实战游戏三维立体效果

原理、实践与用户体验的优化

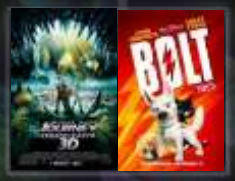
林楠，NVIDIA

提纲

- NVIDIA 3D Vision™
  - 包含驱动、硬件设备和显示器的立体效果解决方案
- 立体视觉基础
  - 原理与公式
- 结合3D Vision进行渲染
  - 如何在立体模式下进行渲染
- 实际问题与解决方案
  - 游戏中遇到的实际问题及解决方案


NVIDIA
立体眼镜是如何工作的？
NVIDIA® 3D VISION™
gameworks.nvidia.com

三维视觉体验



3D电影

3D图像

3D游戏

3D网播

利用NVIDIA GPU的强大处理能力，各种格式的立体数据经过解码和转换操作,都可以在支持3D-Ready显示设备上展现出来

120 Hz LCD显示器

3D DLP高清电视

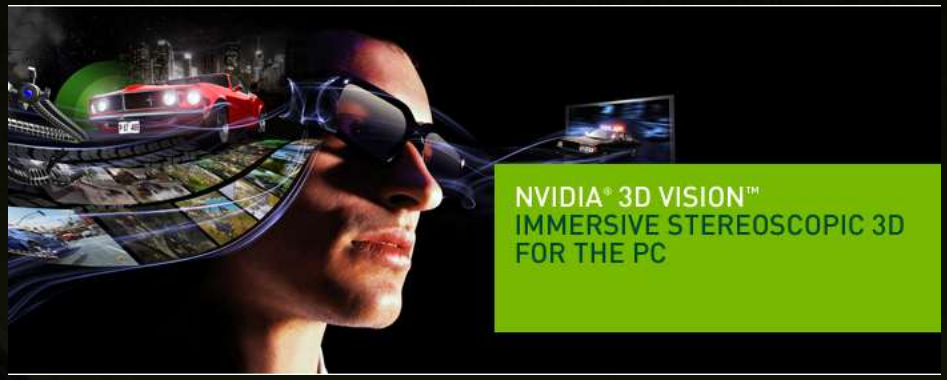
3D投影仪

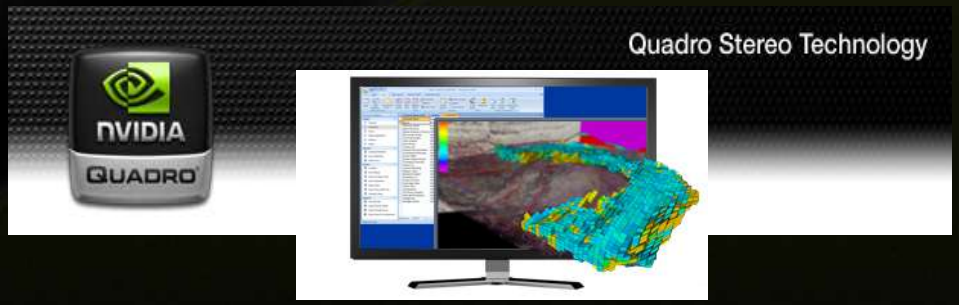
3D立体彩相眼镜

NVIDIA 3D Vision

支持立体效果

- GeForce系列
  - 立体显示驱动
    - Vista和Win7
    - D3D9/D3D10/D3D11
- Quadro系列
  - 具备**GeForce**的所有特性
  - 支持专业**OpenGL**中的**Stereo Quad Buffer**
    - 支持多个同步的立体显示器
    - 多平台
    - **3D Vision**以及其他立体视觉技术





gameworks.nvidia.com

NVIDIA 3D Vision

NVIDIA 3D Vision解决方案

	NVIDIA 3D Vision Discover	NVIDIA 3D Vision
Availability	一些特定NVIDIA显卡会附赠 用来体验3D立体效果	单独出售,专为体验高清3D立体设计
3D 眼镜类型	NVIDIA 优化立体色相(红/蓝)眼镜	快门式液晶眼镜，带无线接收器
3D模式	GPU进行处理 两张过滤对应颜色后的图像构成3D效果	120 Hz刷新，奇偶帧交替
色彩还原度	有限色彩	全色彩
显示器要求	各种LCD和CRT显示器	3D-Vision-Ready 显示器
NVIDIA GeForce GPU	GeForce 8系列及更高	GeForce 8系列及更高
操作系统	Microsoft Windows Vista Microsoft Windows 7	Microsoft Windows Vista Microsoft Windows 7
观看3D 图像	Y	Y
观看3D 电影	Y	Y
玩3D游戏	Y	Y
其他3D 产品	Y	Y

NVIDIA 3D Vision

工作机理

游戏中的3D数据被发送给立体驱动

驱动使用接收到的3D数据渲染两遍场景（左右眼各一次）

左眼图像

右眼图像

立体视觉设备在偶数帧(0, 2, 4, 等)显示左眼图像,奇数帧(1, 3, 5, 等)显示右眼图像

NVIDIA 3D Vision

工作机理(续.)

右图展示了一个例子，在屏幕显示左眼图像的时候，交替遮挡眼镜的右眼镜片变黑，而左眼正常显示；屏幕显示右眼图像的时候则反而行之。

这意味着对于单眼来说，显示器的实际刷新率是减半的。
(例如：一个120Hz的显示器实际上是单眼60Hz)

用户看到的最终图像即是结合了左右眼信息的立体图像

工作机理(续.2)

- 3D Vision被设计为可以透明地集成到游戏中
  - 游戏引擎不需要知道立体驱动的存在
  - 驱动自动产生左右眼图像
  - 无需或仅需很少的程序调整
- 同时也支持完全手工控制的方法
  - 用以支持高级特效或特殊用法
  - NVAPI中提供了程序接口

NVAPI立体视觉模块

- NVAPI是NVIDIA的核心软件开发包之一，通过NVAPI可以直接访问NVIDIA的GPU和驱动

- 面向3D Vision高级应用
  - NVAPI包含一个3D Vision模块用于控制驱动中的立体成像参数
  - 检测系统是否具有3D Vision功能
  - 管理游戏中的立体成像参数设置
  - 在引擎中动态控制立体成像参数，以便提供更佳的视觉体验

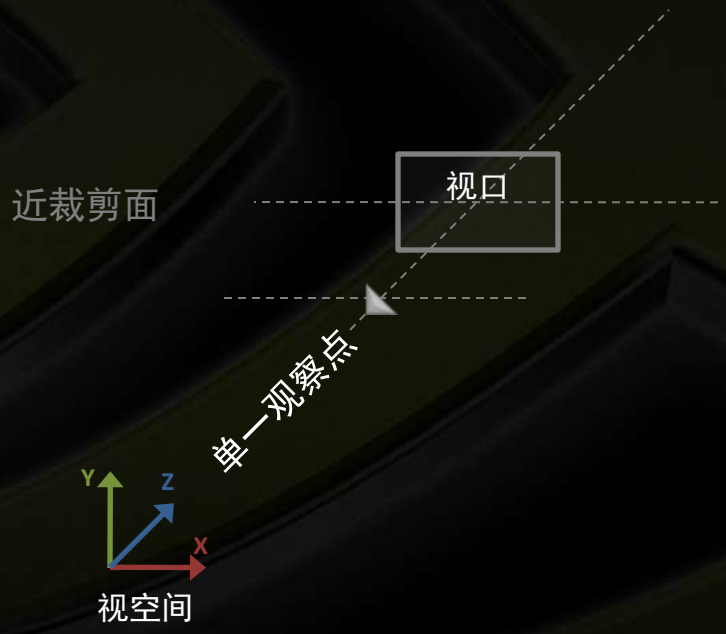
- SDK和文档下载地址
  http://developer.nvidia.com/object/nvapi.html

原理与公式

立体视觉基础

标准单眼渲染

场景从单一观察点(单眼)进行渲染，然后投影到视口的近裁剪面上



立体视觉基础

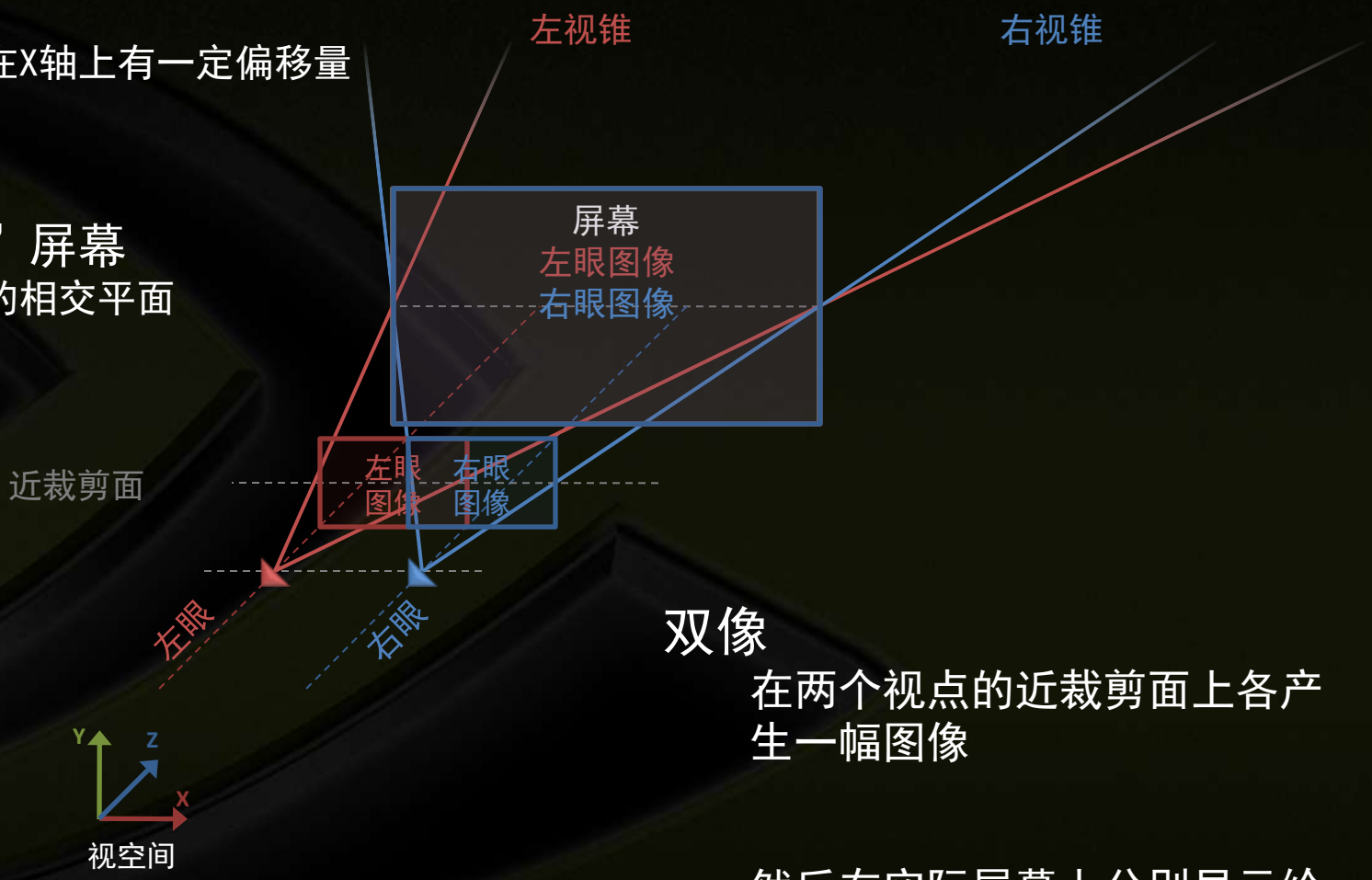
双眼，双图像，单一屏幕

左眼和右眼

相对单眼在X轴上有一定偏移量

一个“虚拟”屏幕

左右视锥的相交平面



双像

在两个视点的近裁剪面上各产生一幅图像

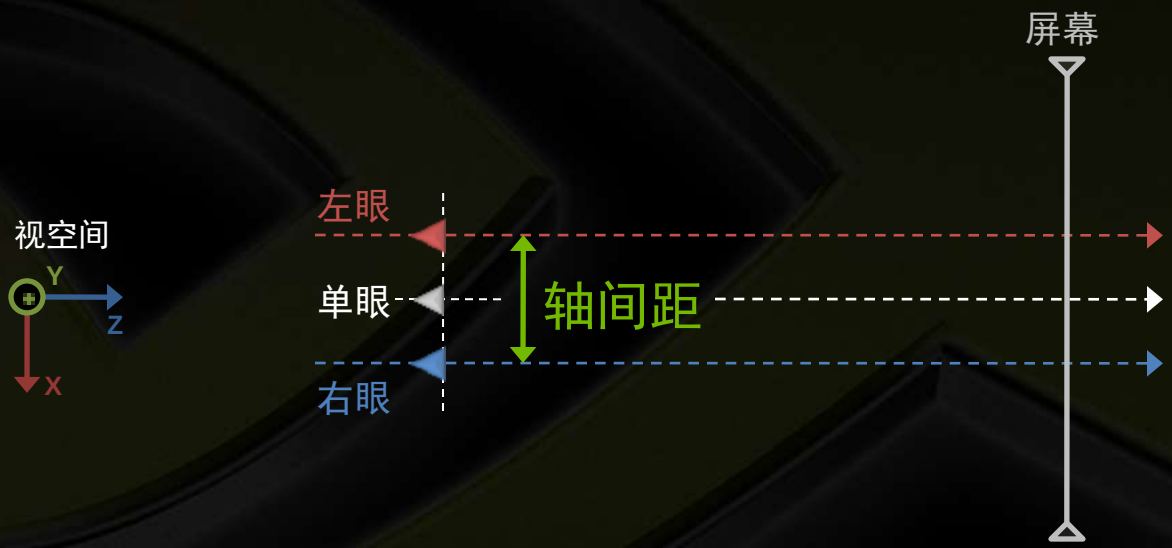
然后在实际屏幕上分别显示给相应的眼睛观看

立体渲染

- 为左右视点各渲染一次场景，产生相应的左右图像
- 对标准管线进行了3处修改
  - 使用立体表面(stereo surfaces)
    - 由输出缓存(render surfaces)复制而来
  - 使用立体Draw Call(stereo drawcalls)
    - 复制原有Draw Call
  - 立体分离度(stereo separation)
    - 用于修改投影矩阵

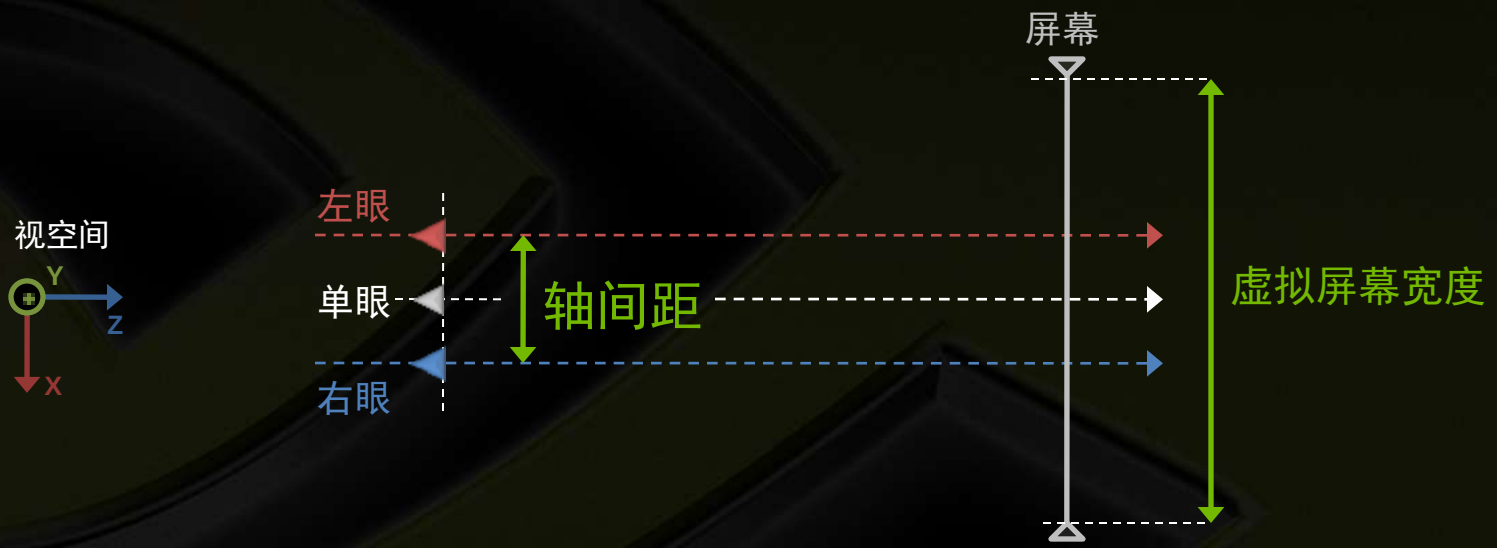
定义：轴间距(Interaxial)

- 视空间内，左右眼之间的虚拟距离
- 左右眼视线及单眼视线相互平行



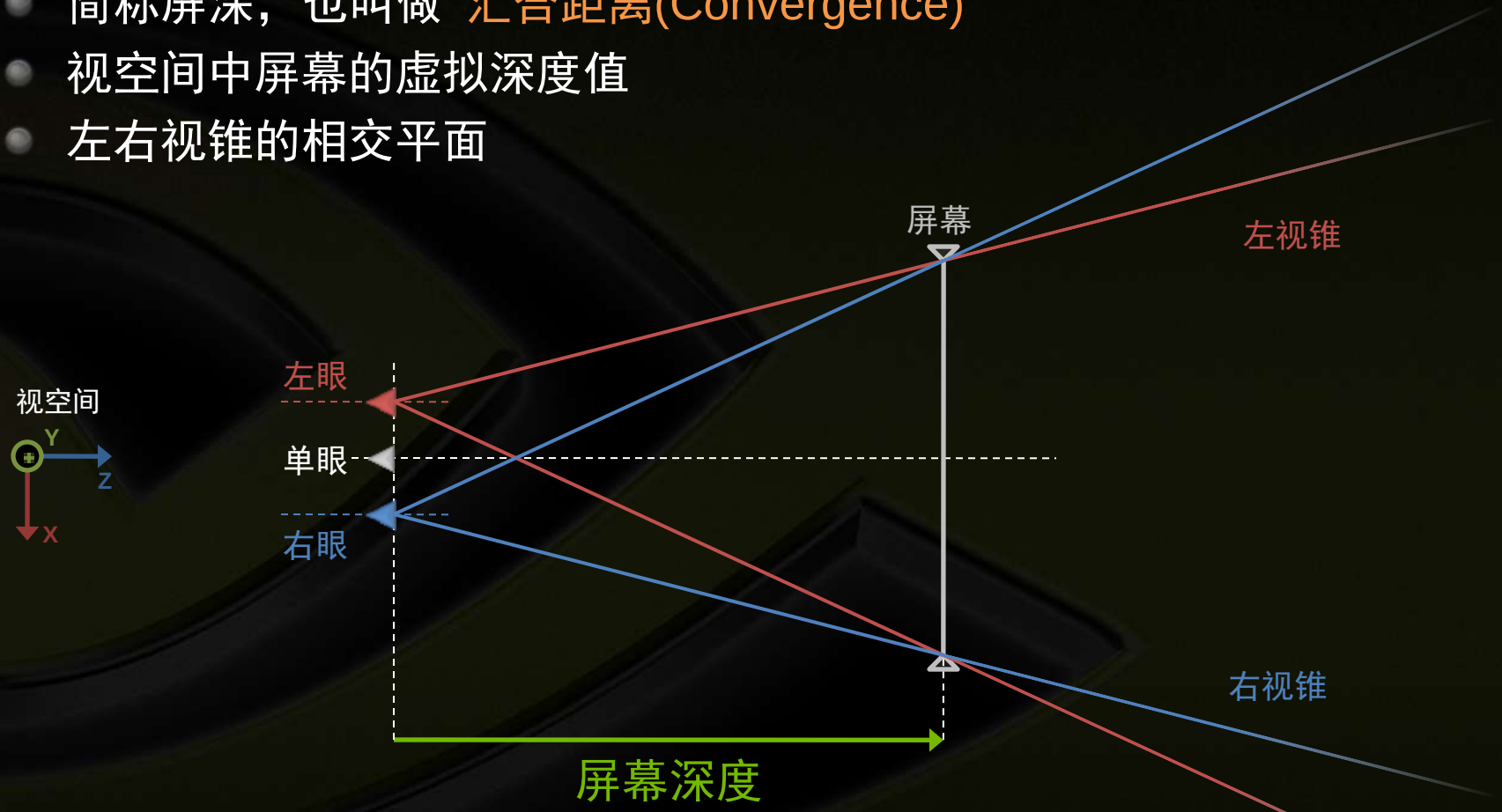
定义：分离量(Separation)

- 规范化的轴间距
  分离量 = 轴间距 / 虚拟屏幕宽度



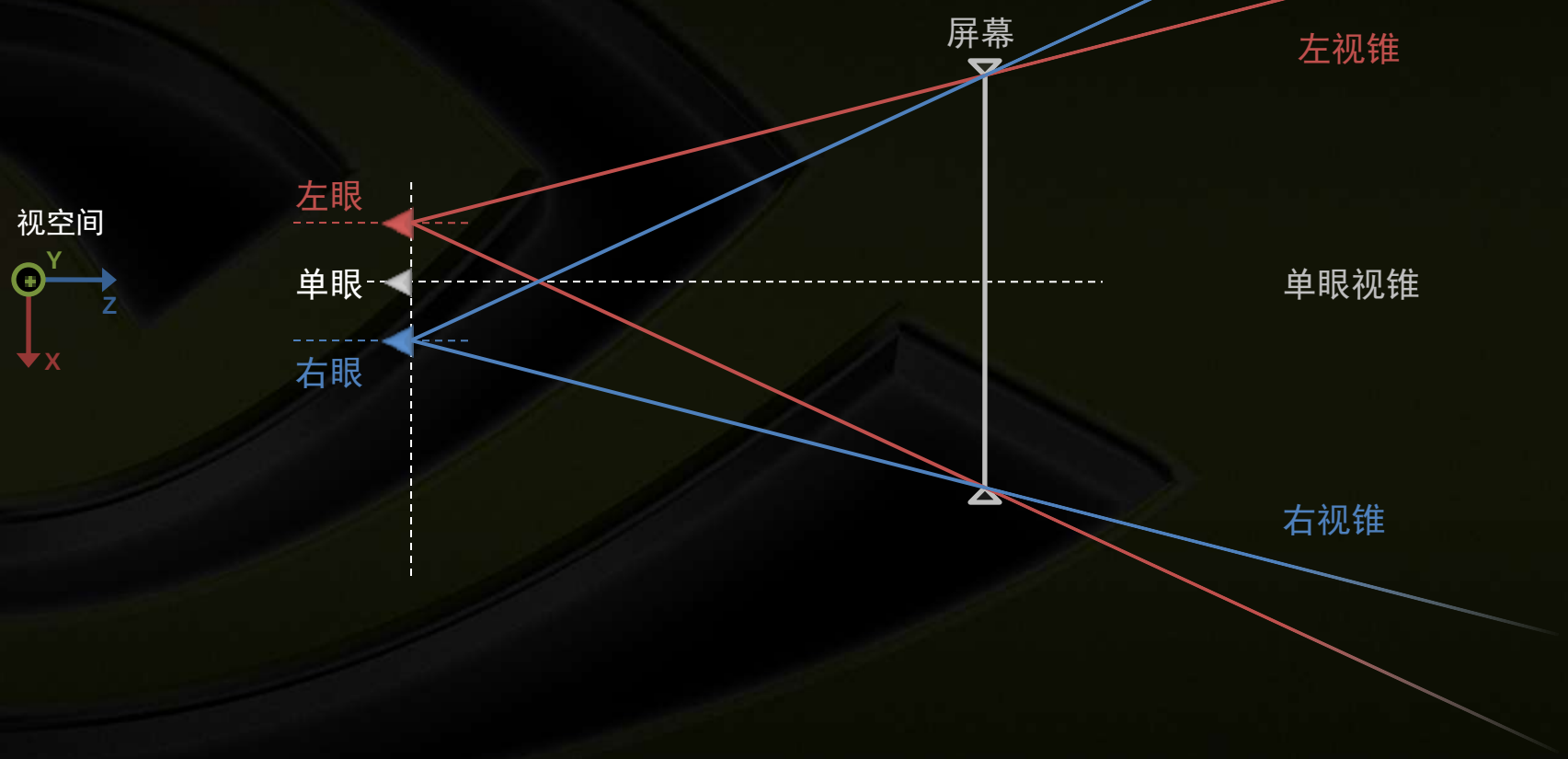
定义:屏幕深度(Screen Depth)

- 简称屏深，也叫做“汇合距离(Convergence)”
- 视空间中屏幕的虚拟深度值
- 左右视锥的相交平面



左右眼投影

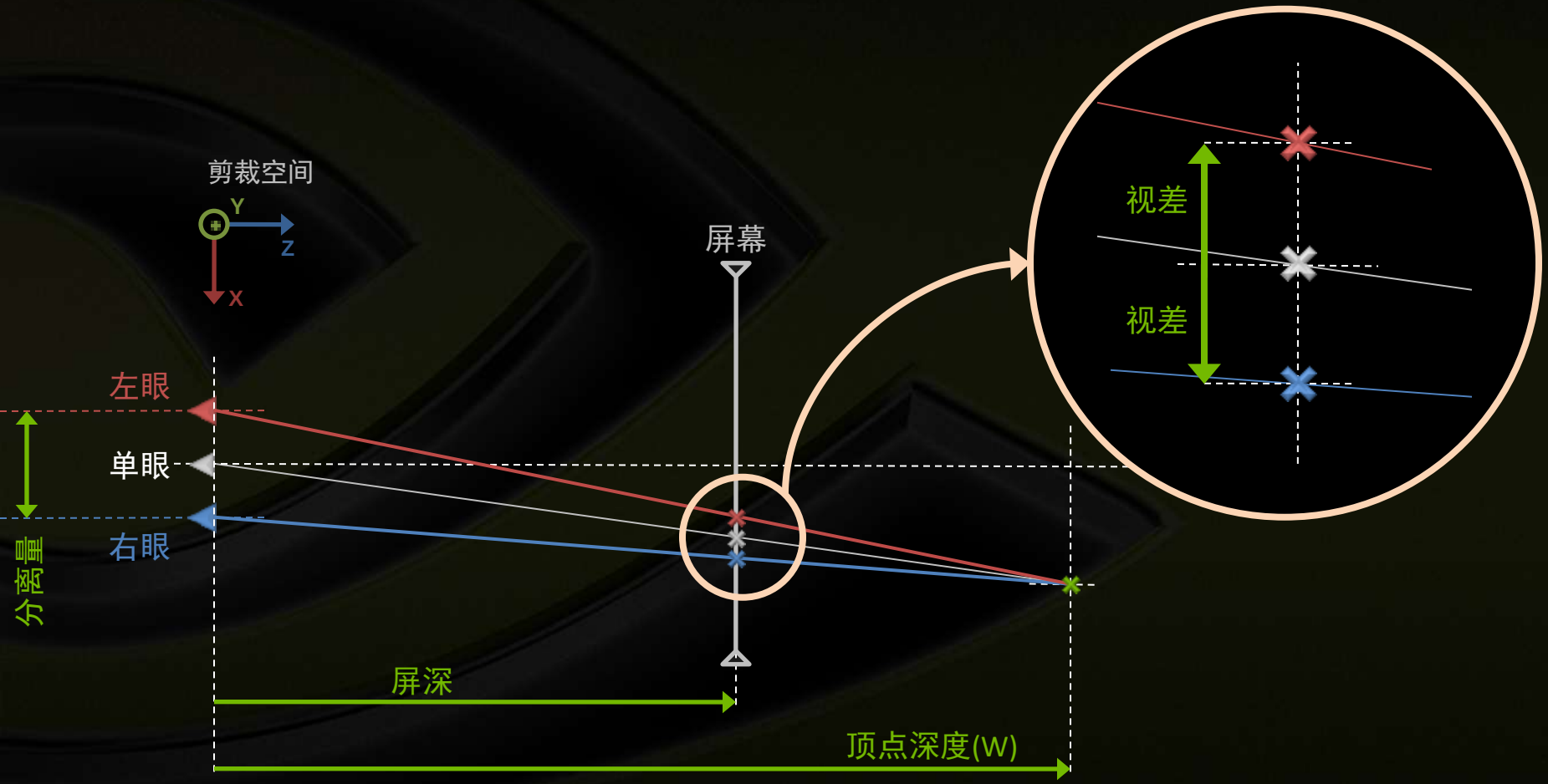
- 修改常规单眼投影矩阵，在水平方向上平移，得到左右眼的投影矩阵
  - 在X轴方向上向左或者向右平移



立体视觉基础

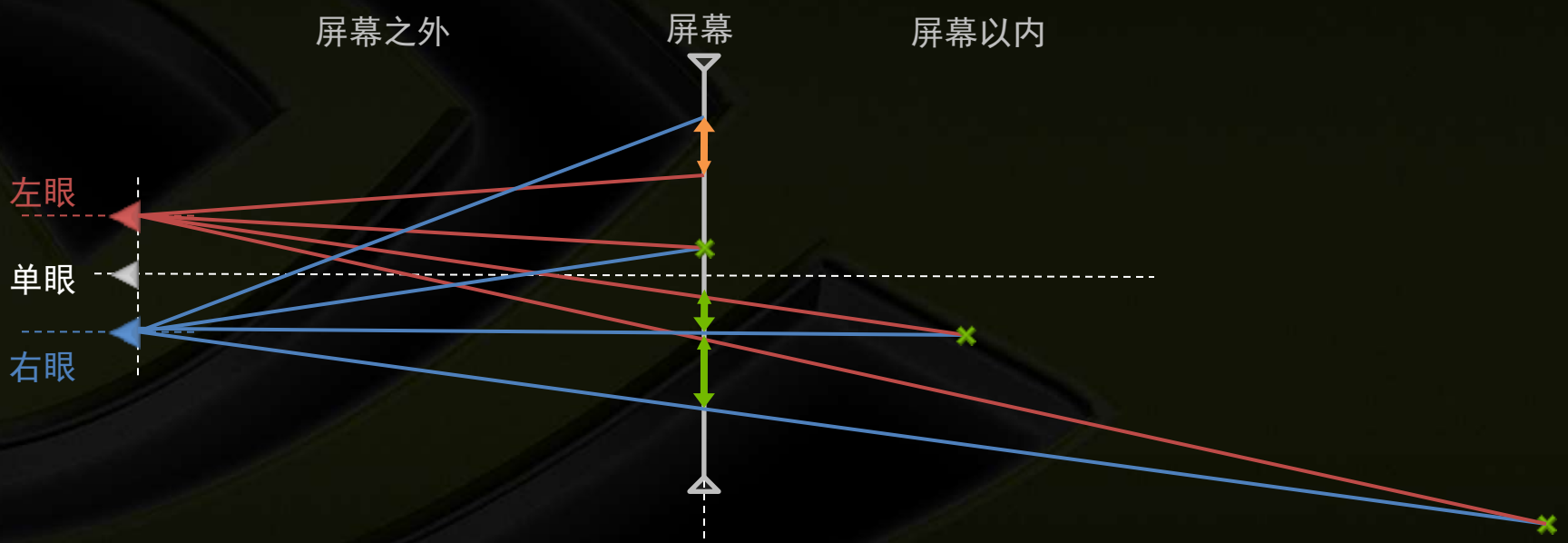
定义:视差(Parallax)

- 同一顶点在屏幕上左右眼两个投影点之间的带符号距离
  - 视差是视空间定点深度的函数
    视差 = 分离量*(1 – 屏深/ W)



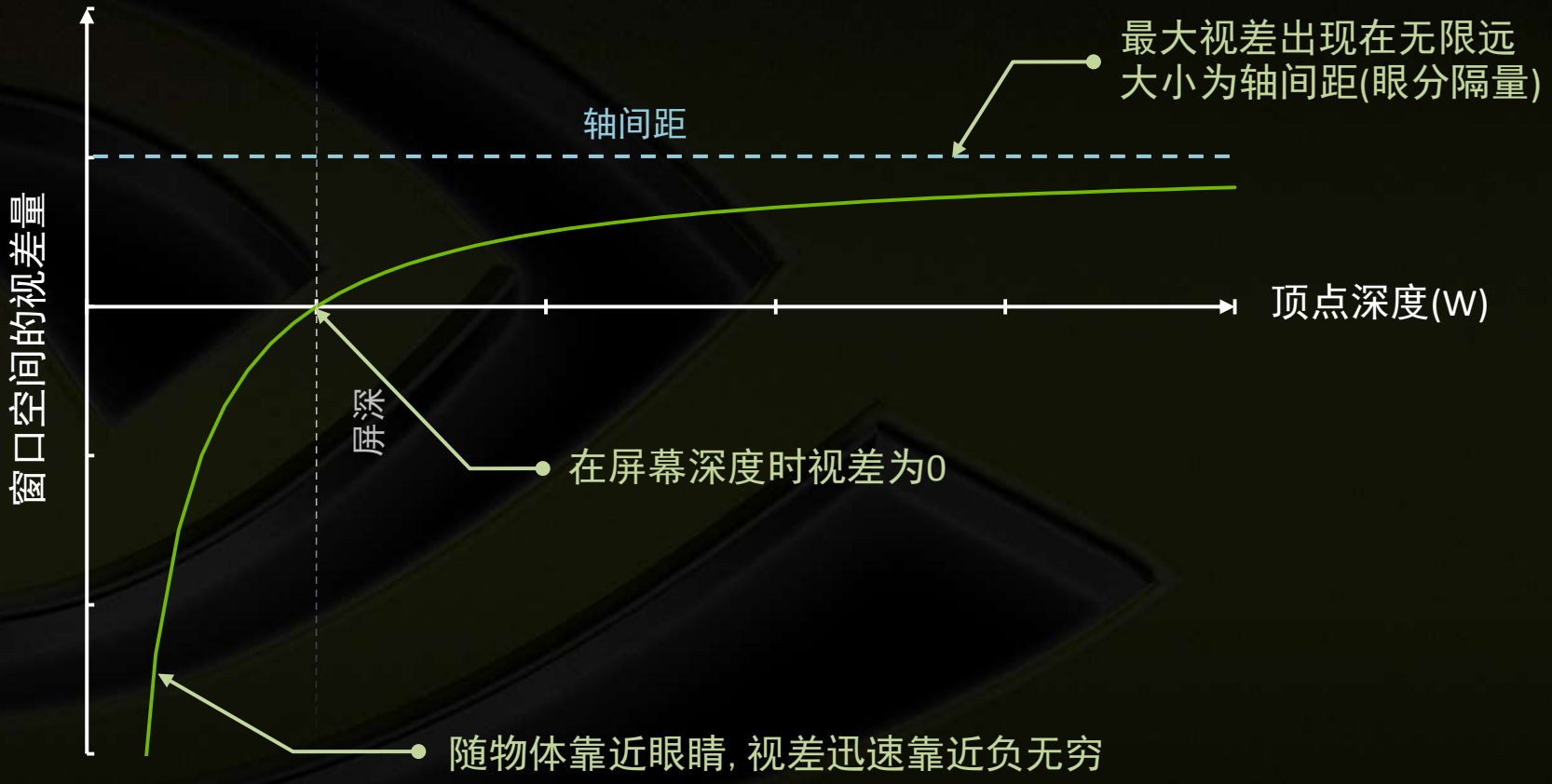
屏幕内外

- 视差会产生相对于屏幕的深度感
- 当视差值为负时, 顶点看起来在屏幕之外



视差方程

- 视差 = 分离量 * (1 –屏深/ W)



定义和公式概要

- 轴间距和分离量
  - 两眼之间的虚拟距离/规范化距离
- 屏深
  - 屏幕平面在视空间内的深度
- 视差
  - 视空间内同一个顶点产生的两个投影点之间的带符号距离
  - 视差 = 分离量 * (1 – 屏深/ W)
- 屏幕内和屏幕外
  - 屏幕内:屏幕和远裁剪面之间
    视差 > 0, W > 1
  - 屏幕外:眼平面和屏幕之间
    视差< 0, 0 < W < 1

具体如何在立体模式下渲染？

结合3D Vision进行渲染

驱动魔术

- 在立体驱动中发生了什么?
  - 自动为游戏引擎中加入立体效果

1. **复制输出缓存(render targets)**
   产生用于立体渲染的左右两个表面(surface)
2. **复制Draw Call(draw call)**
   每个Draw Call执行两遍
3. **使用分离量创造立体效果**
   修改Vertex Shader的输出，把坐标沿x轴分别向左右平移，产生左右眼图像

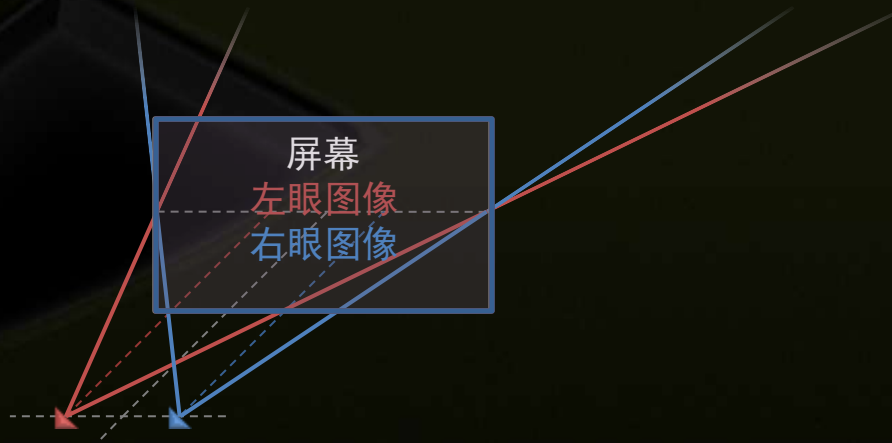
复制输出缓存

- 驱动根据一套特定机制来决定是否进行复制
  - 对游戏引擎透明
  - 主要依据输出缓存的尺寸：
    - 大于或等于**back buffer**尺寸的表面会被复制
    - 正方形的输出缓存不会被复制
    - 小于**back buffer**尺寸的表面不会被复制
- 也可以显式进行复制
  - 由游戏引擎完全掌控
  - **NVAPI**提供必要的接口
- 在以下的内容中，未经特别提及，指的都是自动复制模式

立体输出缓存

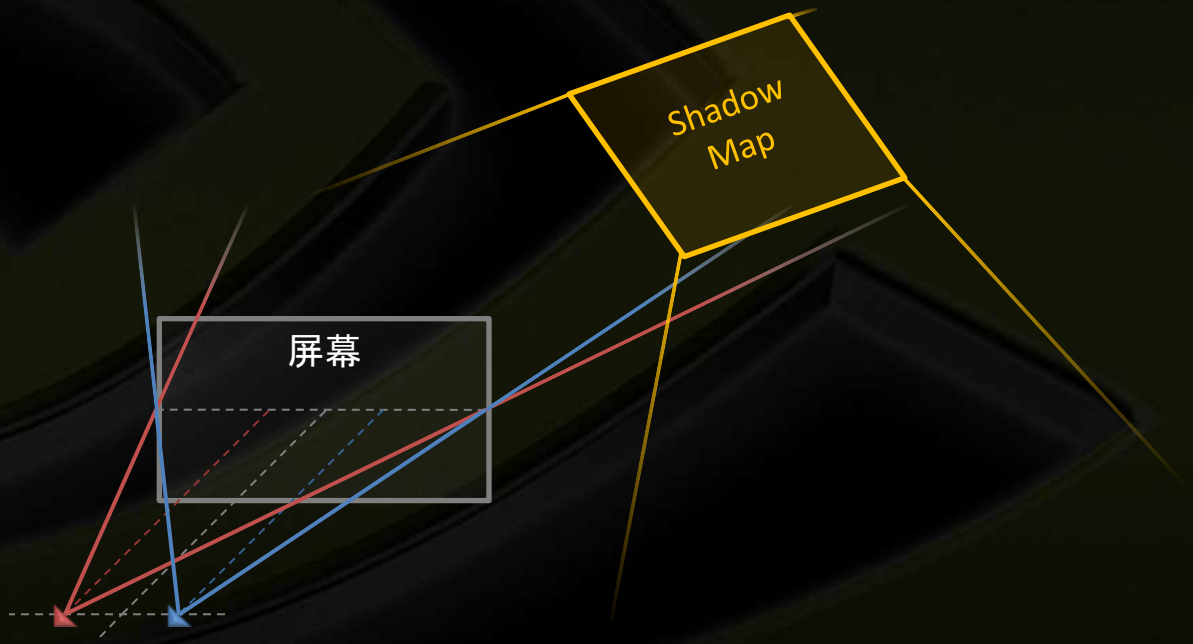
- 视点相关的输出缓存必须进行复制
  - **Back buffer**
  - **Depth Stencil buffer**

- 用于后处理的中间缓存
  - **HDR, blur, bloom, DOF**
  - **SSAO**
  - 屏幕空间内的全屏阴影



普通输出缓存

- 视点无关的输出缓存无需复制
  - **Shadow map**
  - **Spot light textures**



立体与普通:更多实例分析

用例	表面类型	立体投影	复制Draw Call
阴影(shadow map)	普通	否 使用光源的投影矩阵	画一次
主帧缓存 任何前向渲染的输出结果	立体	是	画两次
反射贴图	立体	是 用反射景物的立体投影矩阵	画两次
后处理特效 (绘制一个全屏的矩形)	立体	否 无需投影	画两次
延迟渲染中的光照 (渲染一个光源体)	立体 G-buffers	是 须特别处理逆投影变换 (考虑立体投影)	画两次

结合3D Vision进行渲染

复制Draw Call

- 在NVIDIA立体驱动(自动模式)中进行
  - 对于所有的立体缓存, 每个draw call都为左右眼各调用一次
  - 对普通缓存不做任何修改
- 驱动程序中的伪代码

```
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();
}
```

使用分离量创造立体效果

- 在自动模式下
  - 驱动修改Vertex Shader，添加视差偏移的代码
  - 输出的顶点坐标中，x值被修改为:

Pos.x += EyeSign * Scale * 分离量* (Pos.w – 屏深)

Scale: 用户可调整的一个参数, 用来调节视差效果的强弱

- 在显式模式下
  - 游戏引擎自行在Vertex Shader中用分离量计算偏移量
  - 获取参数

Scale: NvAPI_Stereo_GetSeparation()
Separation: NvAPI_Stereo_GetEyeSeparation()
Convergence: NvAPI_Stereo_GetConvergence()
EyeSign: 左眼为**-1,** 右眼为**+1**

gameworks.nvidia.com

3D物体

- 位于一帧中的所有3D物体都应使用统一的透视投影变换
- 相对于同一场景，所有3D物体的深度值应保持一致性
  - 感觉上距离接近的物体，在立体模式下的深度也应接近
- 多数光照效果时无需修改shader
  - 跟视点相关光照效果，如高光及镜面反光等，通常不会发生可察觉的错误
- 反射和折射应当在立体模式下渲染

伪3D物体: 天空, Billboard等

- 天空盒必须以一个有效的深度值进行渲染，确保其远于场景中的常规物体
  - 必须被立体投影
  - 最好设定为极远值这样视差最大
  - 覆盖整个屏幕
- Billboard(如粒子、树叶等)应当被渲染在与视平面平行的平面上
  - 但看起来效果欠佳
- 浮雕贴图（Relief Map）及类似算法的效果较差
  - 包括Parallax occlusion map, steep map等
  - 需要在pixel shader中进行额外的修正(稍后详细讨论)

多视口中的3D物体

- 某些游戏中也许需要在同一屏幕上显示多个3D场景
  - 显示当前人物肖像的小视口
  - 分屏游戏模式，多个玩家共同使用
- 每个视口可能需要单独的屏深设置
  - 游戏引擎需要小心处理每个视口
  - 使用NVAPI中的函数NvAPI_Stereo_SetConvergence()

3D物体:屏外效果(out of screen effect)

- 让用户与飘出屏幕的幻像搏斗
  - 需要小心调整才能获得良好的效果
- 受显示器边缘限制
  - 会造成奇怪的效果
- 可缓慢地将物体从屏幕内移动到屏幕外，让眼睛逐渐适应
  - 平滑地调整可见度
  - 避免闪烁
- 真实的图像质量有助于提升屏外效果

2D物体

- 2D元素必须以有效深度值进行渲染
  - 给每一个2D 元素一个有效的W值
- 与3D场景无交互
  - HUD等界面元素
  - 放在屏深处(W = 1.0), 不会产生视差
- 与3D场景有交互
  - 鼠标指针放在被指向物体的深度
    不能使用硬件指针
  - 十字准星
  - 场景中的标签或者广告牌
  - 放在场景中深度恰当的地方 W = 视空间中的场景深度

2D物体: 设置正确的深度值

```
float  depth;

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

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

    return Output;
}
```

- 将**depth**设为**1.0**以去除视差

gameworks.nvidia.com

如何解决…

实际问题与解决放案

大量潜在问题

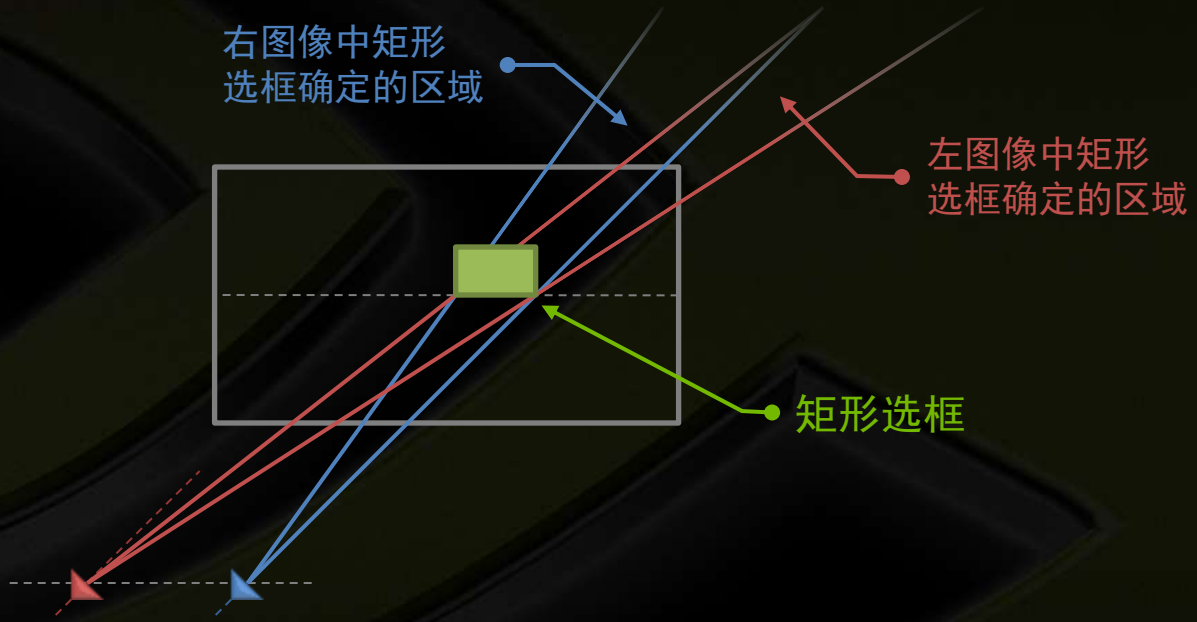
- 在过去的3年中,我们遇到过很多立体成像的问题
  - 十字准星,鼠标指针以及选择框
  - 在游戏引擎中重播立体视频/图像
  - 视锥剪裁
  - 延迟渲染以及类似技术
  - 视点相关的光照/贴图效果
  - 全屏模式下输入法编辑器问题
  - 视觉疲劳和眩晕
- 自动模式对大多数游戏都有良好的效果
  - 但是游戏引擎需要正确处理2D物体的深度

十字准星和鼠标指针

- 在现实世界中，人们不会用双眼瞄准
- 在立体模式中应该如何放置十字准星?
  - 解决方案: 将十字准星放置在瞄准物体的深度上
  - 不够真实, 但感觉“正常”
- 硬件鼠标指针不能设置深度值
  - 对于需要大量鼠标拾取操作的游戏，可考虑手动在场景深度上绘制鼠标指针

选择框

- 没有立体效果时，选择框可以简单地渲染为屏幕上一个2D矩形. 在立体模式下不能这样做
  - 在屏幕空间上，左右视点会通过选择框确定不同的范围
  - 两个视点的选择框的竖直边无法重合



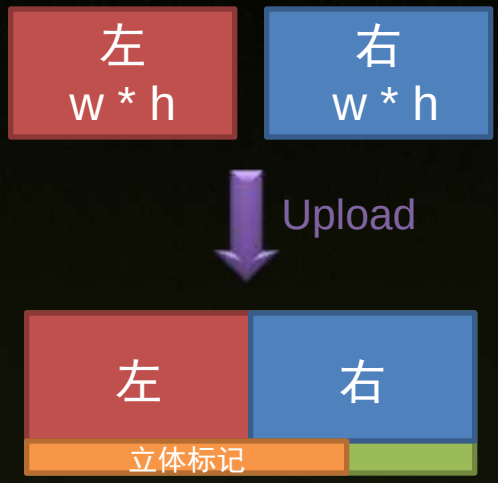
- 应使用矩形选框的投影体来进行精确的选择判断. 参考下文以获取更多细节 http://developer.download.nvidia.com/presentations/2009/GDC/GDC09-3DVision-The_In_and_Out.pdf

播放立体视频或立体图像

- 许多游戏希望以立体模式播放预渲染的视频剪辑
- 在3D Vision中播放现有的立体内容
  - 回放立体视频
  - 使用立体图像作为菜单背景
  - 过场动画

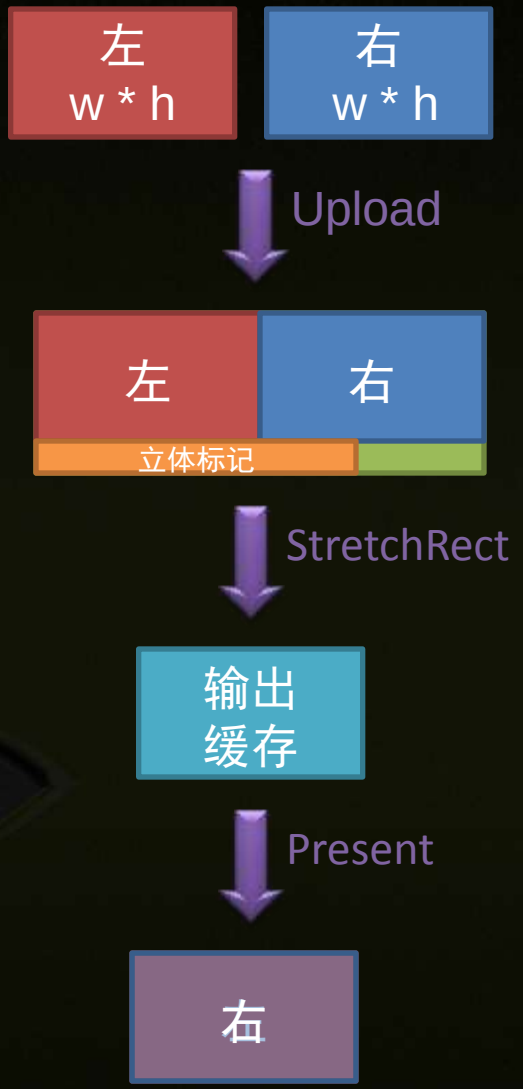
播放立体视频或立体图像(续.)

- 引入立体贴图
- 生成一张D3D贴图, 其中:
  宽度为原图像的2倍
  高度为原图像+1
  左半边是左眼图像
  右半边是右眼图像
  额外一行中存放NV3D标记
- 在创建时加入NV3D标记



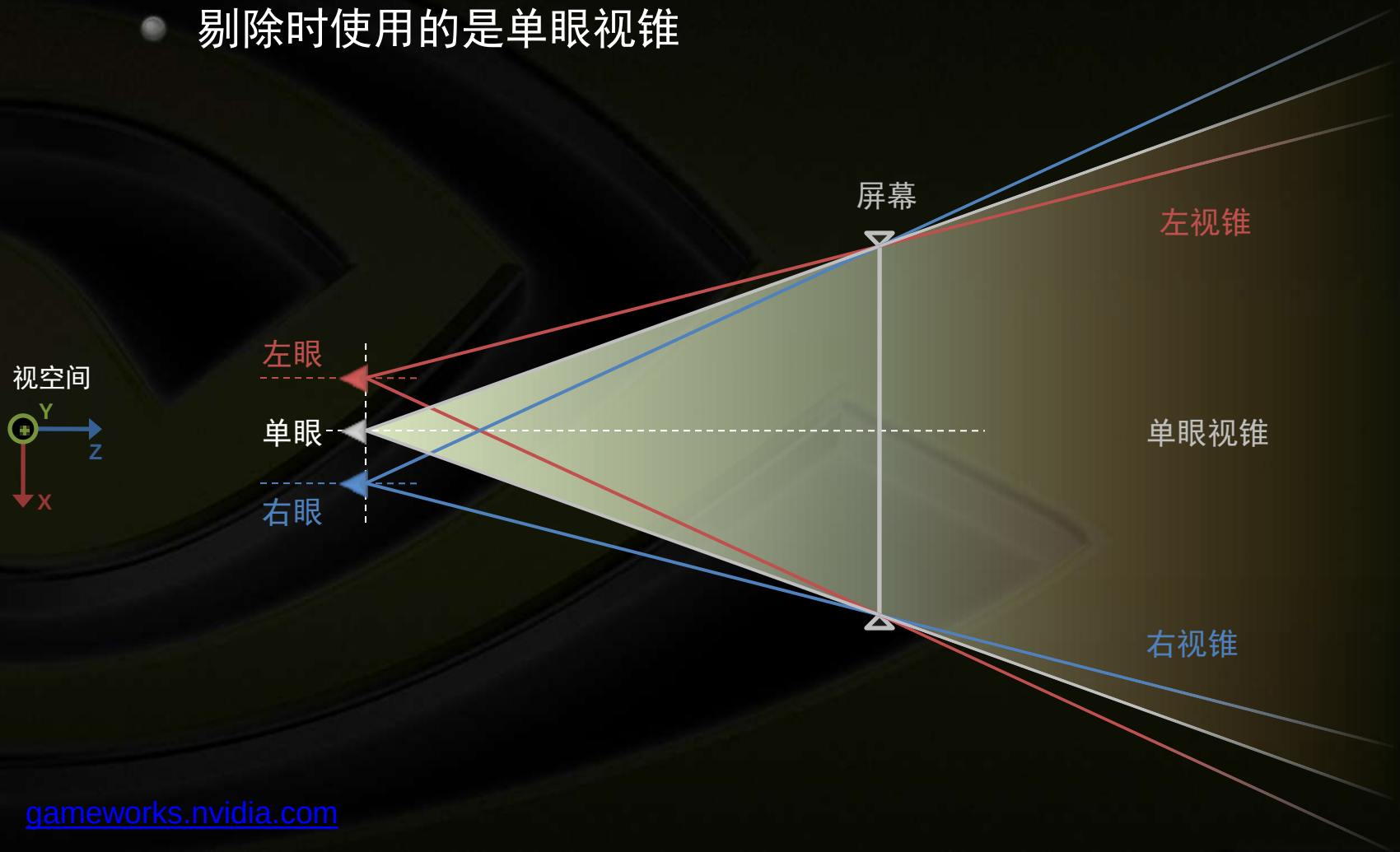
播放立体视频或立体图像(续. 2)

- 拷贝立体贴图到立体输出缓存中
  - 驱动自动处理:
    将贴图左侧放到输出缓存的左表面
    将贴图右侧放到输出缓存的右表面
- 若要在pixel shader中使用立体贴图
  - 需要每一帧先将这张贴图拷贝到一张普通贴图中
  - 驱动自动在画左眼图像时，拷贝左侧贴图数据；画右眼图像时，拷贝右侧贴图数据



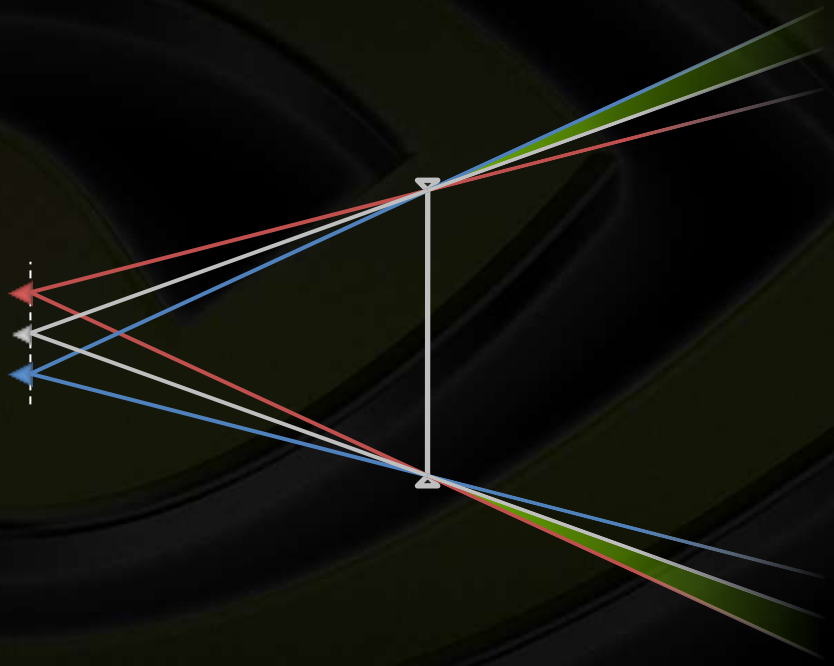
视锥剔除

- 多数游戏引擎都会做视锥剔除
  - 剔除时使用的是单眼视锥

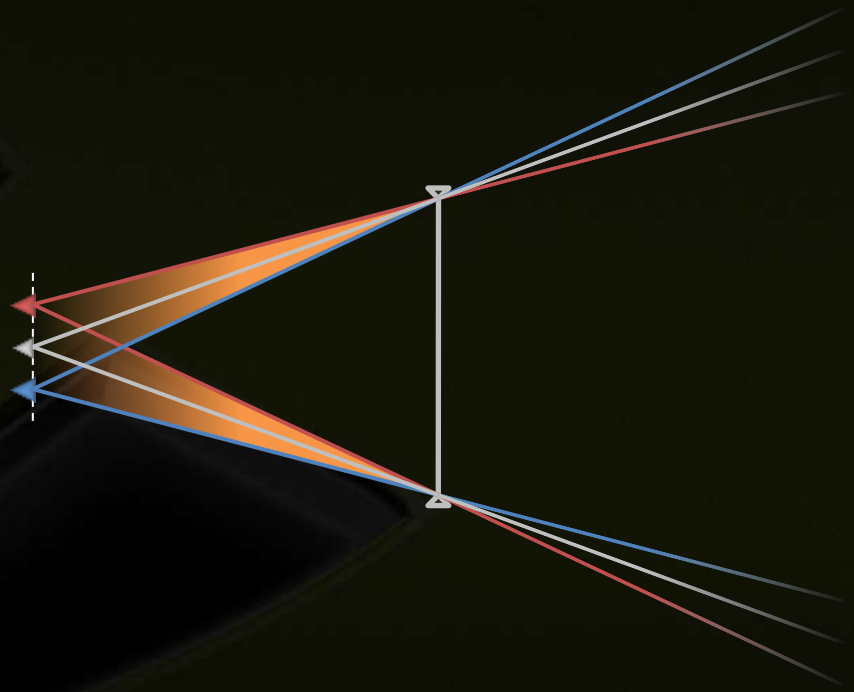


视锥剔除(续.)

丢失部分屏幕内区域



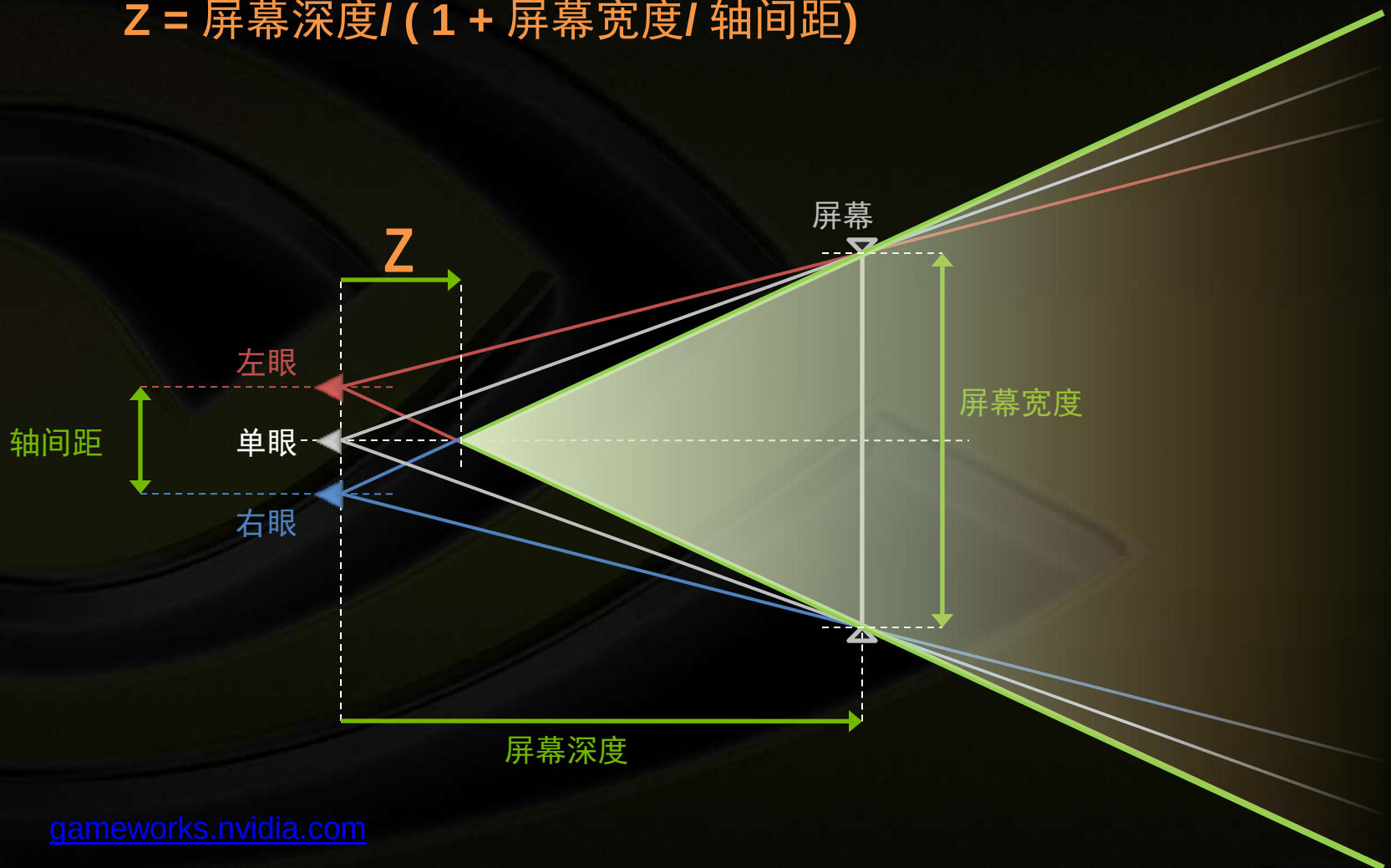
部分屏幕外区域
只对单眼可见



视锥剔除(续. 2)

- 正确的剔除方法：结合左右眼视锥

  Z = 屏幕深度/ (1 + 屏幕宽度/ 轴间距)

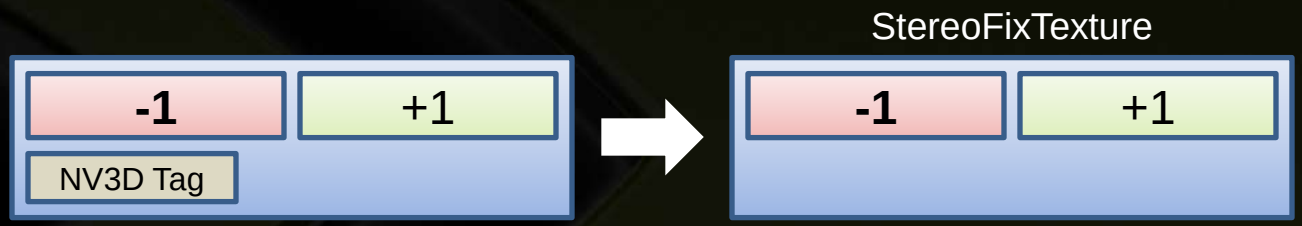


延迟渲染(Deferred Shading)

- 计算光照时，使用屏幕空间位置推导出世界空间位置
  - 但是，屏幕空间位置已按左右眼更改过了!
- 需要在pixel shader中进行特别修正
  - Pixel shader需要了解当前是在渲染左眼还是在渲染右眼
  - 进行坐标变换时需要把视差偏移量加进去
- Pixel shader如何知道是在渲染左眼图像还是右眼?
  - 可以利用立体贴图

延迟渲染(续.)

- 创建一个小的立体贴图
  - 左侧存放**-1.0**, 右侧存放**1.0**
  - **Pixel shader**很容易就能知道是在渲染左眼图像还是右眼



- 画左眼图像时, **Sample(StereoFixTexture) == -1**
- 画右眼图像时, **Sample(StereoFixTexture) == 1**

延迟渲染(续.2)

- **Pixel shader中的数学计算**

将光源形体变换到立体屏幕空间

1. 添加光源形体的视差
   Parallax = Scale * Separation * (Pos_mono.w – Convergence)
   Pos_stereo.x = Pos_mono.x + Sample(StereoFixTexture) * Parallax
2. 光源形体的坐标除以**w**, 变换到**[-1, 1]**
   Pos_stereo.xy /= Pos.w
3. 使用**Pos_stereo.xy** 从**G-Buffer**中读取**SceneDepth**等各种数据

计算世界空间位置

1. 像素坐标乘以场景的**w**, 即景深
   Pos_stereo.xy *= SceneDepth
2. 去除场景视差
   Parallax = Scale * Separation * (SceneDepth – Convergence)
   Pos_mono.x = Pos_stereo.x – Sample(StereoFixTexture) * Parallax
3. 逆变换**Pos_mono** 到世界空间

视点相关光照效果

- 高光和镜面反射
  - 在精确的情况下，应针对左右眼向量分别计算反射向量
  - 然而在多数情况下，使用单一视线向量并不会导致可察觉的错误
  - 无需修改**pixel shader**
- 浮雕贴图（Relief Map）
  - 视差遮挡贴图（POM）等.
  - 在视线-高度图求交时，对左右眼视线应分别处理
  - 使用立体贴图来确定是在渲染左眼还是右眼图像

全屏模式下的输入法编辑器(IME)

- 3D Vision只能在全屏模式下工作
- 并非所有IME都能在全屏模式下正常工作
- 解决方法: 建议用户在立体模式下使用D3D兼容的IME

视觉疲劳与眩晕

- 许多原因会导致视觉疲劳甚至严重眩晕
  - 太大或太小的视角
  - 闪烁的光照环境
  - 缺乏合理的运动模糊特效
  - 物体摆放情况、轴间距和屏深的不正确组合
  - 过多的屏外物体

瞳间距(Interocular)

- 瞳间距:两瞳孔之间的距离
  - 普通人的瞳间距均值大约为**6.0cm ~ 6.5cm**
  - 约等于无限远物体在屏幕上的视差
  - 同一物体的两个像，如果间距接近或大于瞳间距时，人脑就无法将他们重叠在一起

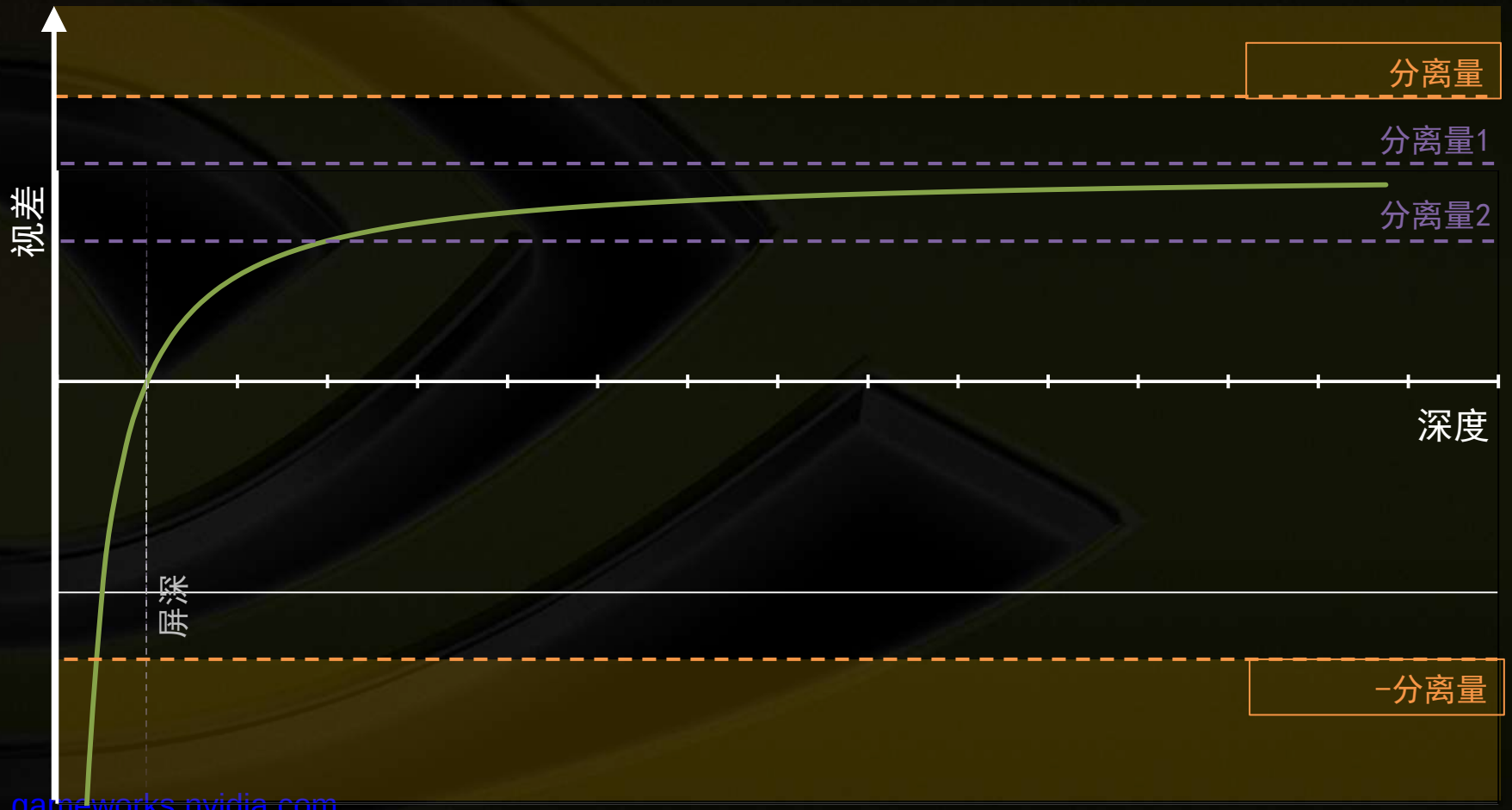
- 轴间距和瞳间距的关系

  轴间距 = 瞳间距 / 屏幕实际宽度

  - 取决于实际屏幕的尺寸。不同用户的轴间距各不相同
  - 让用户自行调节轴间距到一个舒适的范围

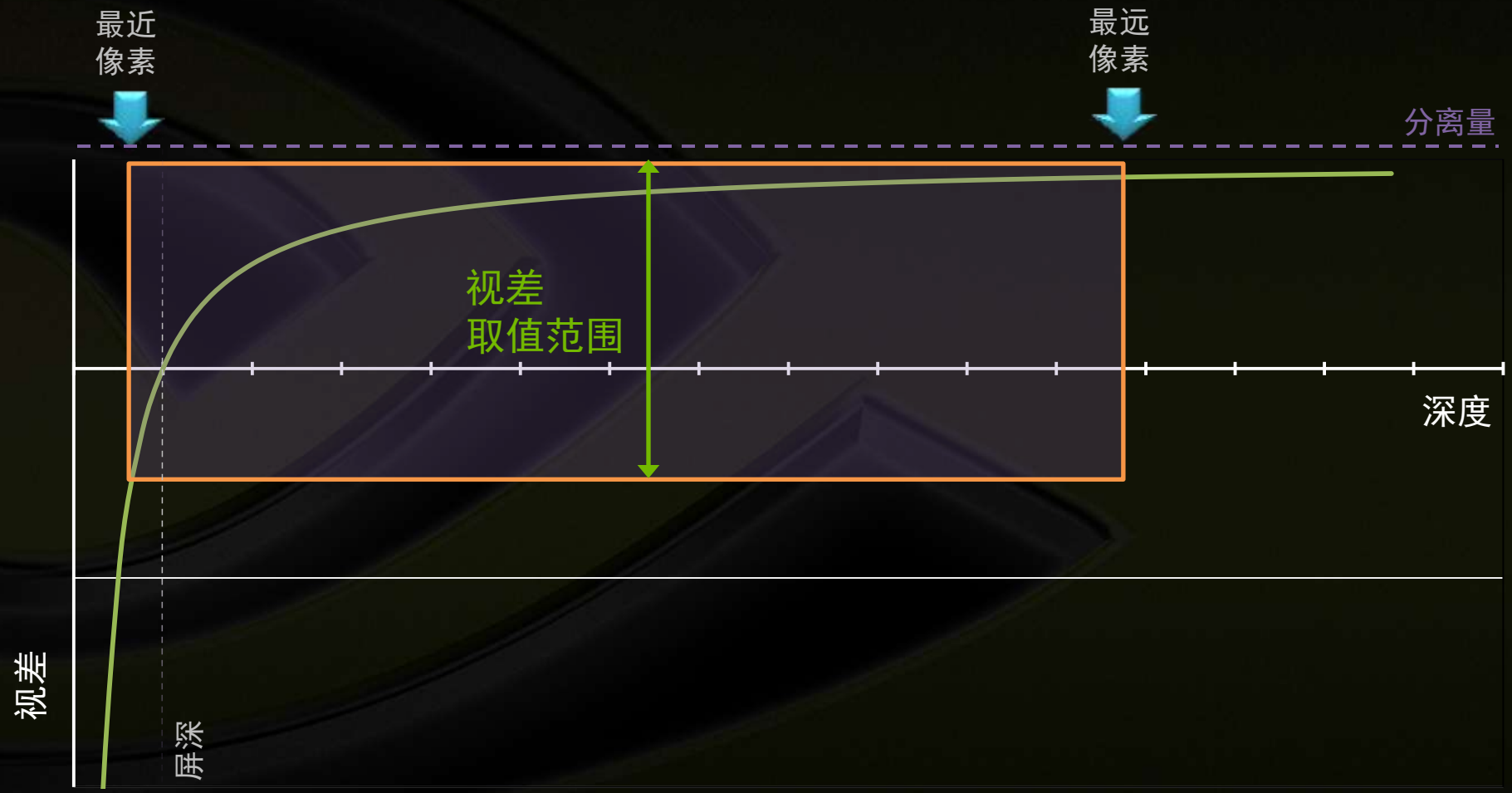
安全视差范围

- 对于一个特定的轴间距，物体的视差存在一个“安全范围”



视差取值范围

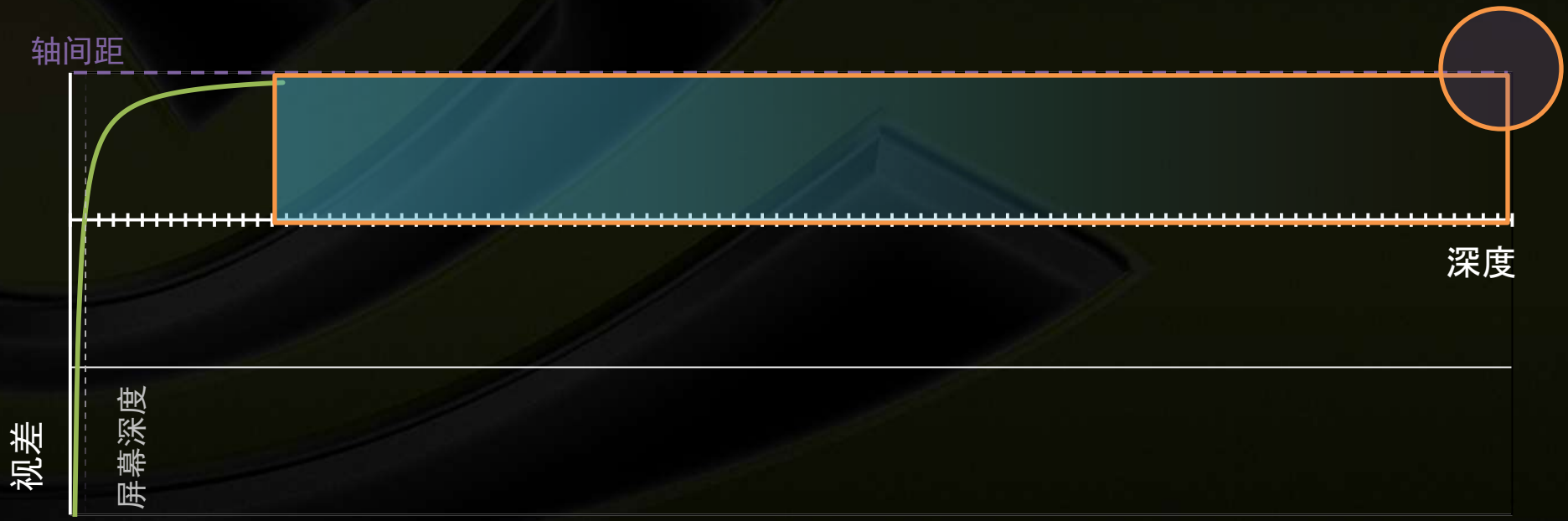
- 视差可在多大范围内变化



减少视觉疲劳和眩晕

视差取值范围：最远像素

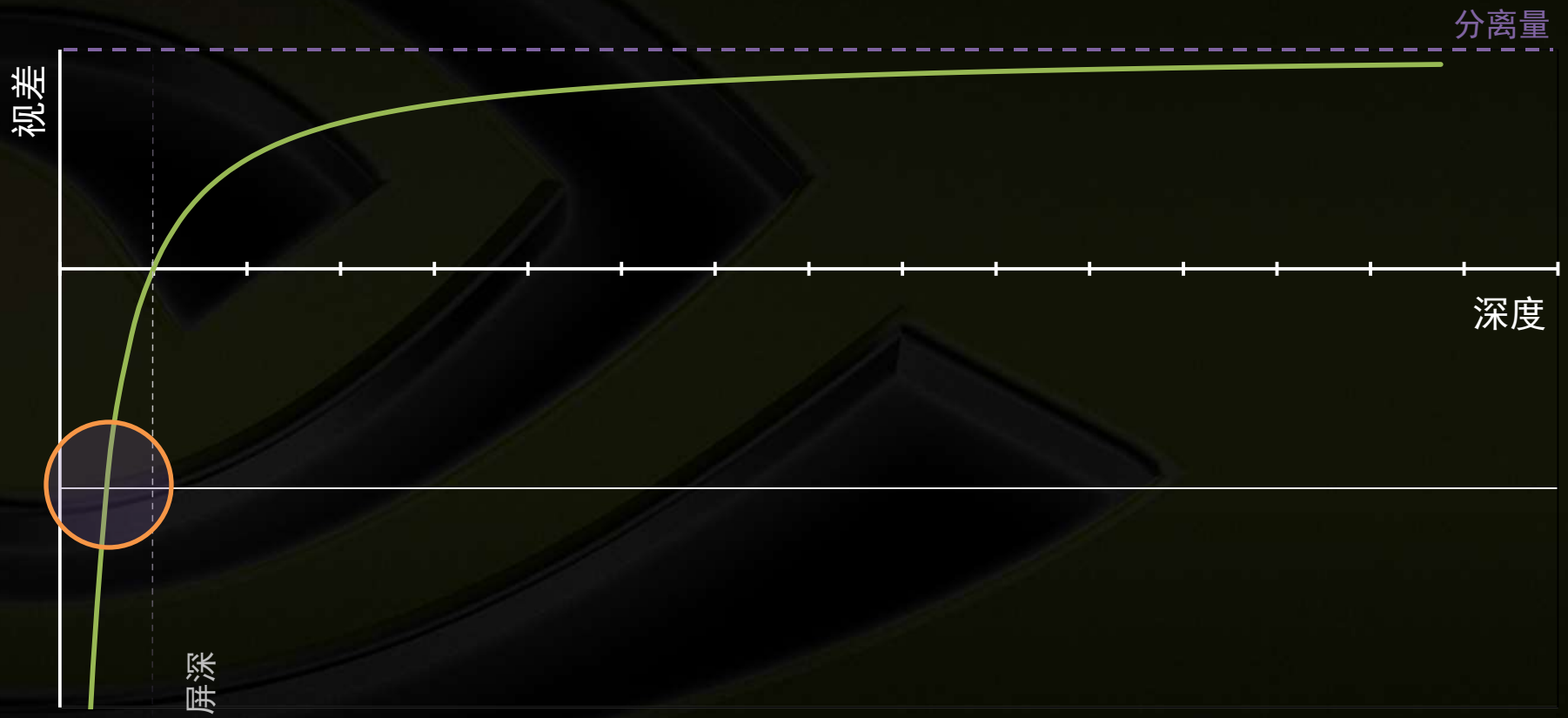
- 深度为100*屏深时，视差值达到轴间距的99%
  - 远于100*屏深的像素，物体看起来几乎没有立体感
- 在10到100*ScreenDepth之间，视差仅变化9%
  - 这个区间的物体立体感变化很细微



减少视觉疲劳和眩晕

视差取值范围：最近像素

- 当深度为屏深/2时，物体位于屏外，视差值等于负的分离量
  - 如果屏外物体的视差很大（大于分离量），可能造成视觉疲劳



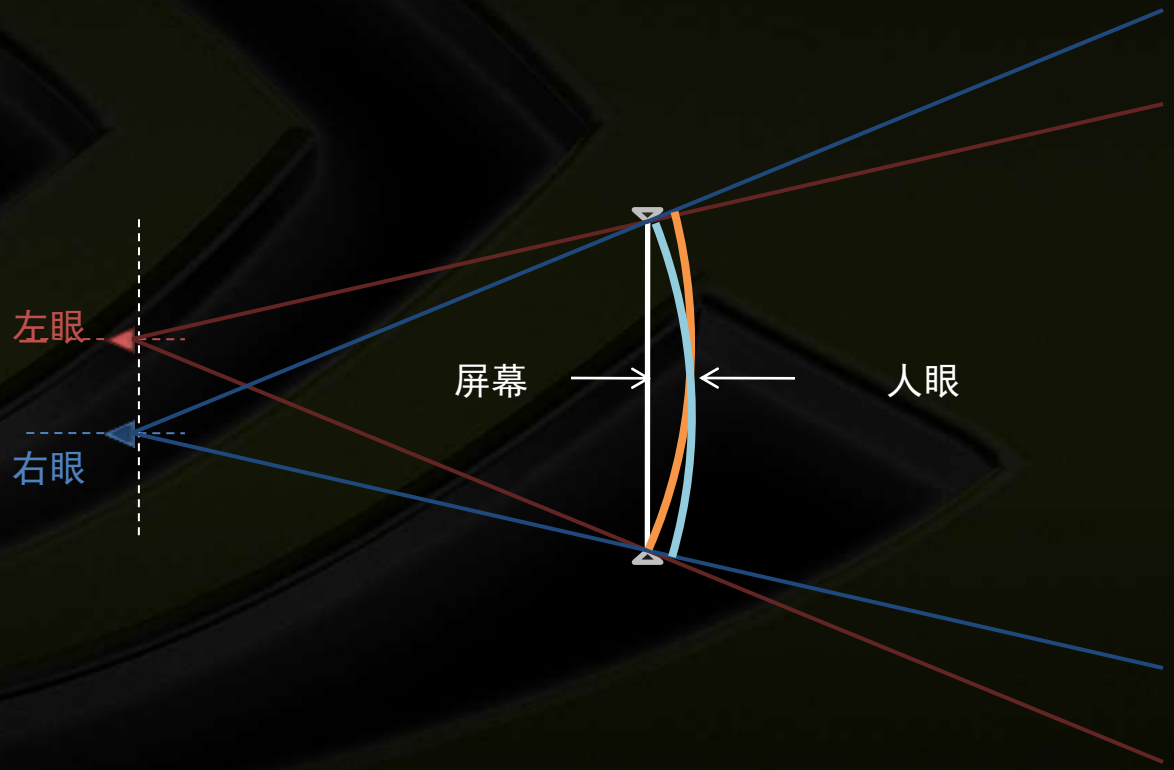
减少视觉疲劳和眩晕

屏深和舒适范围

- 屏深应当由程序决定，依摄像机和场景位置不同而不同
- 确保场景物体在以下范围
  [屏深/2, 100*屏幕深度]

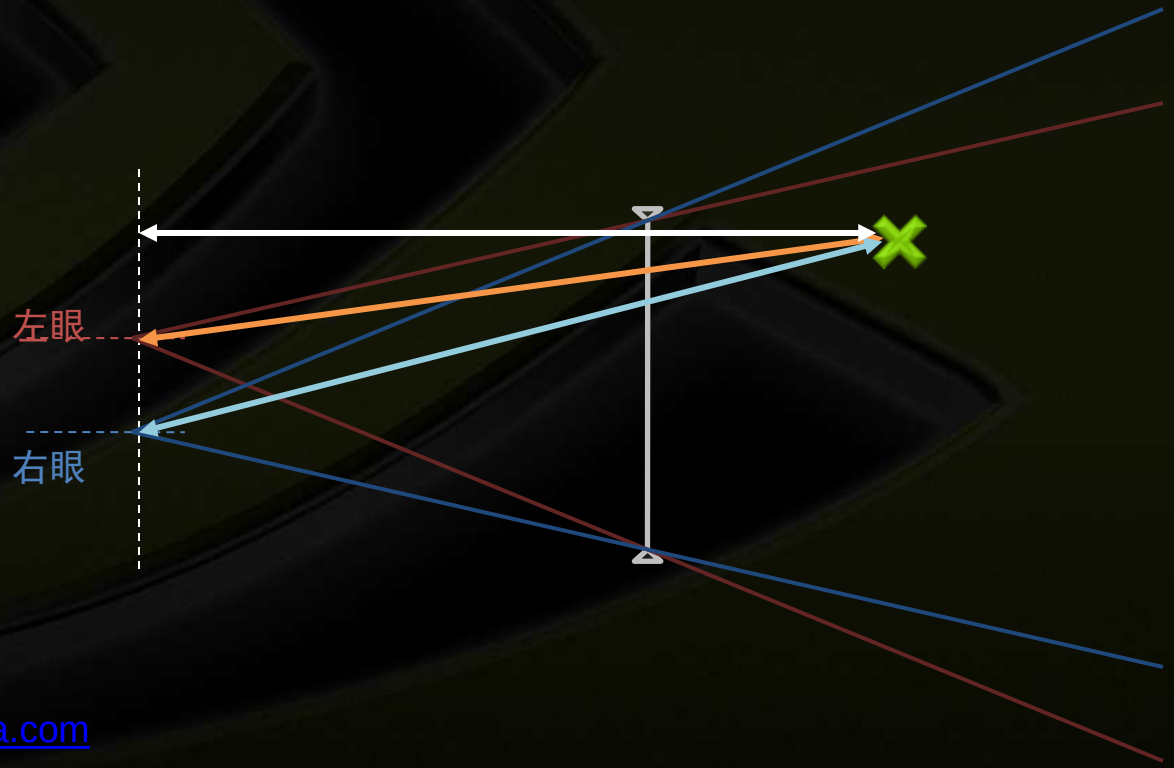
投影面

- 现有的图形管线都假设投影面是一个平面
- 实际上人眼的投影面更接近一个球面



场景深度

- 图形管线中的景深
  - 深度 = 点到眼平面的距离
- 人眼中的景深
  - 深度 = 点到眼睛的距离
- 二者之间差距过大会造成强烈的不适感



总结

- 合理按排景物的分布，尝试不同的屏深和分离量数值，可以有效降低立体成像的副作用
- 通过更多的测试来发现造成不适和眩晕的因素，并及时加以改正。

致谢

- **Samuel Gateau和John McDonald,** 开发者技术部门
- 立体驱动团队中的每一个人!

如何联系我们

- 网站: http://developer.nvidia.com
- 论坛: http://developer.nvidia.com/forums
- 游戏开发者技术支持: DevSupport@nvidia.com

问题?