



NVIDIA®

NVIDIA开发工具
NVIDIA Developer Tools

Koji Ashida

NVIDIA提供了很多工具

NVIDIA Provides Many Tools

- **NVSDK**
- 性能调节工具
- 内容创作工具和插件 (**plugins**)
 - **Melody**
 - **NVTriStrip**
- 创作特效
 - 着色器 (**shader**) 设计和管理
 - 在**DCC**软件中的插件

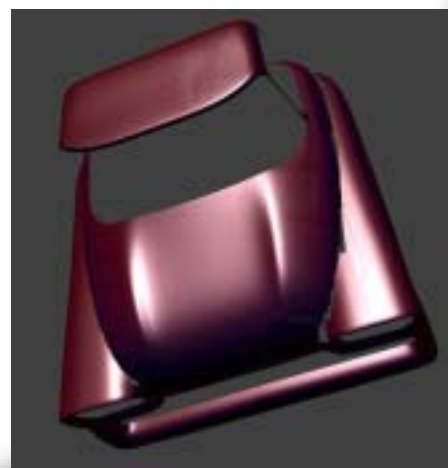
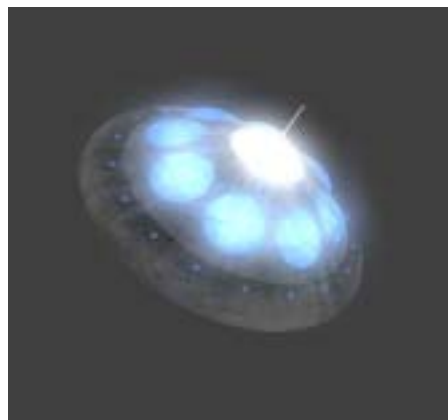


NVIDIA.

NVSDK

实时开发资源

- **GeForce FX**的新的**shader**效果：
Skin, Gooch, Car Paint, Glow, Uber, Bicubic Filtering, 还有更多...
- 在**DirectX**和**OpenGL**中有几百种效果
- 大量的源代码
- 广泛和经常更新的分发
 - 每个版本都有**40,000**次下载
- 工作流程关注于：将艺术变成代码
- **developer.nvidia.com**



Demo: CgBrowser



NVIDIA.

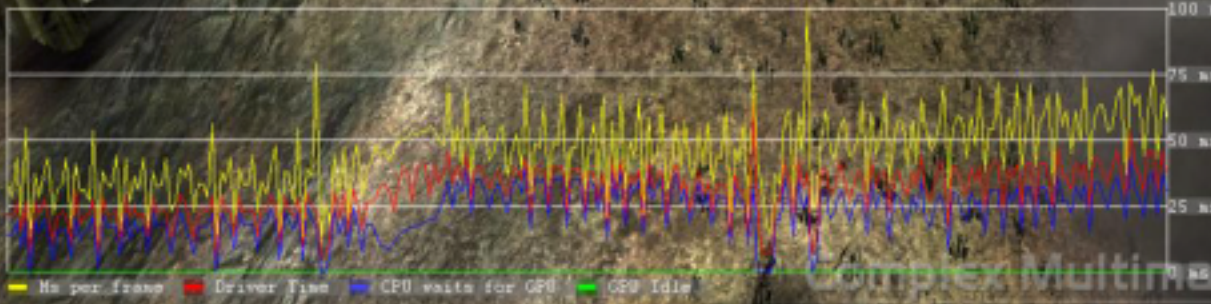
性能调节：NVPerfHUD

Performance Tuning: NVPerfHUD

- 驱动程序现在支持**NVPerfHUD**
- 采用覆盖的方式显示应用程序运行的多种重要的参数统计
- 上部图表显示：
 - **API调用的数目** – Draw*Prim*, render states, texture states, shader states
 - **存储器分配情况** – AGP和视频的
- 下部图表显示：
 - **GPU Idle** –图形硬件没有处理任何事情
 - **Driver Time** – 驱动程序工作情况（状况和资源管理，shader 编译）
 - **Driver Idle** – 驱动程序等待**GPU**完成处理
 - **Frame Time** – 每帧所花时间的毫秒数



fps: 014 tris/frame: 169215



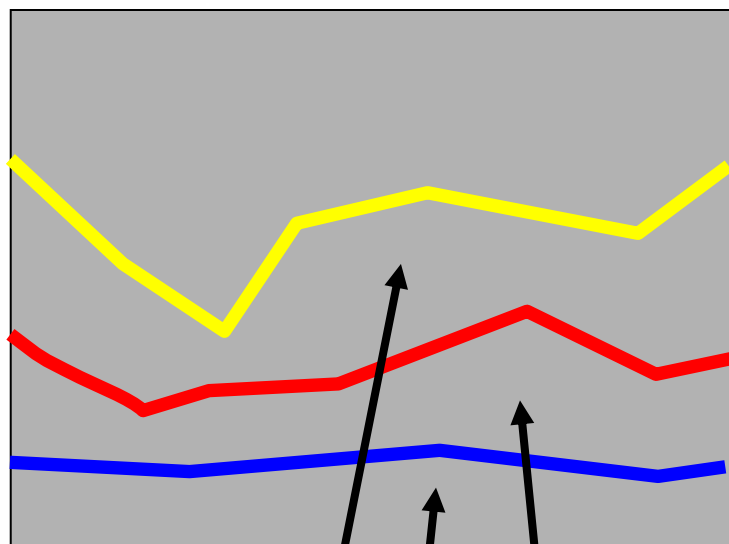
Complex Multitextured Shader



NVIDIA

NVPerfHUD: CPU和GPU使用率

CPU

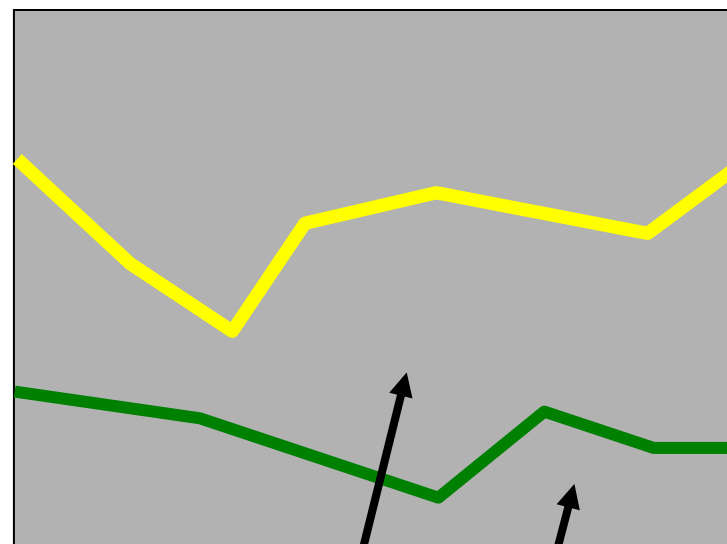


程序
工作

CPU
等待

driver
工作

GPU



着色
工作

GPU
等待

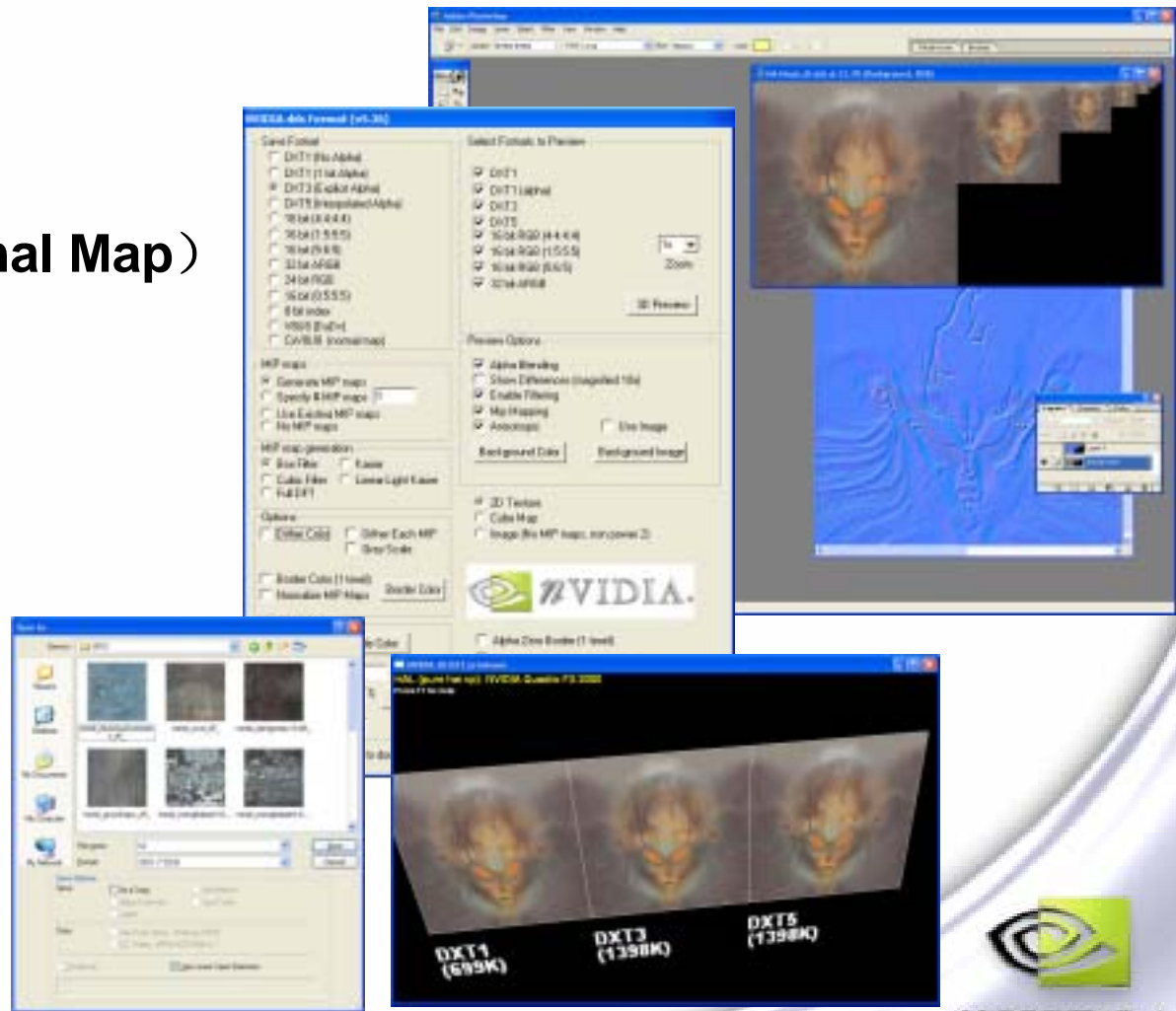


NVIDIA.

纹理工具和插件

Texture Tools & Plugins

- Photoshop插件:
 - DXT压缩 (.dds)
 - 法线贴图 (Normal Map) 创作
 - 3d预览和评估
 - MIP map创作器
- 命令行和.lib
- DDS简明察看器



Demo: Melody



NVIDIA.

Shader开发: FX Composer

Shader development: FX Composer

- **HLSL FX** (FX: 效果) 开发的完整集成开发环境 (IDE)
- 提供为NV3x家族的**shader**时序模拟
- 顶点和像素**shader**的反汇编
- 从**HLSL**代码中提取纹理
- 可以实现渲染成纹理 (**render to texture**) 的效果
- **HLSL Intellisense**
- 允许从**.x**和**.nvb**文件中输入场景
- 支持动画、光照、蒙皮网格化 (**skinned meshes**) 等等.....
- 可以有可插入的几何修改器 (鳍状效果.....)
- 计划文件
- **Fxmapping.xml** – 定制语义/注释贴图



Demo: FX Composer UI components

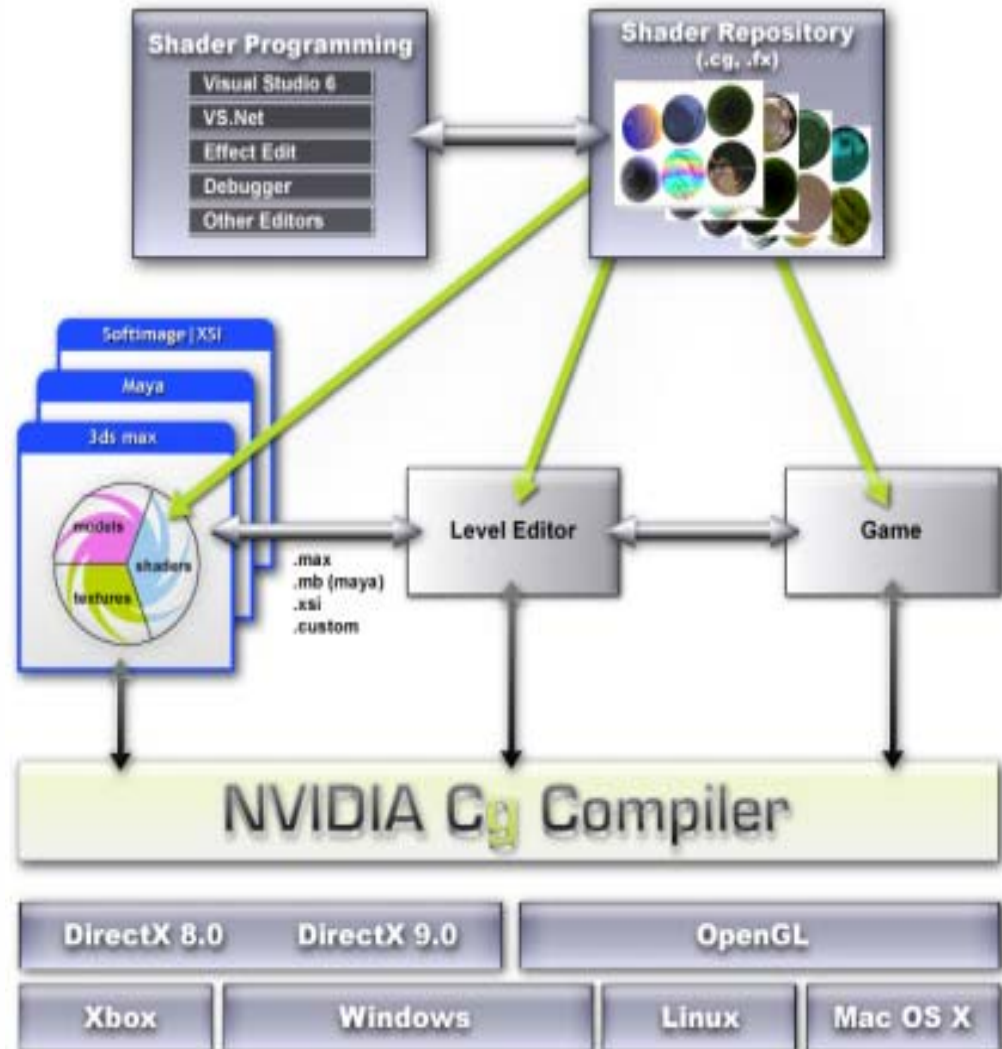


NVIDIA.

效果如何工作的

How FX (Effect) Works

- **Shader**是设计来用于控制**GPU**如何将几何转换到合适的空间以及如何将光栅化后产生的像素进行着色的一种小程序。
- **FX**适用于创作和使用数字图形的每个阶段：
 - 建模**Modeling**
 - 纹理贴图**Texturing**
 - 动画**Animation**
 - 级别设计**Level Design**
 - 游戏引擎**Game Engines**
 - 渲染**Rendering**



FX: 一个完整的着色语言

FX: A Complete Shading Language

- 将顶点/片段（ **fragment** ） **shader**统一在一个完整的外观的下的一个简单的途径
- 可以用于产品创作的所有阶段
 - 已经整合在最流行的**DCC**软件中
- 多渲染流程（ **Multiple render passes** ）的支持
- **Techniques**适应不同的硬件设备并表现出不同的渲染能力
- 简单的文本文件以容易处理和管理



NVIDIA.

FX文件结构

- **FX** 文件看起来类似于程序
- 每个**FX**包含：
 - 用户“**Tweakables**”/跟踪声明：
用户变量和**UI**提示
 - 其它的全局声明
 - 顶点和像素着色器（**Shader**），声明成函数
 - **Techniques**装入**Shaders**、**Tweakables**、**Render Passes**以及图形状态设置。



Demo: a skeleton FX file



NVIDIA.

FX示例—Tweakables

FX Example – Tweakables

```
float4 lightPos : Position  
<  
    string Object = "PointLight";  
    string Space = "World";  
> = {100.0f, 100.0f, 100.0f, 1.0f};  
  
float lightIntensity  
<  
    string gui = "slider";  
    float uimin = 1.0;  
    float uimax = 10000.0;  
    float uistep = 1.0;  
    string Desc = "lamp power";  
    float min = 0.0;  
    float max = 10000.0;  
> = 10.0;
```

- **:** 语义 为自动联编（**binding**）给应用程序的提示
- **<Annotations>** **<注释>** **>** 给与额外的应用程序特别的UI提示
- 语义和注释都是可选的



FX示例—“Un-Tweakables”

FX Example – “Un-Tweakables”

- 一些通用的“**tweakables**”可以被**FX**自动跟踪
- 我们不允许用户调节它们，因此它们是“不可调节的（**untweakables**）”

```
float4x4 worldIT : WorldIT;  
float4x4 wvp : WorldViewProjection;  
float4x4 world : World;  
float4x4 viewIT : ViewIT;
```



示例—仅用于应用程序的全局变量

Example – Application-Only Globals

- 这些值不被**Shader**本身所使用，可以被应用程序用于目录编制、定义**UI**等等。

```
string description = "Shader Template";  
string Category = "Template";  
string keywords =  
    "bumpmap, texture, glossmap, fresnel";
```



FX示例—Shader作为函数

FX Example – Shaders as Functions

```
vertexOutput basicVS(appdata IN,  
    uniform float4x4 WorldViewProj,  
    uniform float4x4 WorldIT,  
    uniform float4x4 World,  
    uniform float4x4 ViewIT,  
    uniform float3 LightPos  
) {  
    vertexOutput OUT;  
    OUT.WorldNormal = mul(WorldIT, IN.Normal).xyz;  
    float4 Po = float4(IN.Position.xyz,1.0);  
    float3 Pw = mul(World, Po).xyz;  
    OUT.PtLightVec = LightPos - Pw;  
    OUT.TexCoord = IN.UV.xy;  
    OUT.WorldView = normalize(ViewIT[3].xyz - Pw);  
    OUT.HPosition = mul(WorldViewProj, Po);  
    return OUT;  
}
```



NVIDIA.

FX Example – Technique

```
technique Main
{
    pass p0
    {
        VertexShader = compile arbvpl basicVS(wvp,
            worldIT,world,viewIT,lightPos),
        ZEnable = true;
        ZWriteEnable = true;
        CullMode = None;
        PixelShader = compile arbfpl basicPS(colorSampler,
            envSampler,diffStrength,
            specStrength,specExpon,metalness,
            reflStrength,reflMin,fresExp,
            ambiLightColor,surfColor,
            lightColor,lightIntensity);
    }
}
```

- Profiles
- 也可以用 asm{...}
- 渲染声明



Demo: blue circle

- 简单地发射变换顶点
 - $\text{Position} = \{x, y, z, w\} * \text{Winv} * V * P$
- 将像素颜色设置为蓝色
 - $\text{Color} = \{0.0, 0.0, 1.0\}$
- 球体看起来是圆盘



Demo: per pixel phong shading

- 同样的顶点计算
- **Phong**颜色模型
 - **Color = C * (L . n + (H . n)^S)**
 - **C** – surface color
 - **L** – light direction
 - **n** – surface normal
 - **H** – half angle $(E + L) / 2$ where **E** is eye direction
 - **S** – specular constant
- 这通常是基于顶点完成以得到较高的效率

Demo: using tweakables

- 将光照参数设置为**tweakables**



NVIDIA.

Demo: multi-pass rendering



NVIDIA.

Demo: moving object

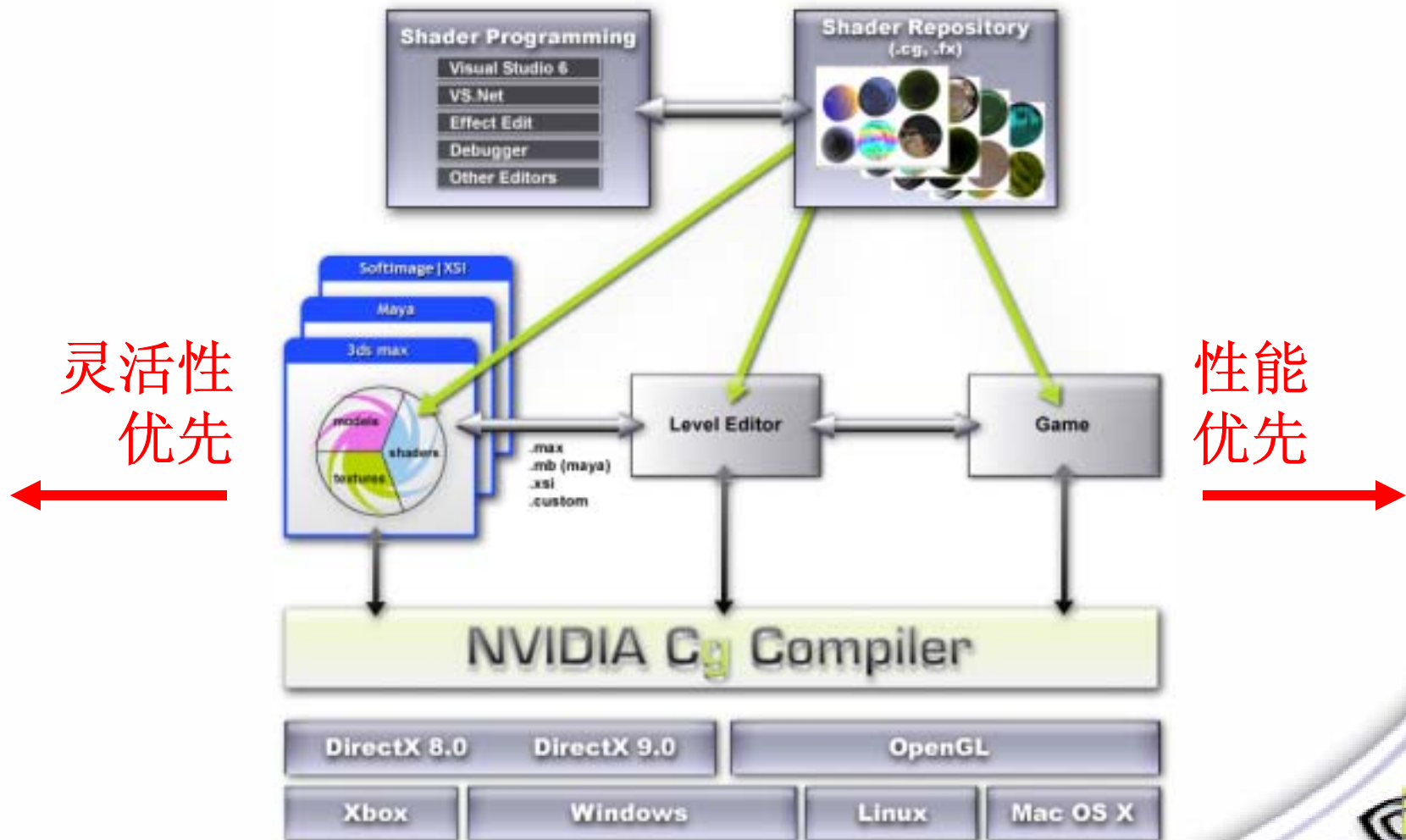
- 使用时间 (**Time**) 来控制顶点位置



NVIDIA.

两种FX工作流程

Alternate FX Workflows



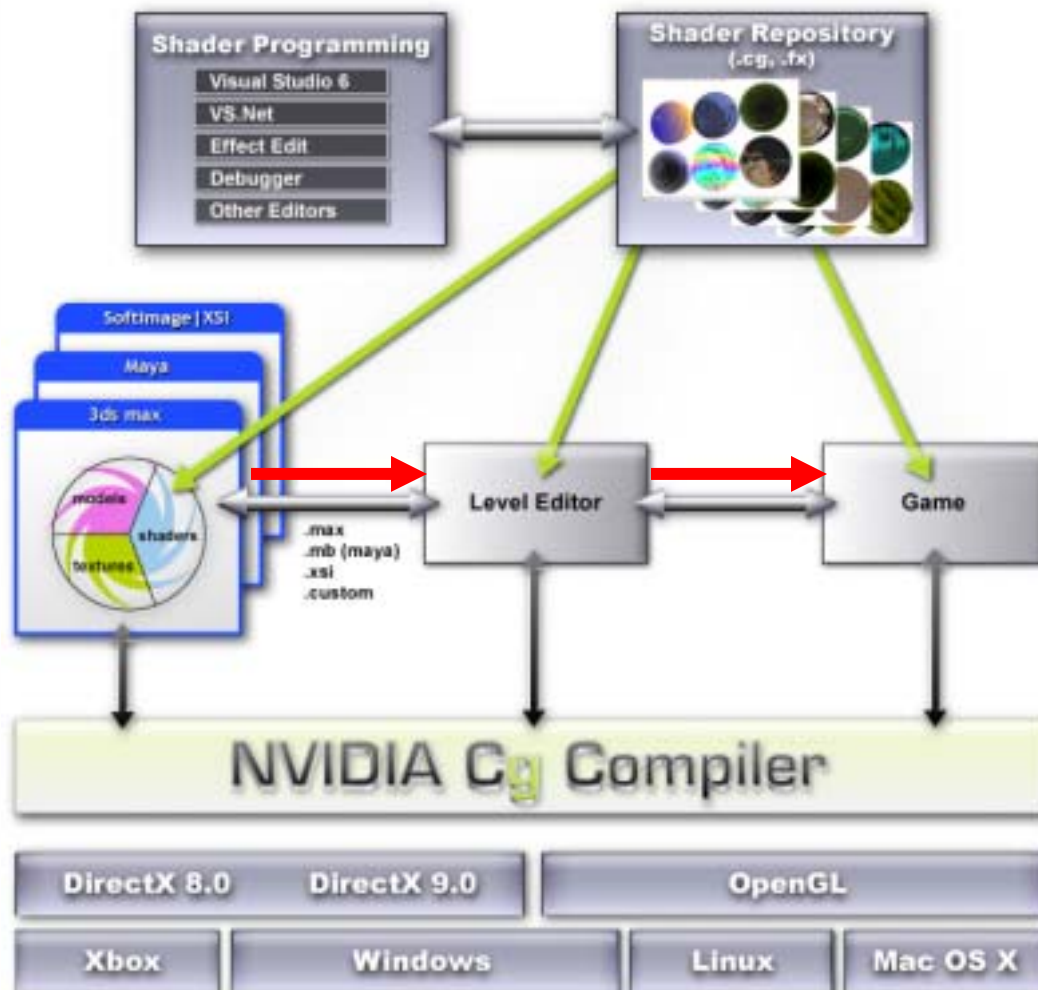
NVIDIA.

FX 工作流程1：艺术家控制

FX Workflow 1: Artist Control



艺术家定义的
Shaders



NVIDIA.

艺术控制下的工作流程

Art-Controlled Workflow

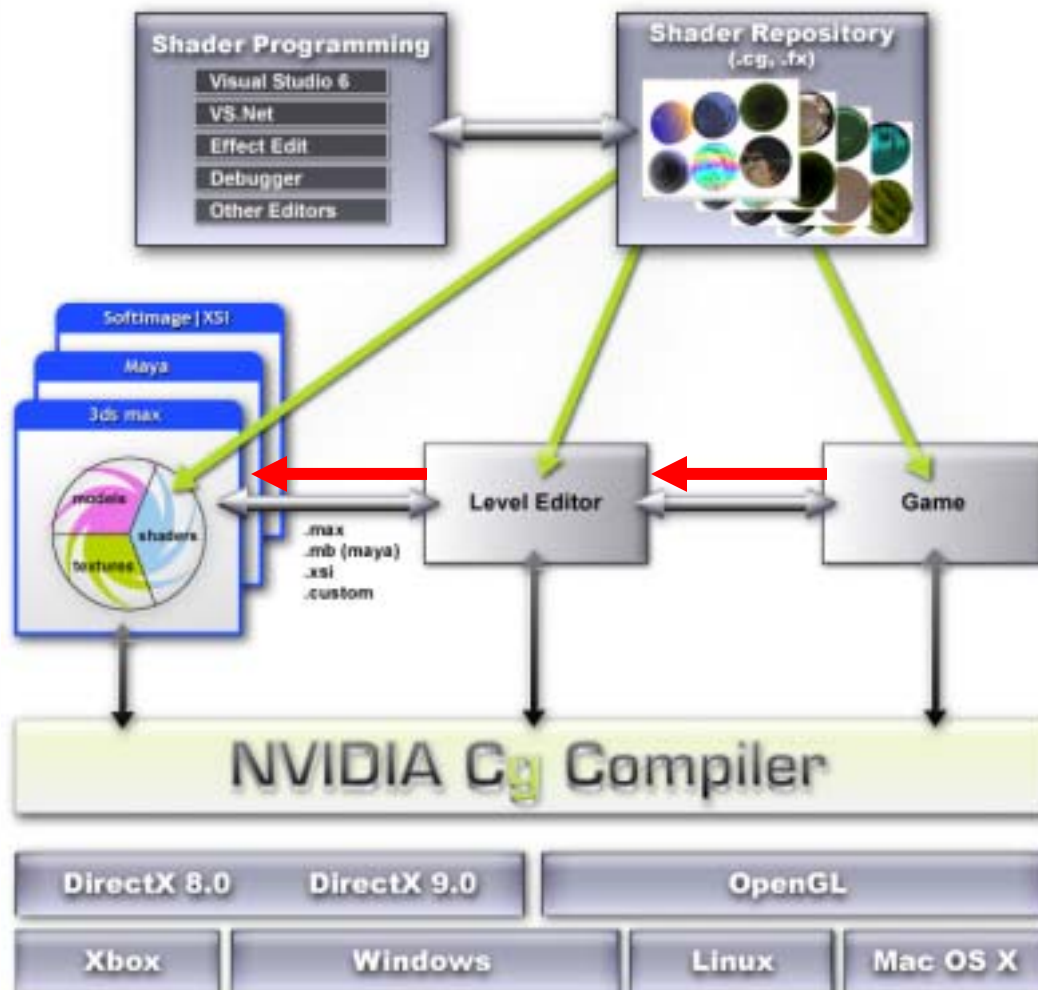
- 艺术家可以根据他们的艺术构思选择或者创作合适的 **Shader**
- **Shaders**可以随后用于后续工作（**Shader**可以是**FX**、整合的**HLSL**或者是编译好的汇编代码）
- 最终产品的感官就与艺术家设想的一样



NVIDIA.

工作流程 2: 程序员控制

Workflow 2: Programmer Control



程序员设计的
Shaders



NVIDIA

程序员控制的工作流程

Programmer-Controlled Workflow

- **Shader**是考虑到性能、特别的光照模型、游戏机兼容性、真实世界的材质模拟等方面进行定义的。
- **HLSL**或者汇编代码合成在**FX**中传递到上游的艺术家那里。
- 艺术家可以随后根据硬件/性能需求进行专门的设计，在他们设计的应用程序中观看最终的效果。
- 艺术家同样也可以通过**techniques**同时为不同的平台设计艺术效果，包括：**Xbox, Playstation**, 快速或者较慢**PCs**



灵活的Runtime编译 (FX)

At Development Time

Shader程序 源代码

```
//  
// Diffuse lighting  
//  
float d = dot(normalize(frag.N),  
normalize(frag.L));  
if (d < 0)  
    d = 0;  
c = d*tex2D(t, frag.uv)*diffuse;  
...
```

At Runtime

- 初始化阶段:
 - 编译和调入
HLSL程序
- 为每一帧:
 - 与**Shader Runtime API**一起调入程序参数
 - 设置渲染声明
 - 调入几何
 - 渲染



NVIDIA.

脱线编译 Compiling Offline

At Development Time

HLSL或者FX
源代码

Shader编译器

Shader程序
汇编代码

Shader编译器
(nvasm.exe, psa.exe)

Shader二进
制代码

```
//  
// Diffuse lighting  
//  
float d = dot(normalize(frag.N),  
normalize(frag.L));  
if (d < 0)  
    d = 0;  
c = d*tex2D(t, frag.uv)*diffuse;  
...
```

```
...  
DP3 r0.x, f[TEX0], f[TEX0];  
RSQ r0.x, r0.x;  
MUL r0, r0.x, f[TEX0];  
DP3 r1.x, f[TEX1], f[TEX1];  
RSQ r1.x, r1.x;  
MUL r1, r1.x, f[TEX1];  
DP3 r0, r0, r1;  
MAX r0.x, r0.x, 1.0;  
MUL r0, r0.x, DIFFUSE;  
TEX r1, f[TEX1], 0, 2D;  
MUL r0, r0, r1;  
...
```

```
012b40 00 00 00 00 00 00 00 00  
012b50 42 CD 09 84 51 3F 84 3C  
012b60 93 AB D9 01 07 07 70 B2  
012b70 5C A5 D1 1C 58 65 58 F4  
012b80 1F 27 1F 22 22 1F 1F 22  
012b90 12 22 22 12 22 22 22 22  
012ba0 1F 2F 2F 2F 2F 2F 23 FF  
012bb0 37 37 37 37 37 37 2C 2C  
012bc0 30 BE 47 04 4A BE A8 E6  
012bd0 50 78 92 DD 90 89 72 CE  
012be0 03 69 8D EE 46 73 85 F9
```

At Runtime

● 初始化阶段:

● 调入

汇编或者二进制代码

● 对每一帧:

● 将程序参数调入硬件寄存器中

● 设置渲染声明

● 调入几何

● 渲染



NVIDIA

Runtime编译的优缺点

Pros and Cons of Runtime Compilation

● 优点:

- 向后兼容性: 应用程序不需要修改就可以从未来的编译器中得到好处 (未来的优化, 将来的硬件)
- 容易的参数管理
- 游戏玩家可以“改造”效果

● 缺点:

- 调入需要更多时间因为需要编译
- 不能够手工调节编译的结果
- 文本文件可能是不可靠的
- **API**声明是固定的 (对于**FX**文件来说)



结论

Summary

- **NVIDIA**为游戏生产提供了多种工具
- 使用**FX Composer**，您可以：
 - 快速将**shader**的构思变成程序原型并进行试验
 - 使用**tweakables**将**shader**开发工作在程序员和艺术家之间区分开来
- **FX**文件帮助您管理您的艺术资产



NVIDIA.

有什么问题、评论、反馈？

- **Koji Ashida <kashida@nvidia.com>**
- **<http://developer.nvidia.com>**



NVIDIA.