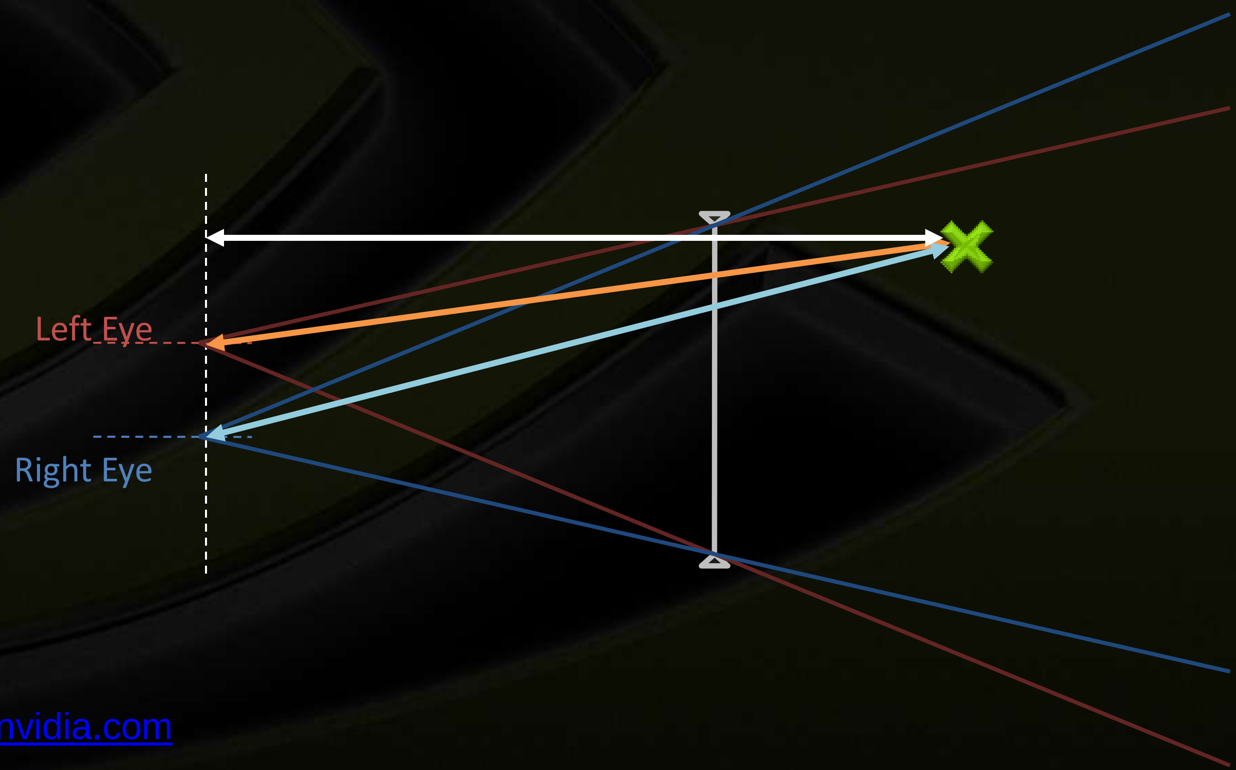
- Current graphics pipeline assumes the projection surface is a plane
- In real world, the projection surface of human eyes is more close to a spherical surface



Reduce Eyestrain & Motion Sickness

Scene Depth

- **Scene depth in current graphics pipeline**
  - **Depth = Distance to eye plane**
- **Scene depth in human eyes**
  - **Depth = Distance to eyes**
- **Higher difference makes more uncomfortable**



QUESTIONS ?

gameworks.nvidia.com

Acknowledgements

- Samuel Gateau & John McDonald in devtech team
- Rod Bogart & Bob Whitehill at Pixar
- Every one in the Stereo driver team !

How To Reach Us

- Online
  - Website: http://developer.nvidia.com
  - Forums: http://developer.nvidia.com/forums