Practical 3D Vision in Games
原理、実装、最適化

NVIDIA　　風間 隆行

NVIDIA

概要

- NVIDIA 3D Vision™
  - 立体視ドライバーとハードウェアの表示ソリューション
- 立体視の基礎
  - 定義と式
- 3D Visionでの描画
  - 何をどのようにステレオモードで描画するか
- 問題と解決方法
  - 実際のゲームであり得る問題とその解法

何が出来るの？どう動くの？

NVIDIA® 3D VISION™

3D Movies

3D Pictures

3D Games

3D Webcast

多角的な展開

GPUのプログラマブルな性質が、あらゆる3Dデータリソースを利用し立体視を様々な3D Readyディスプレイ上で表示することを可能にします

120 Hz LCDs

3D DLP HDTVs

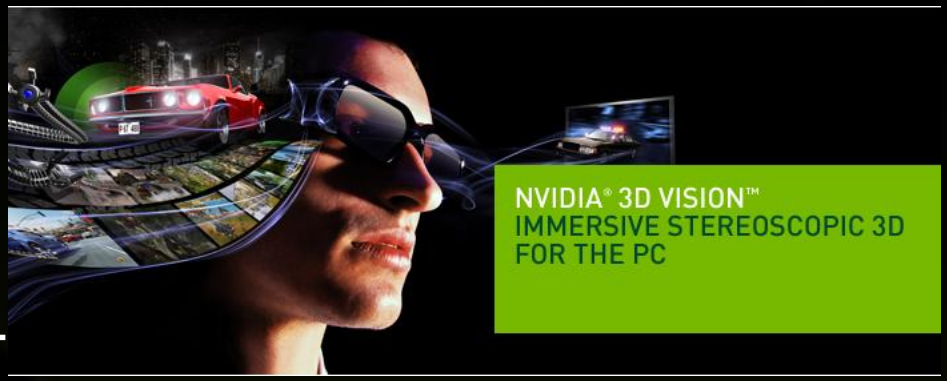
3D Projectors

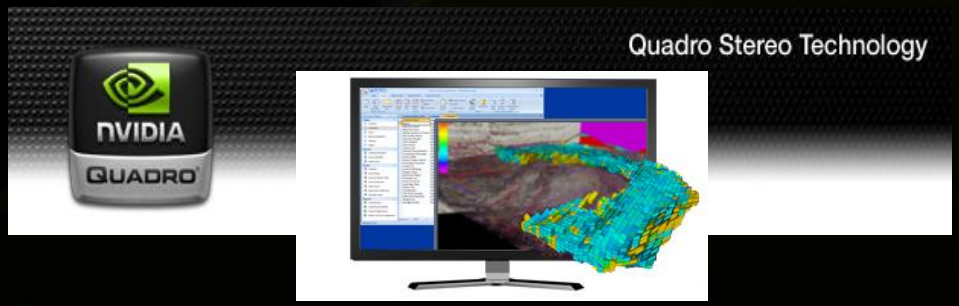
Anaglyph 3D

NVIDIA 3D Vision

立体視のサポート状況

- GeForce
  - ステレオドライバー
    - Vista & Win7
    - D3D9 / D3D10 / D3D11
- 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

どのように動くのか

3D game data はステレオドライバーに送られます

ドライバーは受け取った3D game dataを2回、左目用と右目用にレンダリングします

Left Eye view

Right Eye view

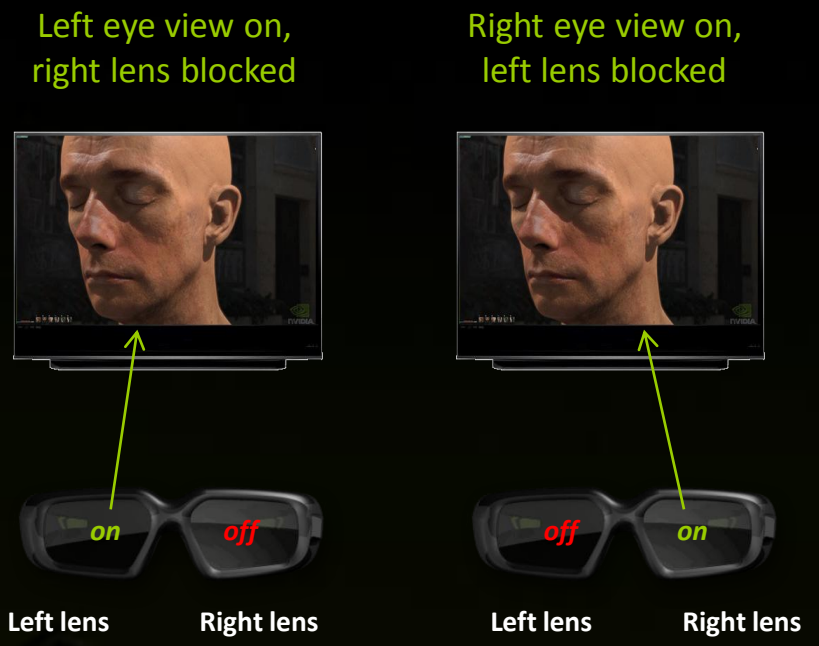
ステレオドライバーは左目を偶数フレームで、右目を奇数フレームで表示します

NVIDIA 3D Vision

どのように動くのか (続き.)

アクティブシャッターグラスは、左目用の絵を表示しているとき右目をブラックアウトさせ、また逆に右目用の絵を表示しているときには左目をブラックアウトします

これはすなわち、各々の目に対してディスプレイはあたかも半分のリフレッシューレートで動作しているかのようになるという事です (例えば120Hzのディスプレイなら各目では60Hz動作に等しくなります)



ユーザーは結果のイメージから、奥行き感を感じることができます

どのように動くのか (続き. 2)

- 3D Visionは透過的にゲームに導入できます
  - ゲームエンジンはステレオドライバーの事を知る必要はありません
  - ドライバーは自動的に左右の画像を作り出します
  - ほんの僅かな（もしくは全く無し）プログラミングが必要なだけです
- ステレオパラメータをコントロールする方法もサポートされています
  - 高度な使用やエフェクトの為に
  - コントロールインターフェースはNVAPIで提供されます

NVAPI Stereoscopic Module

- NVAPI は NVIDIAの主要なSDKで、NVIDIA GPUとドライバーへの直接的なアクセスを提供します

- 高度な3D Visionプログラミングの為には
  - NVAPIは現在ドライバーのステレオコントロール用のモジュールを提供しています
  - システムが3D Vision可能かを判別できます
  - ゲームのステレオプロファイル設定を管理できます
  - より良い立体視を実現するためにステレオパラメータを直接コントロールすることができます

- ダウンロードとドキュメントは
  http://developer.nvidia.com/object/nvapi.html

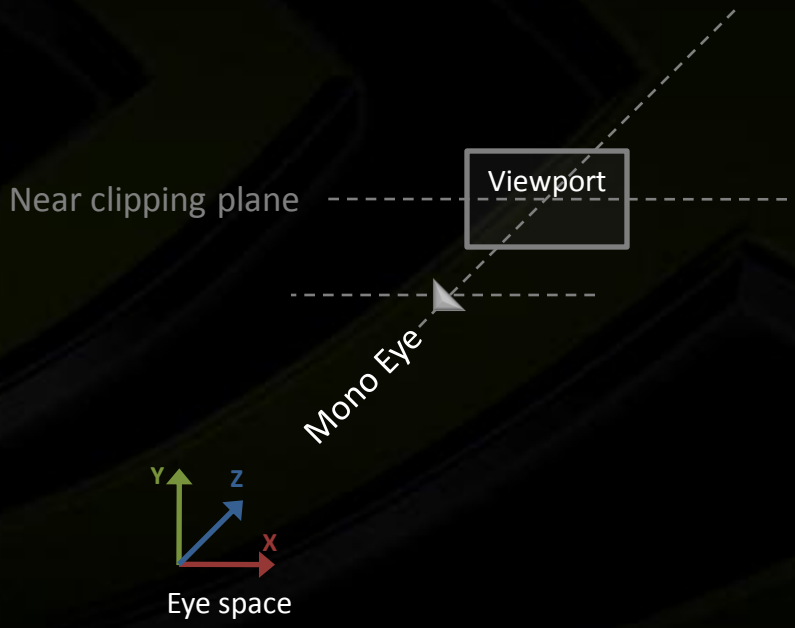
定義と式

立体視の基本

立体視の基本

基本的mono eye(一眼)レンダリング

シーンは一つの仮想的な目で見ることを想定して、ビューポートのnear clipping plane に射影された絵をレンダリングします



立体視の基本

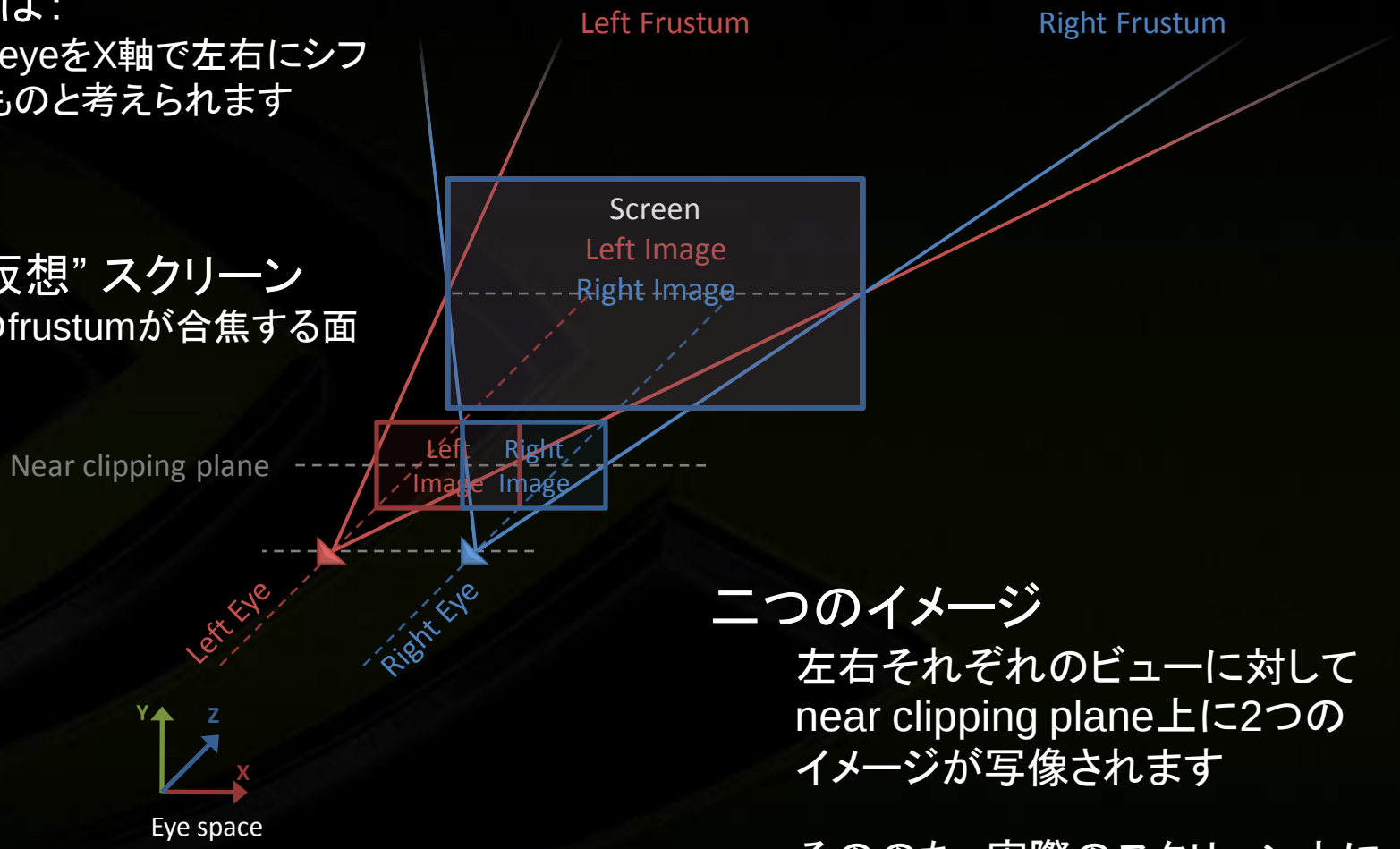
2眼、1スクリーン、2イメージ

左右の眼は：

Mono eyeをX軸で左右にシフトしたものと考えられます

一つの “仮想” スクリーン

左右のfrustumが合焦する面



二つのイメージ

左右それぞれのビューに対してnear clipping plane上に2つのイメージが写像されます

そののち、実際のスクリーン上に左右の眼それぞれの絵が別々に得られます

立体視の基本

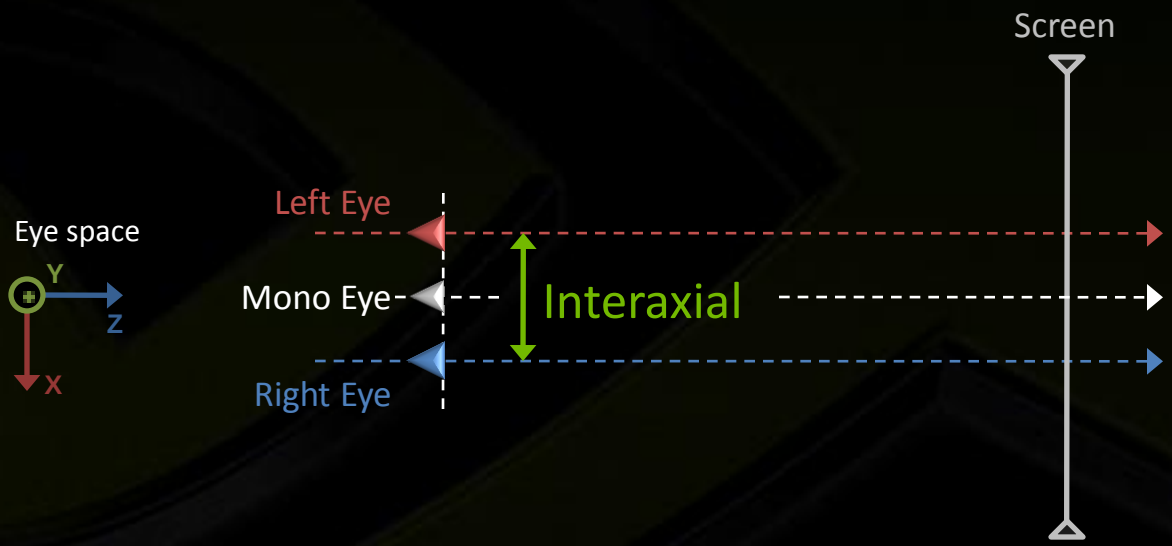
立体視レンダリング

- ジオメトリを2回 左右の目の位置からレンダリングされます
- 3つの修正が基本的なグラフィックスパイプラインに対して行われます
  - stereo surfacesの使用
    - レンダリングサーフェースを複製します
  - stereo drawcalls
    - Drawcallを複製します
  - stereo separationの適用
    - 射影マトリクスを修正します

立体視の基本

定義: Interaxial

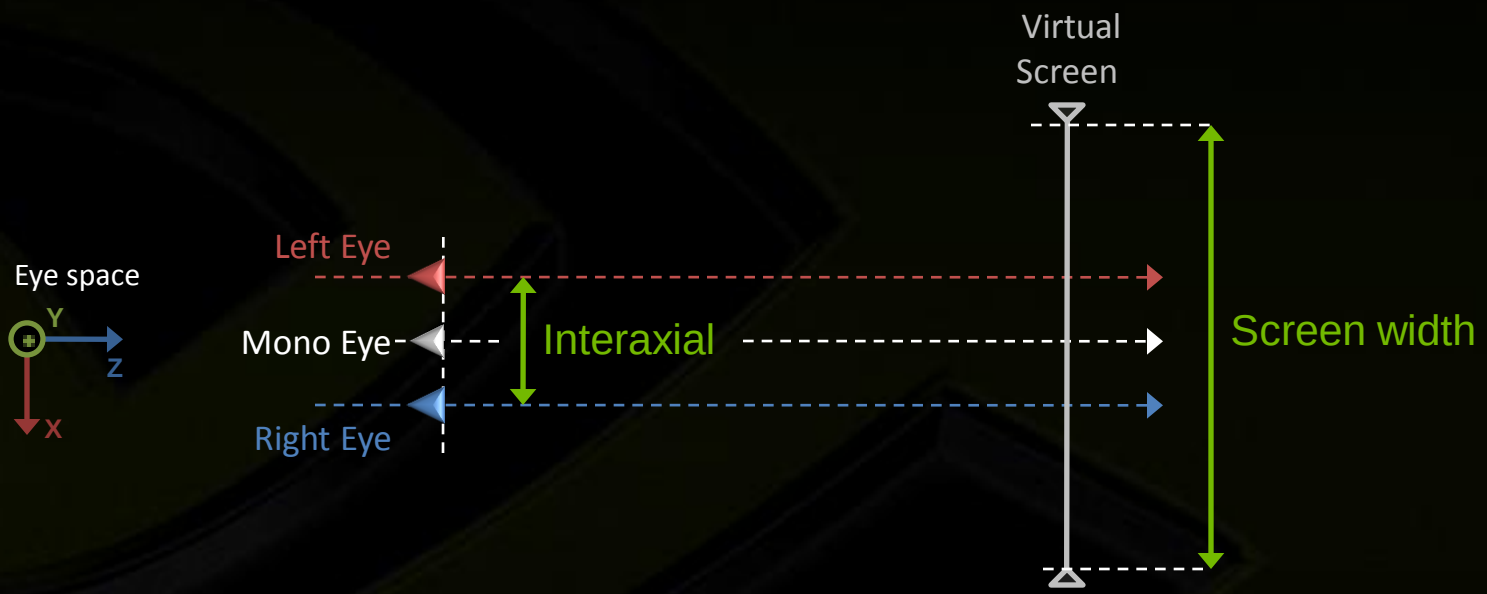
- Interaxialは2つの仮想的な目の距離をeye spaceで表します
- mono eyeもステレオの左右の目も、視線は完全に平行です



立体視の基本

定義: Separation

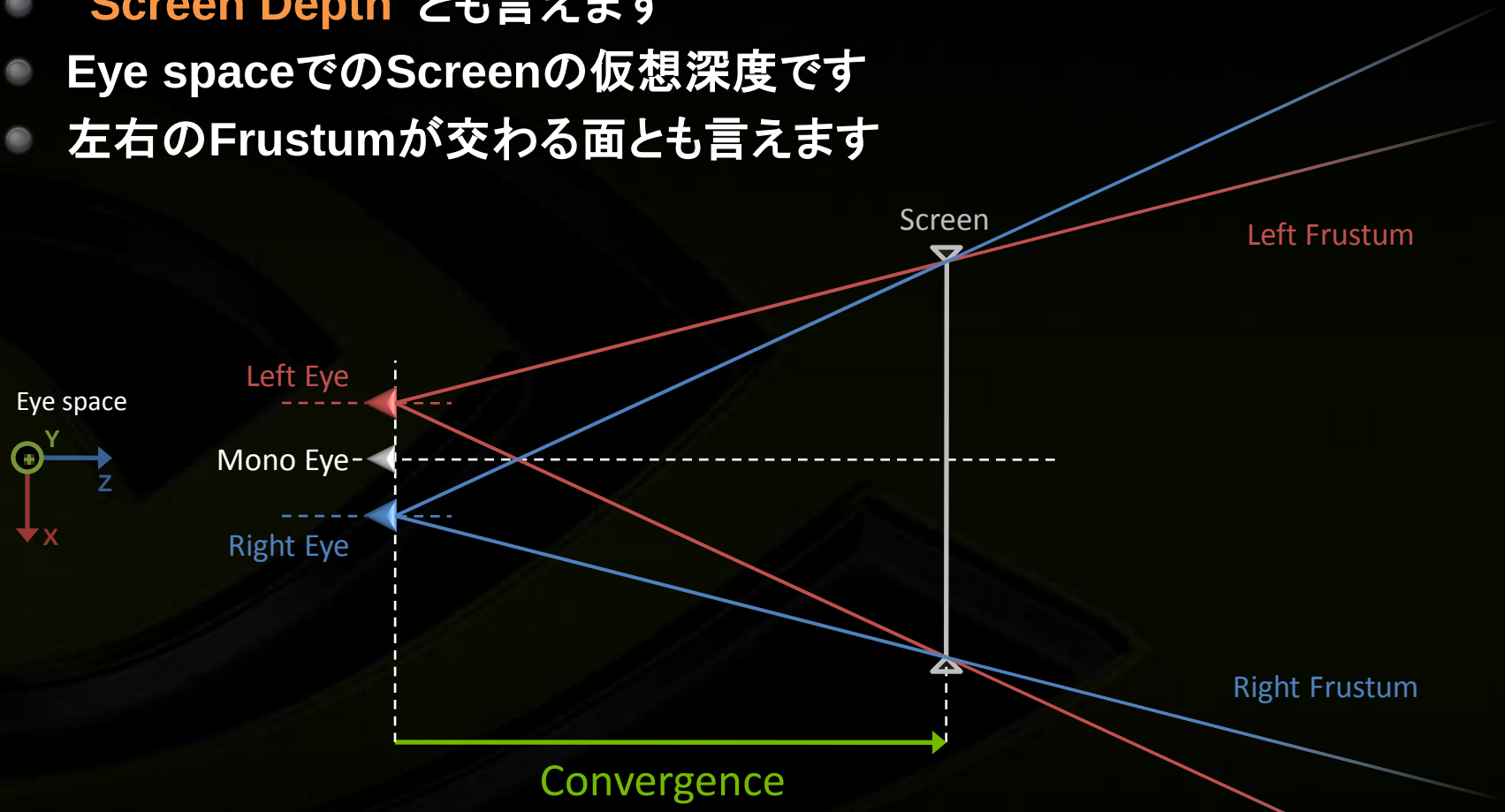
- 仮想スクリーンの幅で正規化したinteraxialのを表します
  Separation = Interaxial / ScreenWidth



立体視の基礎

定義: Convergence

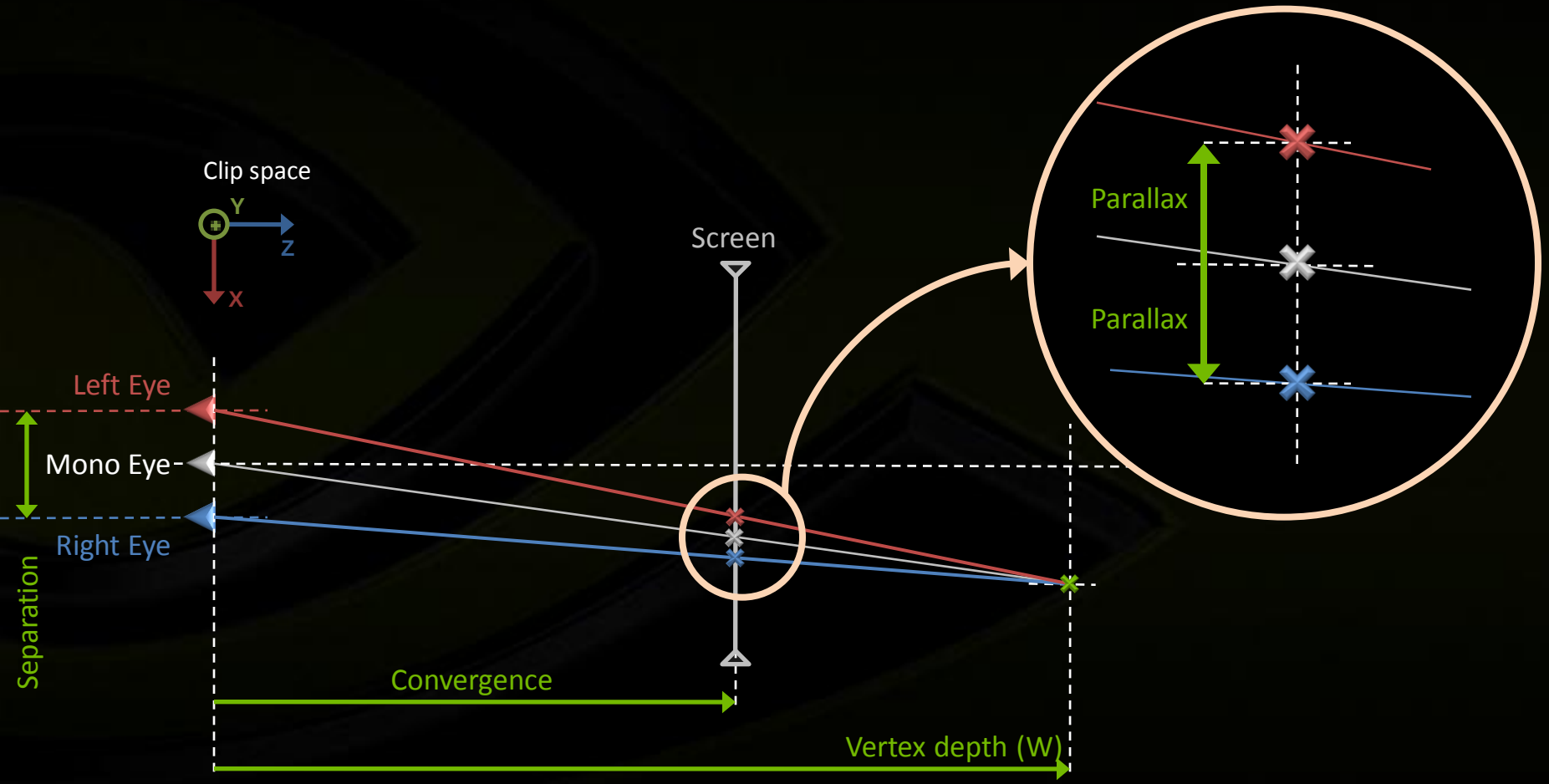
- “Screen Depth”とも言えます
- Eye spaceでのScreenの仮想深度です
- 左右のFrustumが交わる面とも言えます



立体視の基礎

定義: Parallax（視差）

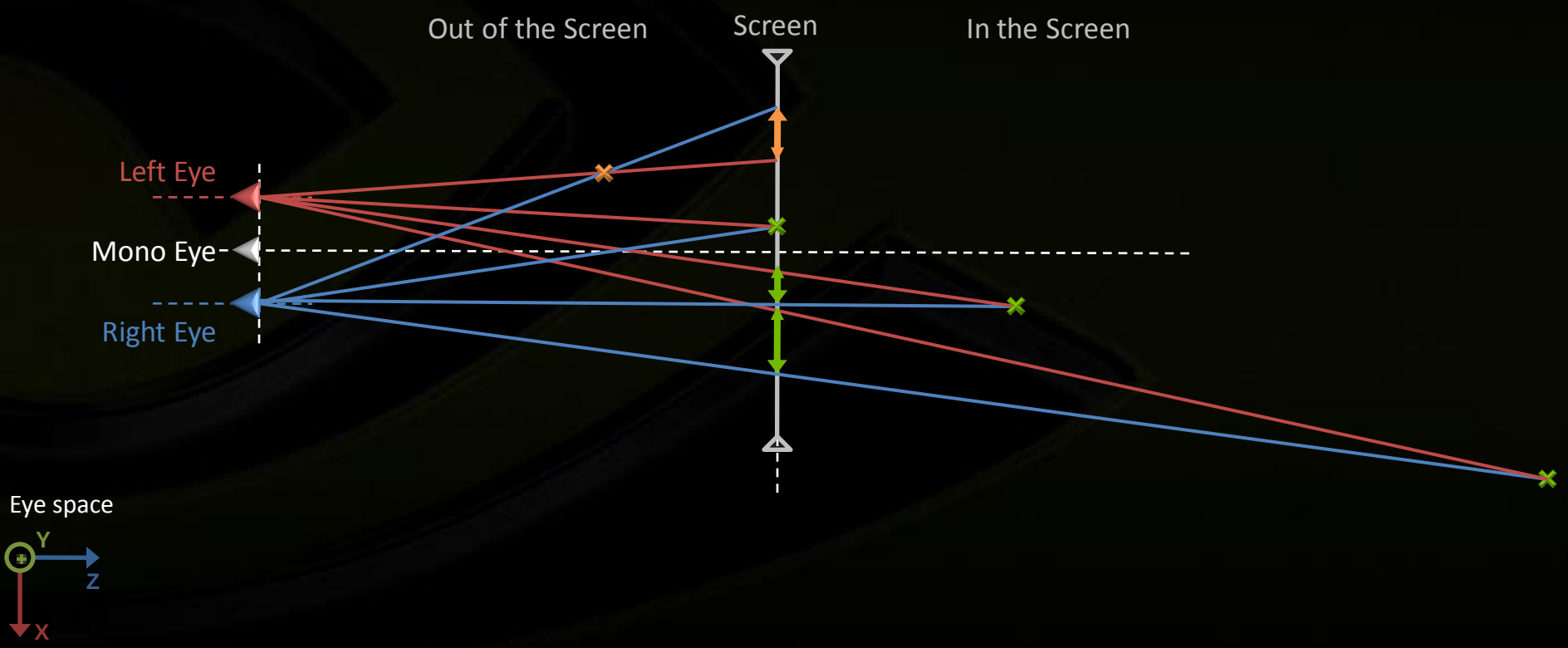
- 頂点のスクリーン上への写像位置の符号付き距離の事です
  - Parallaxはeye spaceで頂点深度の関数として表せます
    Parallax = Separation * (1 – Convergence / W)



立体視の基礎

In / Out of the Screen

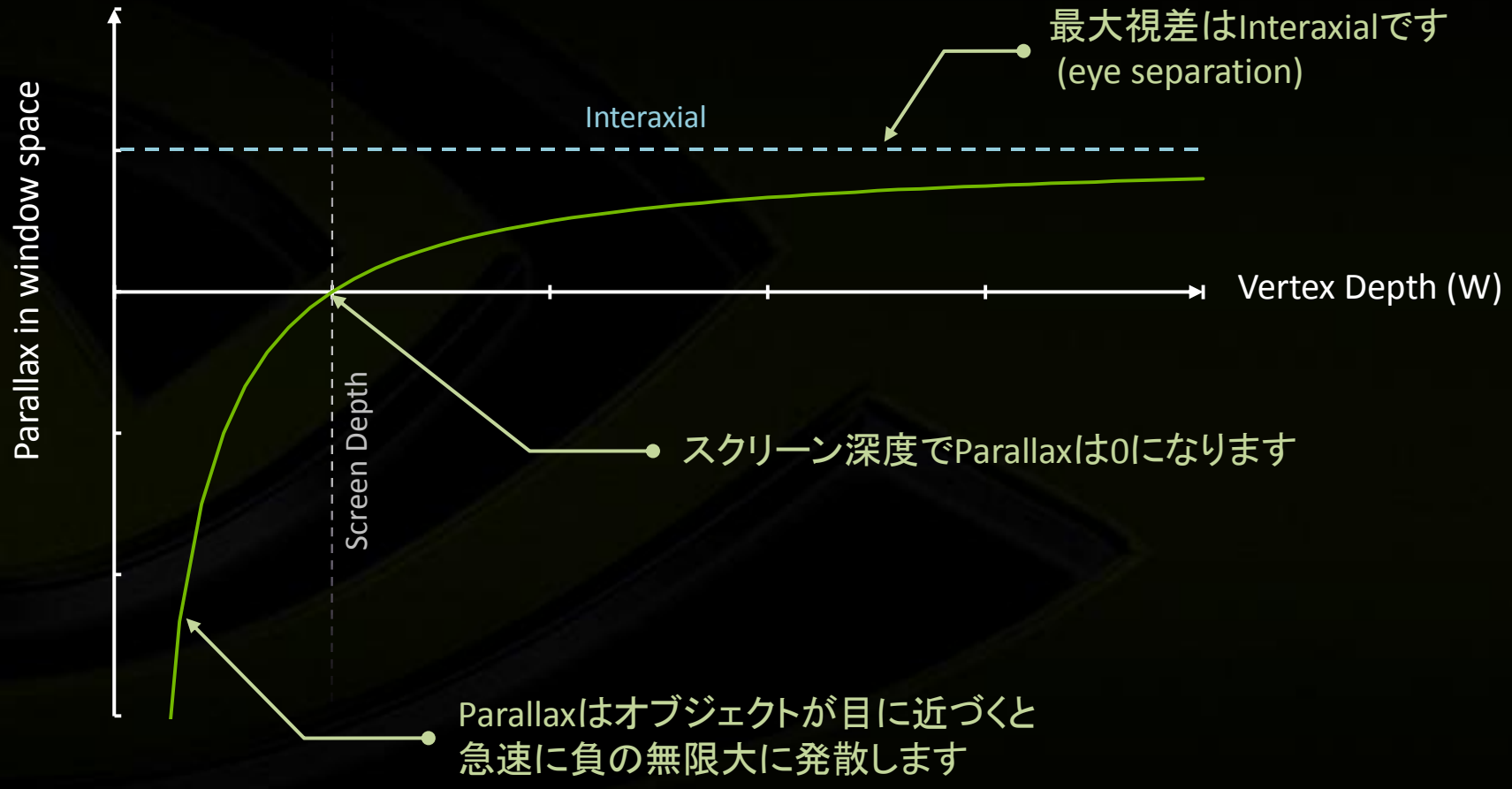
- 視差はスクリーンに対しての奥行き感を生み出します
- Parallaxが負数の場合、頂点は Out of the screen（飛び出し）に見えます



立体視の基礎

視差方程式

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



定義と式のまとめ

- Interaxial と Separation
  - EyeSpaceでの二つの目の仮想距離と正規化された距離です
- Convergence
  - Eye spaceでのスクリーン深度です
- Parallax
  - Eye spaceにおいての、頂点のスクリーンへの射影位置の符号付き距離です
  - Parallax = Separation * (1 – Convergence / W)
- In Screen と Out of Screen
  - In screen: スクリーンと far clip planeの間
    Parallax > 0, W > 1
  - Out of screen: 目とスクリーンの間
    Parallax < 0, 0 < W < 1

何があるのか？どうしたらより良くできるのか？

3D Visionでの描画

3D Visionでの描画

ドライバーマジック

- stereo driverは何をするの？
  - 自動的に立体視エフェクトをゲームに追加しようとします

1. Render Targetを複製します
   ステレオレンダリングの為の左右の目用サーフェース
2. Draw callを複製します
   全てのdraw callは2度実行されます
3. stereo separationを適用します
   Vertex shaderの後で左および右へのX座標のシフトを行ないます

サーフェースの複製

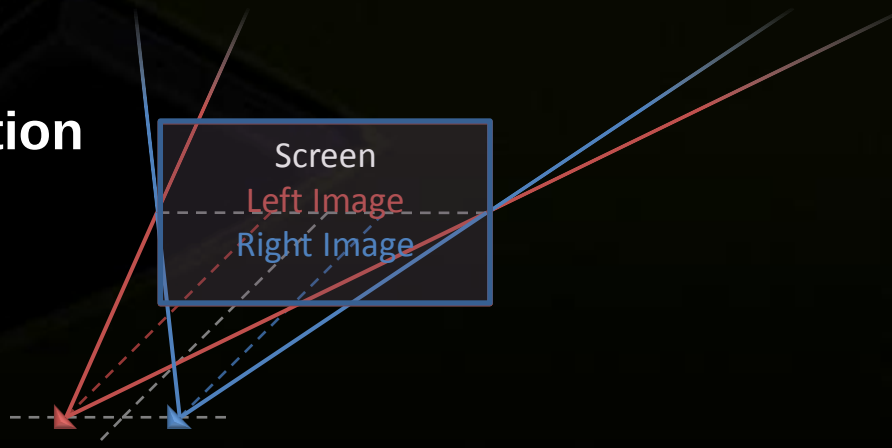
- 推測に基づいたドライバーの自動的複製
  - ゲームエンジンからは透過的
  - 推測はサーフェースサイズに依存
    - バックバッファと同じか大きい サーフェースは複製する
    - 正方形サーフェースは複製 しない
    - 小さなサーフェースも複製 しない
- 明示的に複製または抑制する方法もあります
  - ゲームエンジンは完全に複製をコントロールできます
  - NVAPIが必要なAPIを提供します
- 次に自動複製の詳細をみてみましょう

3D Visionでの描画

Stereo Rendering Surfaces

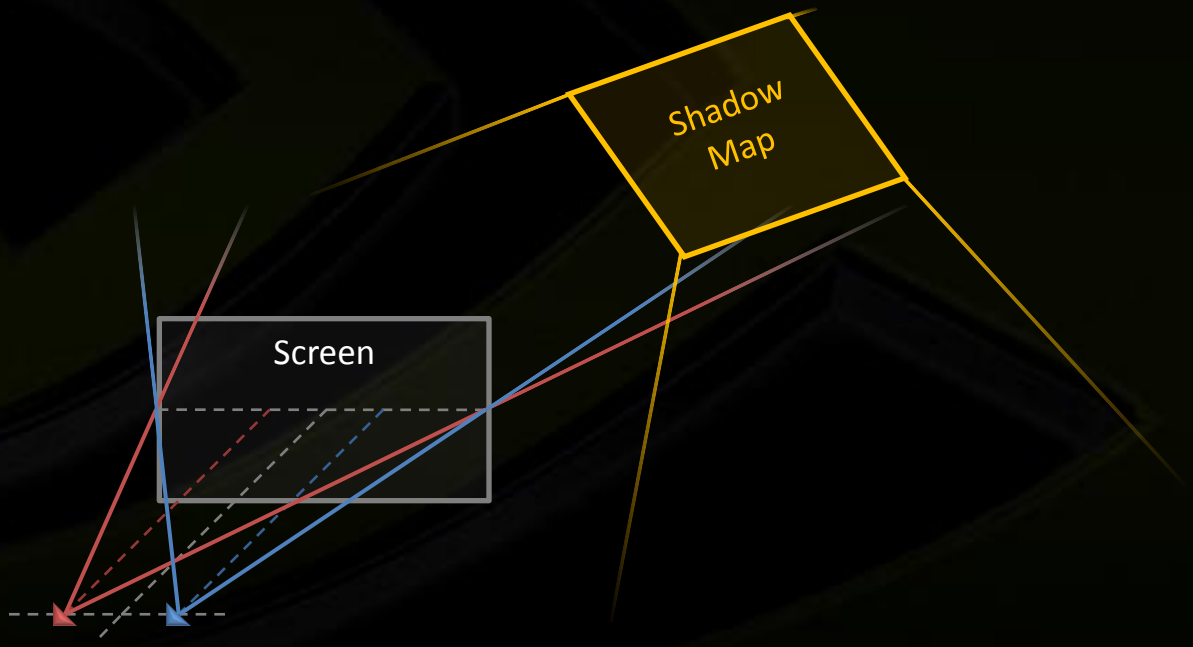
- ビューに依存したレンダーターゲットは複製される必要があります
  - バックバッファ
  - Depth Stencilバッファ

- full screenサイズのPost Effect用のrender targetもあります
  - HDR, blur, bloom, DOF
  - SSAO
  - Screen space shadow projection



Mono Rendering Surfaces

- ビュー非依存 レンダーターゲットは複製される必要がありません
  - Shadow map
  - Spot light textures



3D Vision での描画

Stereo or Mono: More Case Studies

Use Case	Surface Type	Stereo Projection	Stereo Drawcalls
Shadow map	Mono	No Use Shadow projection	Draw once
Main frame あらゆる Forward パス	Stereo	Yes	Draw twice
Reflection map	Stereo	Yes Generate a stereo reflection projection	Draw twice
ポストエフェクト (フルスクリーンの描画)	Stereo	No No Projection needed at all	Draw twice
deferred shadingでの Lighting (Drawing a light sprite)	Stereo G-buffers	Yes 注意：Stereo Unprojection が必要	Draw twice

3D Visionでの描画

Draw Callの複製

- NVIDIA stereo driverで (自動)
  - stereo surfacesに対しては 全てのdraw callは左右サーフェース用に2度発行されます
  - mono surfacesは変化ありません
- Driverでの擬似コード

```
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();
}
```

Stereo Seperationの適用

- 自動で行う場合
  - Vertex シェーダにドライバーは視差シフトコードを追加します
  - Vertex出力は次のように変更されます:

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

Scale: ユーザーが変更可能な視差パラメータ

- 明示的に行う場合
  - ゲームエンジンがVertex Shaderでseparationを適用します
  - パラメータの取得

Scale: NvAPI_Stereo_GetSeparation()
Separation: NvAPI_Stereo_GetEyeSeparation()
Convergence: NvAPI_Stereo_GetConvergence()
EyeSign: -1 for left eye, +1 for right eye

3D Objects

- あるフレームでの全ての3D objectは一つのperspective projectionで描画しなくてはなりません
- 全ての3D objectはシーンに対して一貫した相対深度を持たなくてはなりません
- ほとんどのライティングはシェーダの変更を必要としません
  - ビュー依存のライティング、例えばハイライトや スペキュラーは 大概mono eyeでの計算をそのまま使えるはずです
- Reflectionとrefraction は(厳密には)Stereo的に描画されないといけないことになるでしょう

擬似3D Objects: 空、ビルボード…

- **Sky box** はそれなりに一貫性のある深度値を持って描画されなくてはならないでしょう
  - Stereo化されなくてはいけません
  - 一番良いのは視差が最大になるような非常に遠くの距離です
  - 当然スクリーン全体を覆う必要があります
- **Billboard** (particles, leaves)
  - あまり良い結果を生み出しません
- **Relief mapping系**
  - Parallax occlusion map, steep map, 等
  - ピクセルシェーダで何らかの操作が必要になるでしょう

（後述します）

複数のビューポートでの3D Objects

- ゲームによってはスクリーン上に複数のシーンを描画するかもしれません、例えば
  - 小さなviewportでキャラクターのポートレート表示
  - 複数対戦用に分割されたスクリーン
- 各々のビューポートは、それぞれ固有の convergenceが必要です
  - ゲームエンジンは各ビューポートを管理する必要があります
  - NVAPIの NvAPI_Stereo_SetConvergence() を使う必要があります

3D Objects: 飛び出しエフェクト

- ユーザーの脳は飛び出しエフェクトに対して多大な努力をすることになります（目の疲れ、酔いを引き起こす）
  - 説得力のあるエフェクトを提供するには、注意深く扱わなくてはなりません
- モニターのエッジでclipされるObjectの飛び出し
  - すごく変に見えます
- 目がなれる時間を提供するために、inside screenからoutside screenへのオブジェクトの移動はゆっくり行ってください
  - スムーズな飛び出しエフェクトへの移行
- リアリスティックなレンダリング

2D Objects

- オーバーレイされるような2D オブジェクトは正しくdepth値を持たなくてはなりません
  - 2Dオブジェクトでも正しいW値を設定してください
- 3Dシーンに影響しないためには
  - HUD, UI elements ： スクリーンに置く
    Monoに見えるように W = 1.0 になるようにします
- 3D sceneに影響させたい場合は
  - マウスカーソルはオブジェクトのdepthに描画してください
    HWカーソルを使ってはいけません
  - Crosshair、シーン内のラベル等
  - W をeye spaceでのオブジェクトdepthに設定してください

2D Objects: 正しいDepthの設定法

```
float  depth;

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

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

    return Output;
}
```

- **depth = 1.0で視差無しになります**

どうやって直す・・・？

問題とその解決法

問題と解決法

多くの問題

- ここ3年ぐらいで、多くの立体視の問題に直面しました
  - Crosshair、カーソル、objectセレクション
  - ゲーム内でのStereo VideoやImageのReplay
  - Frustum culling
  - Deferred shading
  - ビュー依存のlighting/textureエフェクト
  - full screen modeでのIME
  - 眼精疲労と乗り物酔い
- ドライバーによる自動ステレオ化はたいていのゲームに対して正しく働くのですが、
  - 少なくとも2D objectは、気にして扱わないといけません

問題と解決法

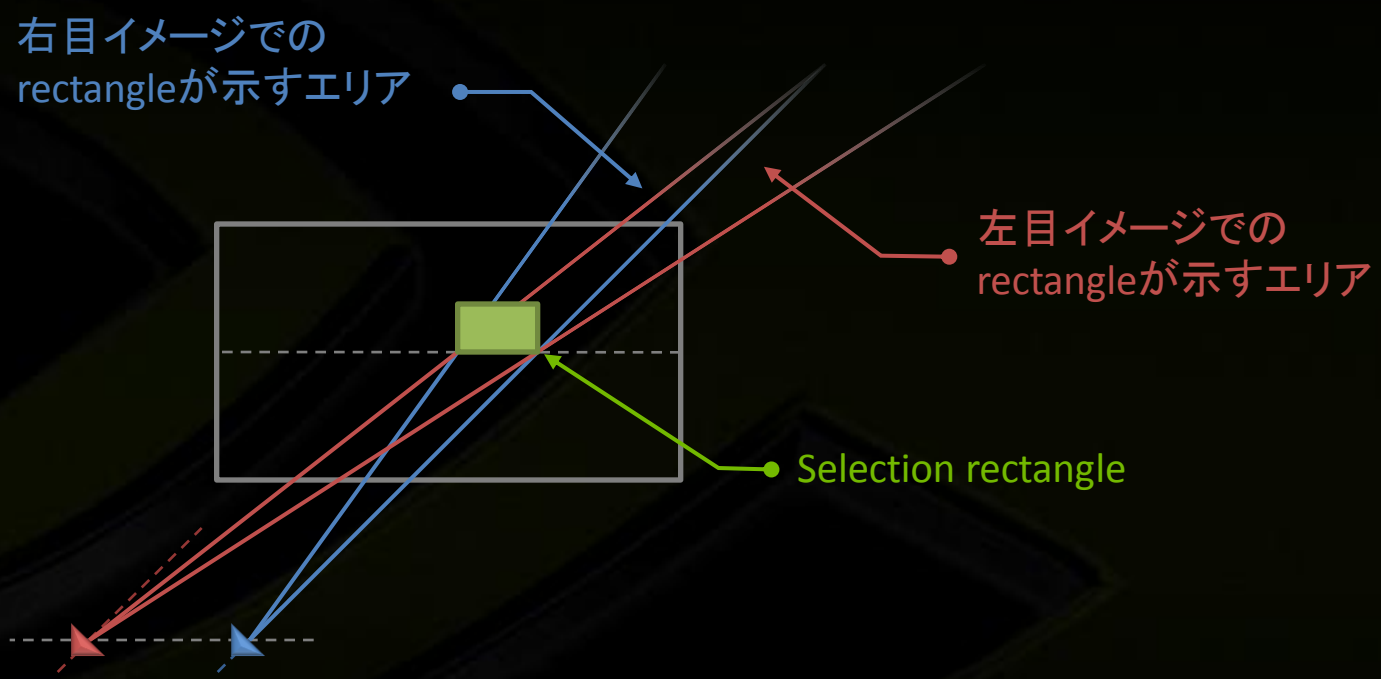
Crosshairとカーソル

- 実際の世界では、両目で物を狙ったりはしません
- ステレオモードでCrosshairはどこにおいたら・・・？
  - 次善策: crosshairをシーン深度においてしまう
  - リアルには感じれないかもしれないが、“正しく”感じられます
- ハードウェアカーソルには深度がありません
  - カーソルでオブジェクトをピックするようなゲームは、ゲームエンジンでカーソルを管理して正しいdepthに描画することを考えなくてはなりません

問題と解決法

セレクション

- Monoではセレクションは単純ですが、ステレオはちょっと考えなくてはなりません
  - スクリーン上で同じ位置でも、実際の位置は左右の目で違う



- 正確なセレクションはrectangleによるvolume castを使わないといけないでしょう:
  http://developer.download.nvidia.com/presentations/2009/GDC/GDC09-3DVision-The_In_and_Out.pdf

問題と解決法

ステレオのビデオやイメージの表示

- pre-rendered movieは多くのゲームで使われます
- 3D Visionですでにあるステレオコンテンツを再生したい場合もあるでしょう
  - ステレオムービーの再生
  - ステレオフォトの表示
  - Prerecorded cut scenes

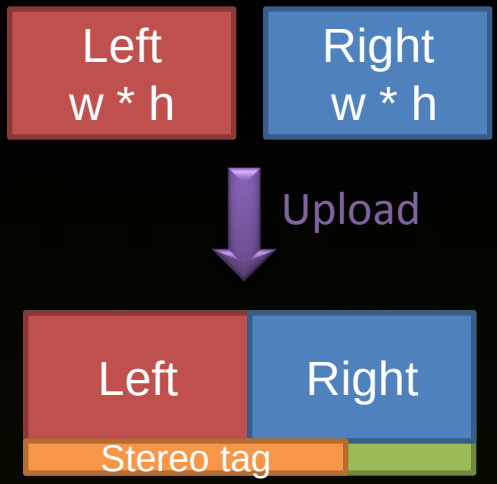
問題と解決法

ステレオのビデオやイメージの表示(続き.)

- stereo textureの紹介

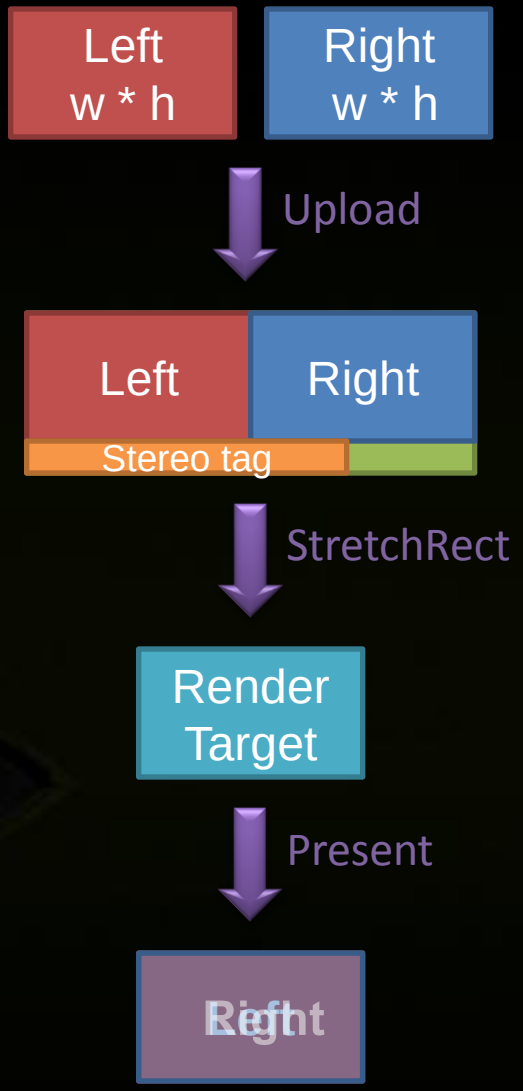
- D3D textureを
  width * 2
  Height + 1
  Left image を左半分に
  Right image を右半分に
  NV3D tag を 追加列に

- Texture作成時にNV3D tagを作ります



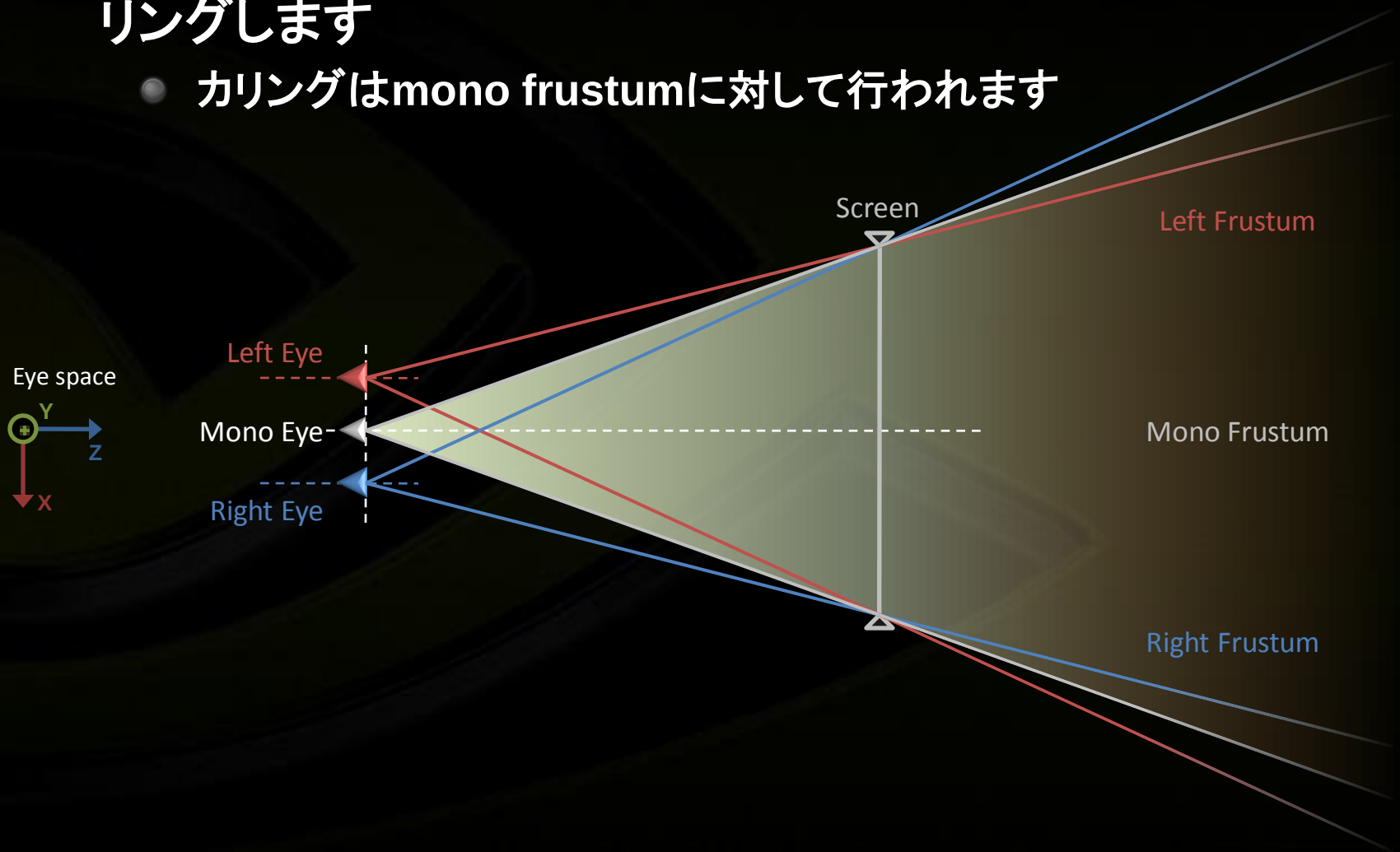
ステレオのビデオやイメージの表示(続き2.)

- ステレオテクスチャーをステレオレンダーターゲットにコピー
  - そうすればドライバーは自動的に
    左半分 -> 左のサーフェース
    右半分 -> 右のサーフェース



Frustum Culling

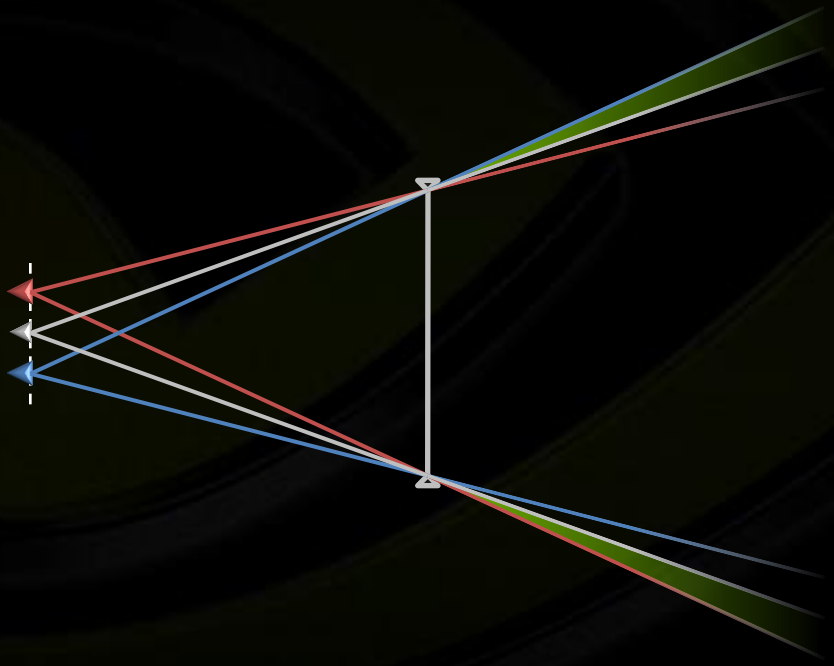
- 多くのゲームエンジンはView Frustum外のオブジェクトをカリングします
  - カリングはmono frustumに対して行われます



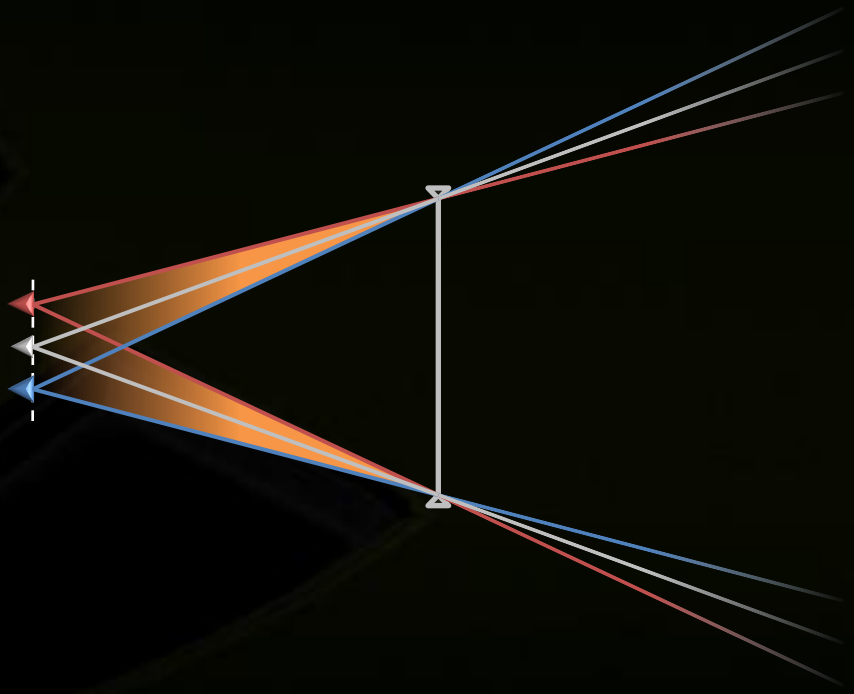
問題と解決法

Frustum Culling (続き.)

失われる可能性のある箇所



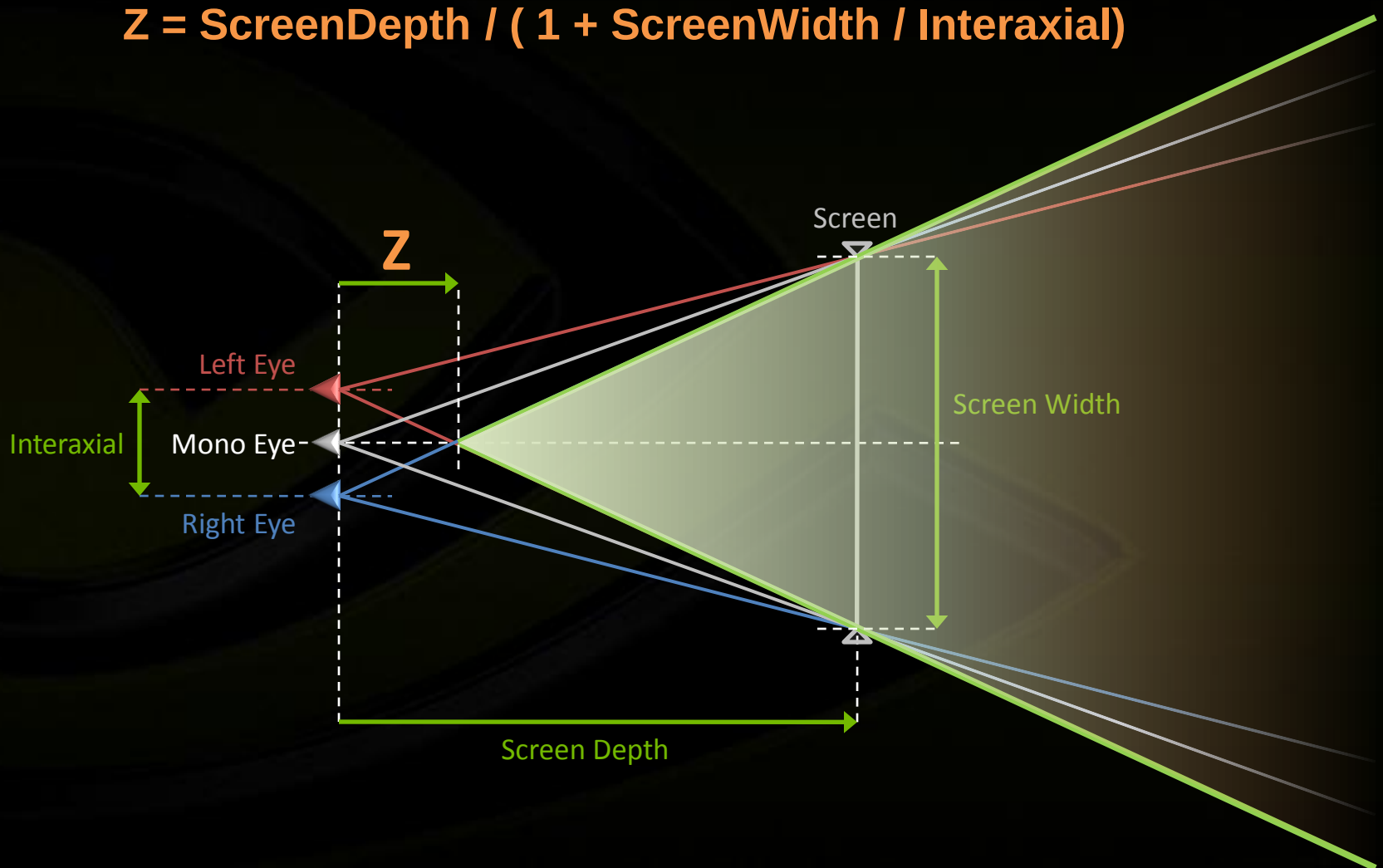
片方の目にしか出てこない
Out of screenの箇所



問題と解決法

Frustum Culling (続き. 2)

- カリングの為の frustumを修正: 複合(compounding) frustum
  Z = ScreenDepth / (1 + ScreenWidth / Interaxial)



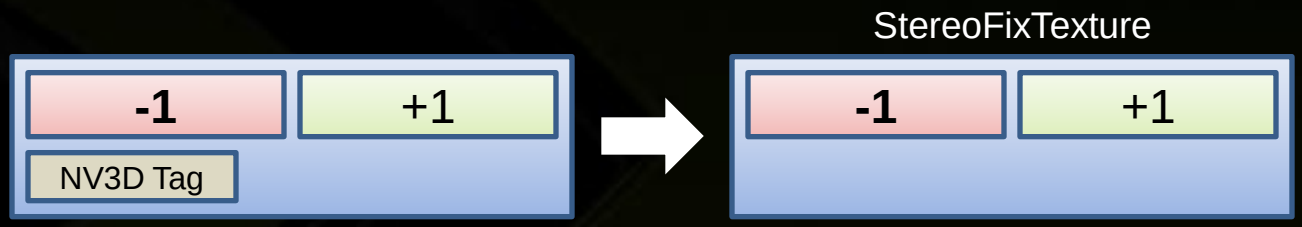
問題と解決法

Deferred Shading

- lightingパスでscreenポジションからworldポジションへの変換があります
  - Stereo化されているとScreenポジションが左右の目の位置で変更されてしまっています！
- Pixelシェーダの明示的な変更が必要です
  - Pixelシェーダに現在それが左目用の描画なのか右目用なのか通知しなくてはなりません
  - Screen<->Worldの変換でparallax shift(視差ずらし)を考慮しなくてはなりません
- 現在のDrawCallが左右どちらかを知るには？
  - Stereo textureで判定可能です

Deferred Shading (続き.)

- まず小さなstereo textureを作ります
  - 左半分を -1.0、右半分を1.0で埋めます
  - これを使えばPixelシェーダは簡単にどちらのdraw callなのかを知ることができます



- 左用draw callなら：Sample(StereoFixTexture) == -1
- 右用draw callなら：Sample(StereoFixTexture) == 1

問題と解法

Deferred Shading (続き. 2)

- Pixelシェーダでの計算

light volumesをstereo screen空間に変換する方法

1. light geometryの視差は
   Parallax = Scale * Separation * (Pos_mono.w – Convergence)
2. parallax shiftを適用します
   Pos_stereo.x = Pos_mono.x + Sample(StereoFixTexture) * Parallax
3. w除算を行って [-1, 1] 空間に正規化します
   Pos_stereo.xy /= Pos.w
4. Pos_stereo.xyを使って SceneDepthを含む G-Bufferからデータを読みます

world 空間での位置の計算方法

1. w除算の逆変換
   Pos_stereo.xy *= SceneDepth
2. そのdepthでの視差を計算
   Parallax = Scale * Separation * (SceneDepth – Convergence)
3. 視差を外します
   Pos_mono.x = Pos_stereo.x – Sample(StereoFixTexture) * Parallax
4. Pos_mono をworld空間に戻します

問題と解決法

ビュー依存のシェーディングエフェクト

- ハイライトとスペキュラー
  - 正確なレンダリングの為には、reflectionベクトルは左目と右目から計算されなくてはなりません
  - しかしながら、mono eyeベクトルでの計算でも知覚できるほどのアーティファクトが出ることはありません
  - pixelシェーダを変える必要はないでしょう
- Relief mapping
  - Parallax occlusion map, steep map, cone map, 等
  - 視線ベクトルは正しい目の位置(左と右)から作られる必要があるでしょう
  - stereo textureのテクニックで左右どちらのdrawcallか確認しましょう

Full Screen ModeでのIME

- 3D Visionはfull screen(exclusive)モードでの動作が必要です
- ところが全てのIMEが正しくexclusiveモードで動作するわけではありません
- Workaround: ユーザーにD3Dコンパチなを使って遊んでもらうことを提案する必要があります

問題と解決法

目の疲れと酔い

- 様々な要因が目の疲れと酔いを引き起こします
  - 非常に大きい、ないしは逆に小さすぎるFoV
  - 蛍光灯などによるフリッカー
  - 正確でないmotion blur
  - Interaxial、convergenceとオブジェクトの位置等が正しくない場合
  - 飛び出し効果のオブジェクトが多すぎる場合

目の間隔（瞳孔間距離）

- Interocular distance: 瞳孔間距離
  - 平均すると成人同交換距離は 6.0cm ~ 6.5cm
  - これは、無限遠にあるオブジェクトの最大視差と等しいです
  - 人間の目は瞳孔間距離よりも近くのオブジェクトをうまく結像することができません

- Interaxial(仮想距離)とinterocular(実際の距離)の関係

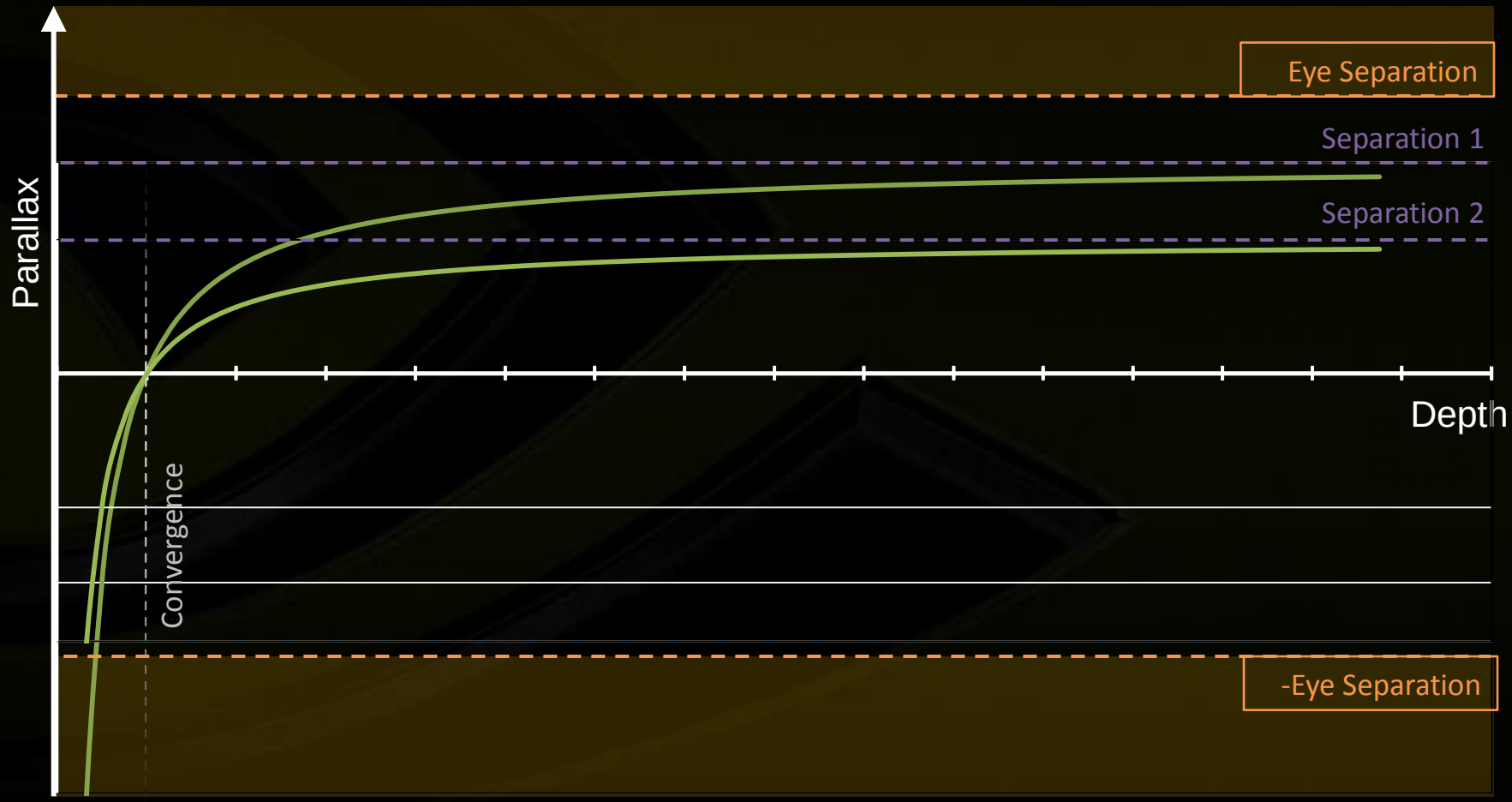
  Interaxial = Interocular / RealScreenWidth

  - どれだけ大きなスクリーンかに依存します。また interaxialはユーザによって大きく違います
  - Interaxialはユーザに調整させるようにしましょう

目の疲れと酔いの軽減

安全なParallaxのレンジ

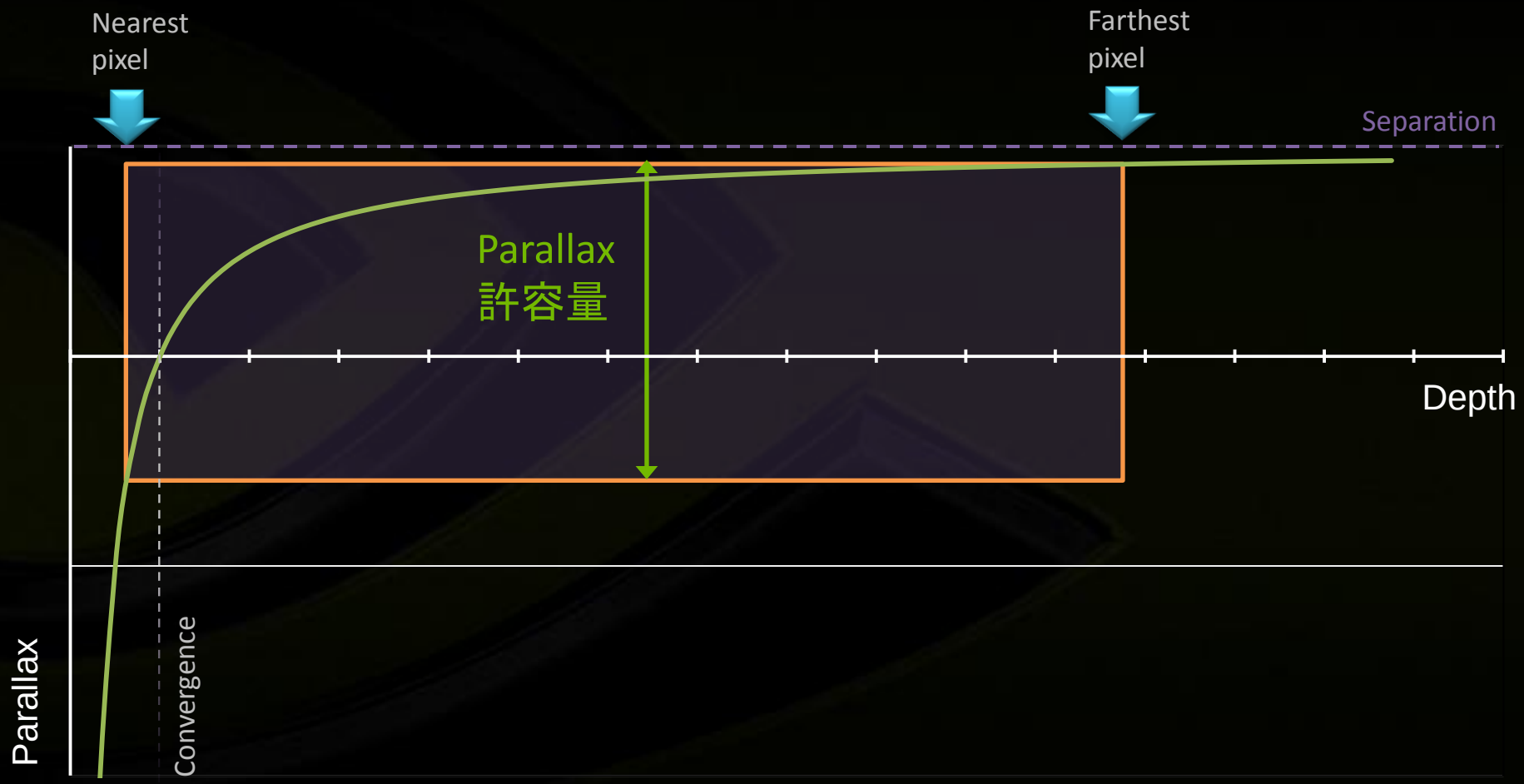
- Interaxialに対して オブジェクトの視差は”安全なレンジ”があります



目の疲れと酔いの軽減

Parallaxの許容量

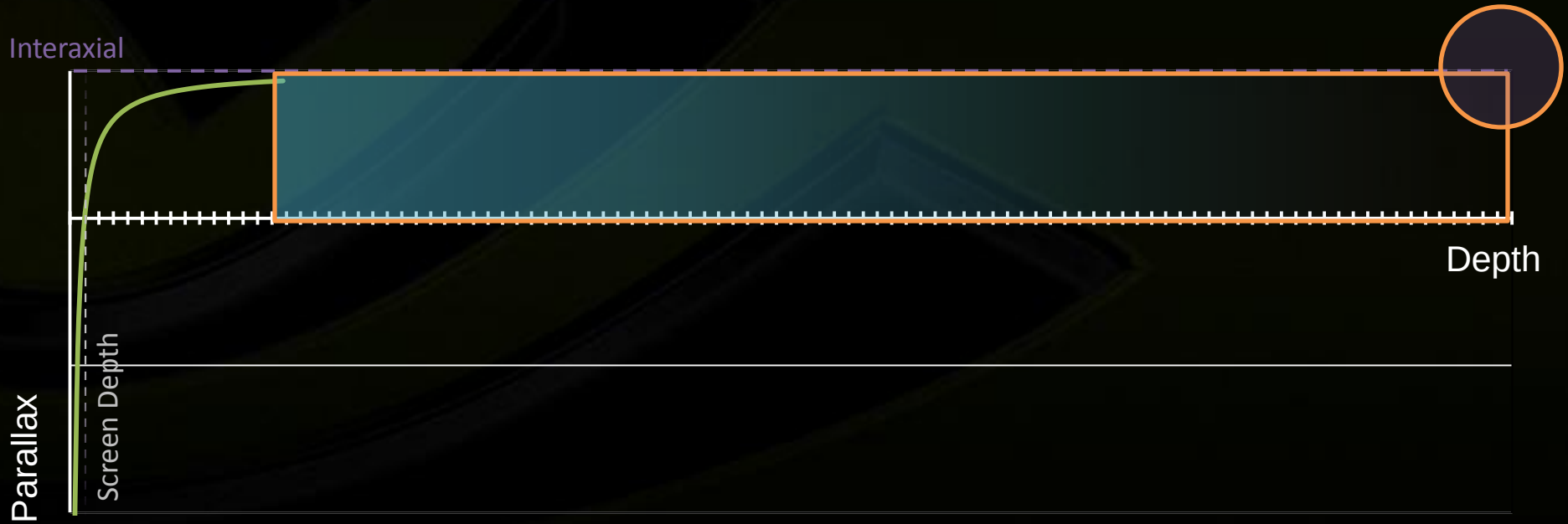
- どれぐらいの視差変化量が許容されるか



目の疲れと酔いの軽減

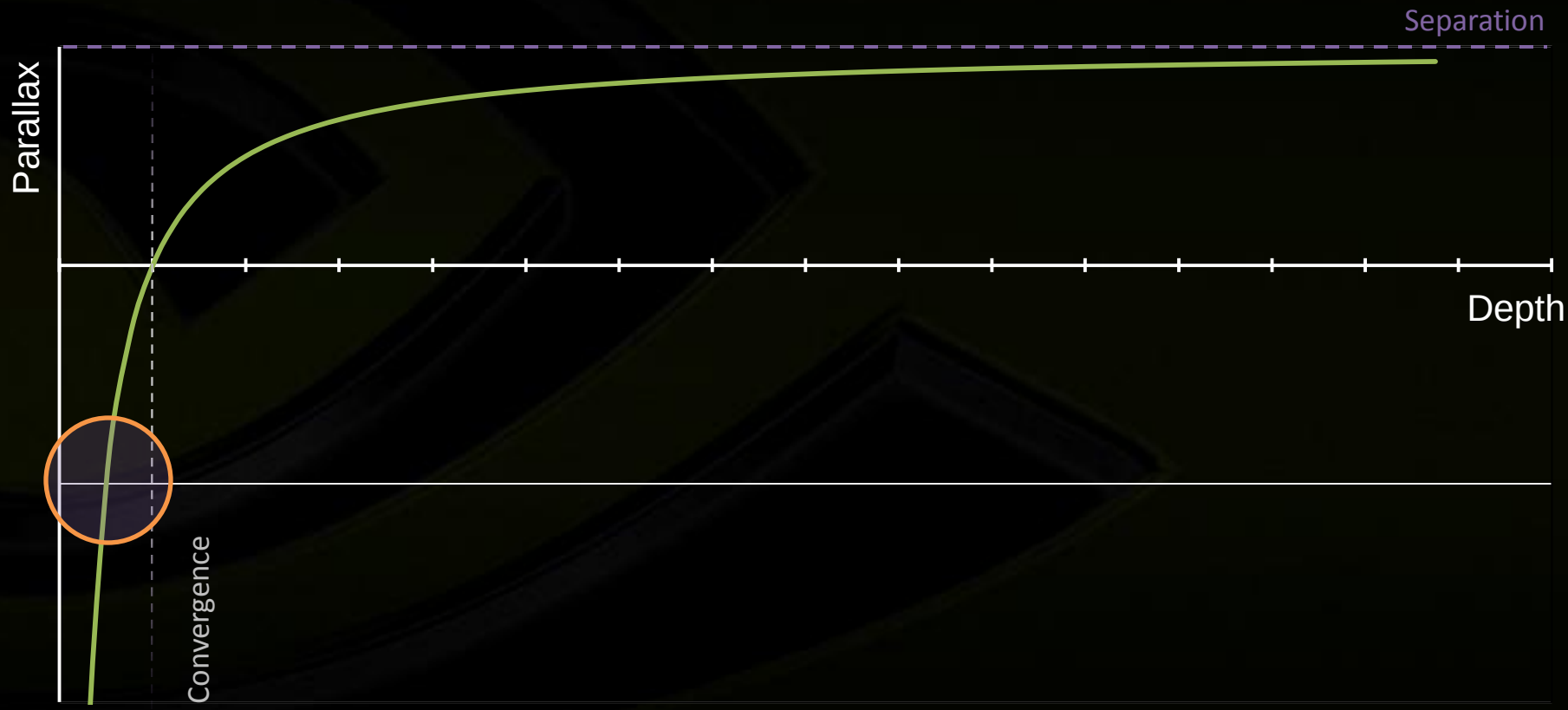
Parallax許容量: 遠いPixel

- 100 * ScreenDepthの位置では Parallaxはinteraxialの99% になる
  - 100 * ScreenDepthよりも遠いエレメントは全てほぼ平らに見えることになります
- 10から100 * ScreenDepthでは Parallaxは 9%だけの変化です
  - このレンジのObjectの奥行き感はわずかになります



Parallax許容量: 近いpixel

- Convergence/2の位置でParallaxは –Separationと等しくなります
  - これ以上は、飛び出しの量が大きくて目の疲れを引き起こします



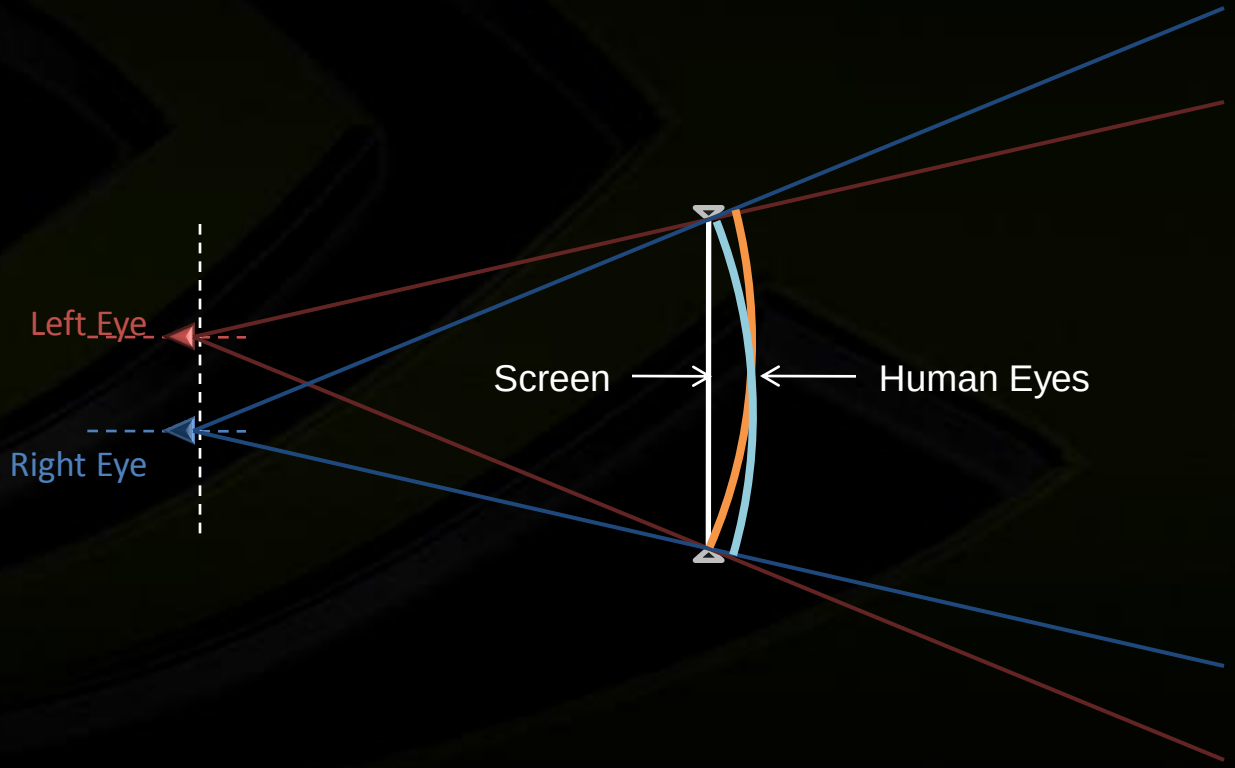
快適な値

- Screen depth (convergence) はapplication がcamera とsceneによって決定するべきです

- シーン内オブジェクトが次のレンジにいるようにしましょう
  [ScreenDepth / 2, 100 * ScreenDepth]

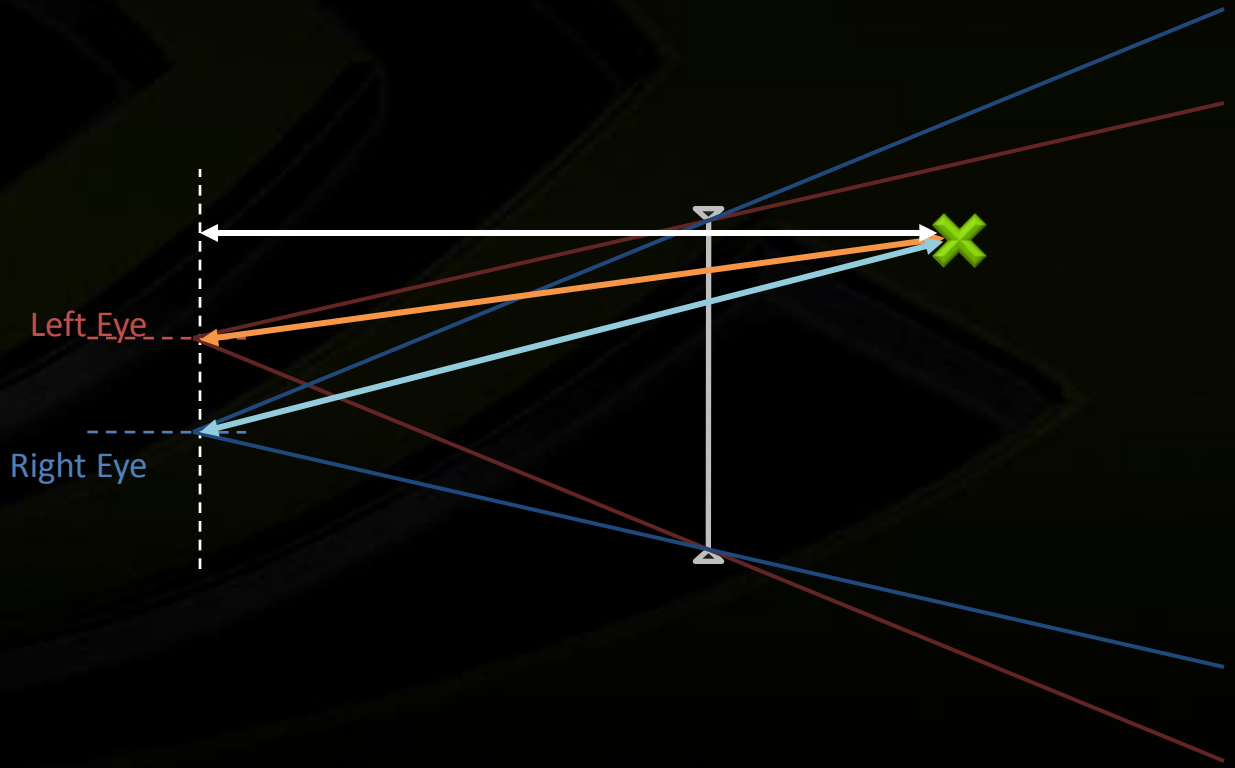
Projection Surface

- 現在のgraphics pipelineは、Projection Surfaceが平らであることを想定しています
- 実際の人間の目のprojection surfaceはspherical surfaceです



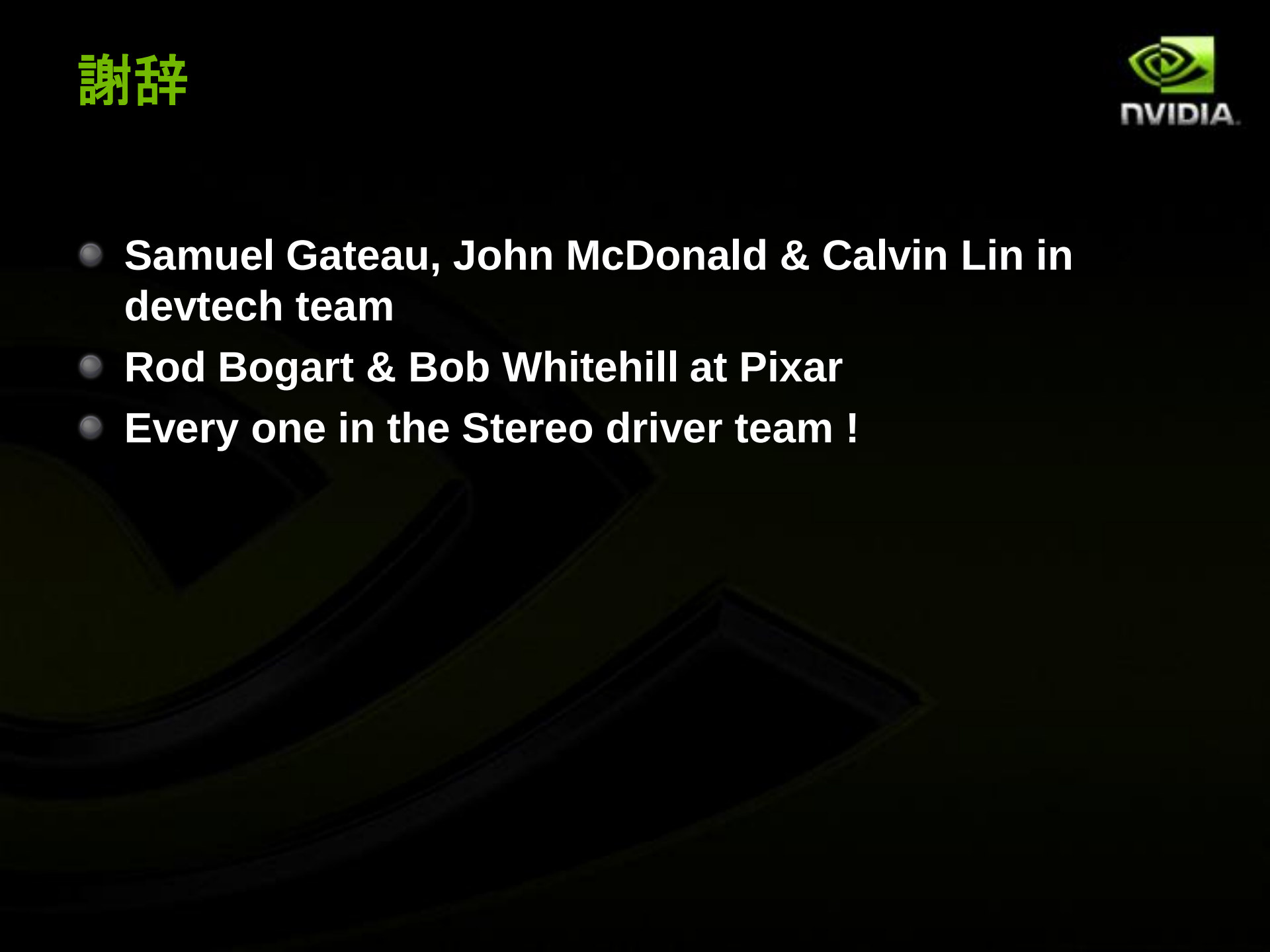
Scene Depth

- 現在のgraphics pipelineではScene depthは
  - Depth = Distance to eye plane
- 人間の目でのScene depthは
  - Depth = Distance to eyes
- この差が大きいと、不快感が増します




QUESTIONS ?

謝辞



- Samuel Gateau, John McDonald & Calvin Lin 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
- **Mail**
  - tkazama@nvidia.com

NVIDIA 風間 隆行