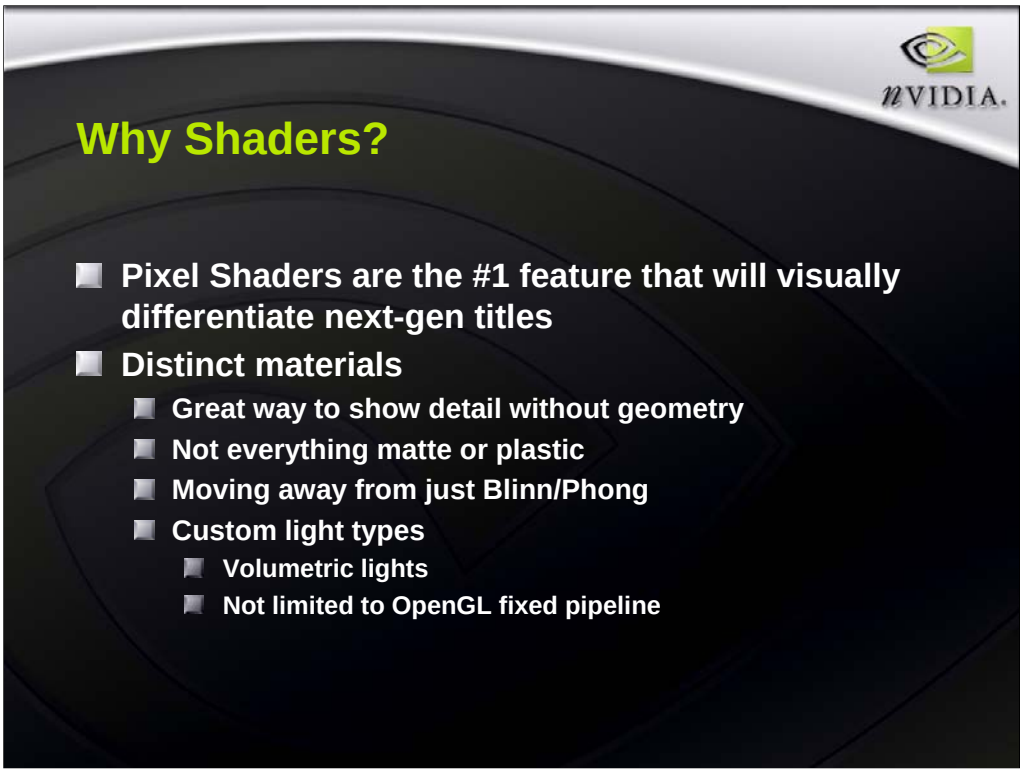

把shader集成到你自己的游戏引擎中

Bryan Dudash
NVIDIA Developer Technology


This is a fairly high level talk. Will cover some details and show some examples. But in general, this session is meant to make you aware of some of the details that you need to wory about when writing a shader system.
这是一个相当高水平的讲座。将会涉及细节并展示一些例子。
但是主要的目的是使你们对某些细节有一定的了解，当你们自己写shader系统的时候需要考虑这些细节

议程

•为什么需要shader?
•Shader到底是什么？
　　图形学的发展
•使用shaders




为什么需要shaders?

•像素shaders是区别于下一代名称的第一个特征

•不同的原料

不需要几何学就可展示细节得好方法

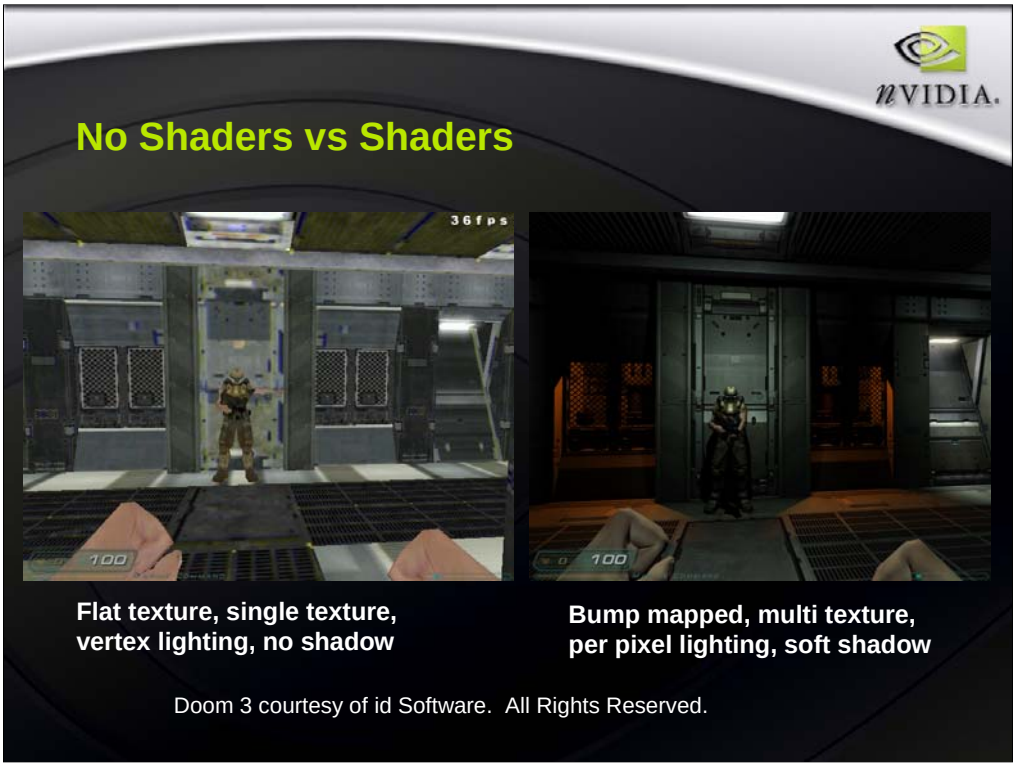
并不是所有的东西都是粗糙的或者可塑的

从Blinn/Phong着色方式解脱出来

自定义光源类型

立体光源

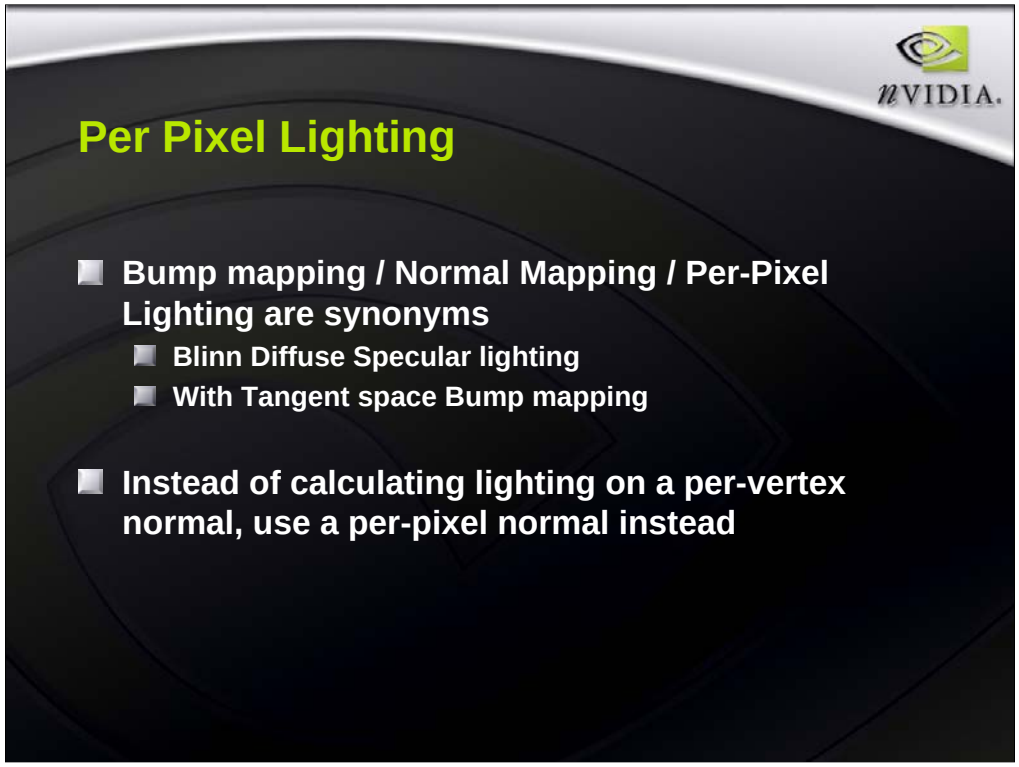
不限于OpenGl确定的渠道




没有shaders vs. shaders

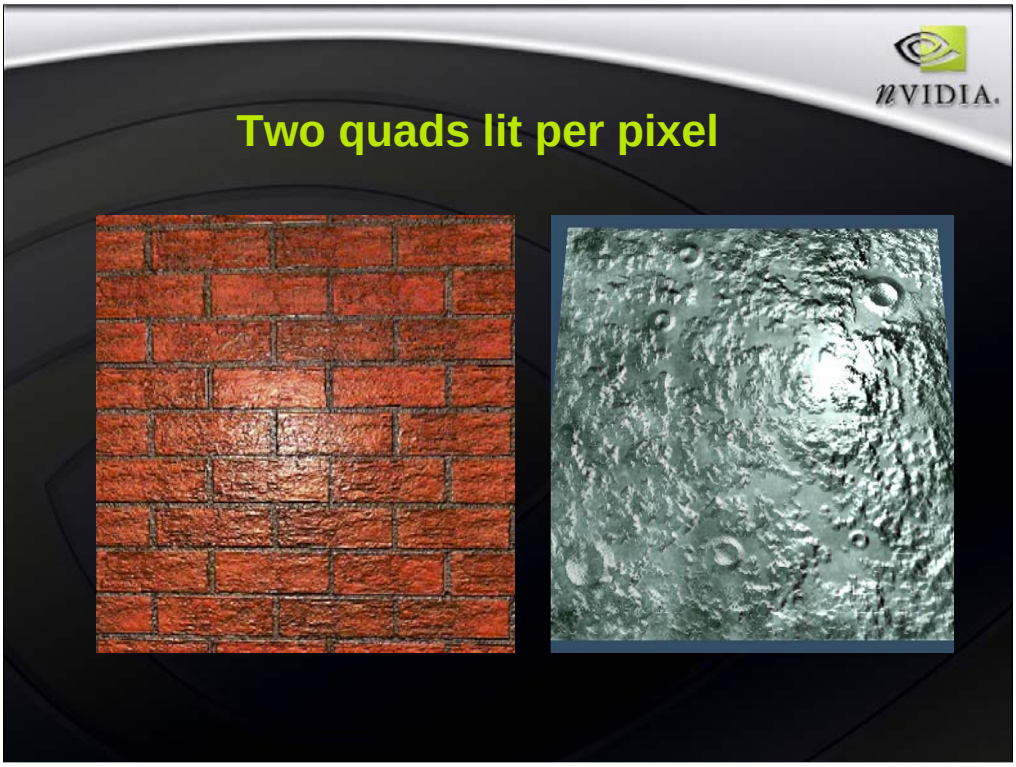
平坦的纹理，单文理　　立体贴图，多纹理

顶点光照，没有阴影　　单像素光照，弱阴影

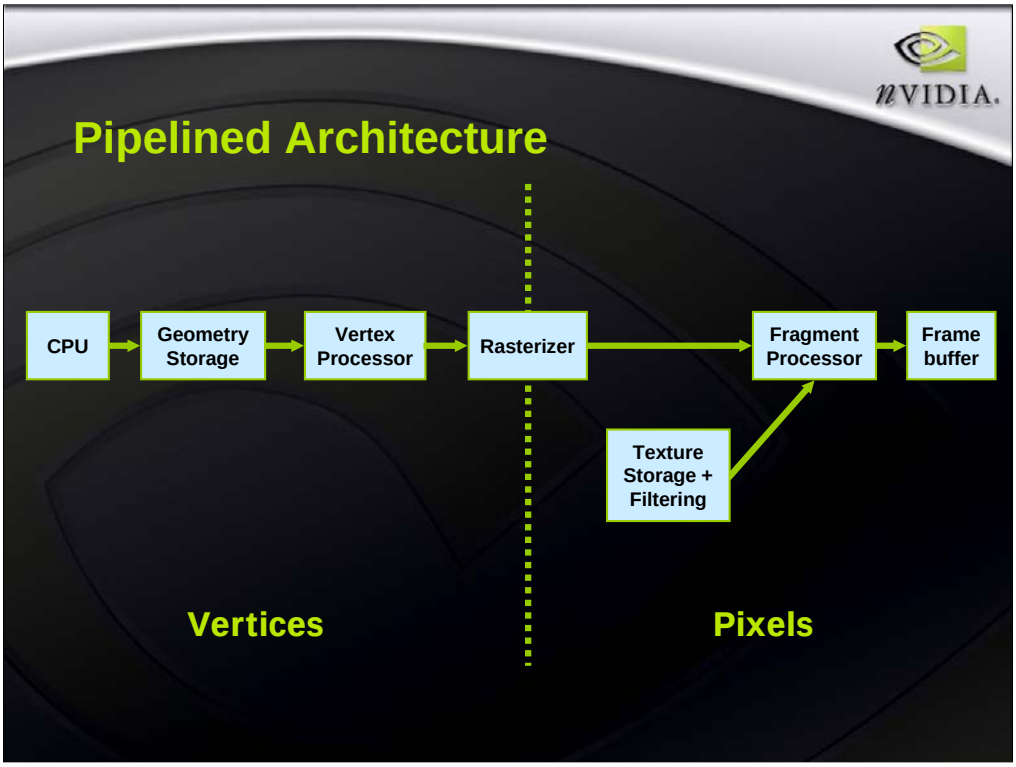



单像素光照

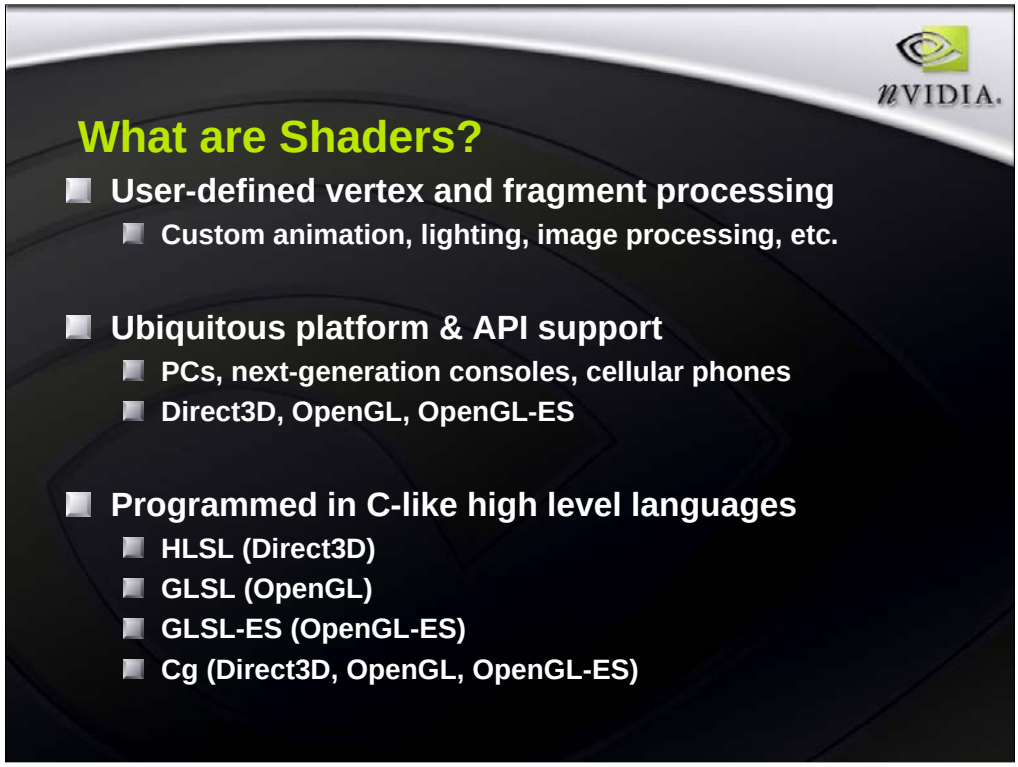
- 立体贴图/法线贴图/单像素光照都是同一个意思
  Blinn技术扩散反射光
  具有切线空间立体贴图
- 不需要计算在单顶点法向量上的光照，而是使用单像素法向量代替




两个四方快，每个像素被照亮


NVIDIA.
Pipelined Architecture
CPU
Geometry Storage
Vertex Processor
Rasterizer
Fragment Processor
Frame buffer
Texture Storage + Filtering
Vertices
Pixels

Pipeline构架




什么是shaders?

•用户定义的顶点和碎片处理

自定义动画，光照，图像处理，等

•普遍存在的平台&API支持

电脑，下一代控制台，便携式电话

Direct3D,OpenGL,OpenGL-Es

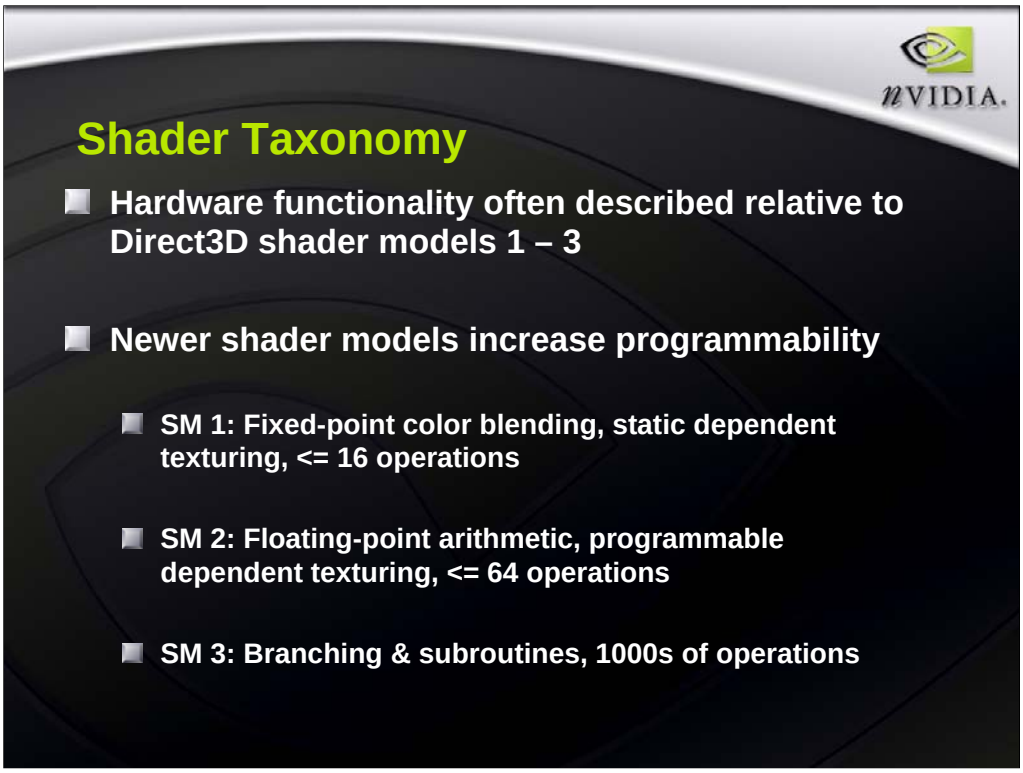
•用c高级语言编程

HLSL (Direct3D)

GLSL (OpenGL)

GLSL-ES (OpenGL-ES)

Cg (Direct3D, OpenGL, OpenGL-ES)




Shader分类

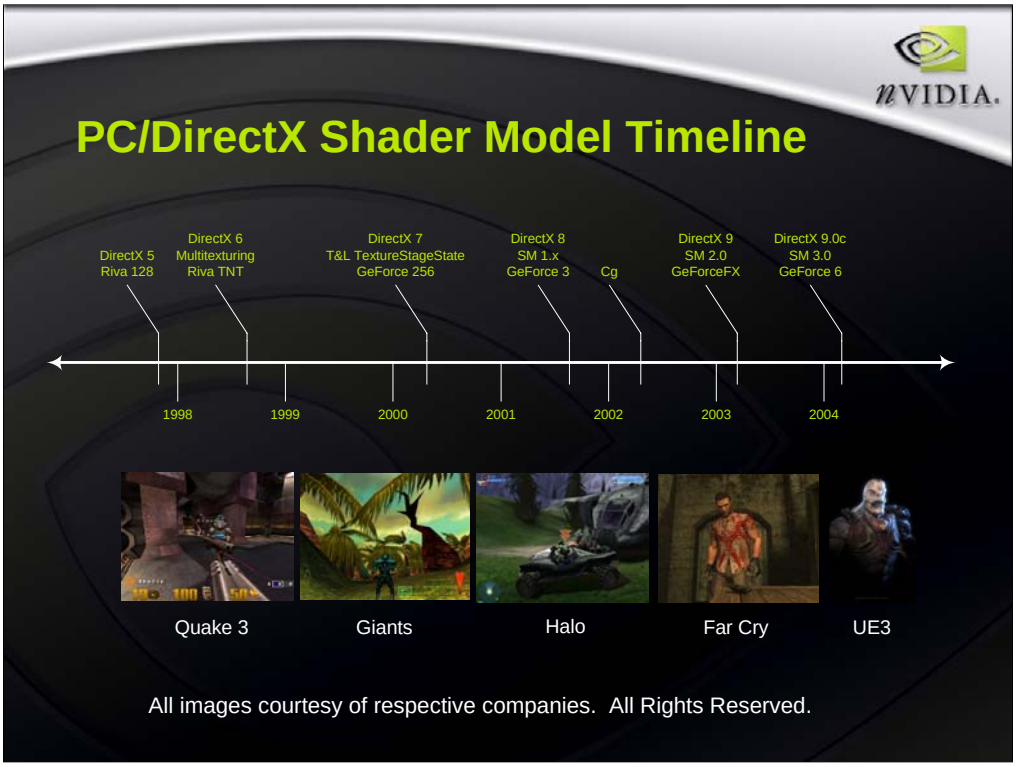
•硬件功能性通常被描述相对于Direct3D,shaders建模成1-3类

•新的shader模型增加了可编程性

SM1:定点颜色混合，静态纹理，小于等于16次操作

SM2:浮点算术，可编程的纹理，小于等于64次操作

SM3:分支&子程序，1000s操作




PC/DirectX shader模型时间轴


DirectX 8,SM 1.x/OpenGL1.4

- 可编程顶点shaders
  上限：128个浮点指令
- 可编程像素shaders
  上限：16个定点向量指令和4个纹理
  3维纹理支持
  上限：一阶的dependent纹理
- 高级的渲染-纹理支持
- 实例硬件

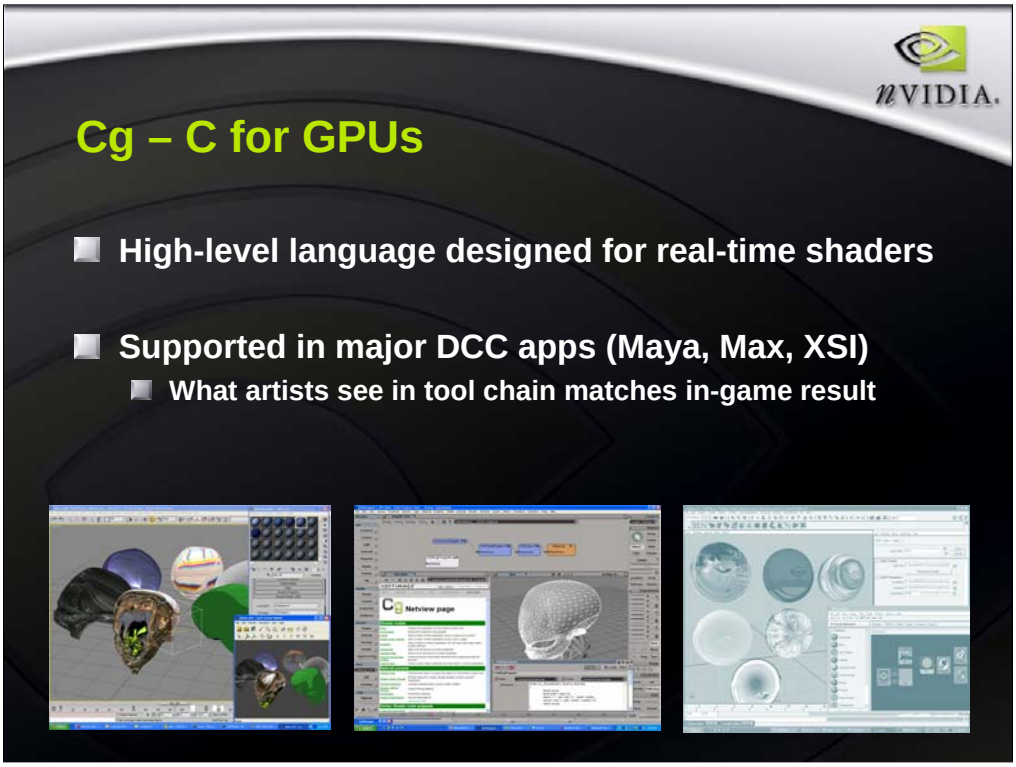
GeForce 3, ATI Radeon 8500, XGI Volari V3, Matrox Parhelia


SM1.x-era 游戏：Halo光晕

- 顶点shaders常用来增加冰面的fresnel反射
- 像素shaders常用来增加太阳的光亮
- 渲染-纹理常用来夸大手枪的射程
- Dependent纹理常用来动画$照亮水面


NVIDIA.
DirectX 7 vs DirectX 8
Halo courtesy of Microsoft. All Rights Reserved.

DX7和DX8比较

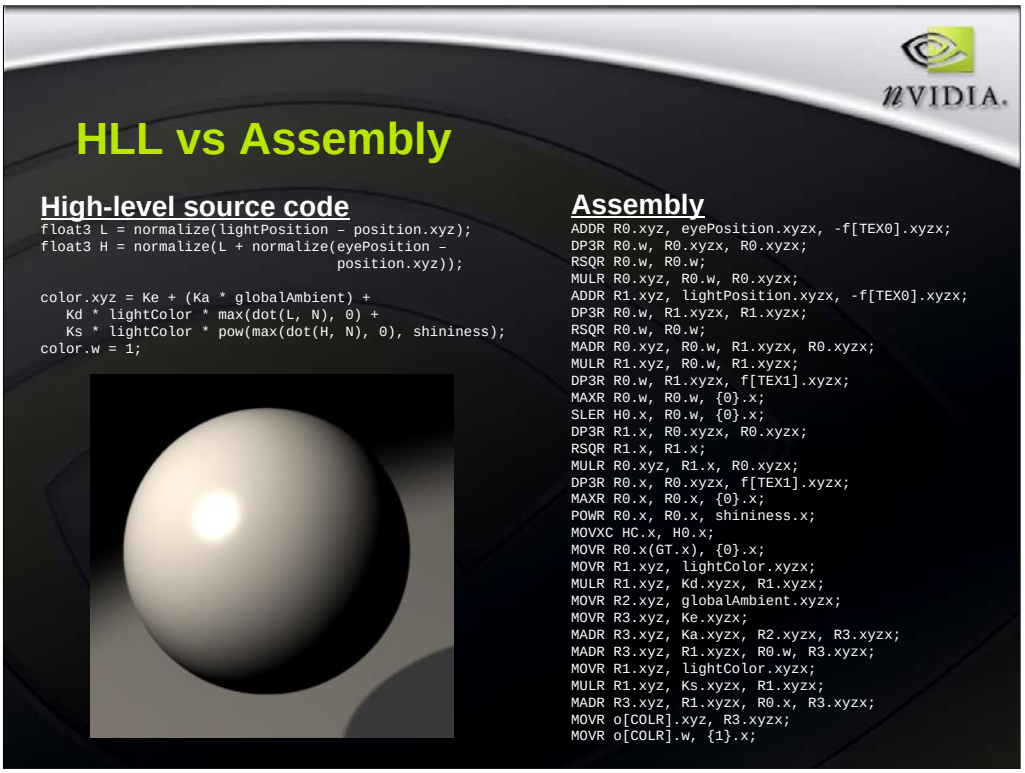



Cg—C for GPUs

•高级语言设计，实时渲染引擎

•大多数DCC apps（Maya, Max, XSI）支持

美工们在工具链上所看到的和游戏结果很吻合

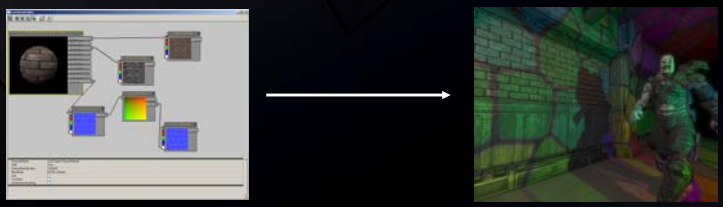



HLL vs. Assembly

高水平的源代码和汇编语言比较

## Impact of HLLs

- Dramatic increase in shader adoption
  - Tens of games per year to hundreds
- Shift in game development
  - Shaders become content requirement, not tech feature
  - "What do I want?", not "what can I do?"
  - Gives control of the look of the game to artists

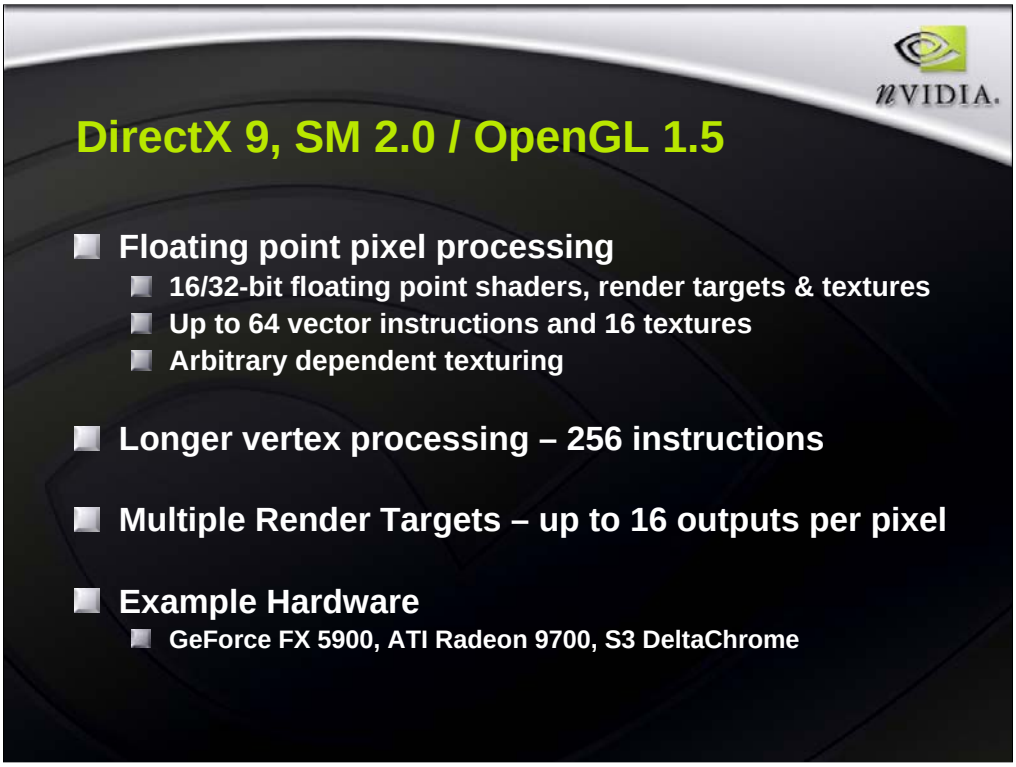




HLLs的影响

•在shader采用方面动态增加
　大量游戏每年增长到数百

•游戏发展的转变
　shaders已经变成一种需求，而不是技术学院的特征
　我们想要什么，而不是我们能做什么
　美工能控制游戏界面的外观




DirectX9,SM2.0/OpenGL1.5

•浮点像素处理

16/32位浮点shaders,渲染目标/纹理

上限：64向量指令和16中纹理

任意的dependent 纹理

•久的顶点处理-256个指令

•大量的渲染目标—上限：每个像素16个输出

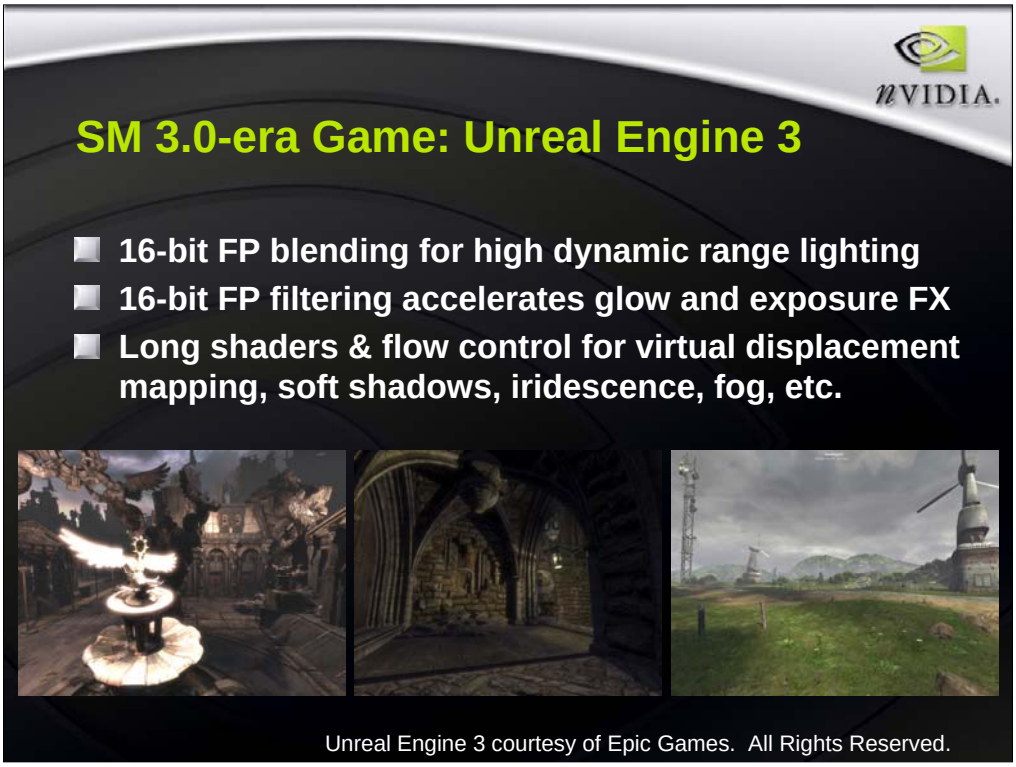
•实例硬件

GeForce FX 5900, ATI Radeon 9700, S3 DeltaChrome


DirectX9.0c,SM3.0/OpenGL2.0

- 统一的shader编程模型
  - 像素&顶点shader流控制
  - 无限长顶点&像素shaders
  - 顶点shader纹理查找表
- 浮点滤波&混合
- 几何实例
- 实例硬件

GeForce 6800, GeForce 7800 GTX



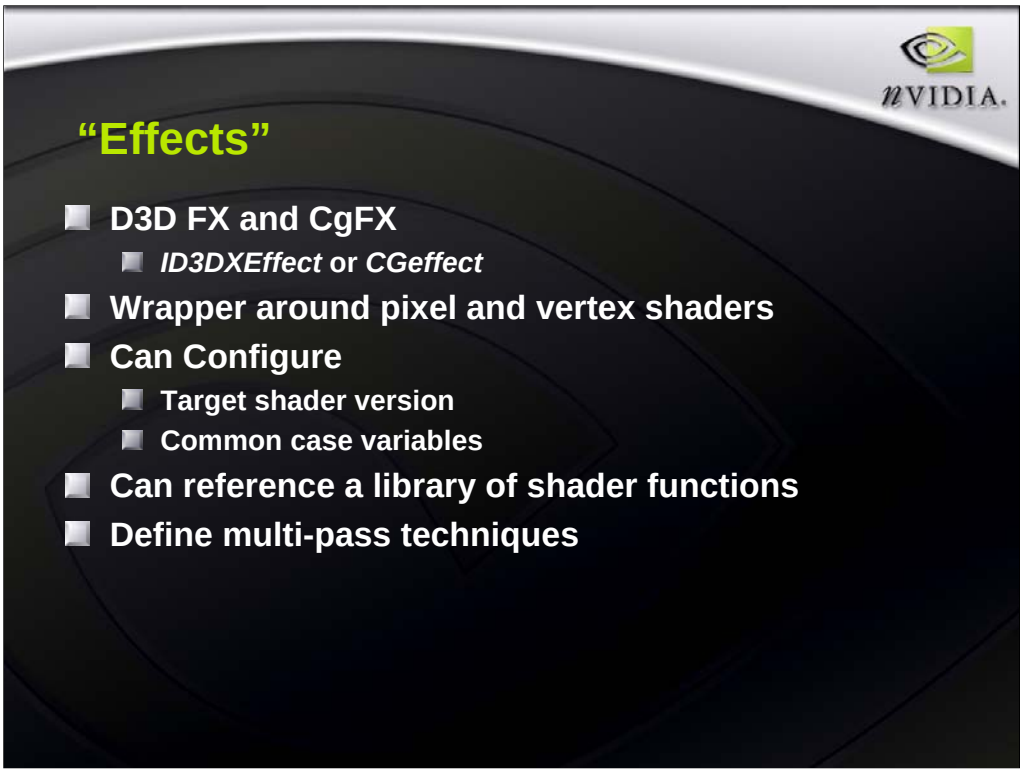

SM3.0-era 游戏：Unreal Engine3 虚幻引擎3

•16位 FP混合了高动态范围的光照

•16位 FP滤波加速了发光和暴露了FX

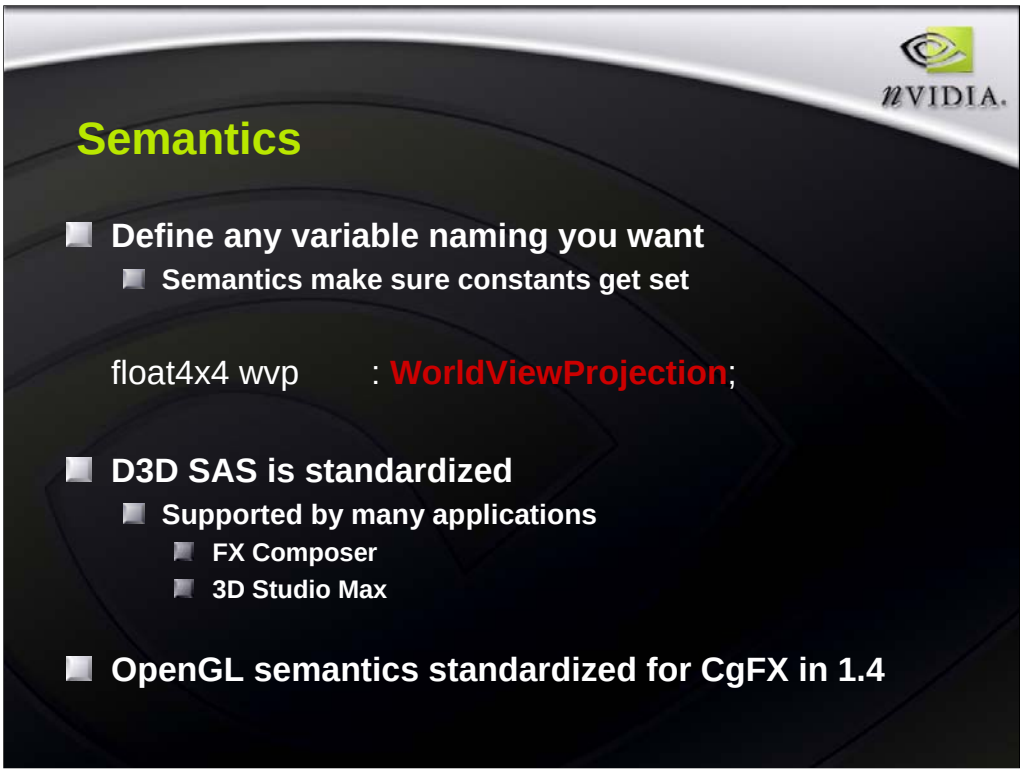
•长的shaders&流控制 适用于虚位移映射，弱阴影，彩虹色，雾等


使用shaders




On the code side, wrapping your high leel language in a FX file allows the shader author to control more about the execution environment. Can control passes, data bindings, etc.
在代码方面，高级语言封装在FX文件中，可以让shader作者去控制更多地执行环境。
控制通道，数据绑定，等。

"效果"
•D3D FX和 CgFX
*ID3DXEffect* 或者 *CGeffect*
•像素封装和顶点shaders
•能够设定
目标shader version
common case变量
•可以参考大量的shader函数
•定义多通道技术



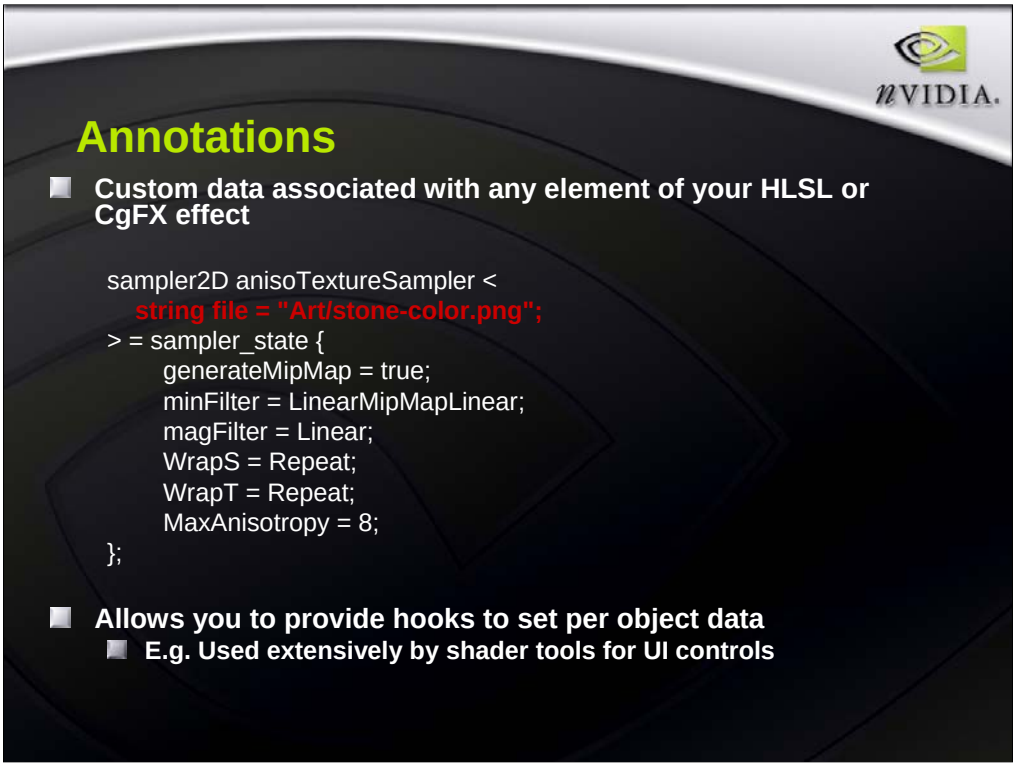

Key here is to let developers know that they can use many 3rd party tools to develop shaders, or let their technical artists develop the shaders, and then SAS can allow them to just drop in the shader without too much fuss. This allows only minimal guidelines to shader developers.

这里的重点是要开发者们知道他们可以使用很多的3rd party tools去开发shaders,或者让他们的技术美工去开发shaders,

然后SAS使得他们能迅速熟悉shader，不需要惊慌。这些只给shader开发者们展示了最少的指导要点。

What about a person(or team) dedicated solely to developing the shaders. They don't have to know C++ or the codebase, just how to write and develop in CgFX…

一个人（或团队）独自专注于shaders的开发会



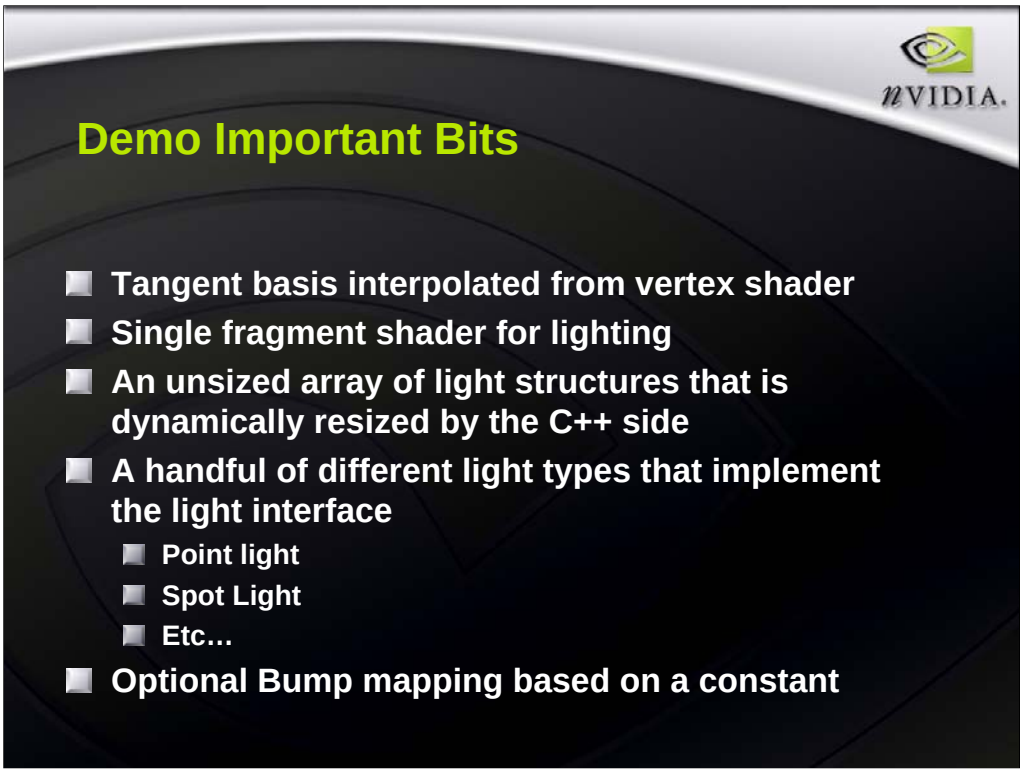

注解：

•结合你自己的HLSL或CgFx中的要素的自定义数据

•你将会对设定每个物体数据感兴趣

比如，利用shader工具控制用户界面，被广发应用


CgFx 语义演示




演示重要的Bits

•从顶点渲染引擎进行正切基插值

•用于光照的单断片渲染引擎

•一系列尺寸不一的光亮结构，根据附带的c++动态改变大小

•不同的光照类新，实现光界面

点光源

聚光灯

等


单光照函数

•对于基础颜色采样Albedo map

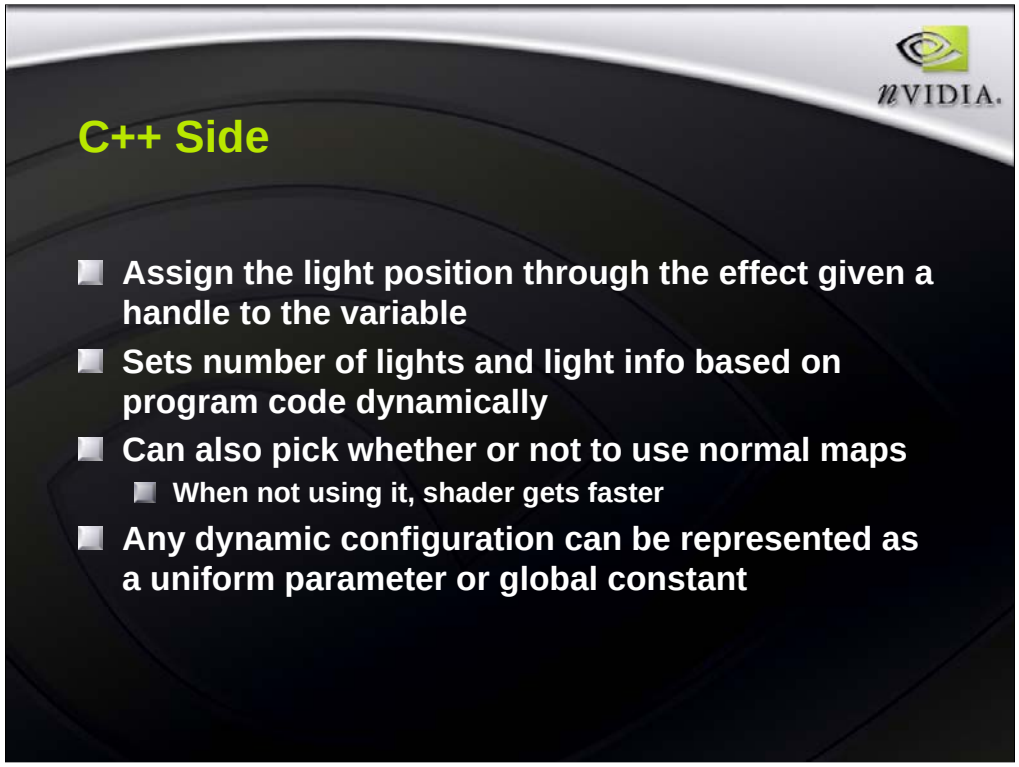
•归一化插值向量

　正切空间基向量

•基于法线贴图，随意地perturb我们的法向量

•迭代，并收集扩散光和反射光

•结合颜色和光亮值，产生最终的结果




附带的C++

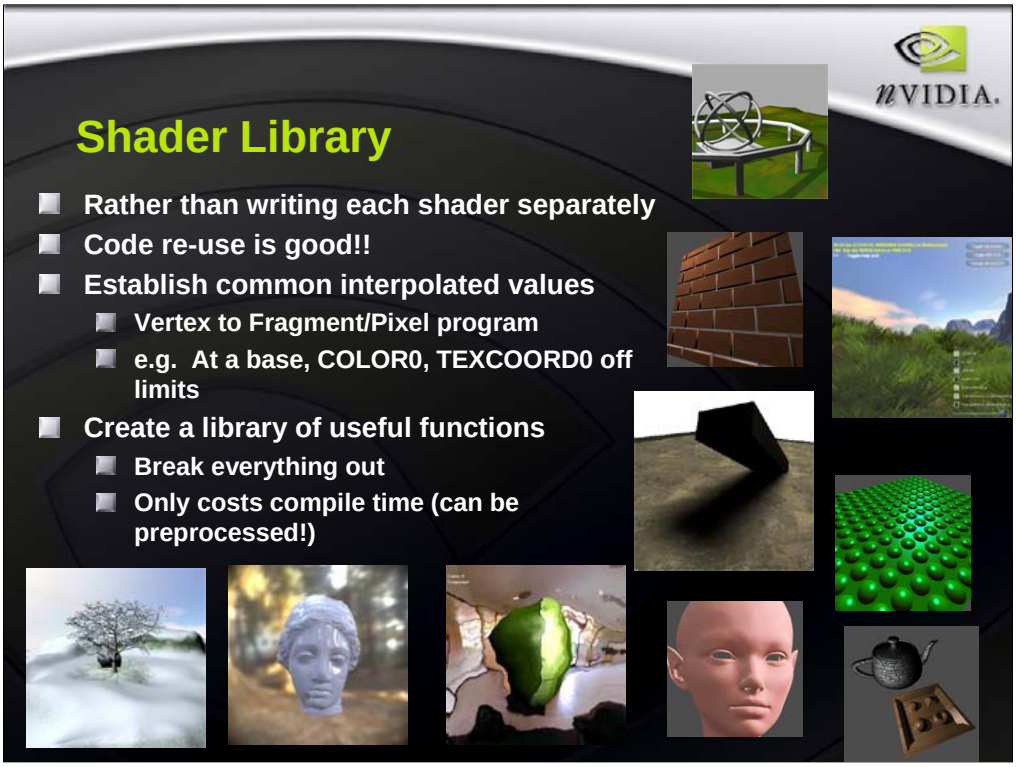
•根据变量的句柄，通过效果分配光源的位置

•基于程序代码动态地设置光源的数量和相关信息

•可以选择是否使用法向量贴图

当不用时，shader速度变快

•任意一个动态的配置可以表示成一个统一的参数或全局常量




Shader 库

•不需要单独写每个shader

•代码重复使用性很好

•建立通用的插值

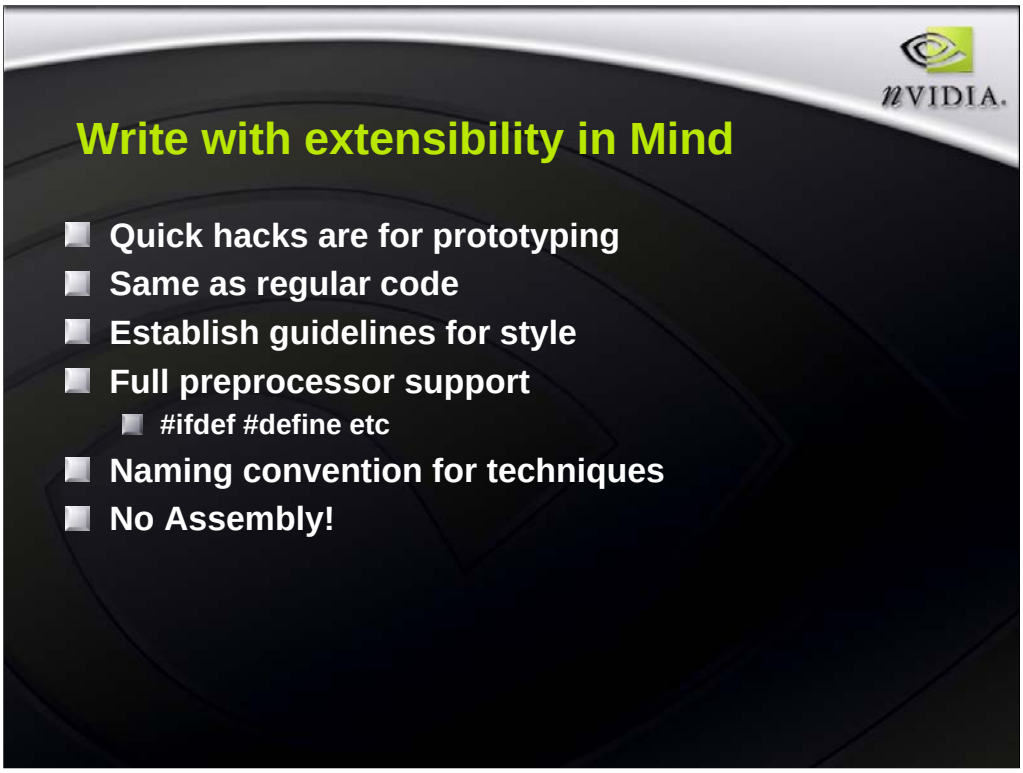
顶点到碎片/像素 程序

比如，基本的，COLOR0,TEXCOORD0 off limits

•建立一个有用的函数库

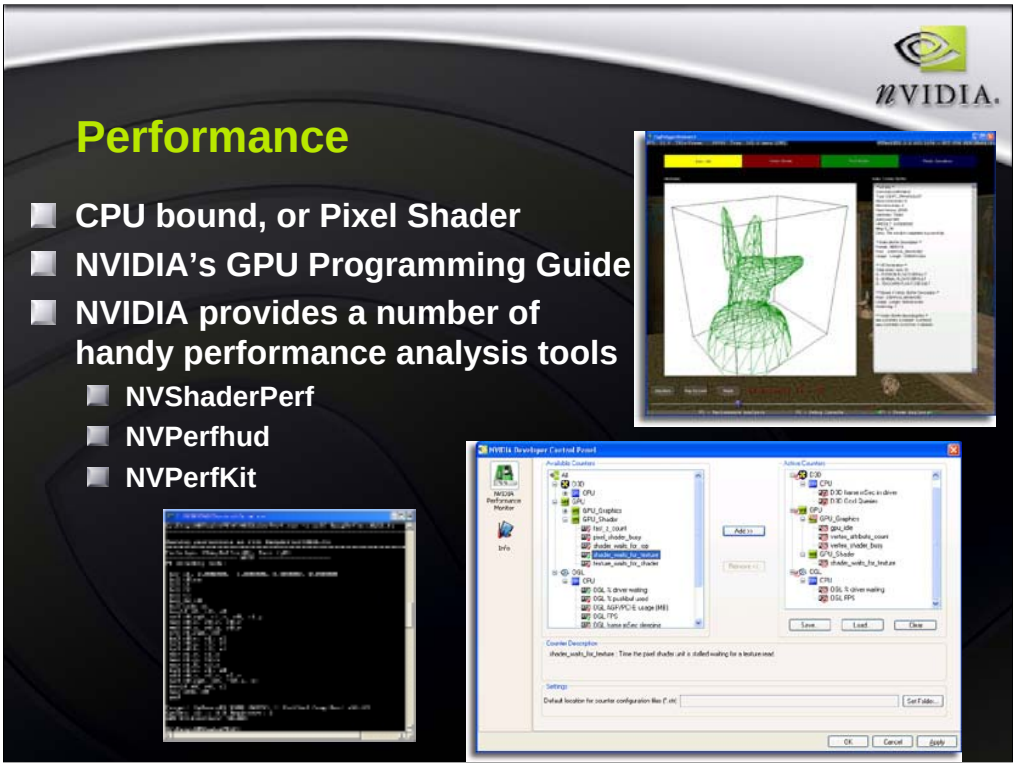
分解

只需要花费编译时间（能够预处理）




可延展性

- 快速使用现成的原型
- 和常规代码一样
- 为风格建立指南
- 完全预处理程序支持
  #ifdef #define etc
- 按照惯例来命名
- 无汇编




性能

- CPU绑定，或像素渲染引擎
- NVIDIA的GPU编程指南
- NVIDIA提供了大量的性能分析工具
  - NVShaderPerf
  - NVPerfhud
  - NVPerfKit

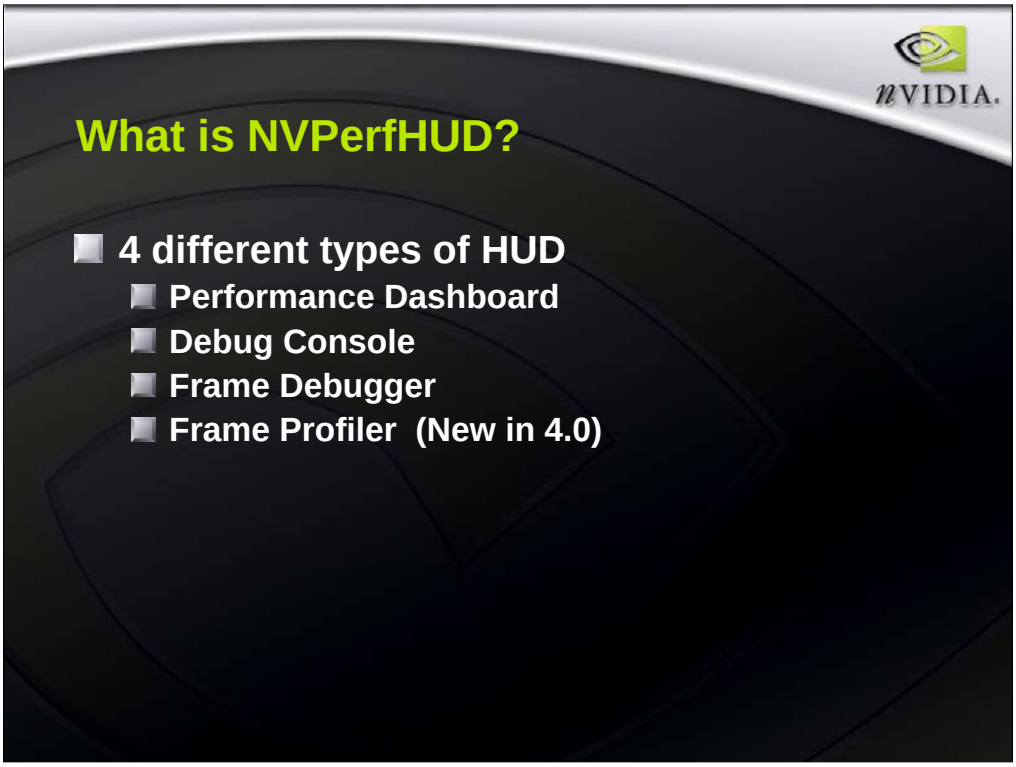

**NVPerfHUD**

- 什么是NVPerfHUD?
- 它是如何工作的?
- 发布时间表


# 什么是NVPerfHUD？

**PERFormanceHeads Up Display** 的缩写

- 在应用程序之上的重叠图和对话框
- 交互式平显 **Interactive HUD**




什么是NVPerfHUD？

- 4种不同类型的 平显HUD
  1) 性能仪表盘（Performance Dashboard）
  2) 调试终端（Debug Console）
  3) 帧调式器（Frame Debugger）
  4) 帧探察器Frame Profiler (4.0版本新增)

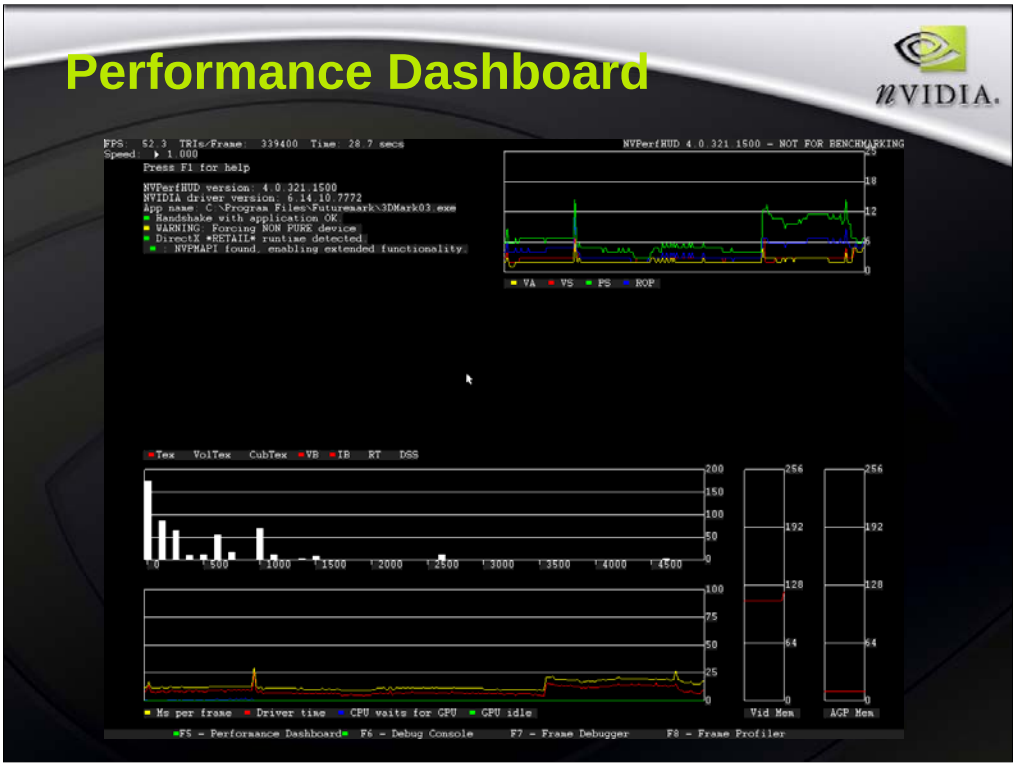

如何使用？

•运行应用程序时，同时运行**NVPerfHUD**

•进行正常的操作，直到你发现以下情况：

**1)** 函数问题：利用帧调试器

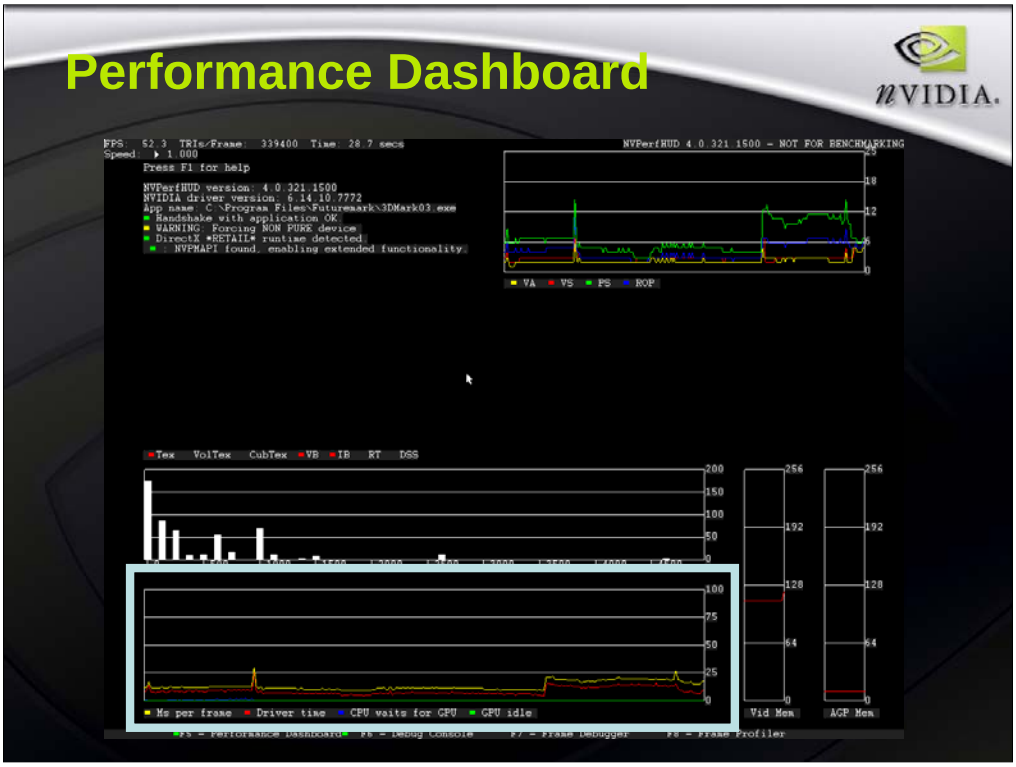
**2)** 低帧率：利用帧探察器




性能仪表盘（**Performance Dashboard**）

before going to the demo id like to show some slides about each of the performance gauges we are going to find.

在我们开始演示之前，我会用几张幻灯片来讲讲我们要看到的一些性能测量程序




**性能仪表盘（Performance Dashboard）**

底部色框所示：

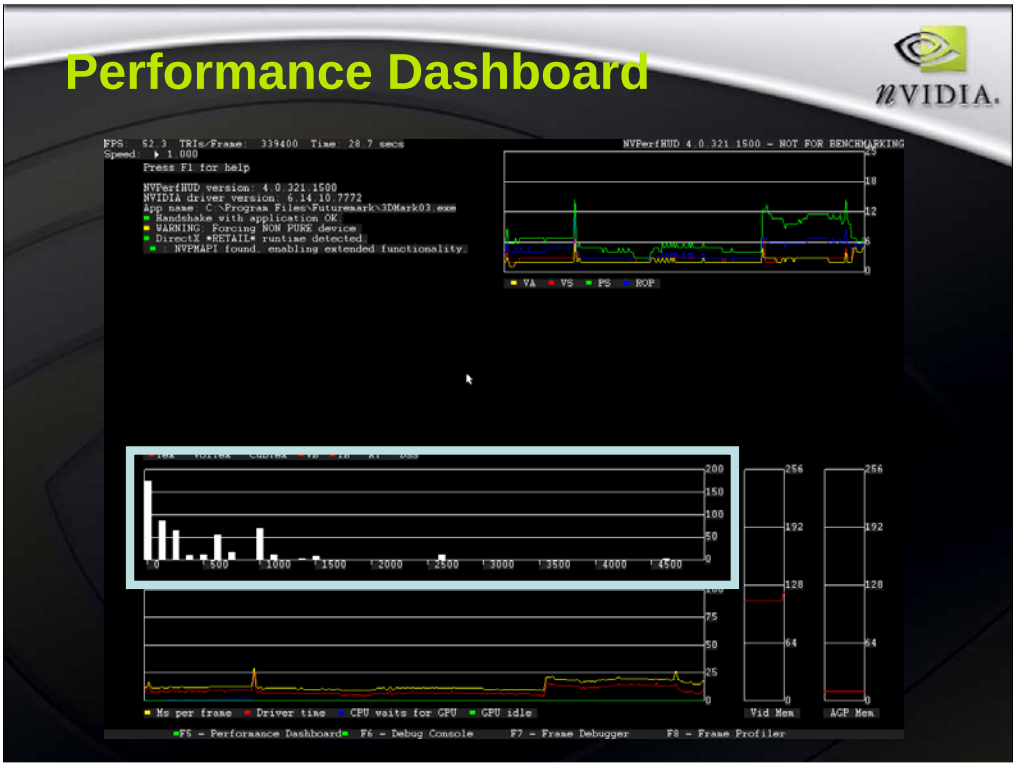
这是一个动态滚动图，每毫秒画一次

frame time（帧速）

driver time （驱动时间）

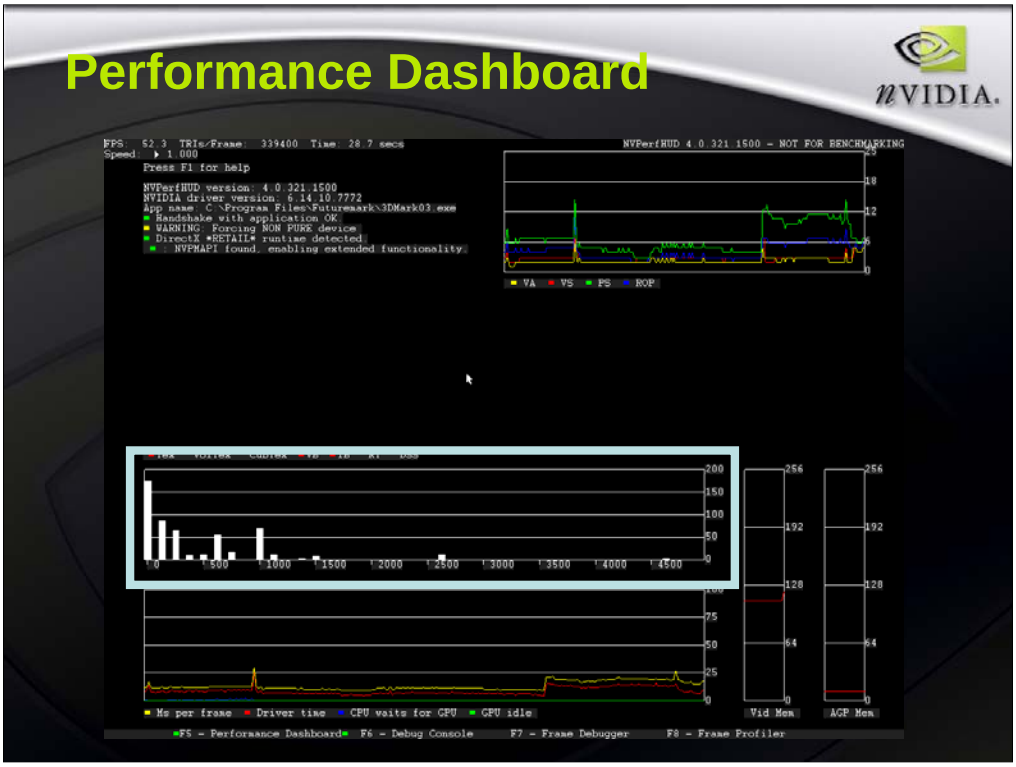
graphics idle （图形等待）

CPU waiting for GPU time （CPU等待GPU时间）




**性能仪表盘（Performance Dashboard）**

上面的色框是另外一个动态滚动图，显示了每一帧的DP数量，如果我按“B”键，就会跳到下一张幻灯片

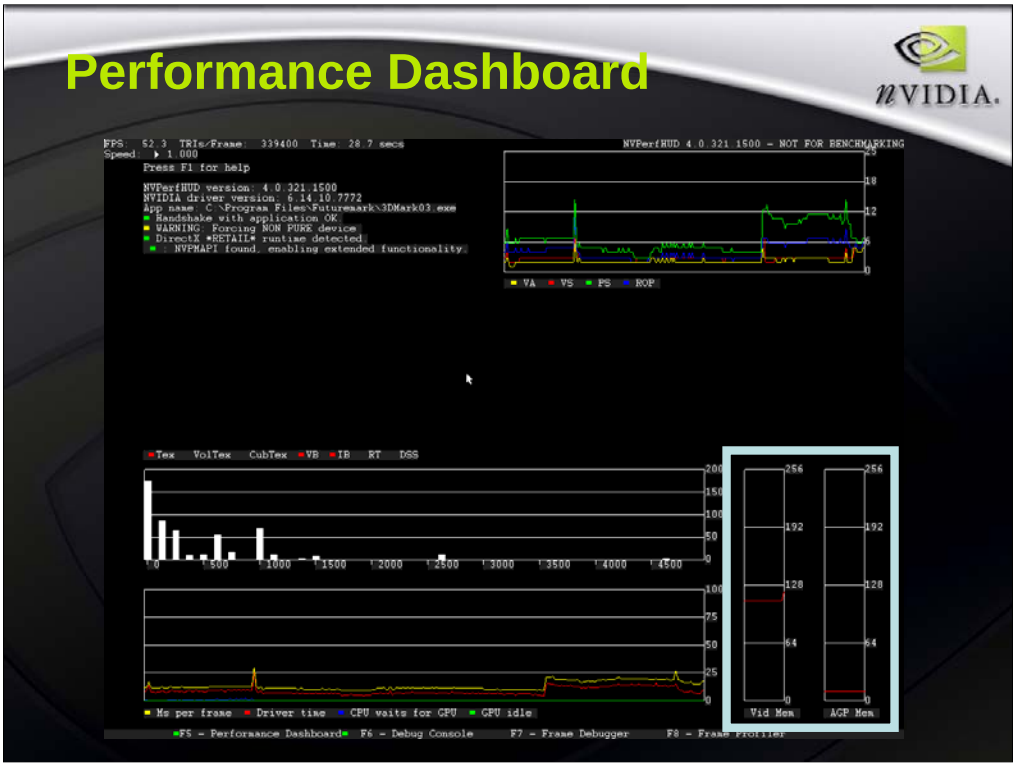



**性能仪表盘（Performance Dashboard）**

然后，我们看到，色框里面的图变成了直方图，

表示每次调用DRAW函数来画出几何体的数目，

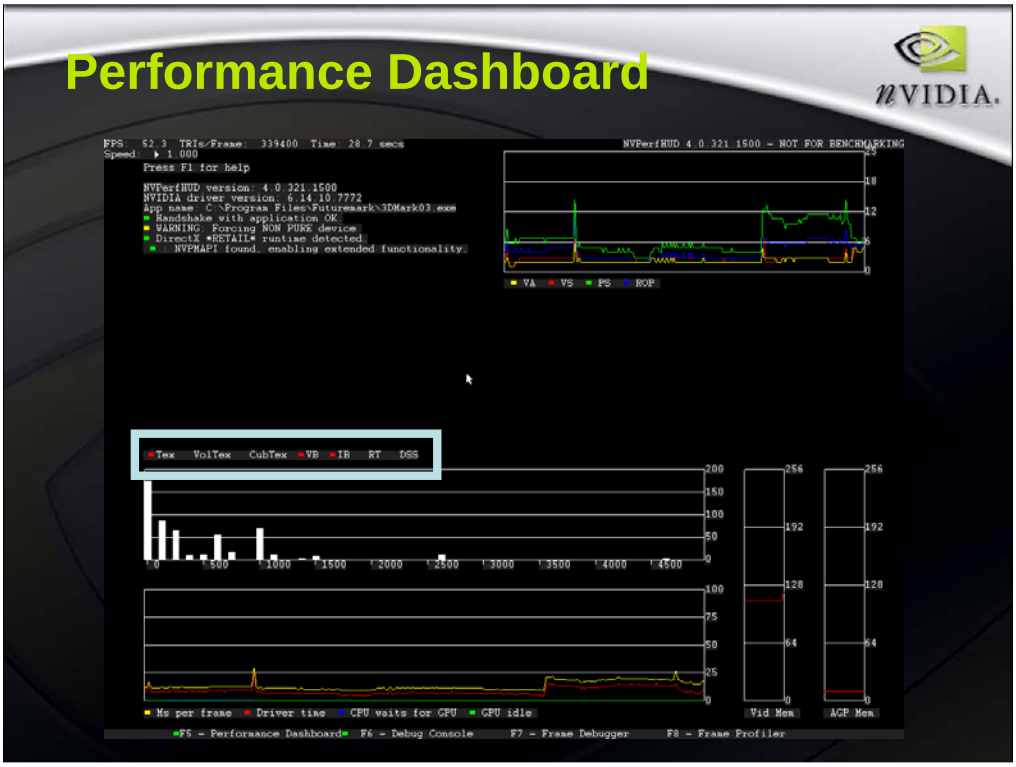
这里我们可以看出，很多调用要画出的三角形数量都在0~100之间。这样，我们可以批处理那些几何体数目少的DP，由此获得高帧率




**性能仪表盘（Performance Dashboard）**

在右边，我们看到另外2个动态滚动图，来显示GPU上可用的和已分配的内存

现在我们来看看资源监视器（resource monitor）




它在这里呢，我们去下张幻灯片，看个详细


性能仪表盘（**Performance Dashboard**）

•资源监视器（**Resources monitor**）

•可以监视的资源包括：

**Textures**

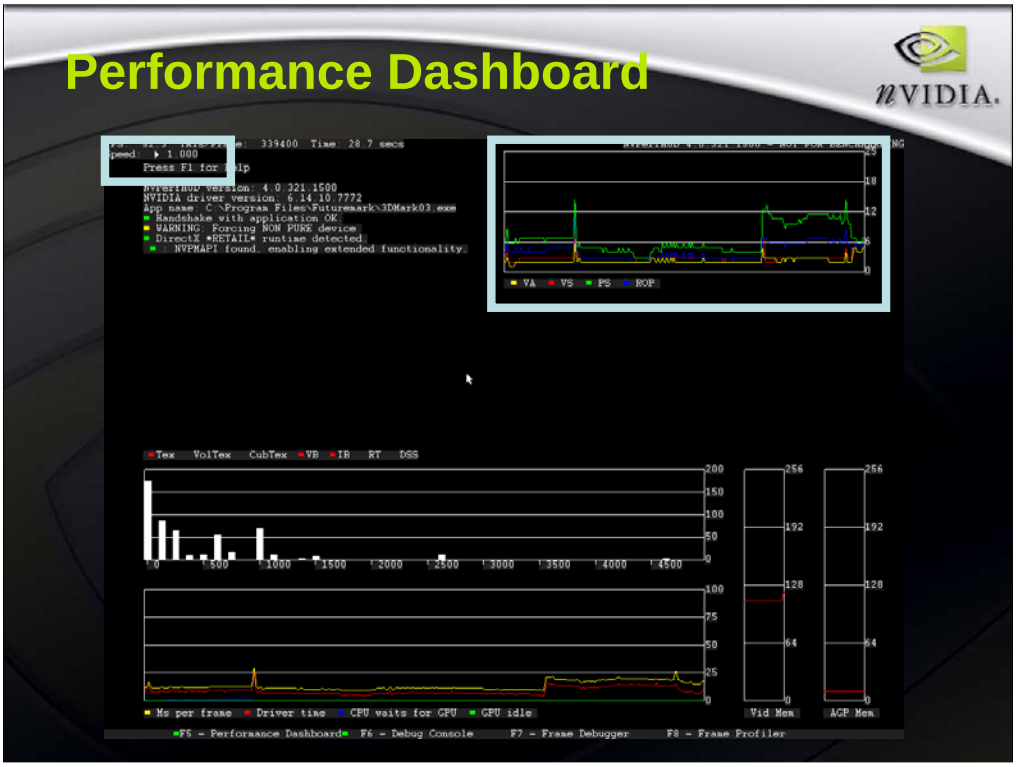
**Volume Textures**

**Cube textures**

**Vertex Buffers**

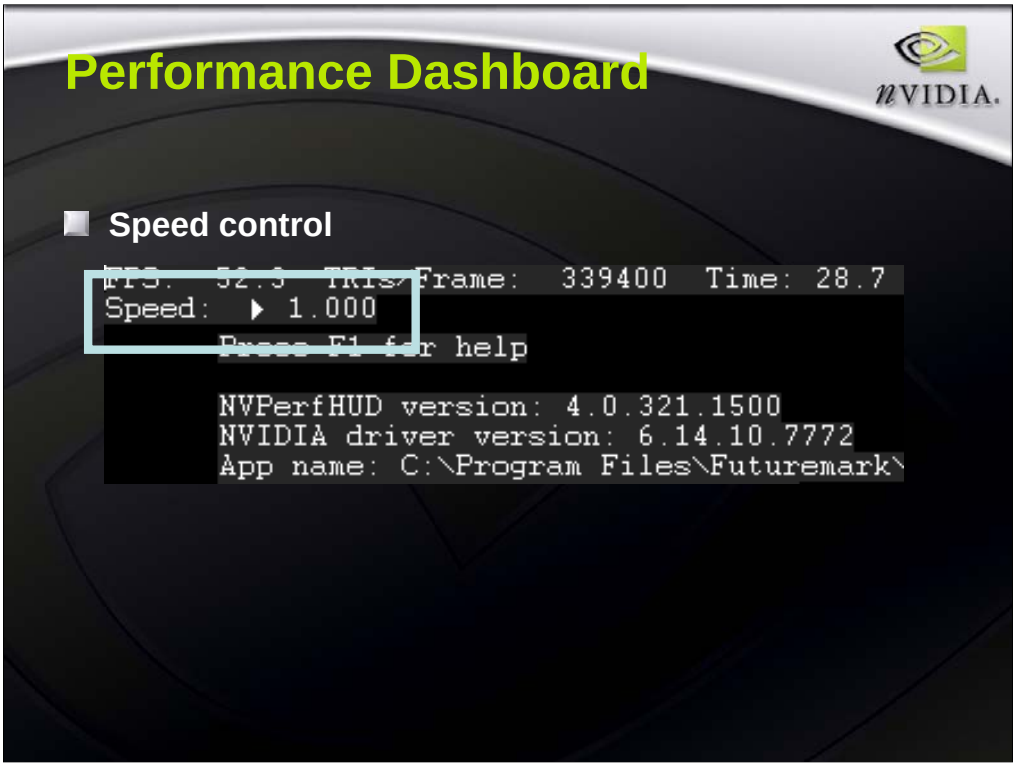
**Index buffers**

**Stencil and depth surfaces**




**性能仪表盘（Performance Dashboard）**

4.0 版本里会有2个新特性

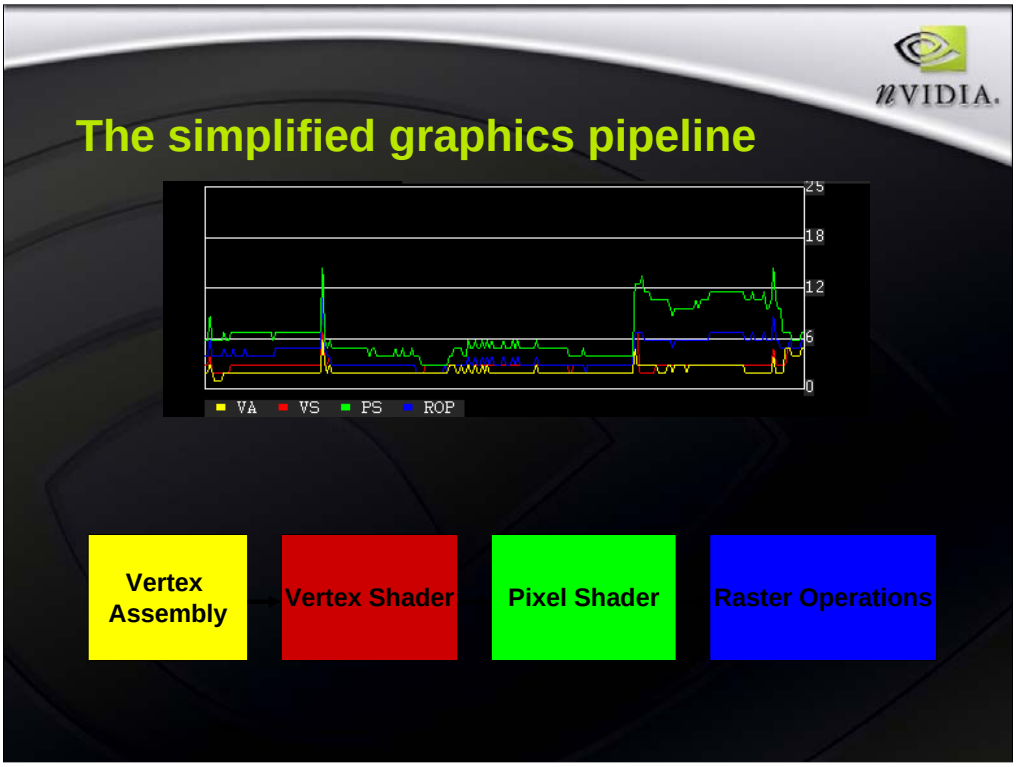



**性能仪表盘（Performance Dashboard）**

另外，在4.0版本里，我们还有一个速度控制功能，可以用来改变应用程序的速度，甚至完全冻结某个程序。

这样，如果我们的游戏里面有个爆炸场面，运行不是很正常，我们就可以将它的运行速度降下来，然后在有问题的地方运行调试程序。

随便我们看下底部的状态条，它告诉我们，当前是哪个模式（4种模式）处于活动状态。



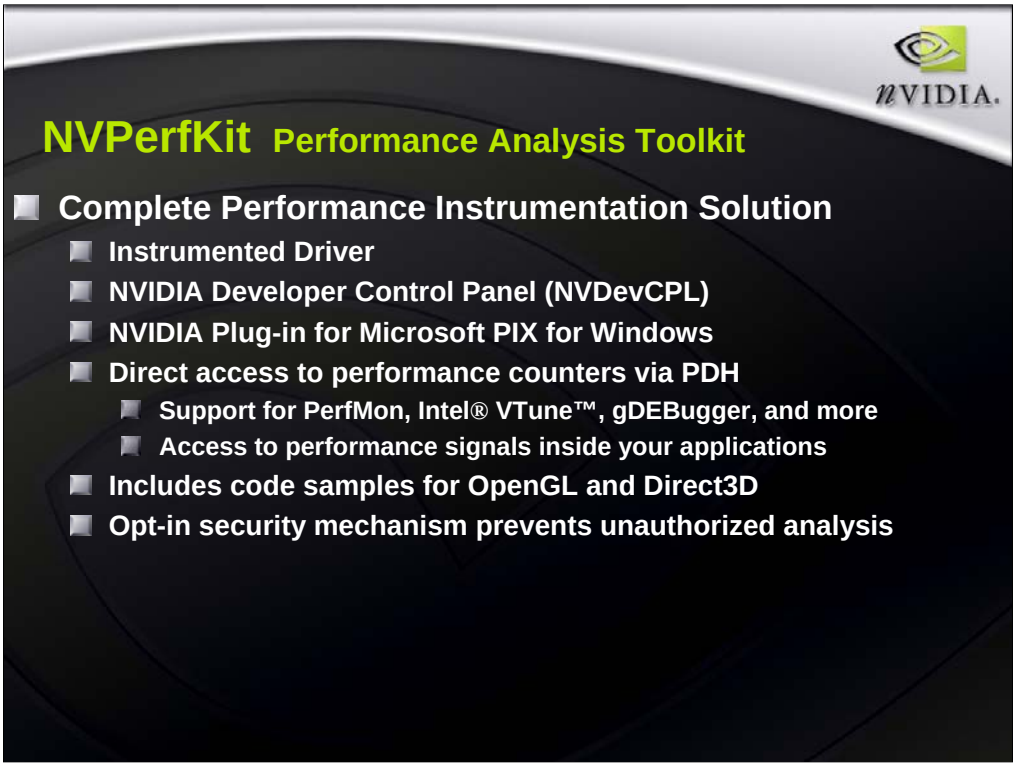

简化的图形通道

顶点组装（Vertex Assembly） 顶点着色器（Vertex Shader）像素着色器（ Pixel Shader） 光栅操作（ Raster Oeerations）

在这张图中，我们可以看到图形通道中4个主要单位的使用效率。


时间表

- Beta测试版：8月
- 正式发布：9月




**NVPerfKit Performance Analysis Toolkit**

•完整的性能管理解决方案

集成的驱动程序**Instrumented Driver**

**NVIDIA Developer Control Panel (NVDevCPL)**

**NVIDIA Plug-in for Microsoft PIX for Windows**

**Direct access to performance counters via PDH**（通过**PDH**直接访问性能计数器）

支持**PerfMon**、 **Intel®VTune™**、 **gDEBugger**和其他更多调式程序

访问程序内部的性能信号指示（**Access to performance signals inside your applications**）

包含**OpenGL**和**Direct3D**的示范代码

**Opt-in**安全机制，防止未授权的分析

instrumented driver（集成的驱动程序）提供了关于应用程序使用GPU的详细信息

使用NVIDIA Plug-in， Microsoft PIX for Windows 里面可以使用NVPerfKit counters

PDH是Performance Data Helper API component of Windows Management Instrumentation (WMI).

opt-in安全机制可以防止未授权的第三方来分析程序


**NVPerfKit Instrumented Driver**

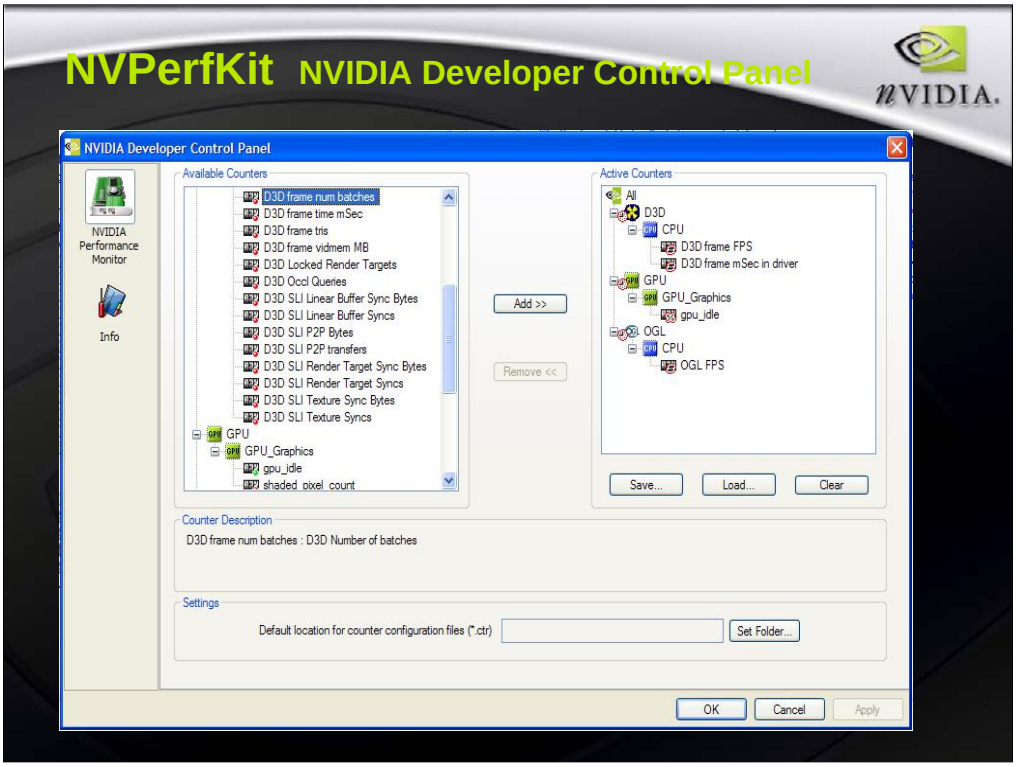
•提供GPU和驱动程序性能计数

•支持OpenGL和Direct3D

•支持 SLI Counters

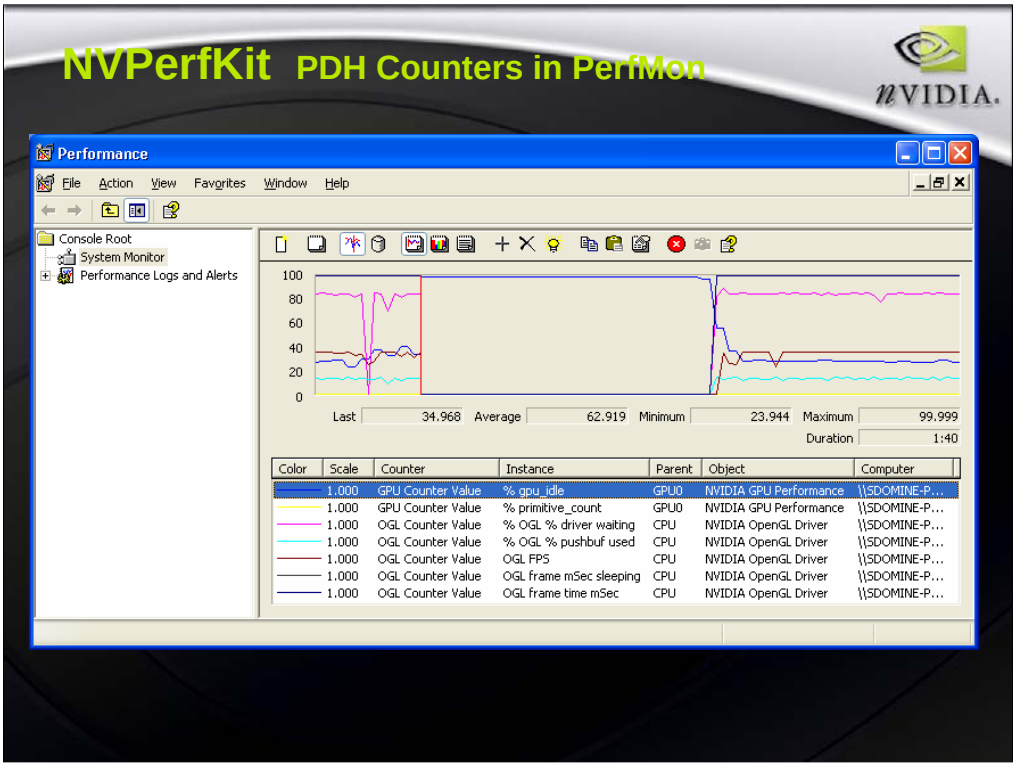
•需要GeForce FX或更高级产品支持

GeForce 6系列及后续产品中还有很多更好的计数器可用

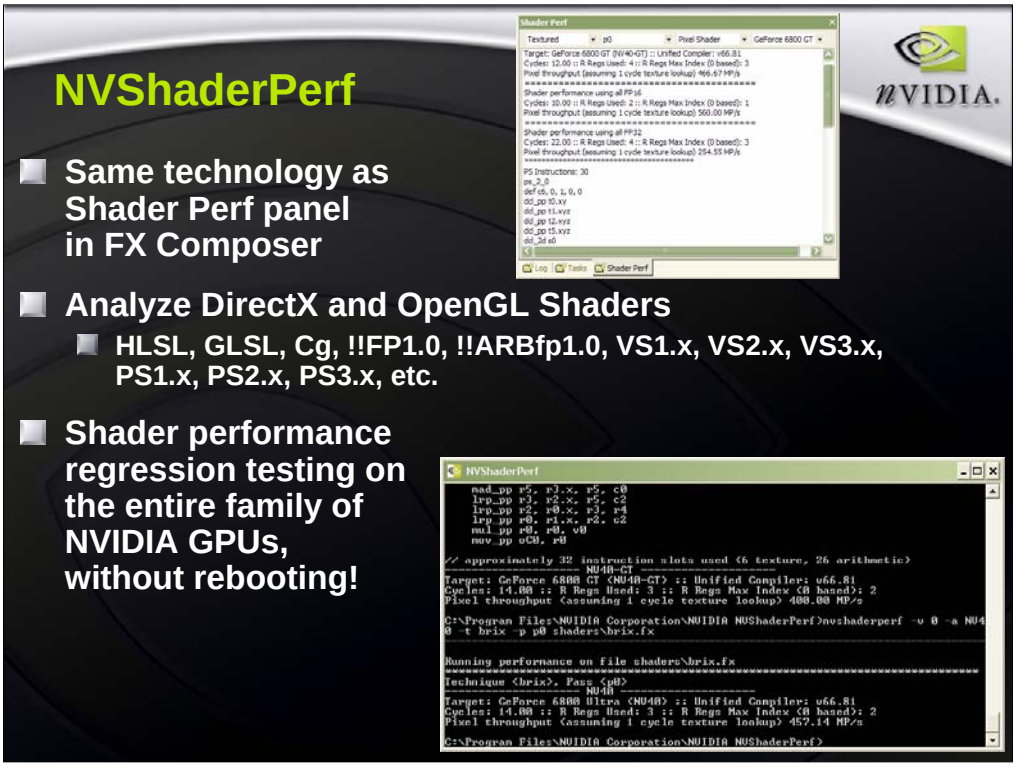



NVPerfKit：NVIDIA Developer Control Panel（开发者控制面板）

利用NVDevCPL来配置计数器，哪些计数器会由驱动程序/硬件来报告。
管理方便，可以保存或重载配置参数。




NVPerfKit： PerfMon 里面的PDH计数器



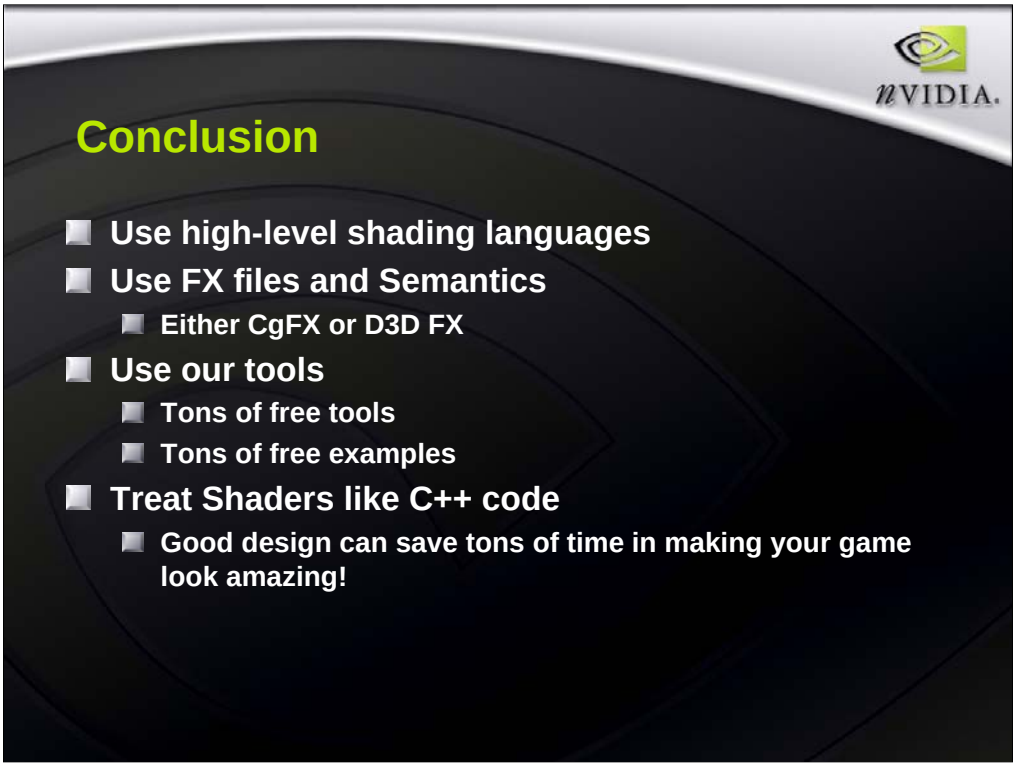

**VShaderPerf**

•与FX Composer 里的Shader Perfpanel使用同样的技术

•分析DirectX和OpenGL 着色器

HLSL, GLSL, Cg, !!FP1.0, !!ARBfp1.0, VS1.x, VS2.x, VS3.x, PS1.x,PS2.x, PS3.x, etc.

•着色器性能退化测试，适用于所有的NVIDIA GPU产品平台，无须重新启动系统！




总结

•使用高级着色语言

•使用FX文件和语意（Semantics）

CgFX或D3D FX都可以

•使用我们的工具软件

大量的免费工具软件

大量的示例程序

•把着色程序看成C++代码

良好的设计可以节约大量的时间，并且可以让你的游戏看上去更吸引人！

## Questions

- http://developer.nvidia.com
- http://developer.nvidia.com/CgTutorial
- **Email:** bdudash@nvidia.com


The Source for
GPU Programming
developer.nvidia.com
• Latest News
• Developer Events Calendar
• Technical Documentation
• Conference Presentations
• GPU Programming Guide
• Powerful Tools, SDKs and more ...
Join our FREE registered developer program for early access to NVIDIA drivers, cutting edge tools, online support forums, and more.
NVIDIA
developer.nvidia.com
©2004 NVIDIA Corporation. NVIDIA, and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation. Nalu is ©2004 NVIDIA Corporation. All rights reserved.