GDC 2011

NVIDIA SPONSORED SESSIONS

Stereoscopic 3D Demystified: From Theory to Implementation in Starcraft 2

Samuel Gateau, NVIDIA
Dominic Filion, Blizzard

PRESENTED BY NVIDIA.

Outline

- Fundamentals of Stereoscopic 3D
  - Stereo projection
  - Depth Perception & how to manage it
- Stereoscopy in a game engine
  - Stereo rendering engine modifications & common pitfalls
  - 3D vision driver
- Starcraft 2

GDC 2011
NVIDIA SPONSORED SESSIONS

3D Stereoscopic Fundamentals

PRESENTED BY NVIDIA.

Stereo Projection

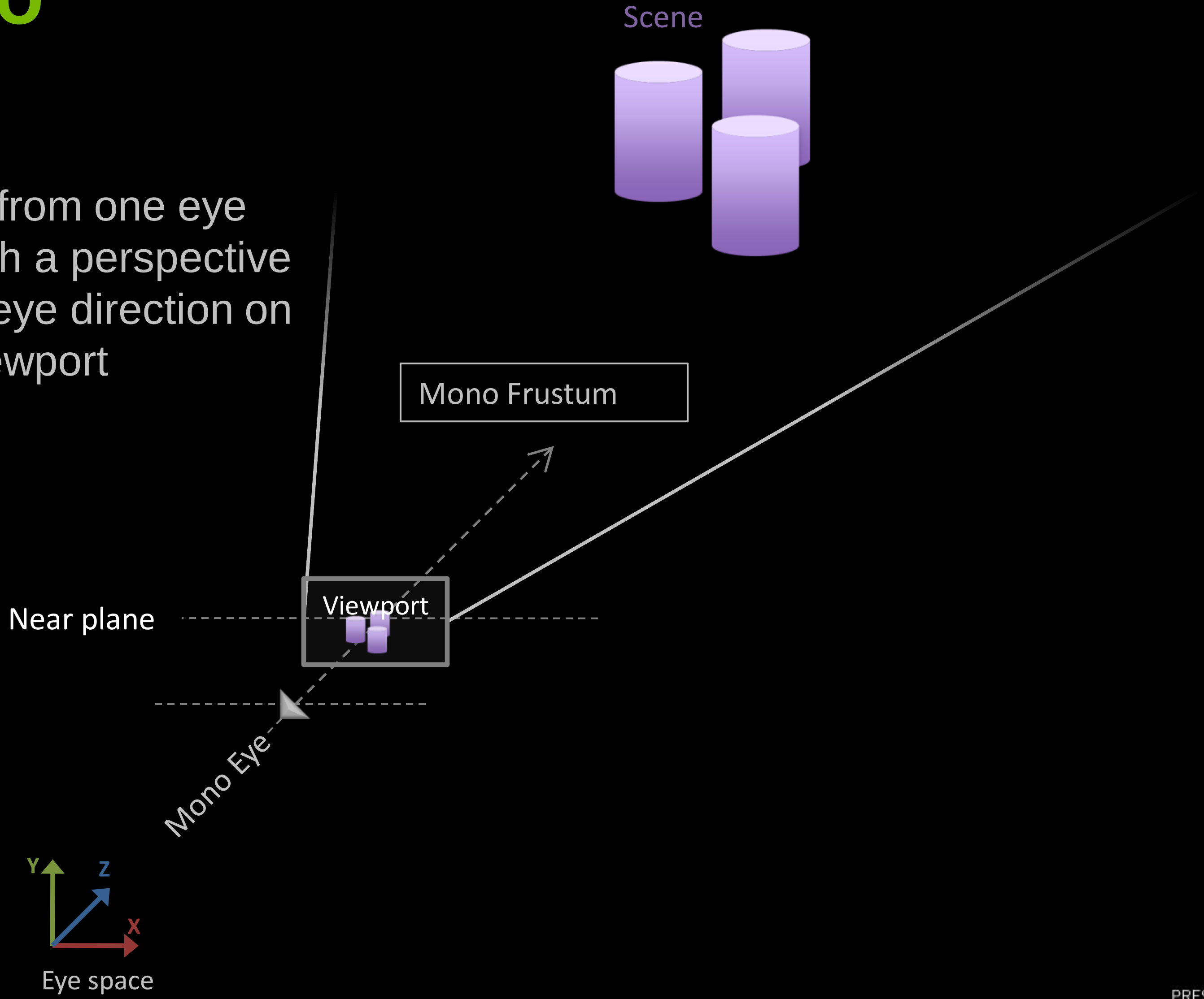
3D Stereoscopic Fundamentals

Changes to the rendering pipe

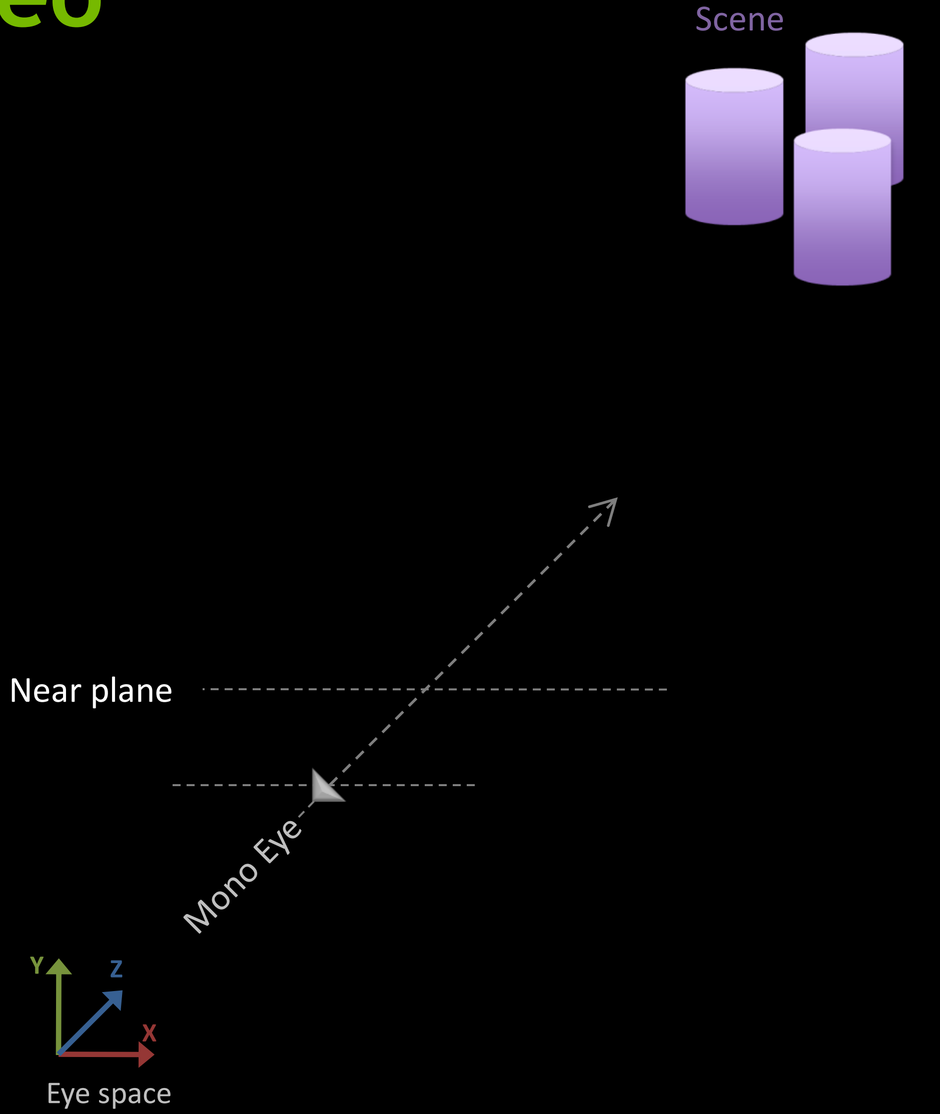
TWO EYES, ONE SCREEN, TWO IMAGES

In Mono

Scene is viewed from one eye and projected with a perspective projection along eye direction on Near plane in Viewport



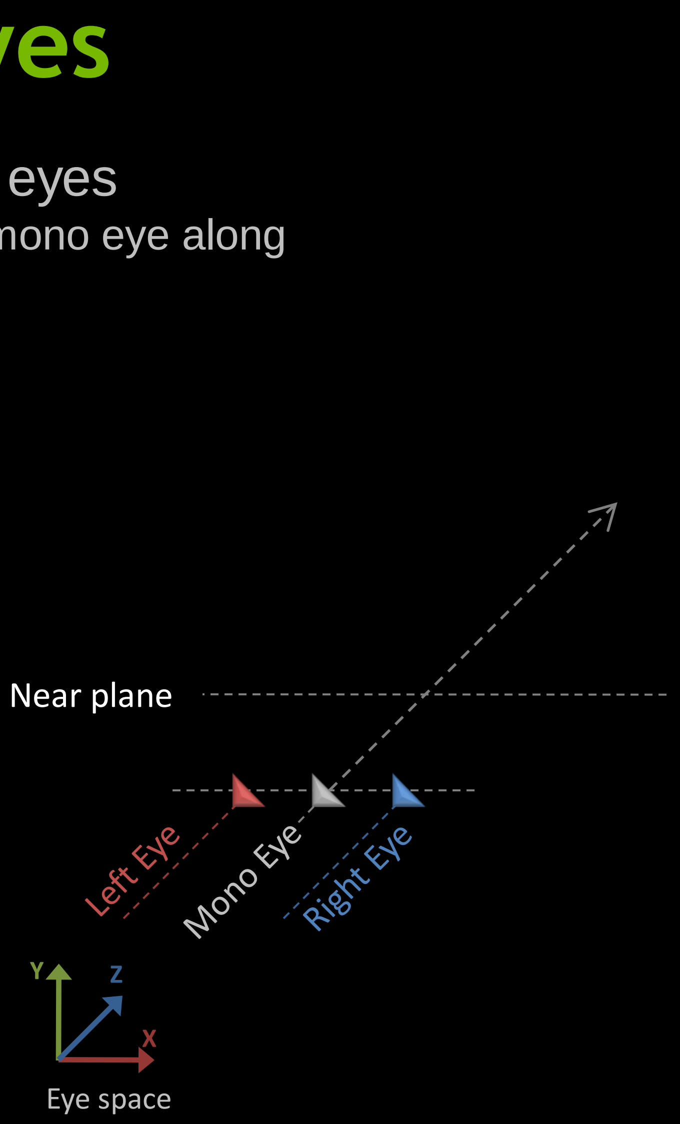
In Stereo



GDC 2011
NVIDIA SPONSORED SESSIONS

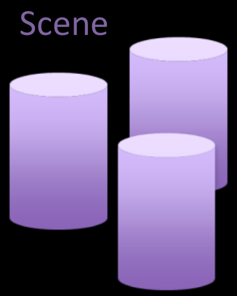
In Stereo:
Two eyes

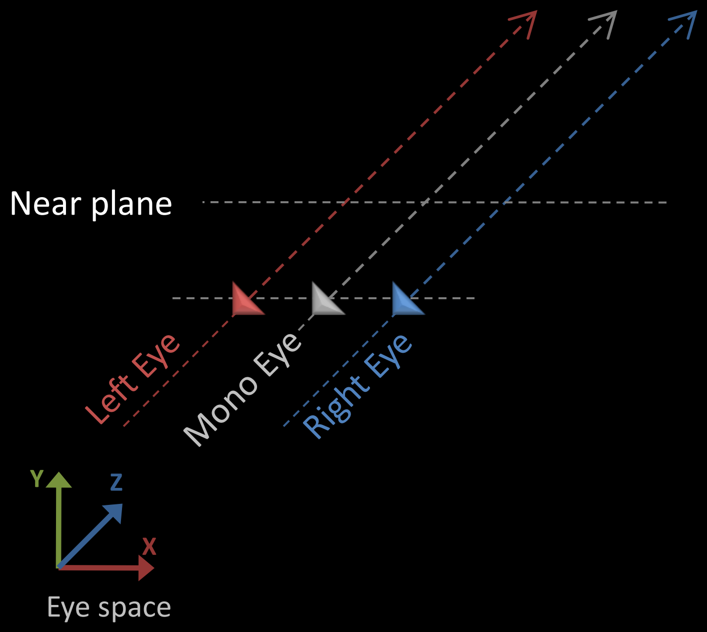
Left and Right eyes
- Shifting the mono eye along the X axis



In Stereo:
Two eyes

Left and Right eyes
- Shifting the mono eye along the X axis
- Eye directions are parallels

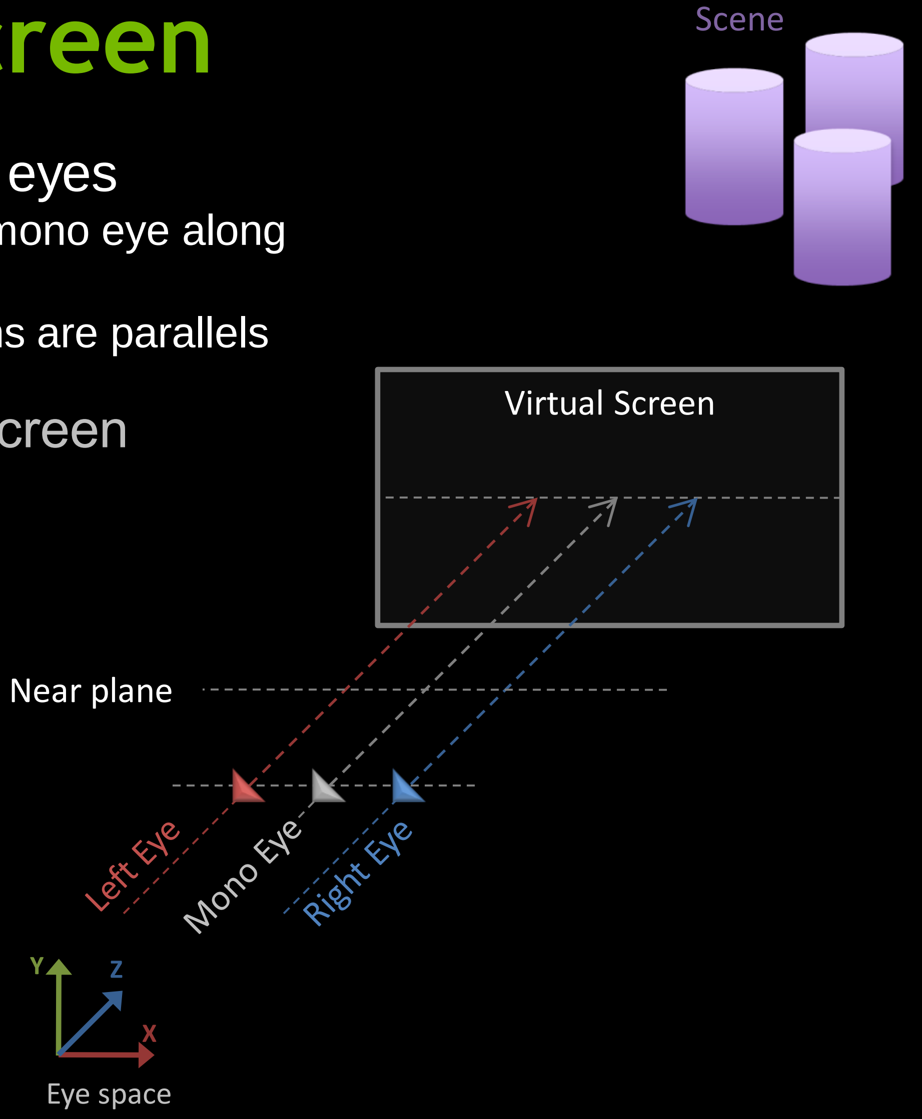




In Stereo: Two Eyes,
One Screen

Left and Right eyes
- Shifting the mono eye along the X axis
- Eye directions are parallels

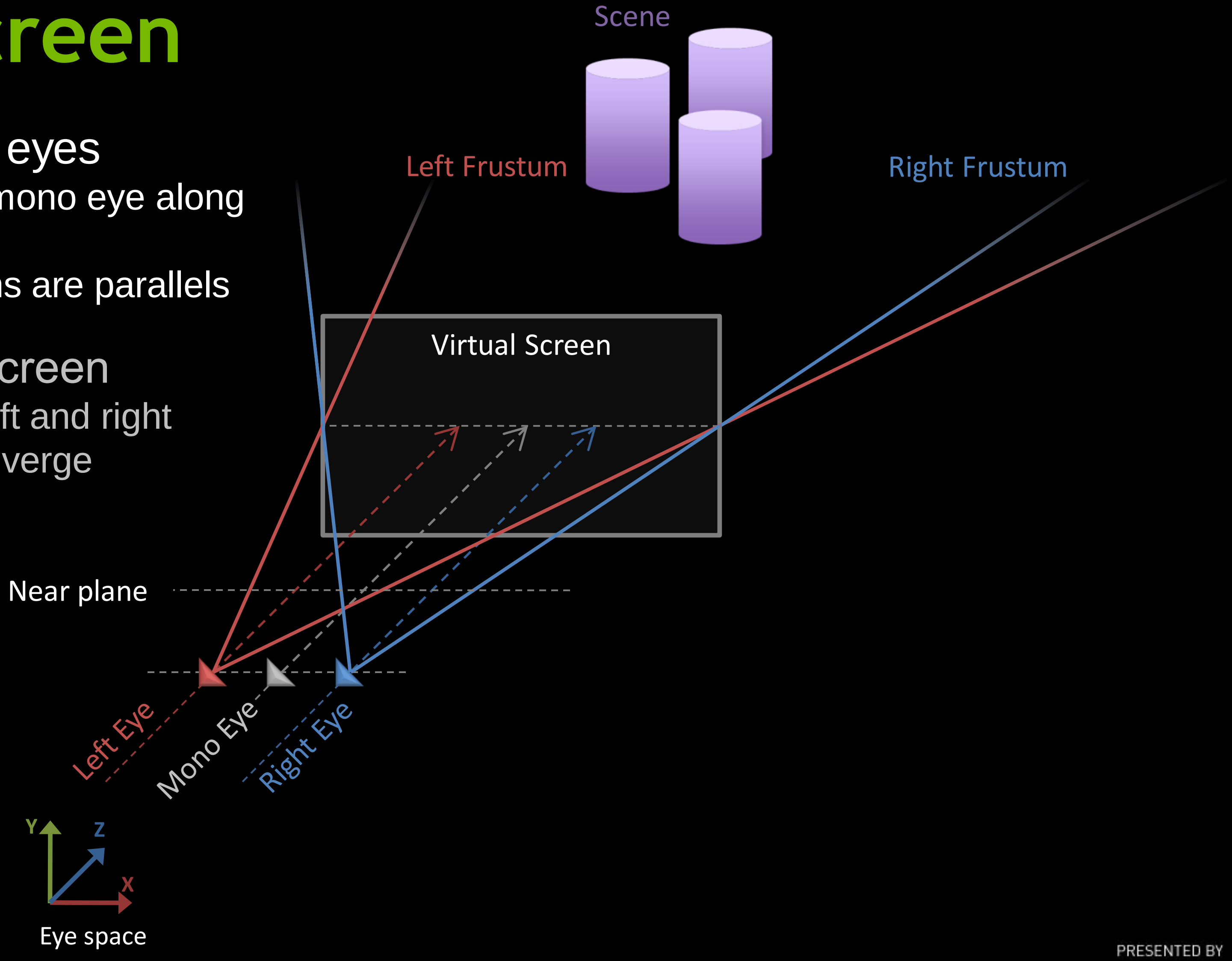
One “virtual” screen



In Stereo: Two Eyes,
One Screen

Left and Right eyes
- Shifting the mono eye along the X axis
- Eye directions are parallels

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



In Stereo: Two Eyes, One Screen, Two Images

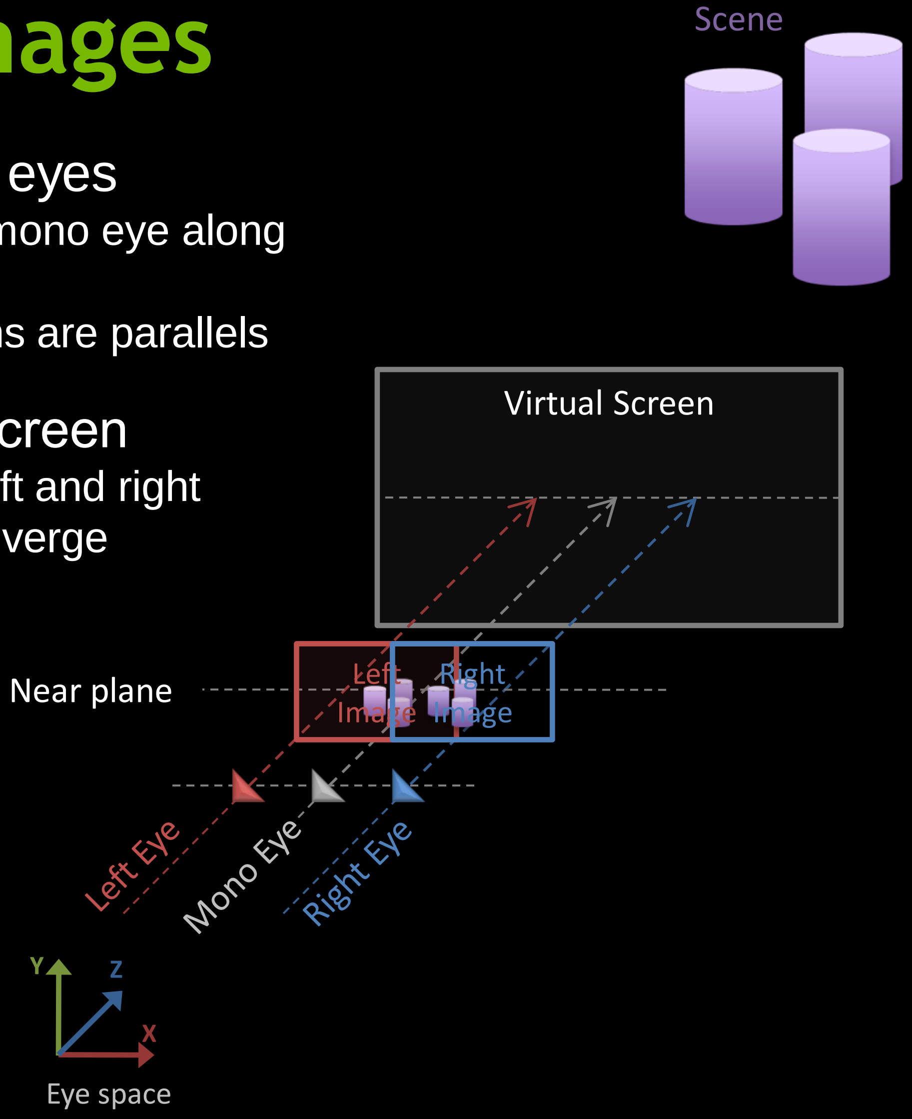
Left and Right eyes

Shifting the mono eye along the X axis

Eye directions are parallels

One "virtual" screen

Where the left and right frustums converge

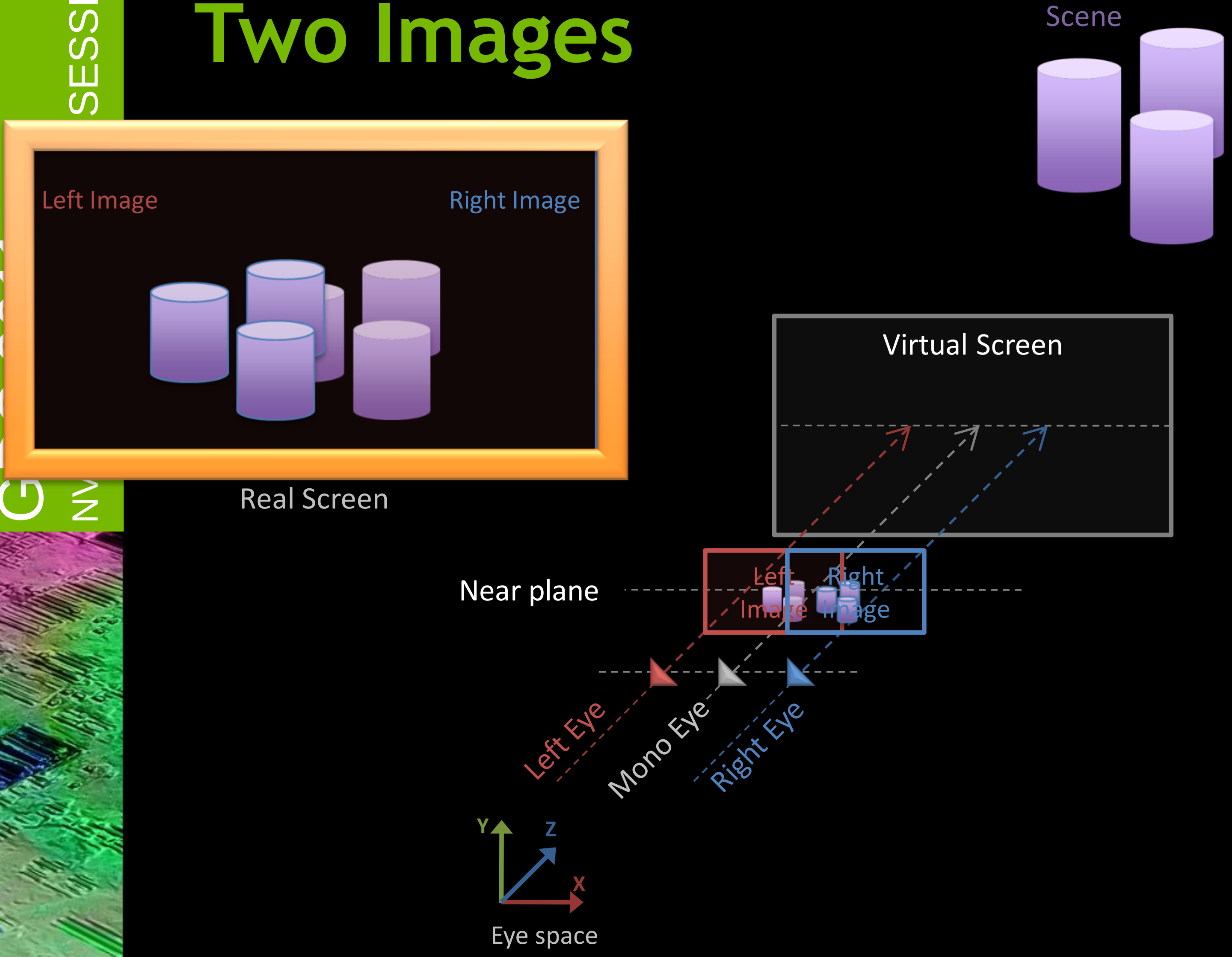


Two images

2 images are generated at the near plane in each views

In Stereo: Two Eyes, One Screen,

Two Images



Two images

2 images are generated at the near plane in each views

Presented independently to each eyes of the user on the real screen

Stereoscopic Rendering

Render geometry twice

From left and right eyes

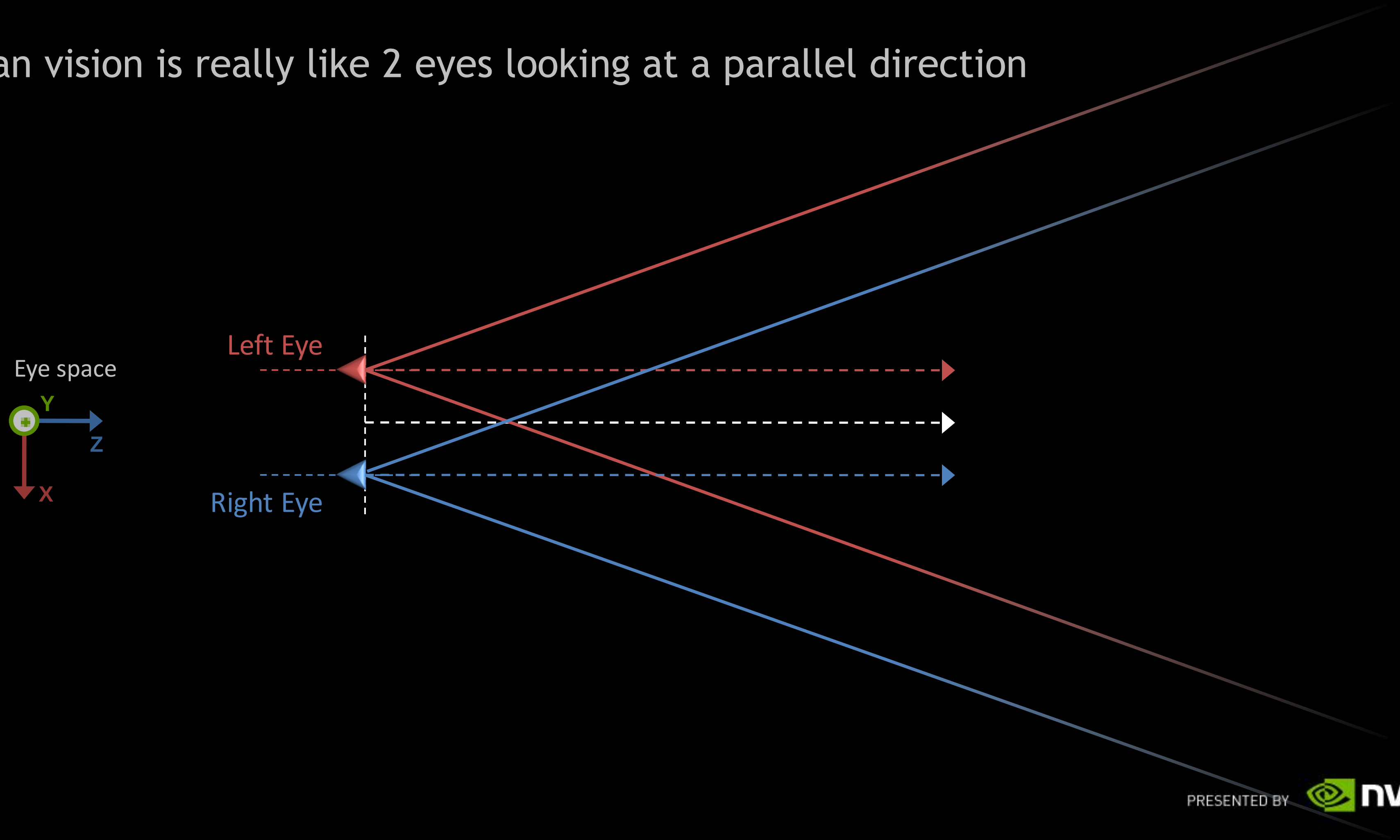
Into left and right images

Basic definitions so we all speak English

DEFINING STEREO PROJECTION

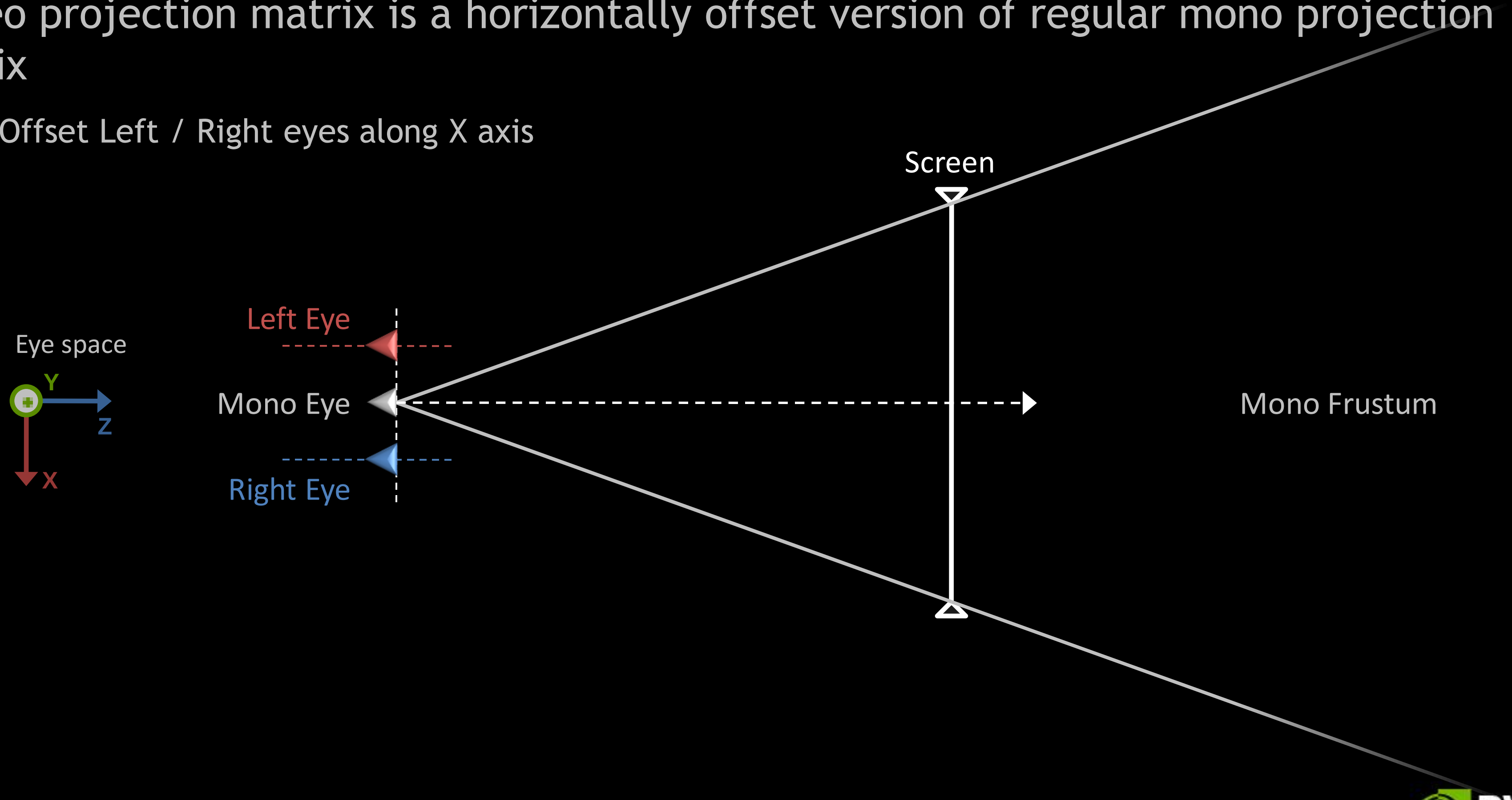
Stereo Projection

- Human vision is really like 2 eyes looking at a parallel direction



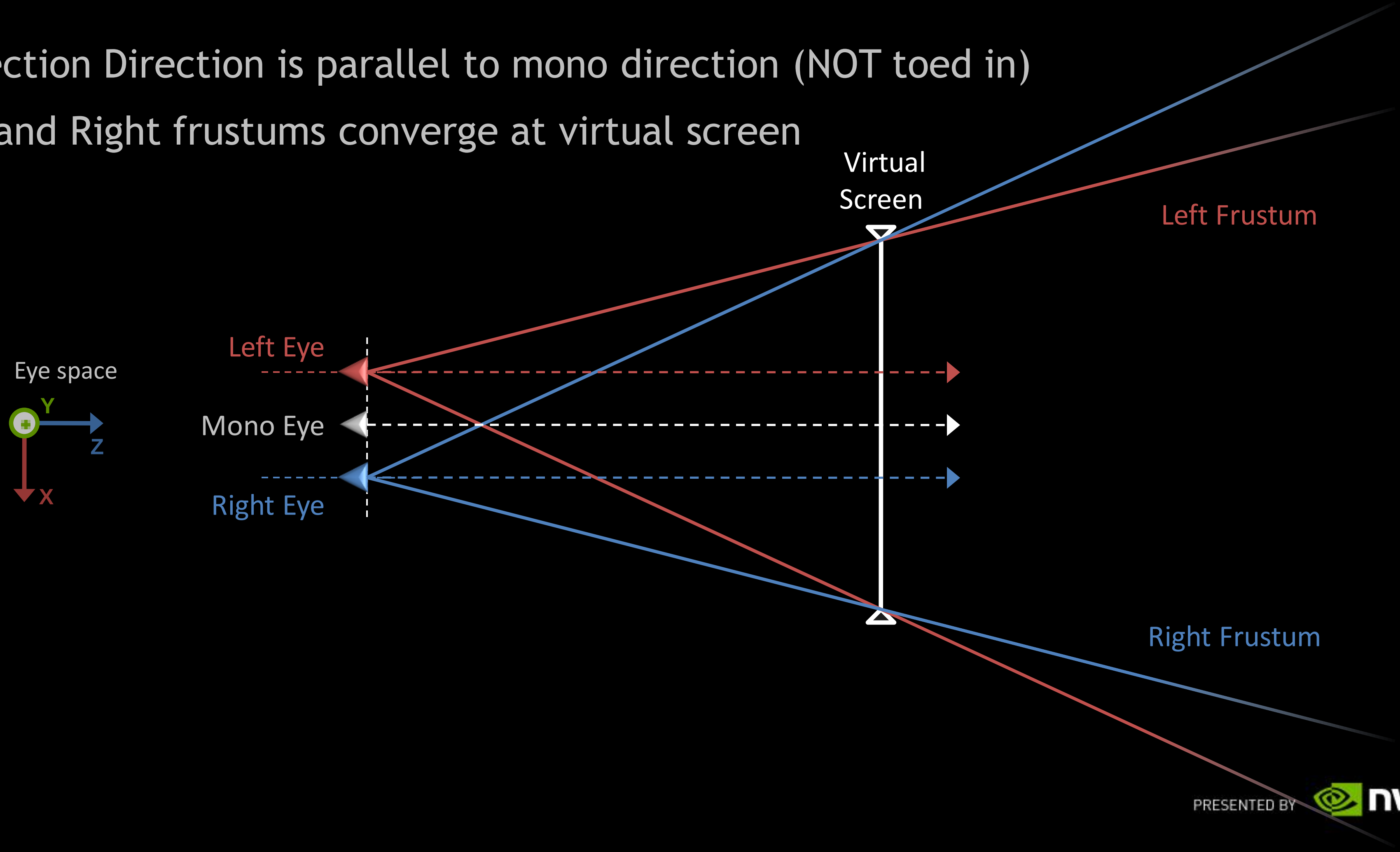
Stereo Projection

- Stereo projection matrix is a horizontally offset version of regular mono projection matrix
    - Offset Left / Right eyes along X axis



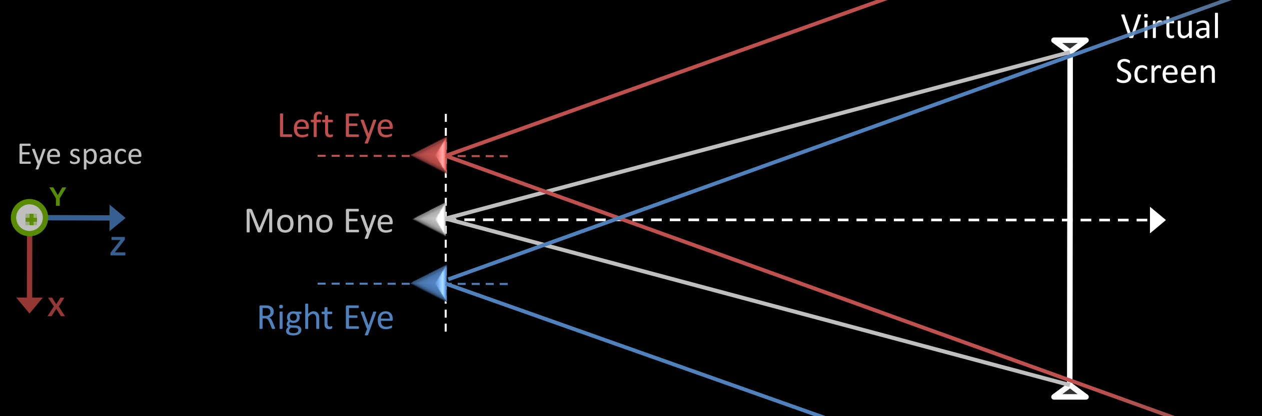
Stereo Projection

- Projection Direction is parallel to mono direction (NOT toed in)
- Left and Right frustums converge at virtual screen



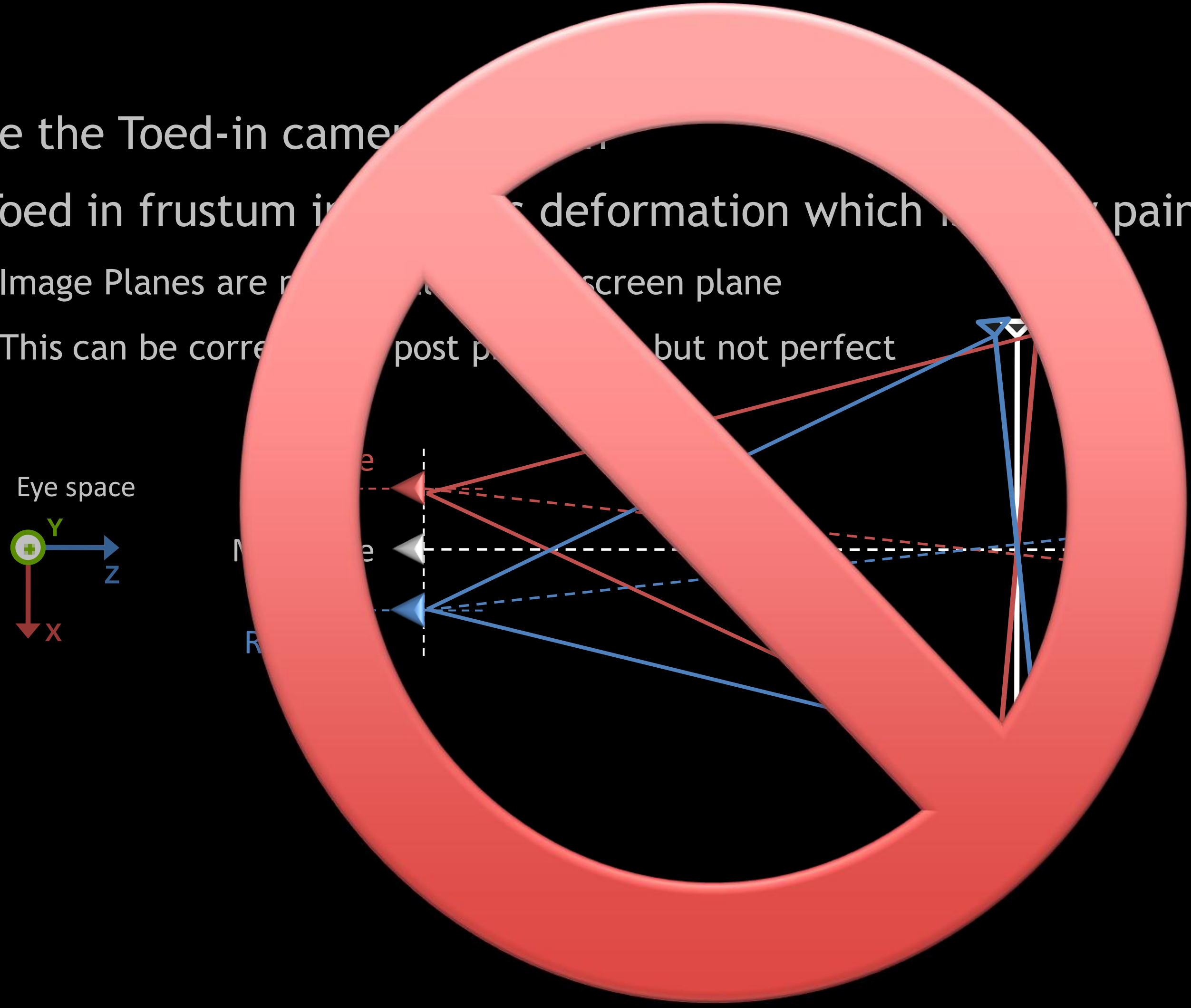
Parallel, NOT Toed in!

- Historically, live camera mounted in parallel stereo would waste a lot of the view field
  - Waste view field is wasted film area



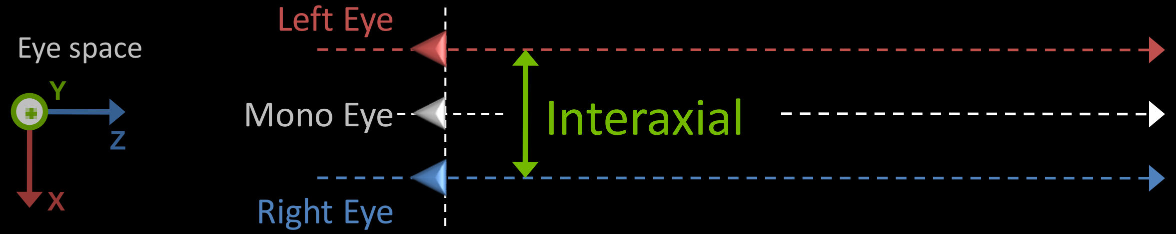
Parallel, NOT Toed in!

- Hence the Toed-in camer
- But Toed in frustum i s deformation which painful
  - Image Planes are n screen plane
  - This can be corre post p but not perfect



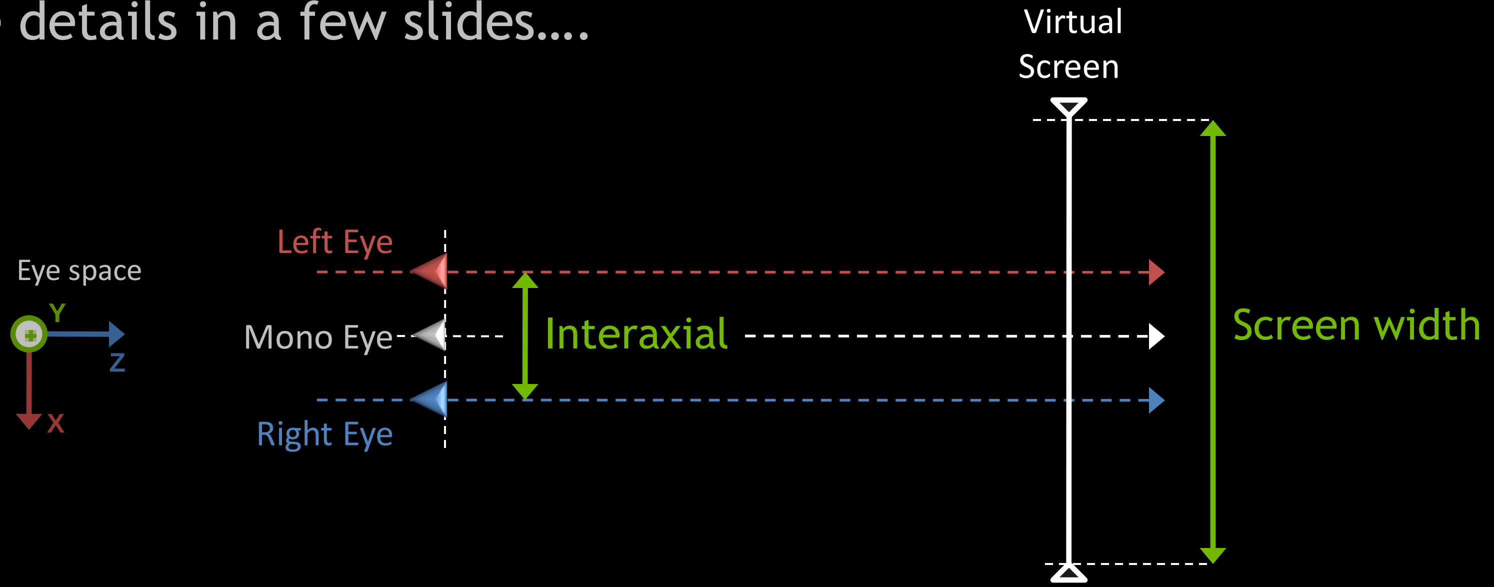
Interaxial

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



Separation

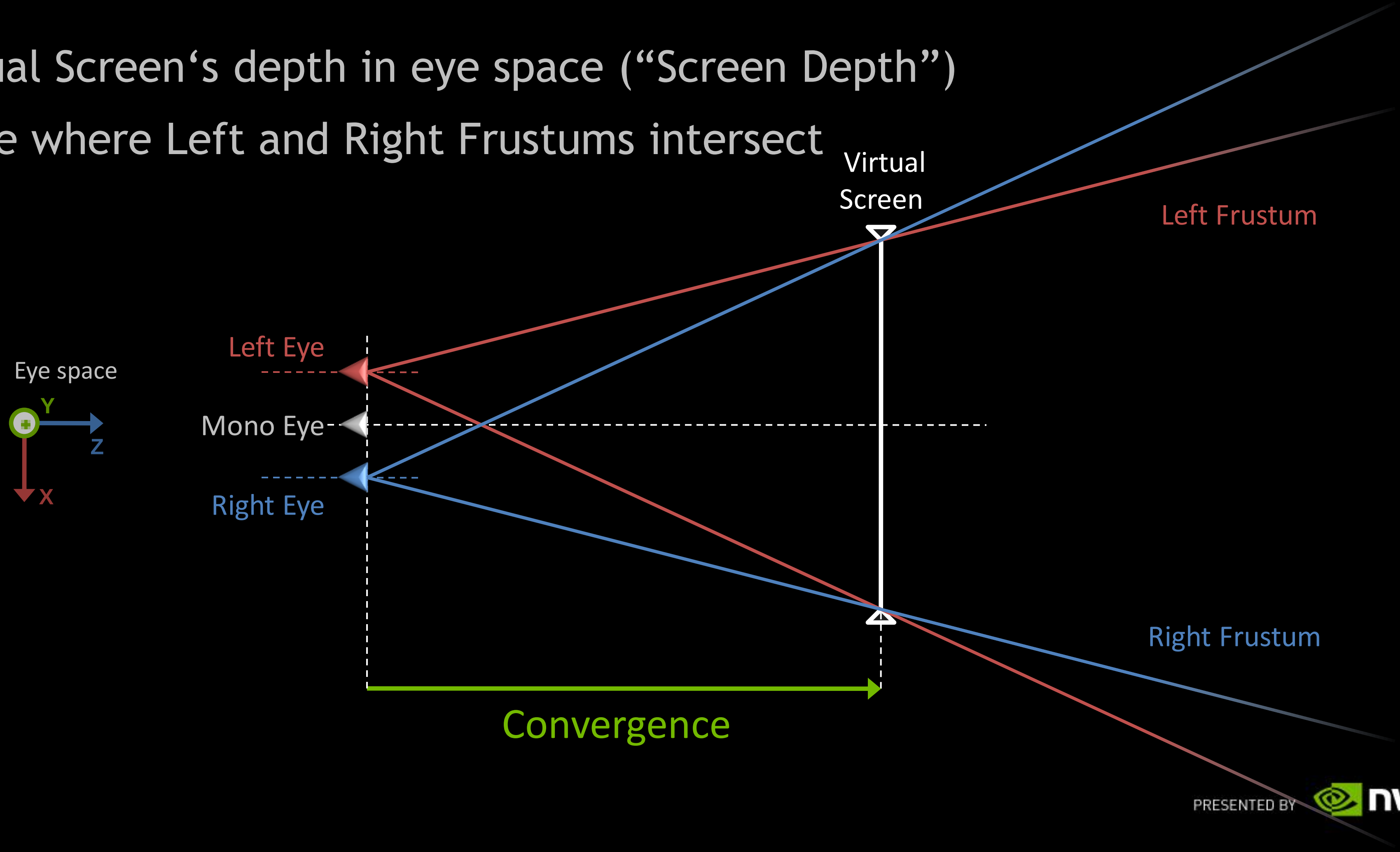
- The normalized version of interaxial by the virtual screen width
- More details in a few slides....



$$Separation = \frac{Interaxial}{Screen\ width}$$

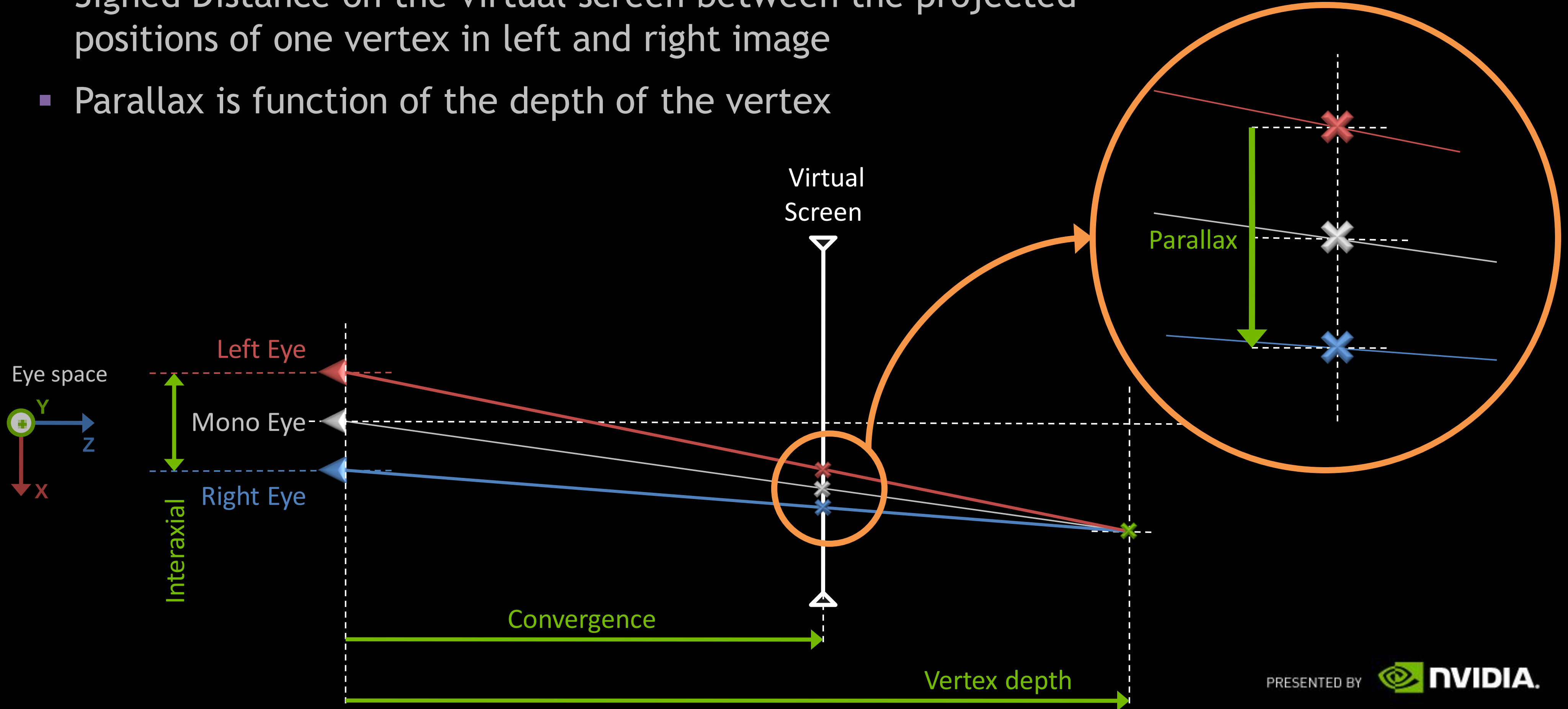
Convergence

- Virtual Screen‘s depth in eye space (“Screen Depth”)
- Plane where Left and Right Frustums intersect



Parallax

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



GDC 2011
NVIDIA SPONSORED SESSIONS

Depth Perception

3D Stereoscopic Fundamentals

PRESENTED BY NVIDIA.

GDC 2011

NVIDIA SPONSORED SESSIONS

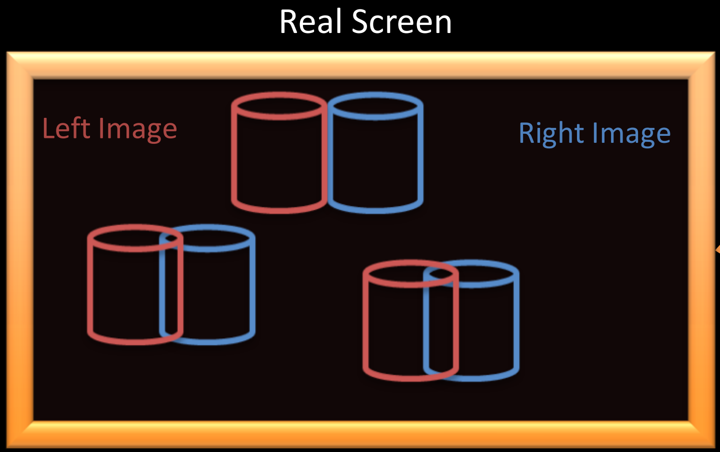
Where the magic happens

DEPTH PERCEPTION

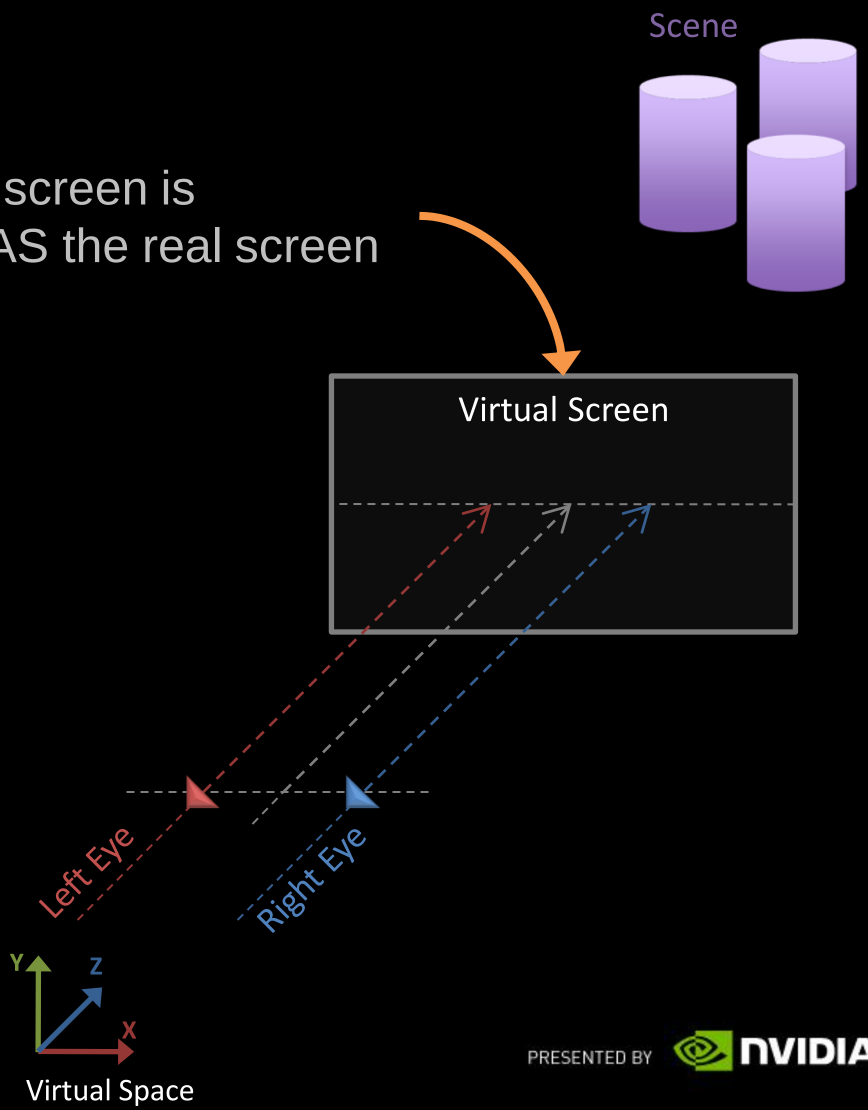
PRESENTED BY NVIDIA.

GDC 2011
NVIDIA SPONSORED SESSIONS

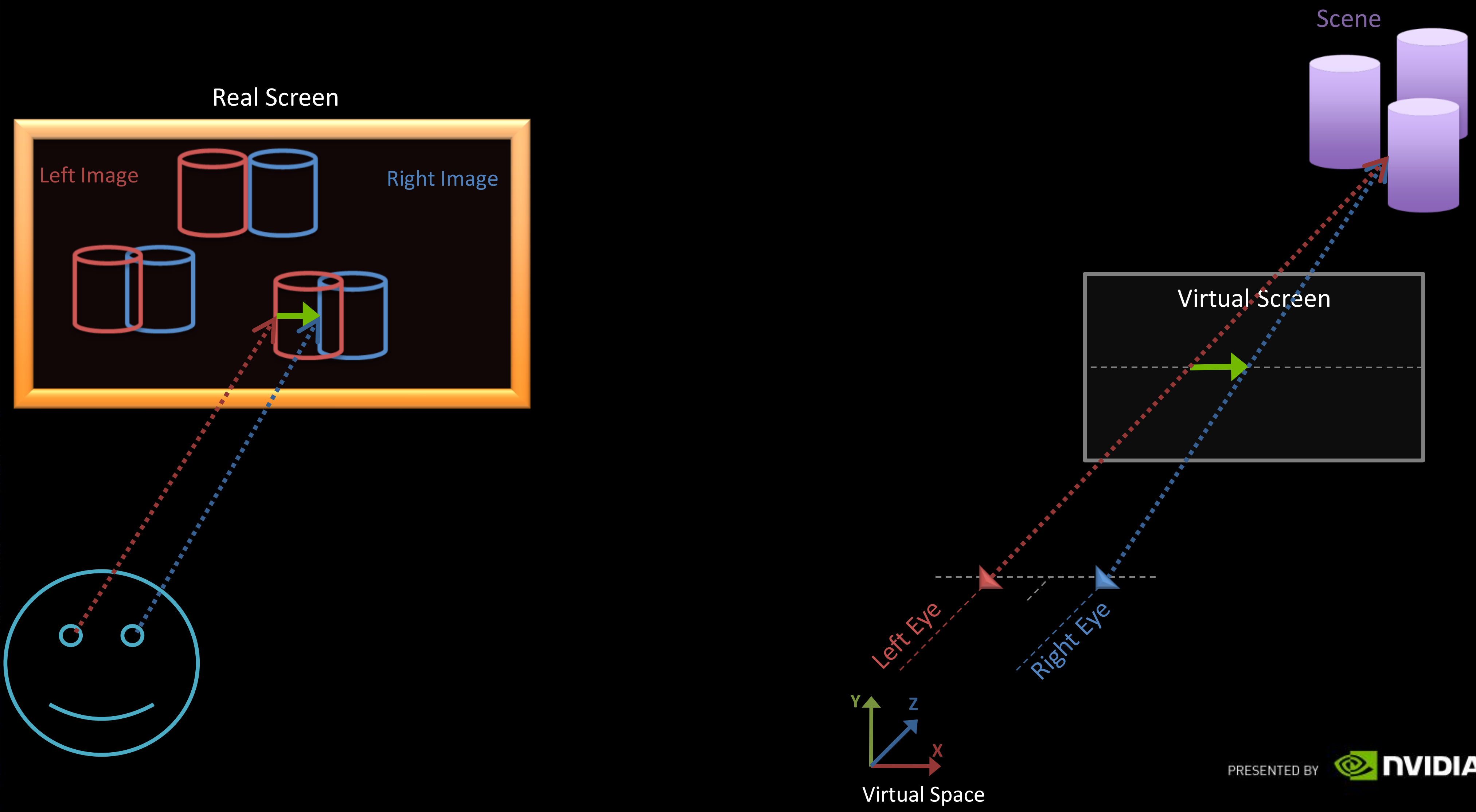
Virtual vs. Real Screen



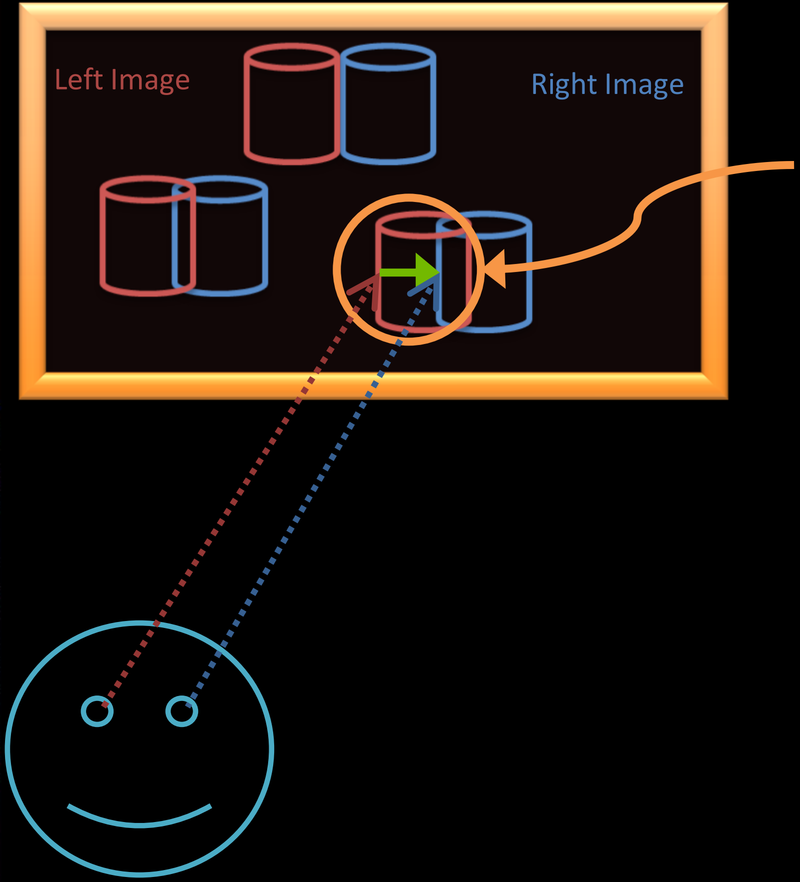
The virtual screen is perceived AS the real screen



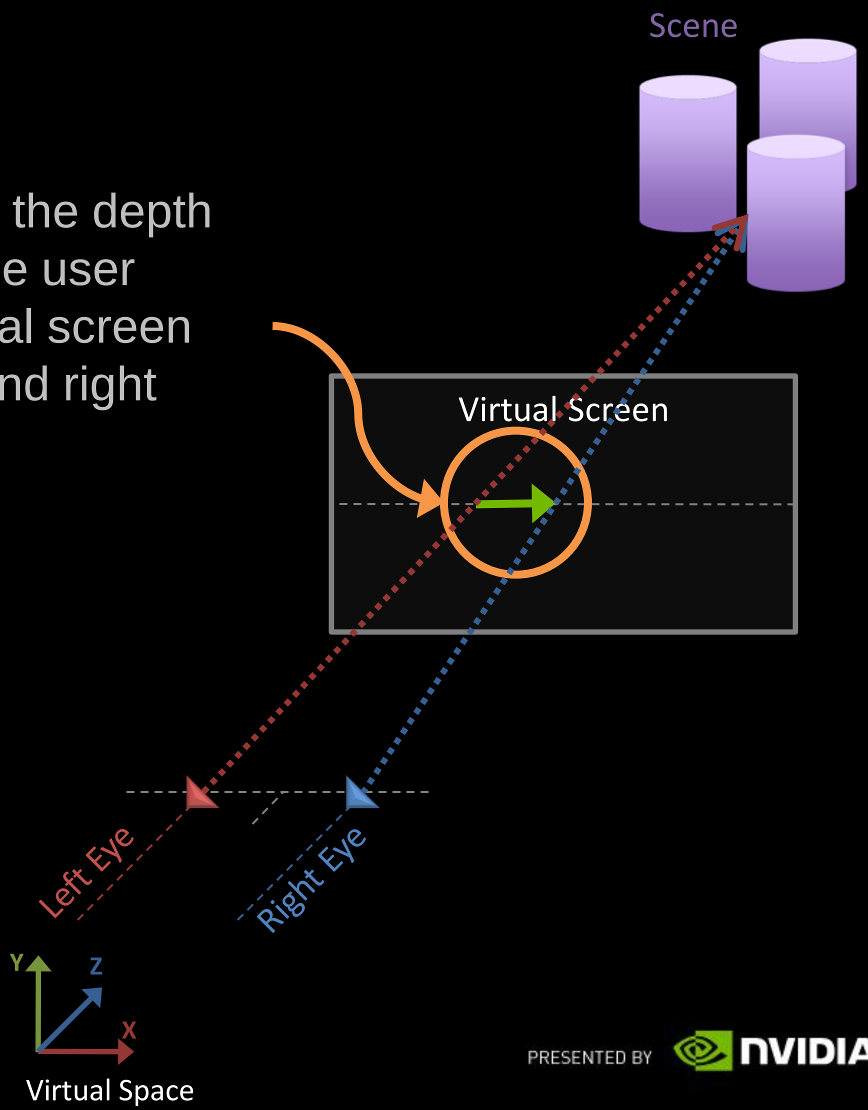
Parallax is Depth



Parallax is Depth

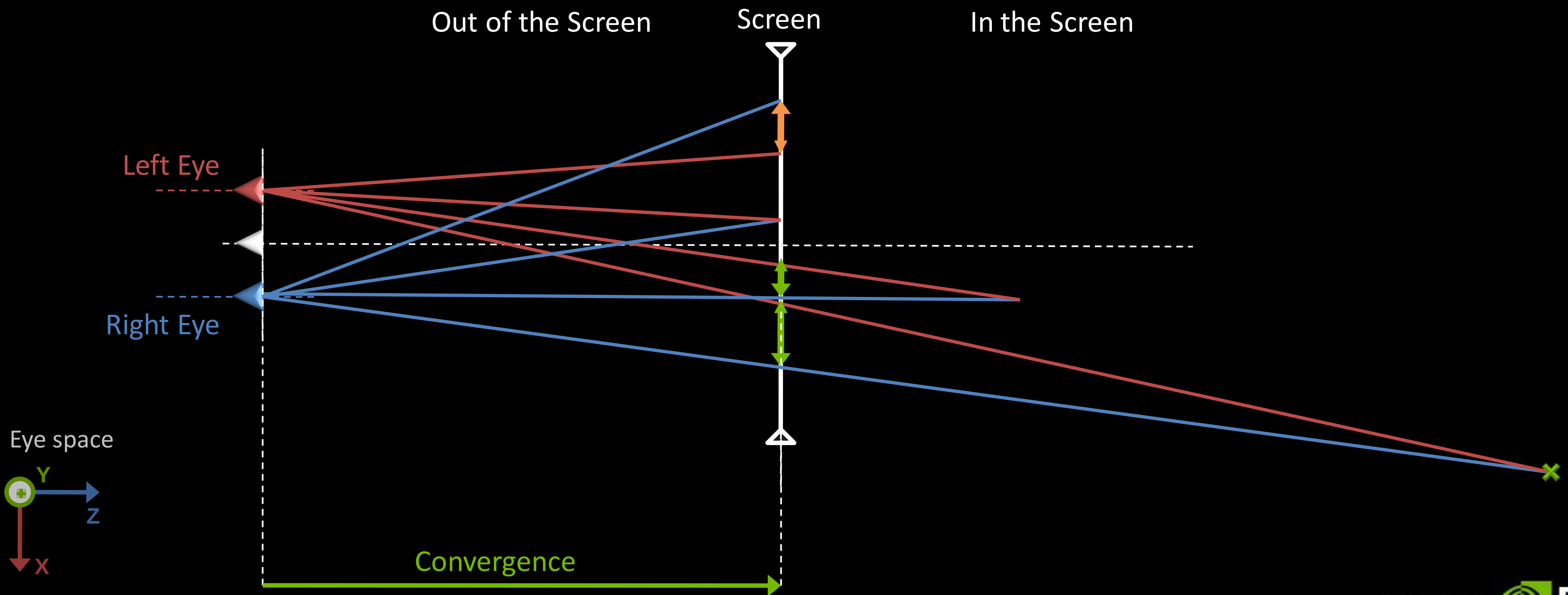


Parallax creates the depth perception for the user looking at the real screen presenting left and right images



In / Out of the Screen

Vertex Depth	Parallax	Vertex Appears
Further than Convergence	Positive	In the Screen
Equal Convergence	Zero	At the Screen
Closer than Convergence	Negative	Out of the Screen



Equations !!!

COMPUTING PARALLAX & PROJECTION MATRIX

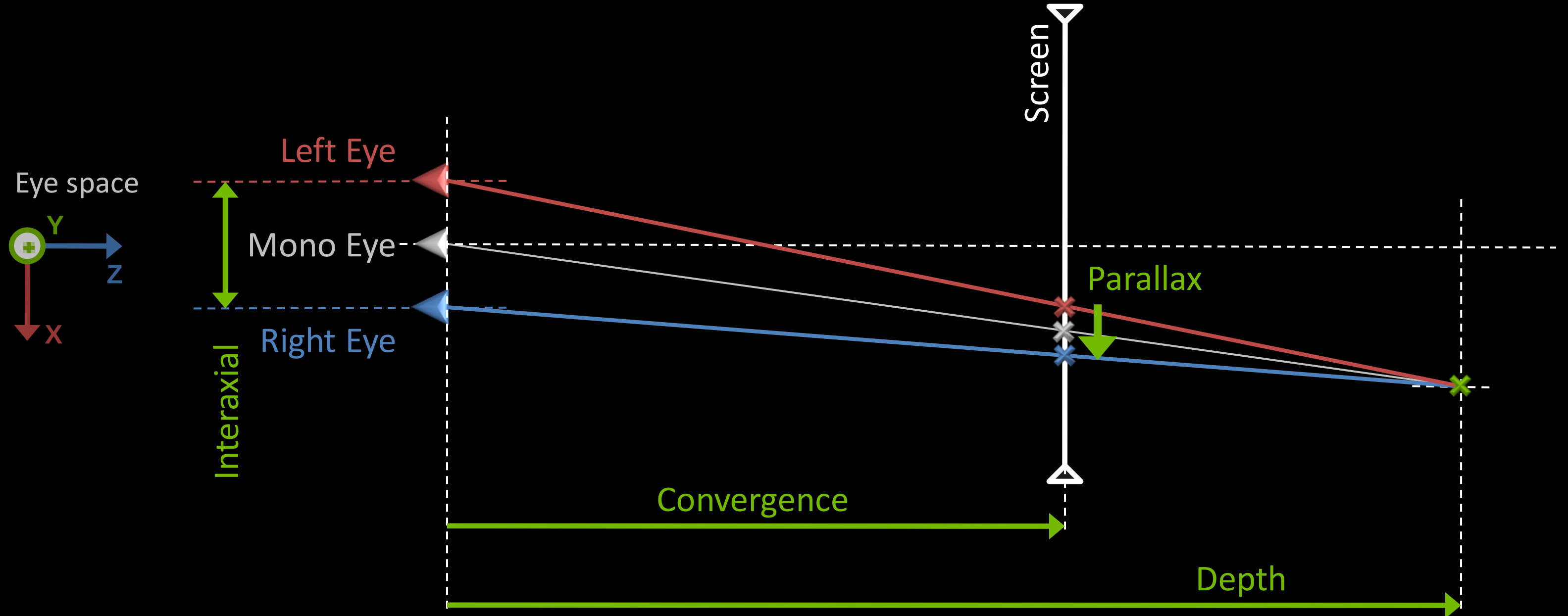
Computing Parallax

Thank you Thales

In eye space:

$$\frac{Parallax_{eye}}{Interaxial} = \frac{Depth - Convergence}{Depth}$$

$$Parallax_{eye} = Interaxial \times \left(1 - \frac{Convergence}{Depth}\right)$$

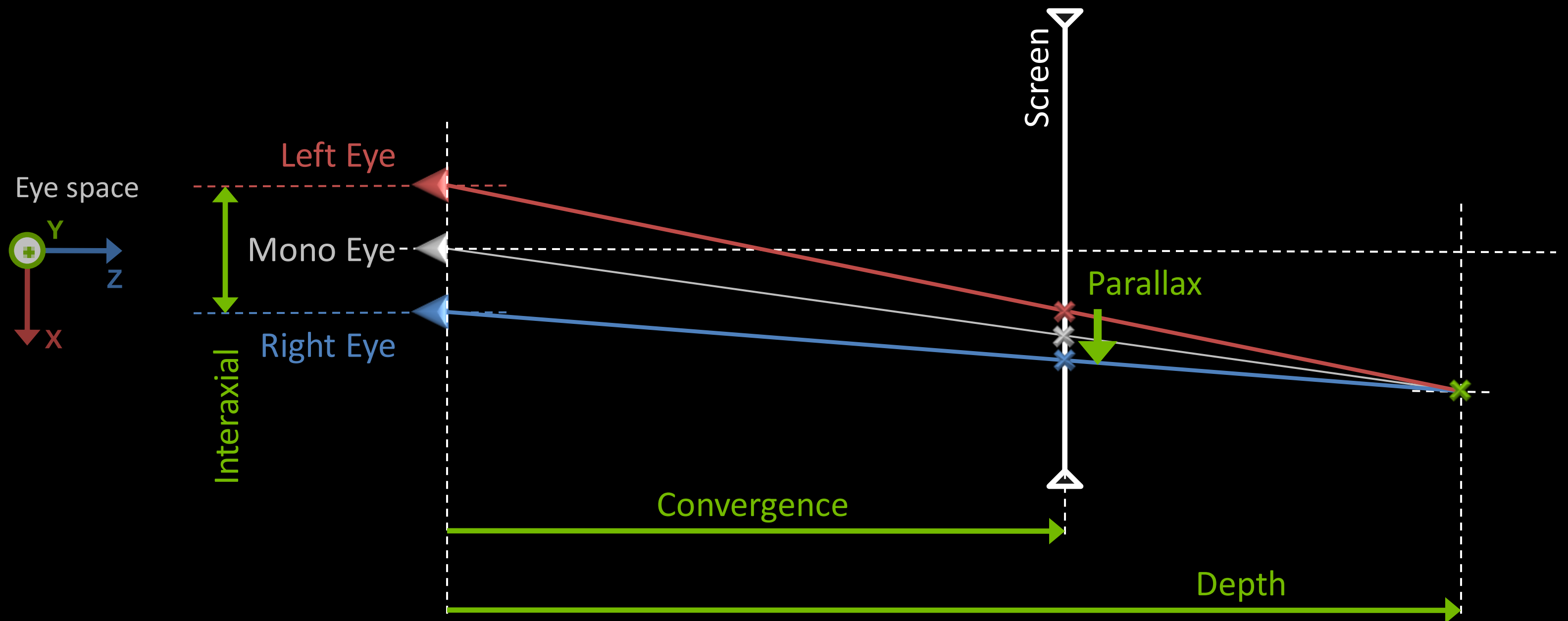


Computing Parallax

In image space (not pixels but in range [0,1])

In image space: $$Parallax_{image} = \frac{Parallax_{eye}}{Screen\ width} = \frac{Interaxial}{Screen\ width} \times \left(1 - \frac{Convergence}{Depth}\right)$$

$$Parallax_{image} = Separation \times \left(1 - \frac{Convergence}{Depth}\right)$$



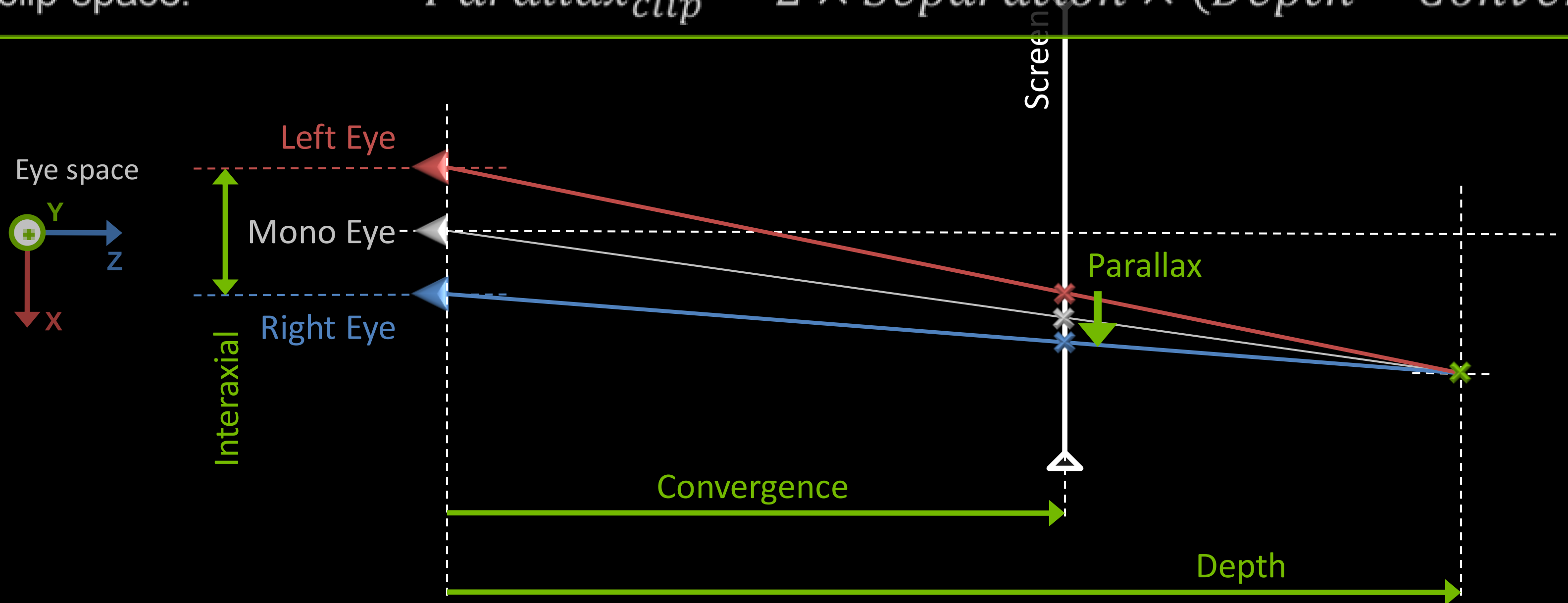
Computing Parallax

And clip space for free

In eye space: $$Parallax_{eye} = Interaxial \times \left(1 - \frac{Convergence}{Depth}\right)$$

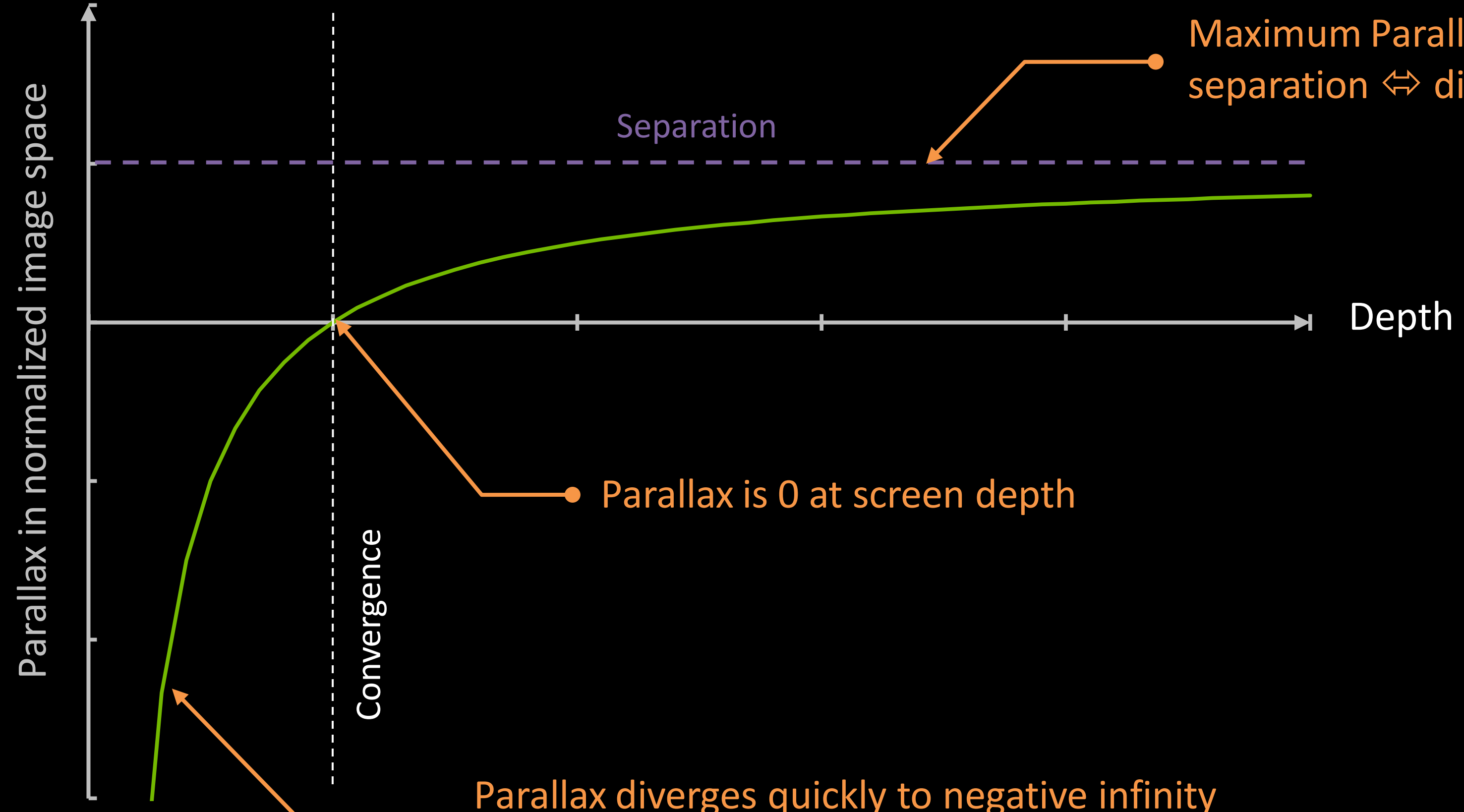
In image space: $$Parallax_{image} = Separation \times \left(1 - \frac{Convergence}{Depth}\right)$$

In clip space: $$Parallax_{clip} = 2 \times Separation \times (Depth - Convergence)$$



Parallax in normalized image space

$$\boxed{Parallax = Separation \times \left(1 - \frac{Convergence}{Depth}\right)}$$



GDC 2011
NVIDIA SPONSORED SESSIONS

Managing the depth

3D Stereoscopic Fundamentals

PRESENTED BY NVIDIA.

GDC 2011
NVIDIA SPONSORED SESSIONS

Take care of your audience

REAL EYE SEPARATION

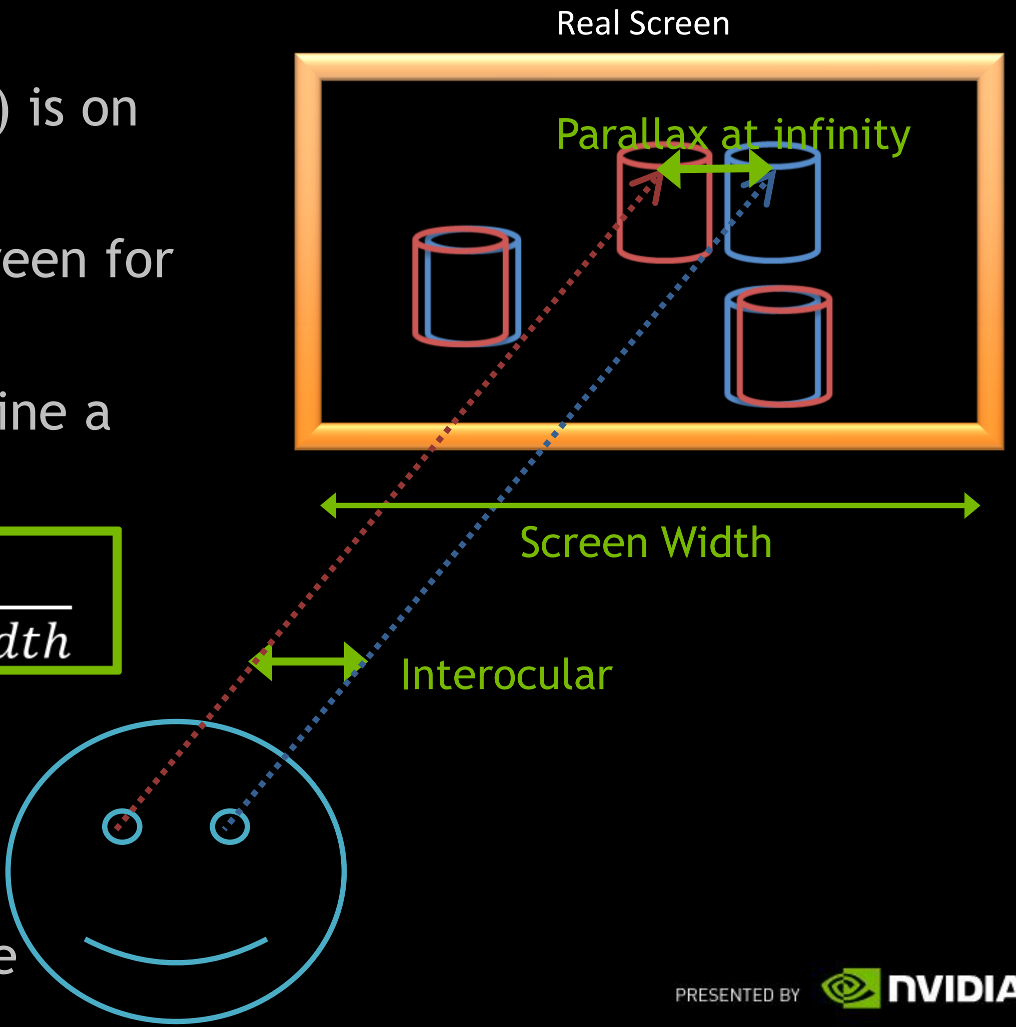
PRESENTED BY NVIDIA.

Real Eye Separation

- Interocular (distance between the eyes) is on average 2.5” ⇔ 6.5 cm
- Equivalent to the visible parallax on screen for objects at infinity
- Depending on the screen width, we define a normalized “Real Eye Separation”

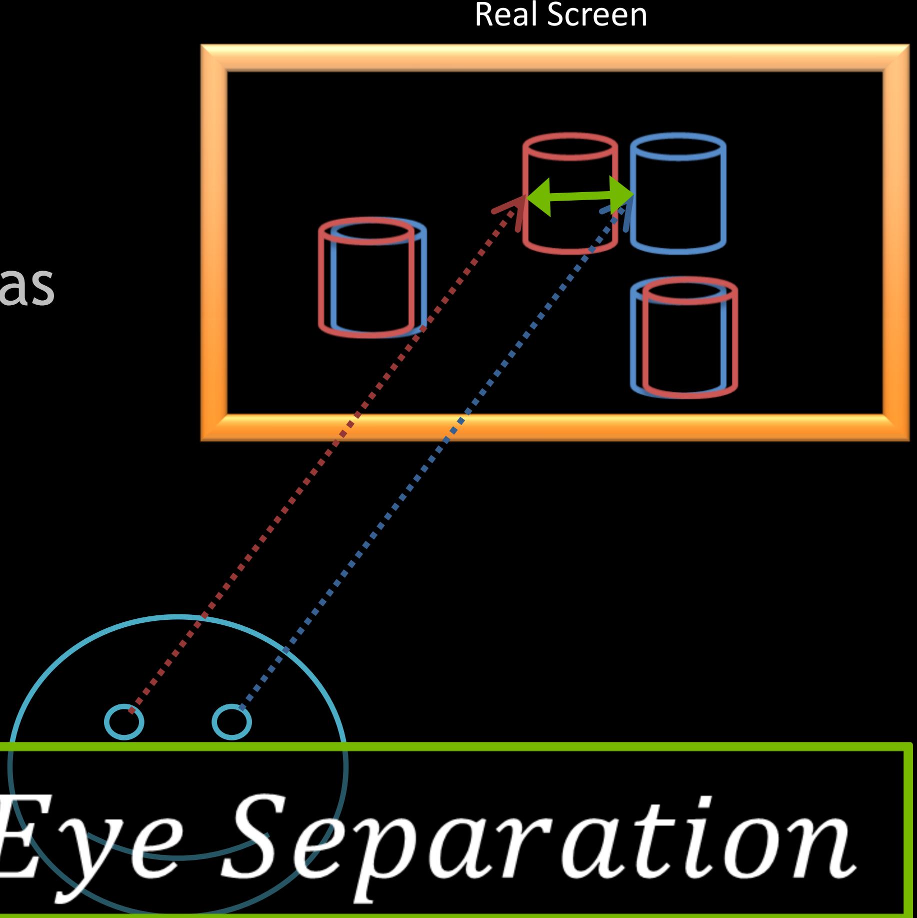
$$\boxed{Real\ Eye\ Separation = \frac{Interocular}{Real\ Screen\ Width}}$$

- Different for each screen model
- A reference maximum value for the Separation used in the stereo projection for a comfortable experience



Real Eye Separation is infinity

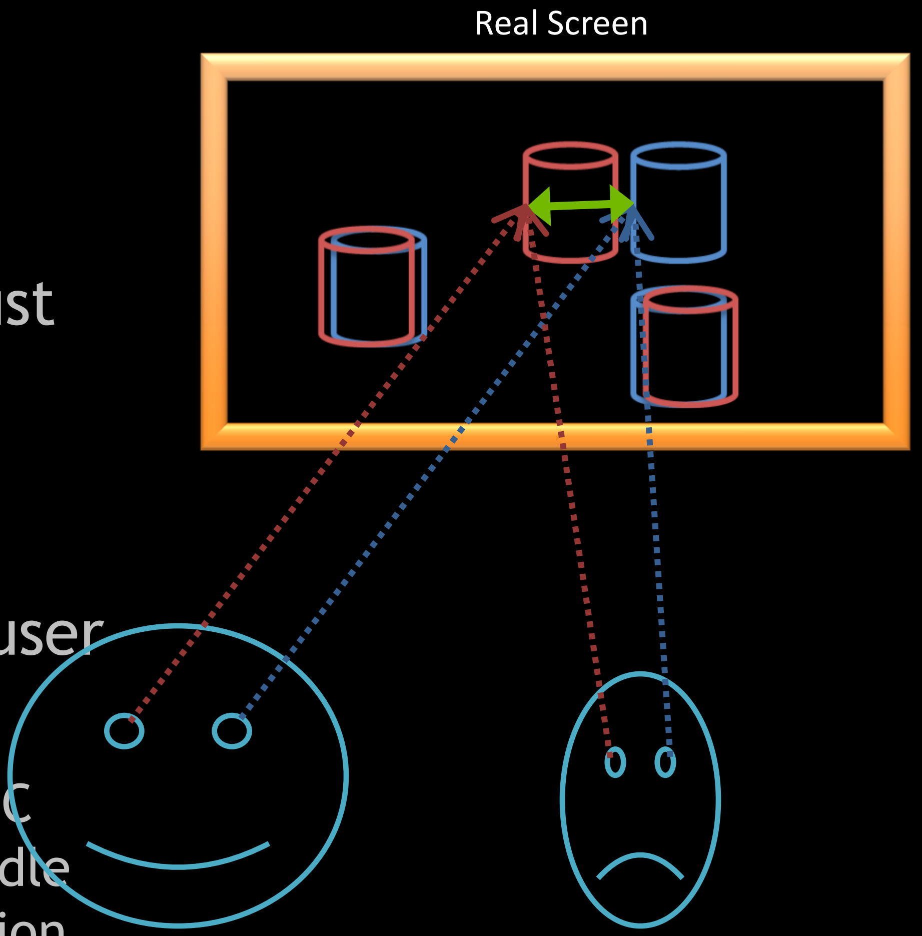
- The maximum Parallax at infinity is Separation
- Real Eye Separation should be used as the very maximum Separation value



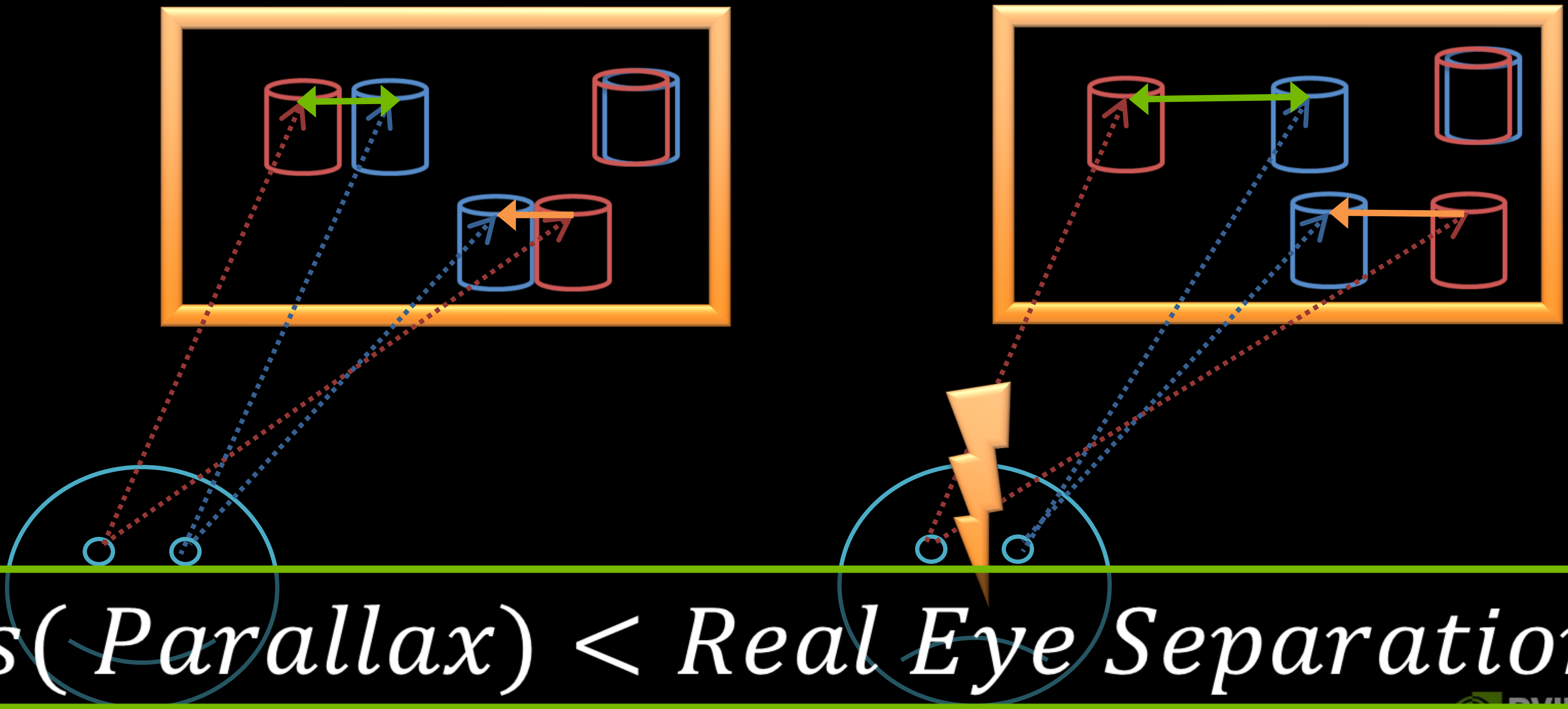
$$Separation < Real\ Eye\ Separation$$

Separation must be Comfortable

- Never make the viewer look diverge
  - People don’t have the same eyes
- For Animation movie, separation must be very conservative because of the variety of the screen formats
  - IMAX vs Home theatre
- For Interactive application, let the user adjust Separation
  - When the screen is close to the user (PC scenario) most of the users cannot handle more than 50% of the Real Eye Separation



Real Eye Separation is the Maximum Parallax



$$abs(Parallax) < Real\ Eye\ Separation$$

Safe Parallax Range

$$abs(Parallax) < Real\ Eye\ Separation$$

GDC 2011

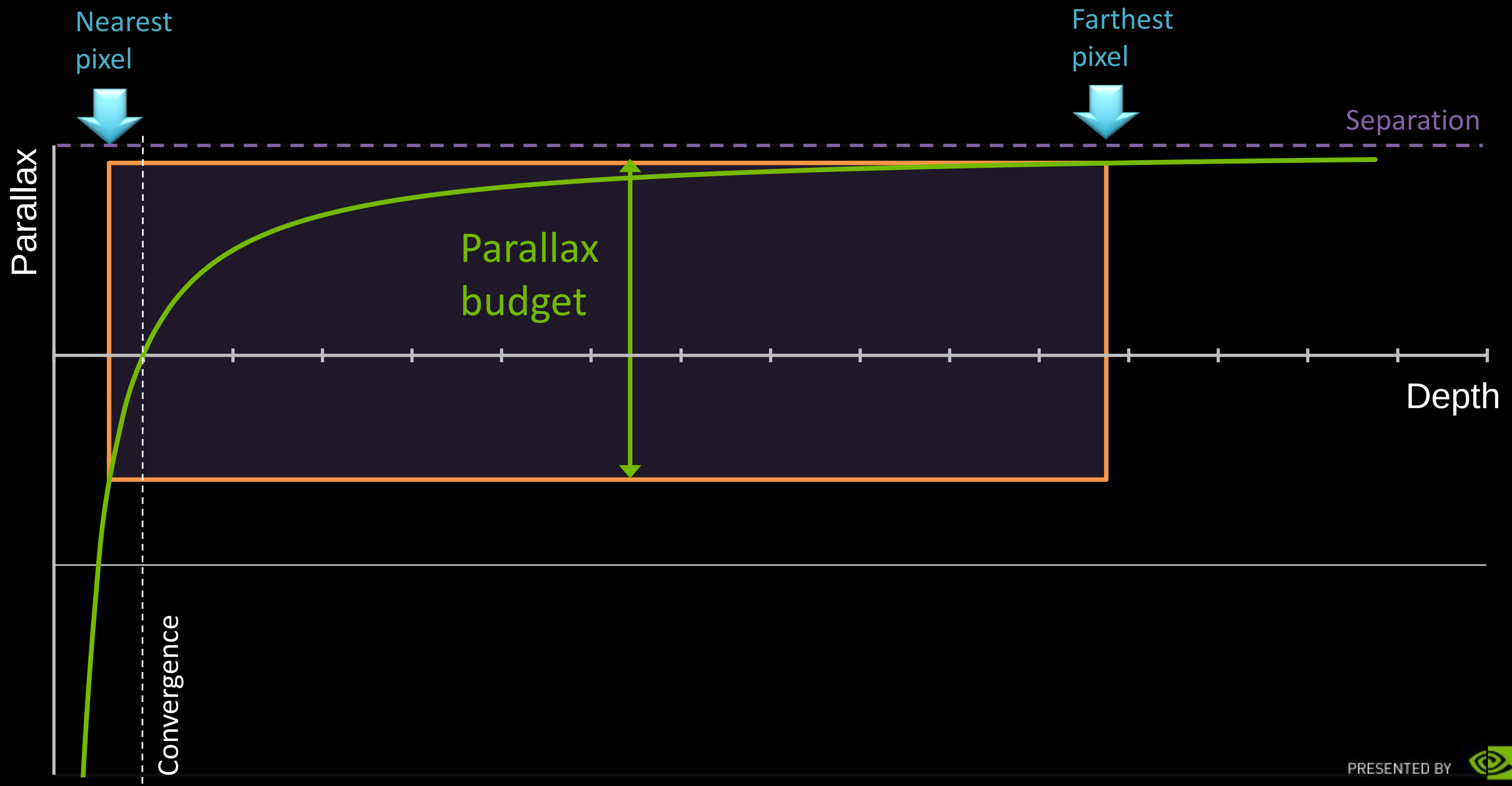
NVIDIA SPONSORED SESSIONS

Convergence and Separation working together

PARALLAX BUDGET

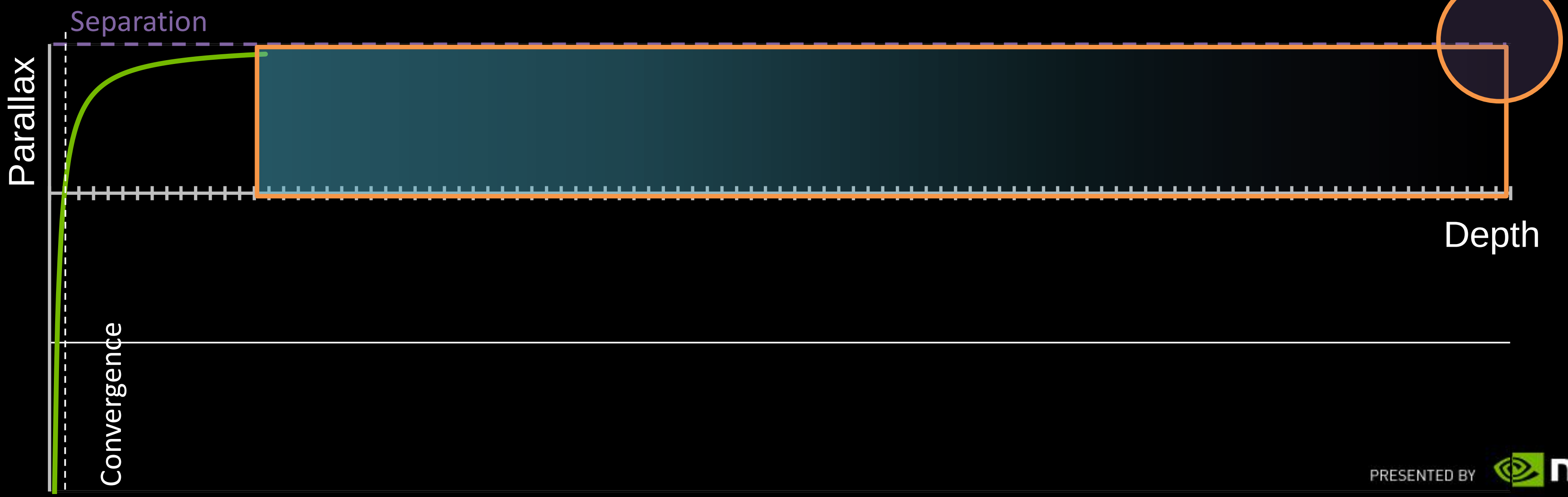
PRESENTED BY NVIDIA.

Parallax Budget
How much parallax variation is used in the frame



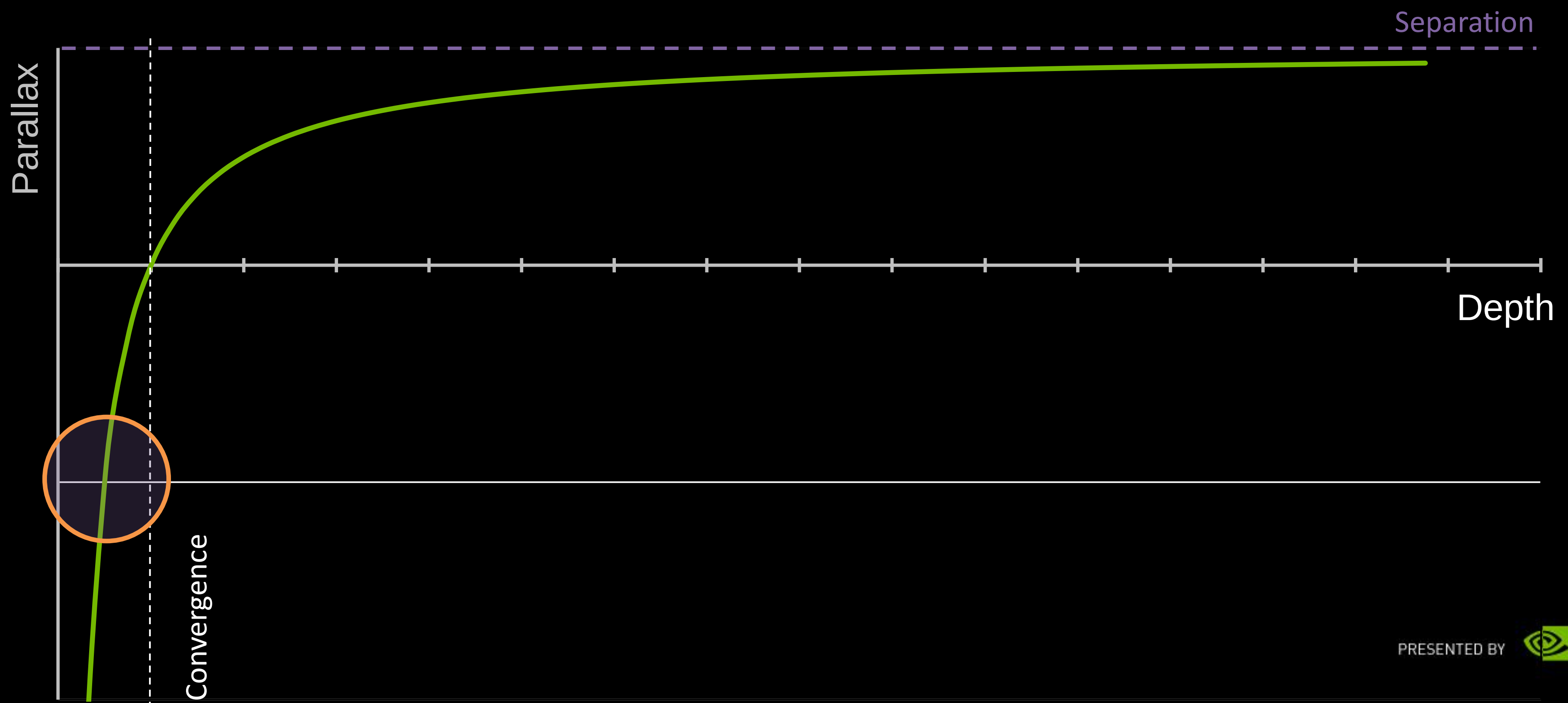
In Screen : Farthest Pixel

- At 100 * Convergence, Parallax is 99% of the Separation
  - For pixels further than 100 * Convergence,
    Elements looks flat on the far distance with no depth differentiation
- Between 10 to 100 * Convergence, Parallax vary of only 9%
  - Objects in that range have a subtle depth differentiation



Out of the Screen : Nearest pixel

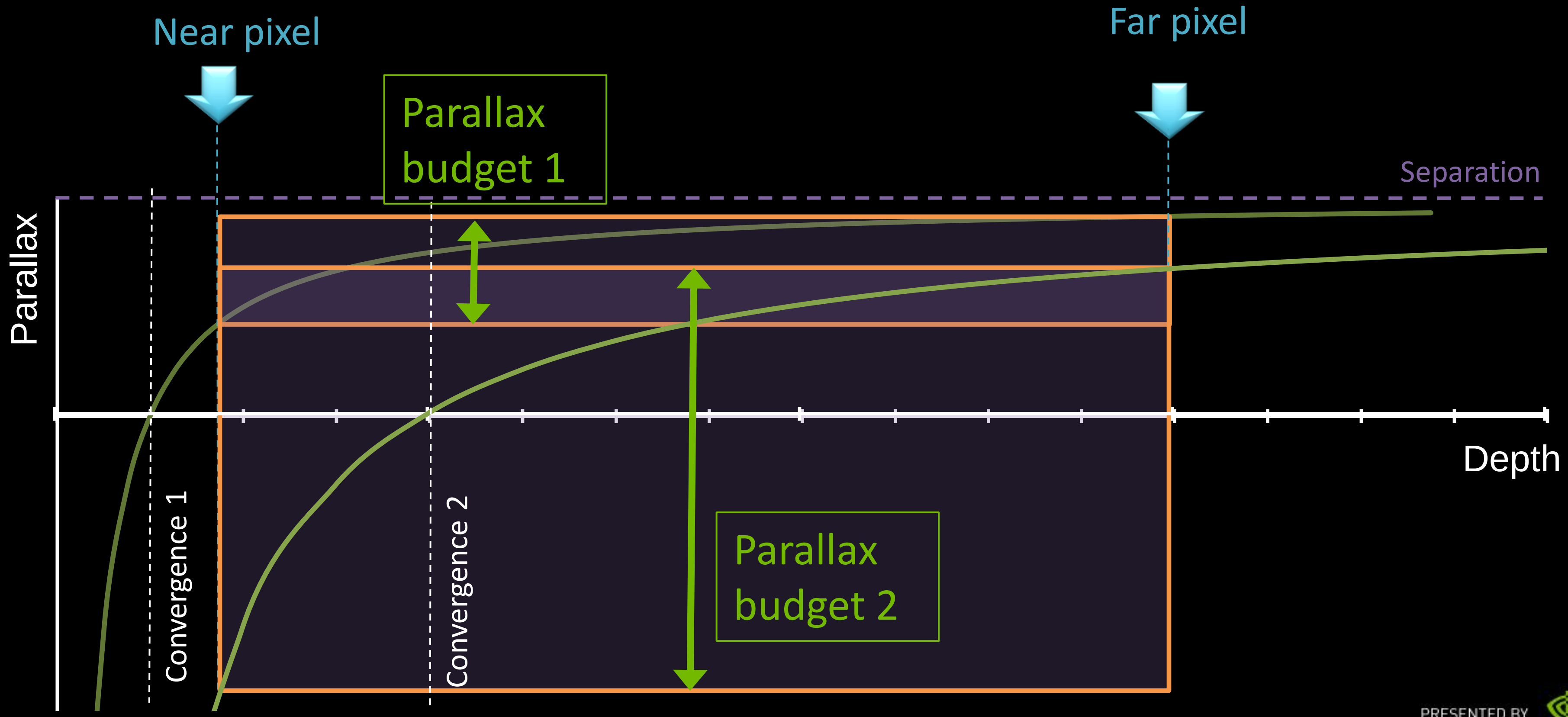
- At Convergence / 2, Parallax is equal to -Separation, out of the screen
  - Parallax is very large (> Separation) and can cause eye strains



Convergence sets the scene in the screen

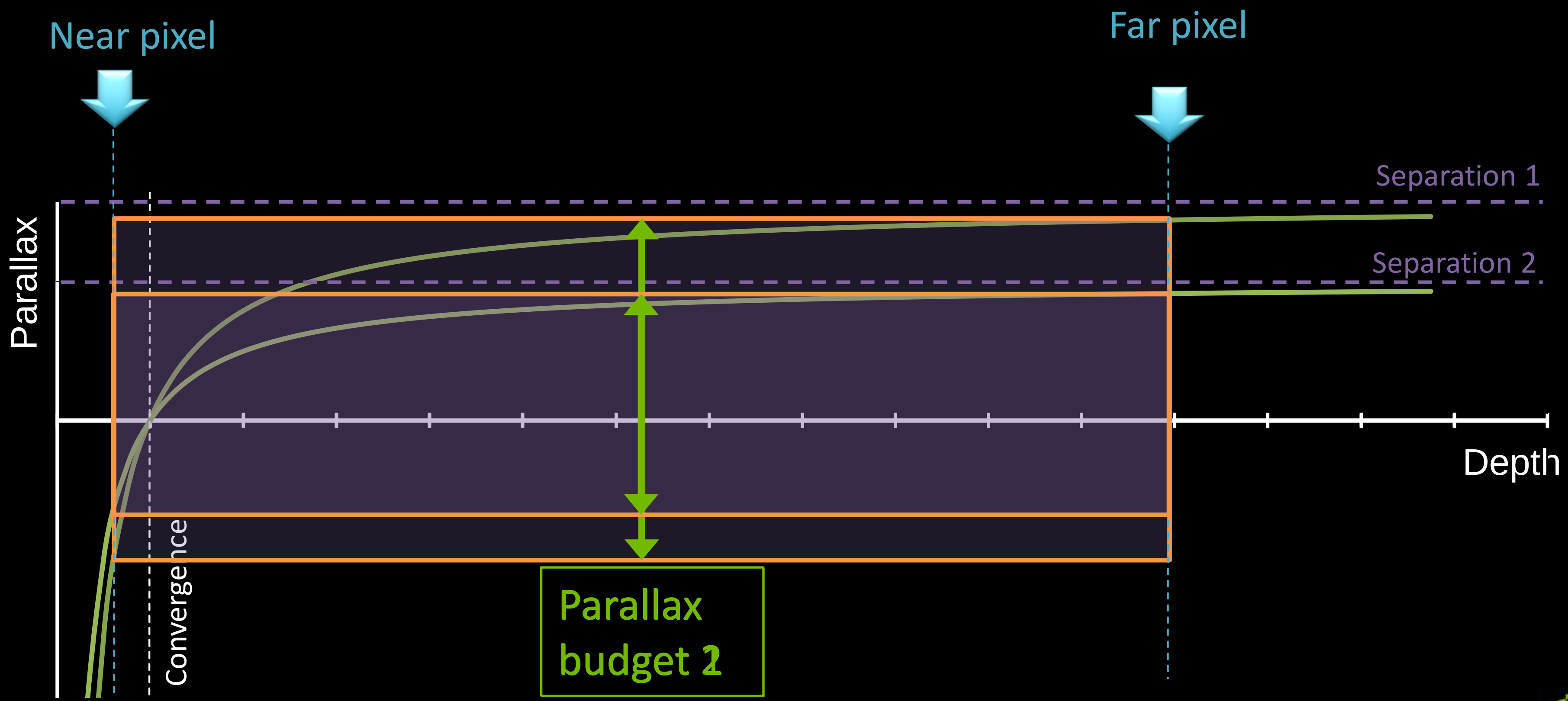
Defines the window into the virtual space

Defines the style of stereo effect achieved (in / out of the screen)



Separation scales the parallax budget

Scales the depth perception of the frame



Adjust Convergence

- Convergence is a Camera parameter driven by the look of the frame
  - Artistic / Gameplay decision
  - Should adjust for each camera shot / mode
    - Make sure the scene elements are in the range [Convergence / 2, 100 * Convergence]
  - Adjust it to use the Parallax Budget narratively
    - Art direction on the stereo
  - Dynamic Convergence is a bad idea
    - Except for specific transition cases

Stereoscopy in a game engine

Let's do it

RENDERING IN STEREO

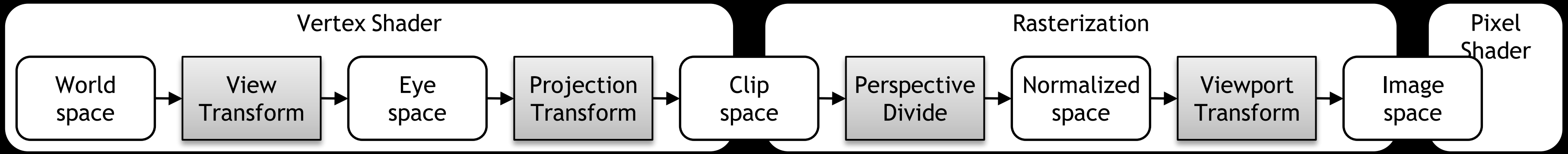
Stereoscopic Rendering

Render geometry twice	Do stereo drawcalls	Duplicate drawcalls Using correct stereo resources side
From left and right eyes	Apply stereo projection	Modify projection matrix
Into left and right images	Use stereo surfaces	Duplicate render surfaces

GDC 2011
NVIDIA SPONSORED SESSIONS

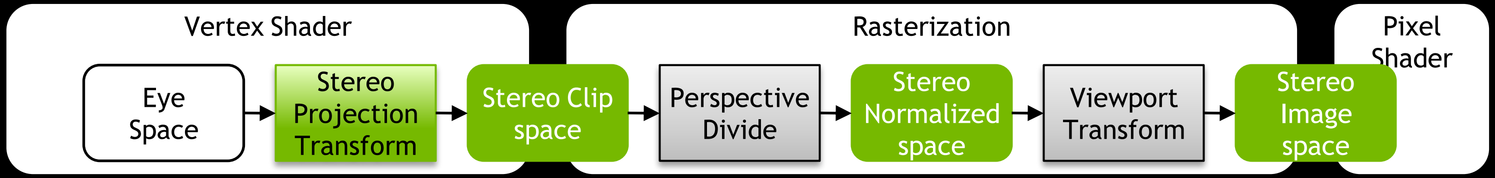
How to implement stereo projection ?

- Start from the mono transformation stack

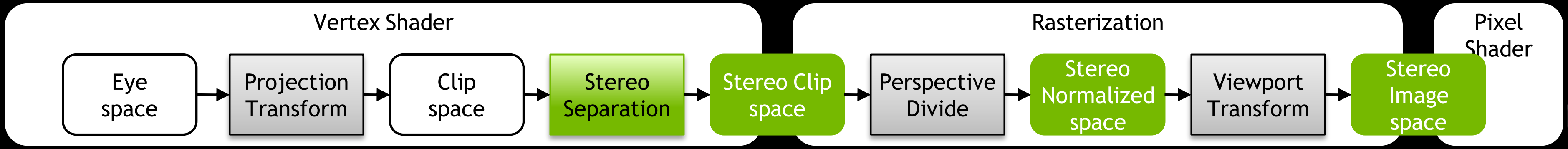


- Inject the side, separation and convergence to get a stereo transformation stack

Stereo Projection Matrix



Stereo shift on clip position



NVIDIA

Stereo Projection Matrix

Right handed column major matrix (OpenGL style)

- Modified version of the Projection matrix for stereo to transform geometry position from eye space to stereo clip space
  - $Pos_{clip\ stereo} = \boldsymbol{Projection}_{stereo} \times Pos_{eye}$

Right handed column major matrix (OpenGL style)

$$\boldsymbol{Projection}_{stereo} = \begin{bmatrix} p11 & 0 & p13 - side * separation & -side * separation * convergence \\ 0 & p22 & p23 & 0 \\ 0 & 0 & p33 & p34 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Side is -1 for left, +1 for right

pij are the coefficients of the standard mono perspective projection

Stereo Projection Matrix

Left handed row major matrix (D3D9 style)

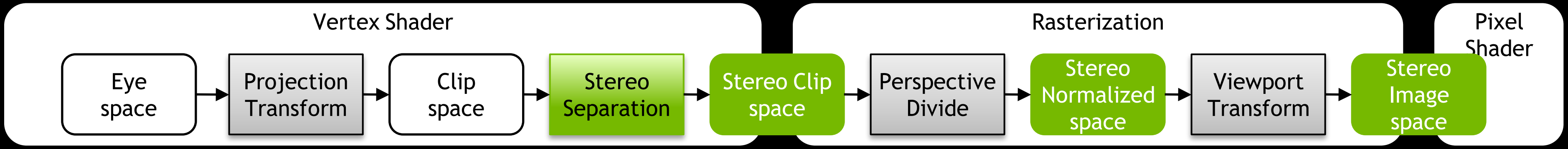
- $Pos_{clip\ stereo} = Pos_{eye} \times \boldsymbol{Projection}_{stereo}$

Left handed row major matrix (D3D9 style)

$$\boldsymbol{Projection}_{stereo} = \begin{bmatrix} p11 & 0 & 0 & 0 \\ 0 & p22 & p32 & 0 \\ p13 + side * separation & 0 & p33 & 1 \\ -side * separation * convergence & 0 & p34 & 0 \end{bmatrix}$$

Side is -1 for left, +1 for right
pij are the coefficients of the standard mono perspective projection

Stereo shift on clip position



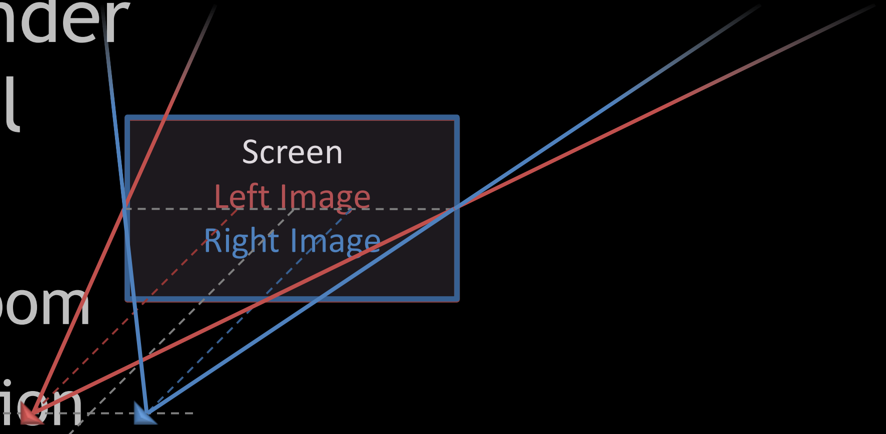
- Just before rasterization in the vertex shader, offset the clip position by the parallax amount

$$clipPos.x \mathrel{+}= Side * Separation * (clipPos.w - Convergence)$$

Side is -1 for left, +1 for right

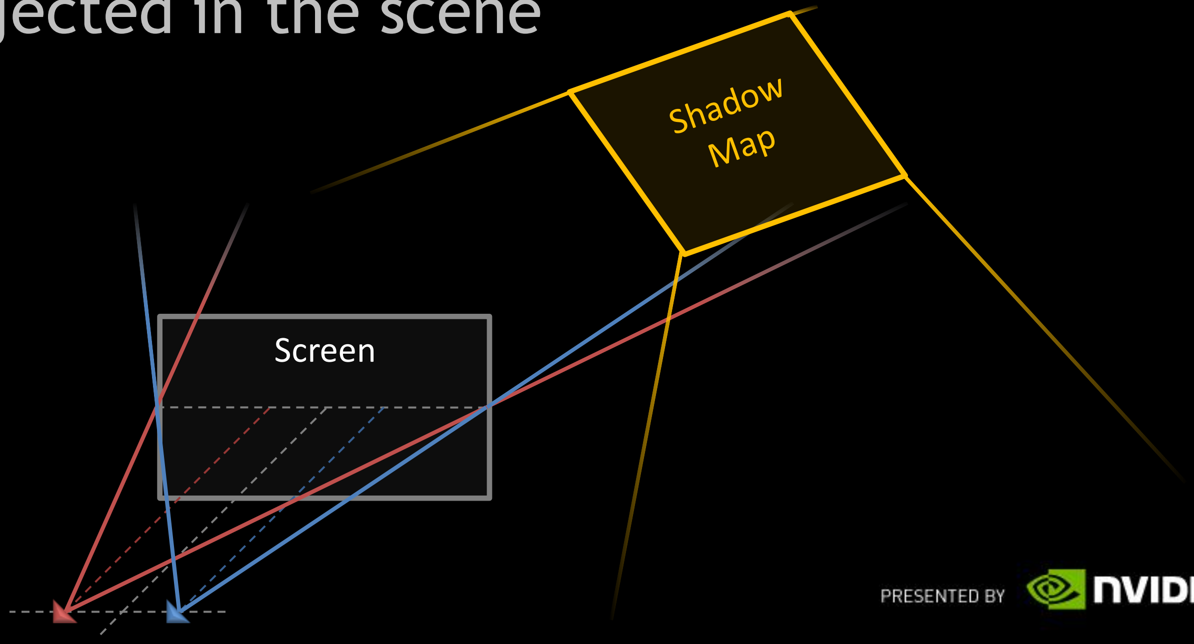
Stereo rendering surfaces

- View dependent render targets must be duplicated
  - Back buffer
  - Depth Stencil buffer
- Intermediate full screen render targets used to process final image
  - High dynamic range, Blur, Bloom
  - Screen Space Ambient Occlusion



Mono rendering surfaces

- View independent render targets DON'T need to be duplicated
  - Shadow map
  - Spot light maps projected in the scene



How to do the stereo drawcalls ?

- Simply draw the geometries twice, in left and right versions of stereo surfaces
- Can be executed per scene pass
  - Draw left frame completely
  - Then Draw right frame completely
  - Need to modify the rendering loop
- Or for each individual objects
  - Bind Left Render target, Setup state for left projection, Draw geometry
  - Bind Right render target, Setup state for right projection, Draw Geometry
  - Probably less intrusive and more efficient in an engine (3D vision driver solution)
- Not everything in the scene needs to be drawn
  - Just depends on the render target type

GDC 2011
NVIDIA SPONSORED SESSIONS

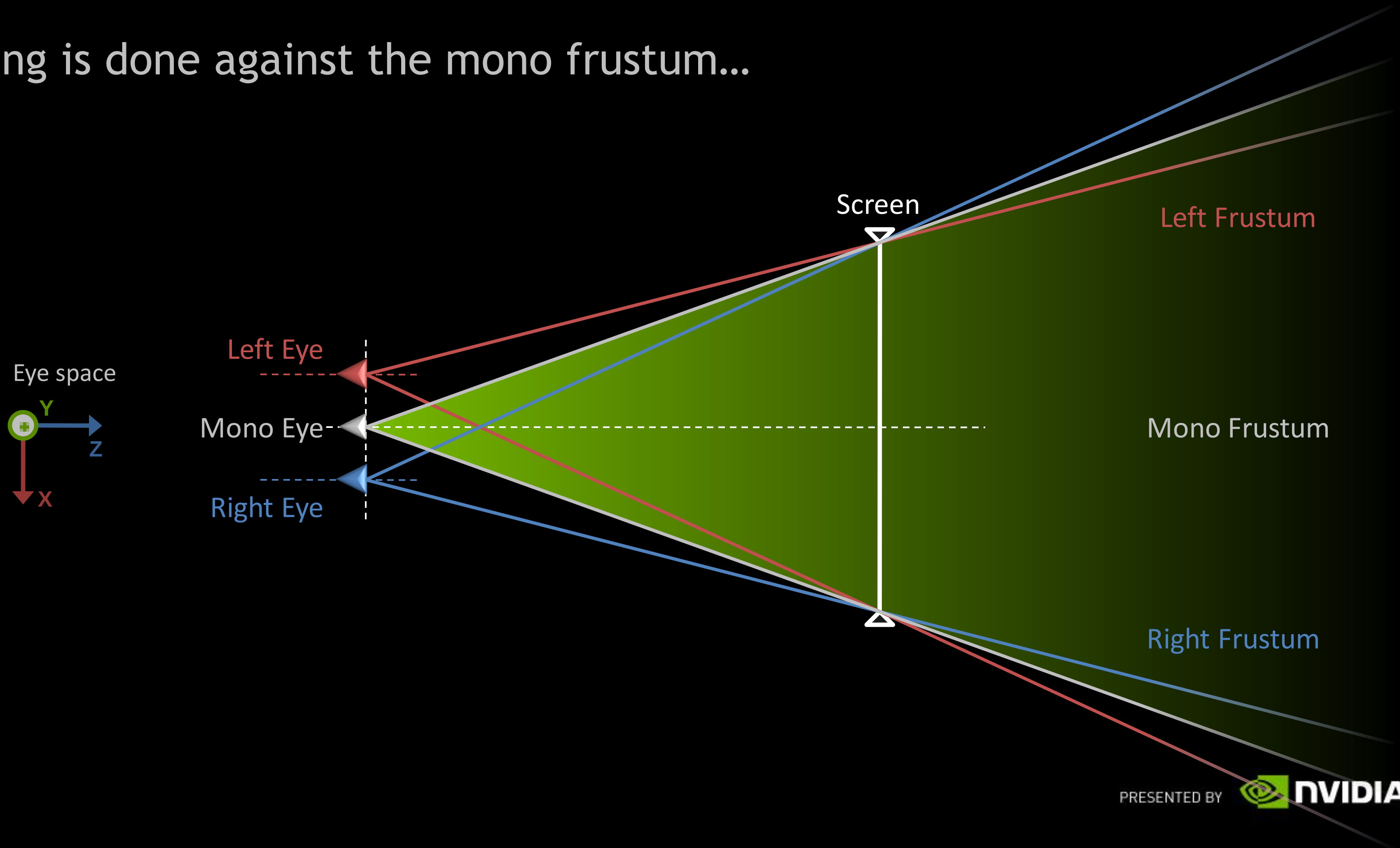
When to do what?

Use Case	Render Target 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
Deferred shading lighting pass (Drawing a full screen quad)	Stereo G-buffers	Yes Be careful of the Unprojection Should be stereo	Draw twice

STEREO CULLING & FLOATING WINDOW

3D Objects Culling

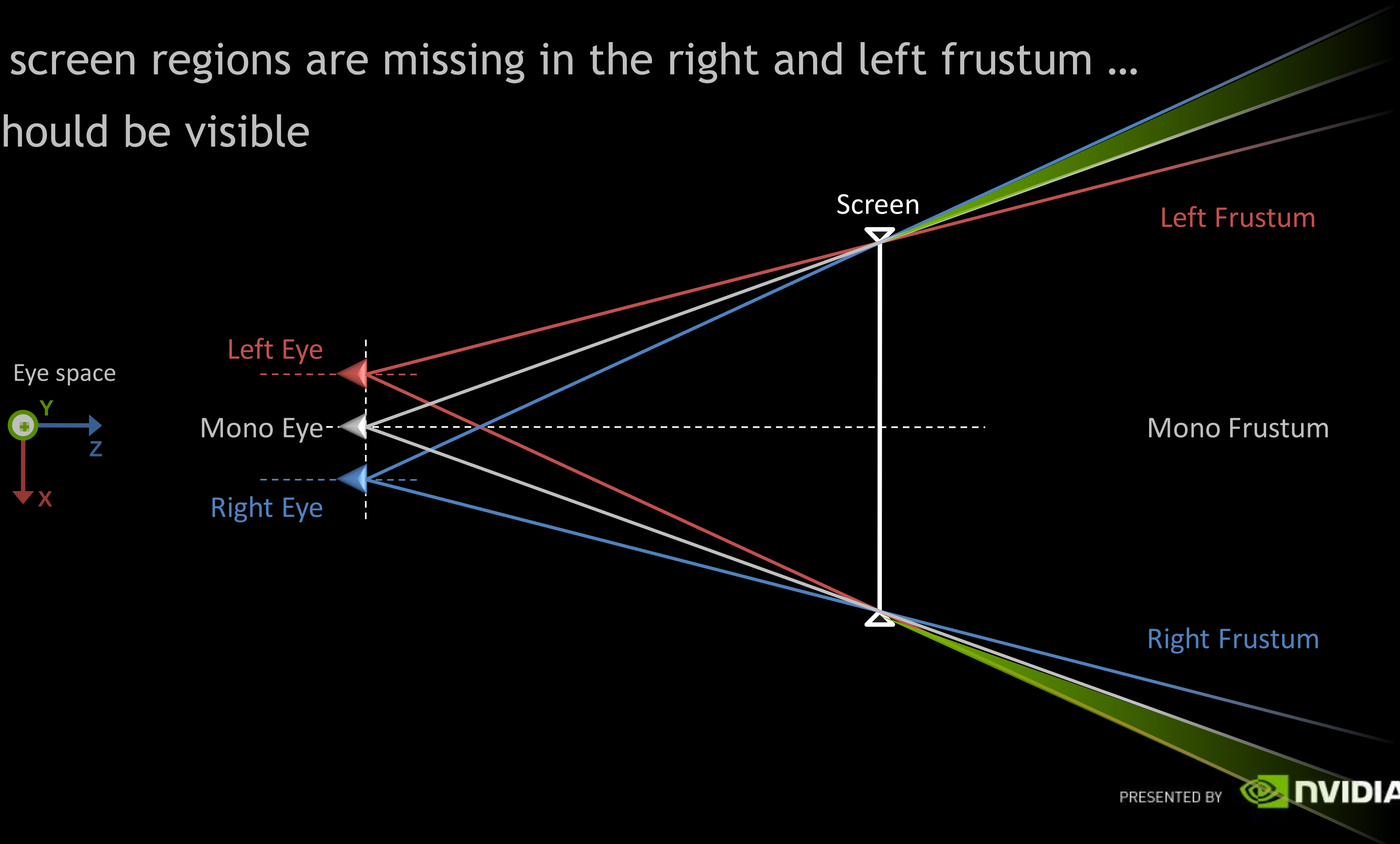
When culling is done against the mono frustum…



3D Objects Culling

… Some in screen regions are missing in the right and left frustum …

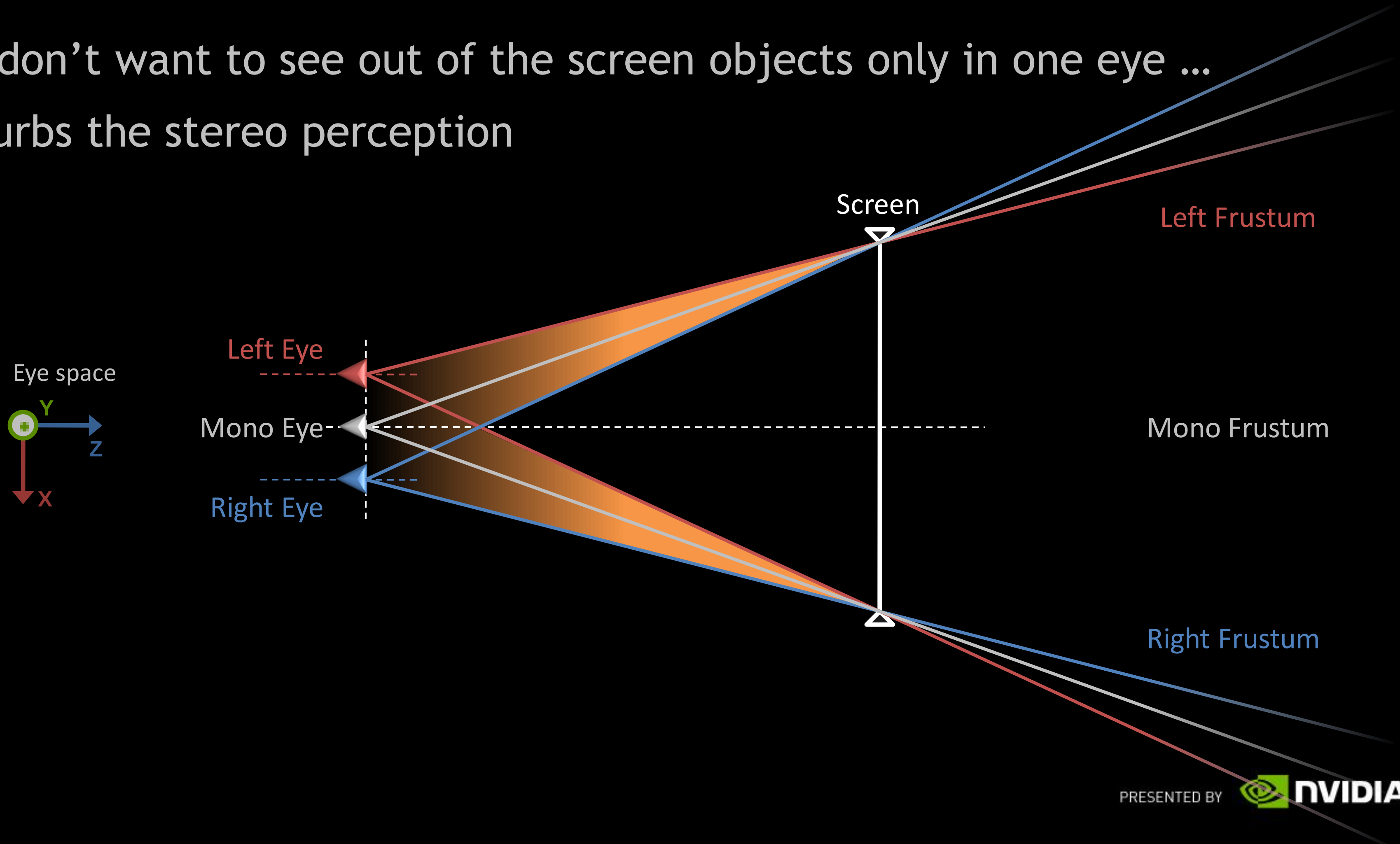
They should be visible



3D Objects Culling

… And we don't want to see out of the screen objects only in one eye …

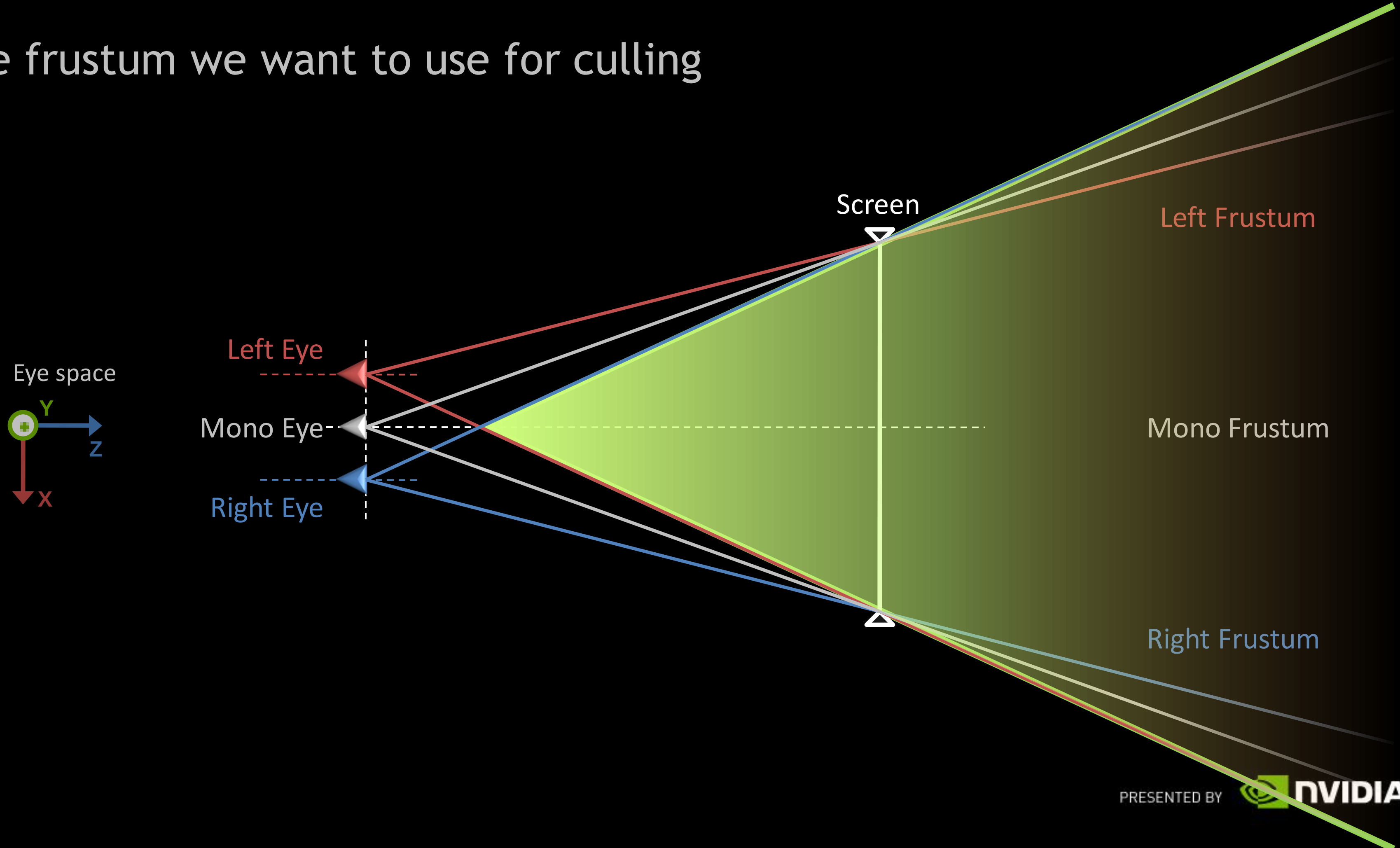
It disturbs the stereo perception



GDC 2011
NVIDIA SPONSORED SESSIONS

3D Objects Culling

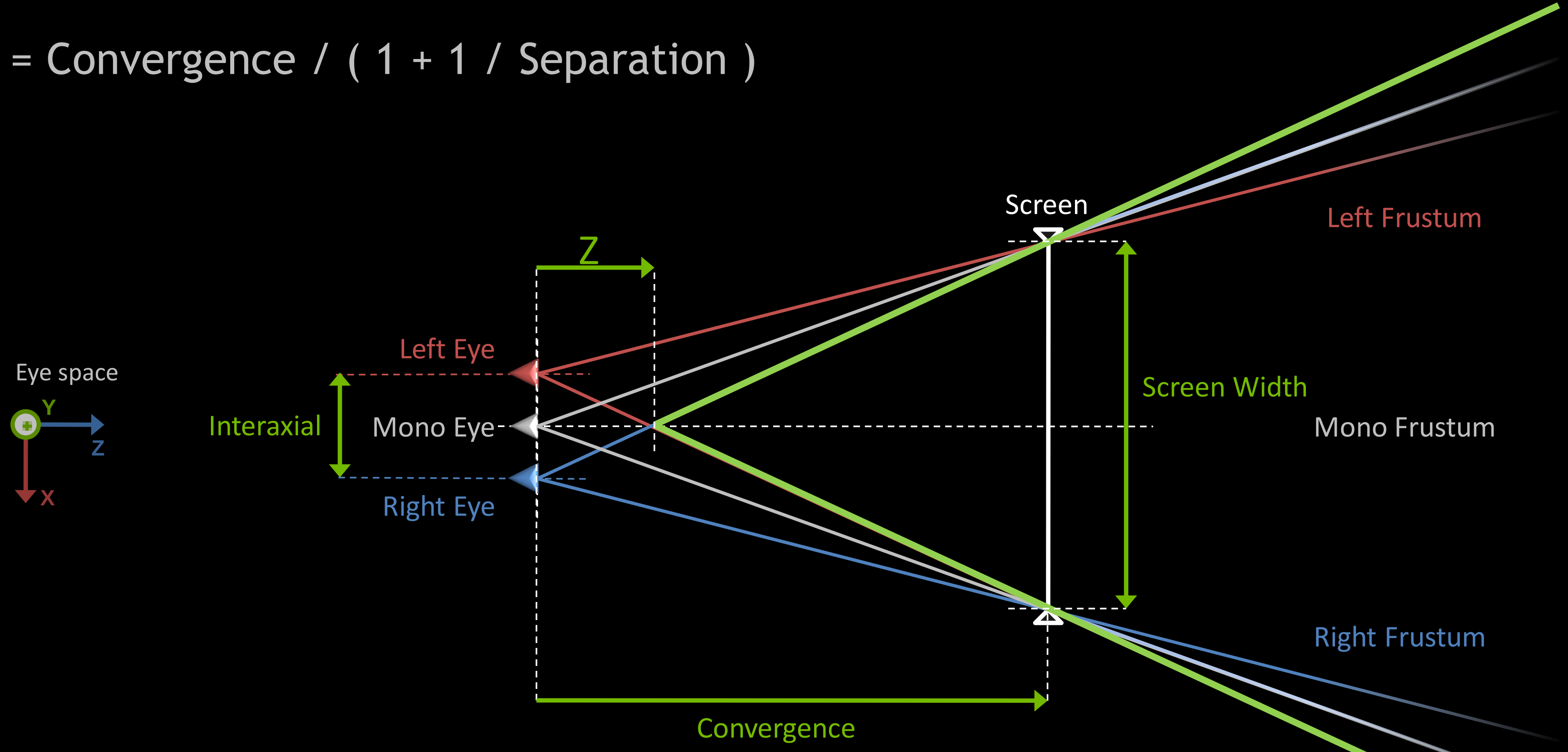
Here is the frustum we want to use for culling



3D Objects Culling

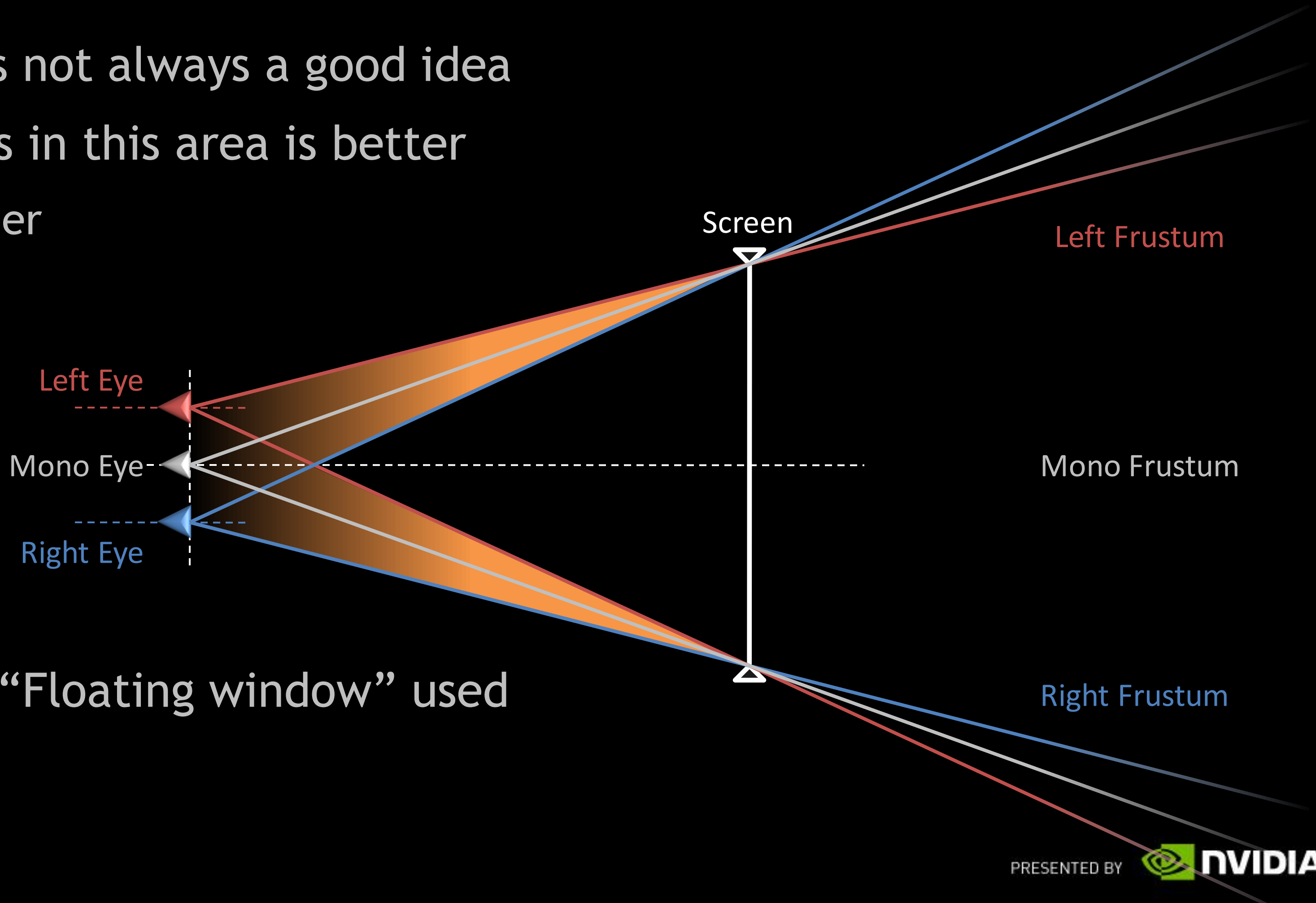
Computing Stereo Frustum origin offset

Z = Convergence / (1 + 1 / Separation)



3D Objects Culling

- Culling this area is not always a good idea
- Blacking out pixels in this area is better
  - Through a shader



- Equivalent to the "Floating window" used in movies

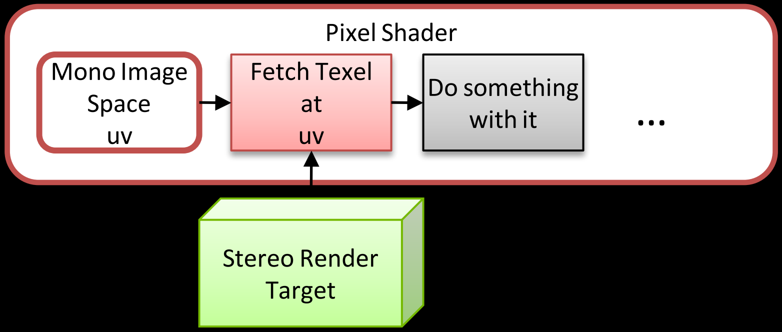
GDC 2011
NVIDIA SPONSORED SESSIONS

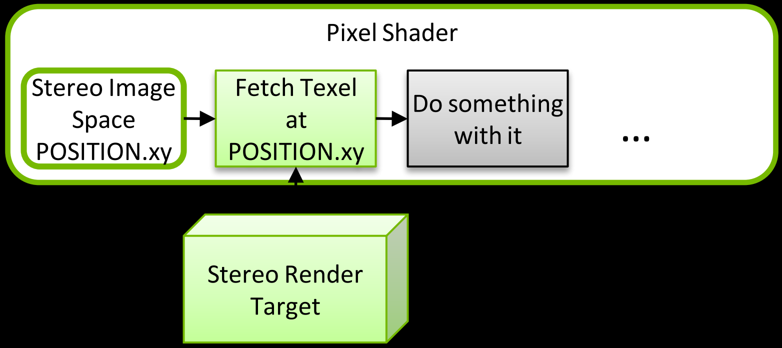
STEREO TRANSFORM STACK TRICKS

PRESENTED BY NVIDIA.

Fetching Stereo Render Target

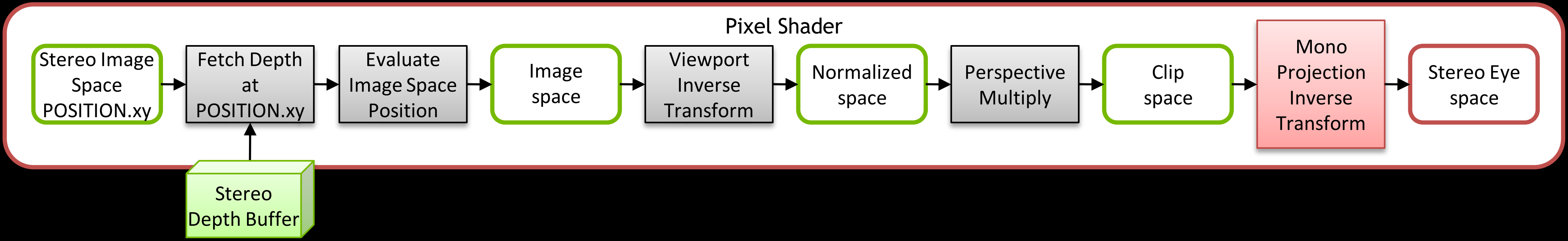
- When fetching from a stereo render target use the good texture coordinate
  - Render target is addressed in STEREO IMAGE SPACE
  - Use the pixel position provided in the pixel shader
  - Or use a texture coordinate computed in the vertex shader correctly

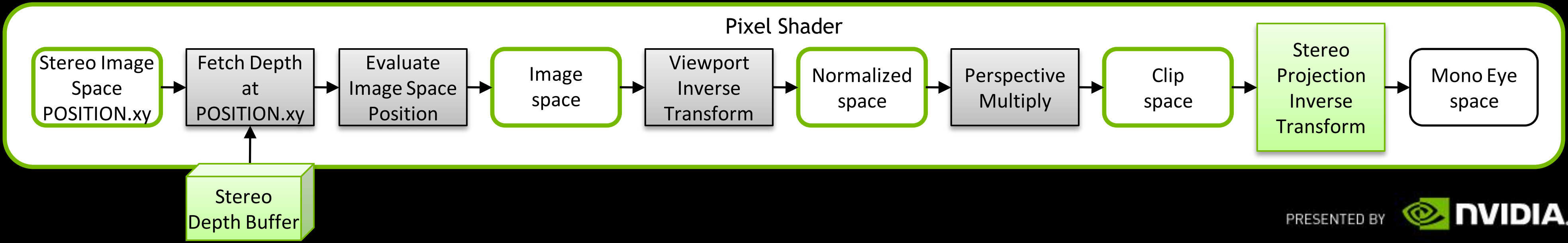




Unprojection in pixel shader

- When doing deferred shading technique, Pixel shader fetch the depth buffer (beware of the texcoord used, cf previous slide)
  - And evaluate a 3D clip position from the Depth fetched and XY viewport position
  - Make sure to use a Stereo Unprojection Inverse transformation to go to Mono Eye space
  - Otherwise you will be in a Stereo Eye Space !





GDC 2011

NVIDIA SPONSORED SESSIONS

Finally something useful

NVIDIA 3D VISION DRIVER

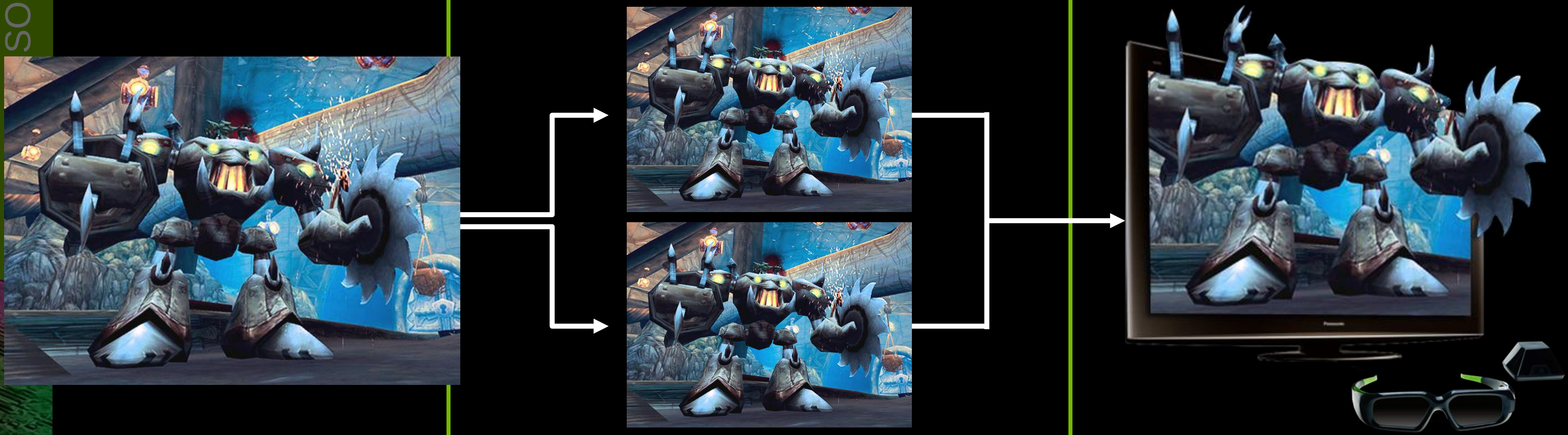
PRESENTED BY NVIDIA.

How Games are Converted to 3D automatically ?

PC Game & DirectX

NVIDIA GPU & 3D vision driver

3D Display and 3D glasses



All DirectX games have lighting, depth, models that are already created in a 3D world

As the data is processed by the GPU, our driver dynamically renders each frame twice.

You can then view the game in 3D. Since conversion happens in real-time, there is even dynamic 3D depth adjustment.

Programming for 3D vision driver

- NVIDIA 3D Vision driver automatically renders two camera positions for any DirectX game or application and manage the stereo images
  - Nothing to do !
- More control on 3D vision driver behavior with NvAPI
  - Activation, separation & convergence control
- And Heuristics defined in game profile
  - That’s when we work together to fine tune the 3D vision driver behavior for your game
- Game developer writes their game as usual, but ensure all effects will work in 3D as much as possible
  - Following best practices

3D vision driver basic Behavior

What	How	When*
Create Stereo surface	Duplicate textures on first usage	• If texture is render target • And not square • If destination of a copy from a stereo surface
Do Stereo Drawcall	Duplicate drawcall Swap stereo resource sides in between drawcalls	• If render target is stereo
Apply Stereo Projection	Shift SV_Position.x at the end of the Vertex shader	• If drawcall is stereo • And If heuristic "Cutoff" is on: if SV_Position.w != 1

* Very basic behavior, much more cases are available through Heuristics

Why use 3D Vision driver ?

- Much more easy than doing it yourself
  - Requires to fix only a few remaining issues with our help
- Much better performances right out of the box
  - Stereo driver works within the low level NVIDIA driver of directX avoiding a lot of the runtime cost
- Marketing and ecosystem support from NVIDIA

3D Vision driver - Developer Resource

- NVIDIA 3D Vision Developer Website

  http://developer.nvidia.com/object/3d_stereo_dev.html
- Developers Conference Presentations
  - Best practices guide
  - GDC, NVISION & Siggraph Presentations
  - New samples coming…
- NvAPI
  - http://developer.nvidia.com/object/nvapi.html

Starcraft II in Stereo

GDC 2011
NVIDIA SPONSORED SESSIONS

It Sucks.

Steroscopic support in StarCraft II

- StarCraft II was initially shipped without any code or design geared for stereoscopic support
- Official support for stereoscopic stereo was added in 1.2.0 patch
- Process to get all the kinks worked out was on the order of 3 weeks fixing issues, designing UI for it, etc.
- Although fairly straightforward to deal with, stereoscopic is still in its infancy in games and a lot of the information has to be gathered from closed, fairly undocumented sources
  - That’s why we had Nvidia fly over their rocket scientists to help us out
- Here is our experience so the process can be much easier for you

What worked out of the box

  - 3D objects
  - Billboards
  - Skyboxes
- All the 3D objects must have a coherent depth relative to the scene
- Don’t fake your 3D
- Lighting effects are visible in 3D so should be computed correctly
  - Highlight and specular are probably best looking evaluated with mono eye origin
  - Reflection and refraction should be evaluated with stereo eyes

What needed fixing
#1- 2D UI

2D UI

- With no stereo projection (or depth zero)
  - Head Up Display interface
  - Menus
- At the correct depth when interacting with the 3D scene
  - Labels or billboards in the scene
- The w component coming out of vertex shader must not be 1

2D to 3D conversion

shader function

```
float4 2Dto3DclipPosition(
    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 relevant if the depth buffer is not used
                // If needed Z = ( depth * f - nf )/(f - n);
                // ( For DirectX )

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

What needed fixing
#2 - Selection, Pointing in 3D

- Selection UI or cursors interacting with the 3D scene don't work if drawn mono
  - Mouse cursor at the pointed object's depth
    Can not use the HW cursor
  - Crosshair
- Need to modify the projection to take into account depth of pointed elements
  - Draw the UI as a 2D element at the depth of the scene where pointed
  - Compute the depth from the graphics engine (can also be computed from depth buffer)

What needed fixing
#3 - Issues with render targets

- Render targets may or may not be sampled differently for each eye depending on context
- Driver currently relies on heuristics based on render target size to determine what to do
- Special registry keys give hints to heuristics for proper behavior per application
- Heuristics are in flux, so currently must work closely with nVidia to determine the proper registry setup

Render targets - bloom

Render targets

- Bloom render target
  - This is purely a 2D post-process with no depth
  - Hence each eye must sample the same point – bloom render texture must not be doubled to be different for each eye
- The default heuristic assumes render targets smaller than the swap chain are stereo – incorrect for this case
- Fixed by NVIDIA in the application profile

Render targets - Water

Render targets

- Water reflection render target
  - Reflection angle varies per eye
  - In this case, render target *must* be doubled so that reflection is offset for each eye
- Again, fixed by the NVIDIA voodoo registry key

Render targets

- Terrain blending render target
  - Our terrain textures are pre-composited from several layers of texture by rendering the layers to off-screen render targets at run-time
  - Another purely 2D operation that has no relationship with the actual scene viewpoint
  - Heuristics assume square render targets are stereo
  - We got stuck – registry voodoo state could not be changed without breaking previous fixes
  - What to do?
    - Use a 2048x2047 render target when using stereo

Render targets

- Future direction
  - Definitely the most counter-intuitive part of fixing up issues for stereo
  - For now, you can at least be aware of the problem and coordinate with NVIDIA
  - In the future, we hope a standard API will be available to give proper direct instructions to the driver on which render targets need to be doubled for stereoscopic support

What needed fixing
#4 - Portraits

- Different 3D scenes rendered in the same frame using different scales
  - Portrait viewport of selected character
- Since the scale is different for each scene, a different convergence must be used to render each scene
- We change the convergence mid frame to render the portrait panel

What needed fixing but didn’t because NVIDIA wouldn’t bribe us

- Ok, we just ran out of time

- StarCraft II’s “story mode” presented other challenges that we didn’t have time to deal with
- Currently StarCraft II just turns off stereo when entering story mode as there are still remaining issues with this mode

Deferred lighting

Deferred lighting

- Deferred rendering must reconstruct a world position when doing the deferred pass
- This reconstructed position must be offset for each eye to be correct
- This is the “unprojection” problem mentioned by Samuel
- Must use special stereo texture that the driver will sample differently for each eye
  - Texture is split in two halves – driver will sample left half for left eye and vice versa
  - Use this texture to find which eye is being sampled in the pixel shader
    - Left half of texture contains -1, right half contains 1
  - See NVIDIA developer website for details

Points of focus within cinematic scenes

- Our story mode has a wide variance in the depth range of camera shots (micro and macro shots) along with varying points of focus
- Accordingly, ideally each camera shot would need a different convergence values for best effect
- Some heuristic sampling of the depth of the scene could be used to dynamically find a convergence value
- Convergence values must be set so that storytelling points of focus within the scene are at natural convergence points
  - More art than science
- The best stereoscopic games are likely to have artists manually find the best convergence value for each shot

Conclusion

It Doesn't Suck.

Game Demo

Questions

Presentation will be available after the show at http://developer.nvidia.com

Ping us for any question at sgateau@nvidia.com