



**nVIDIA®**

**Cross-Platform Development  
using FX Composer 2.0**

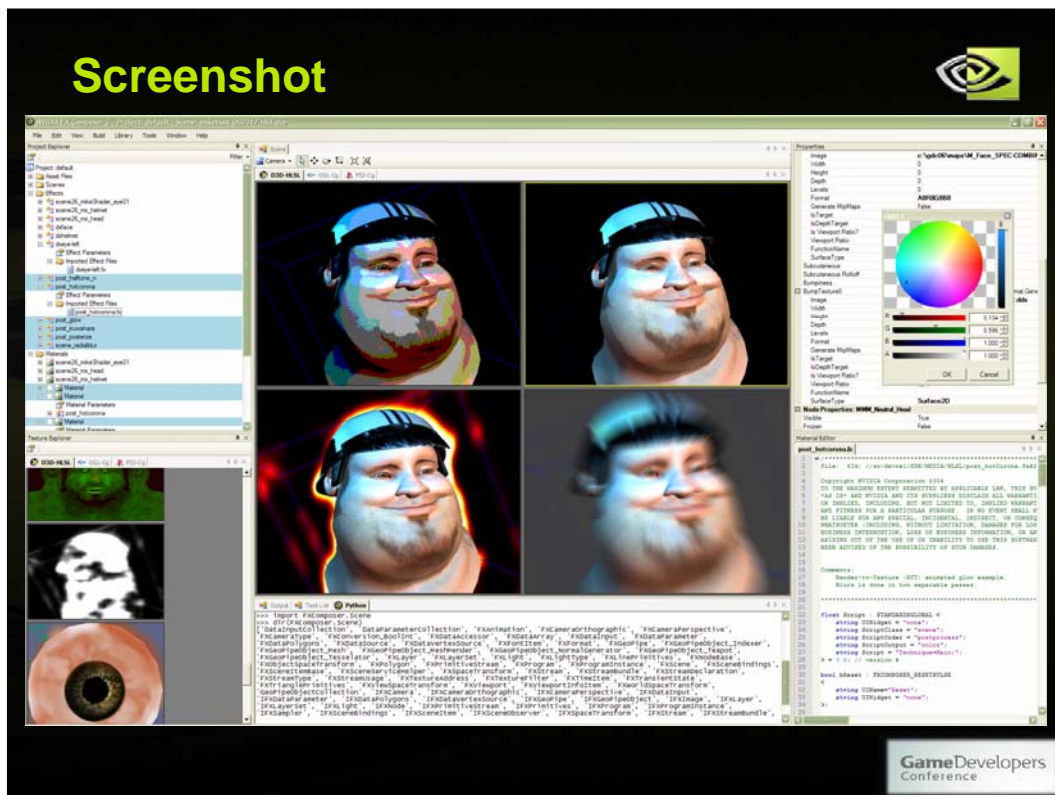
**Chris Maughan · Kevin Bjorke**  
**Alpha 4 · GDC 2006**

**GameDevelopers**  
Conference

Good afternoon & welcome.



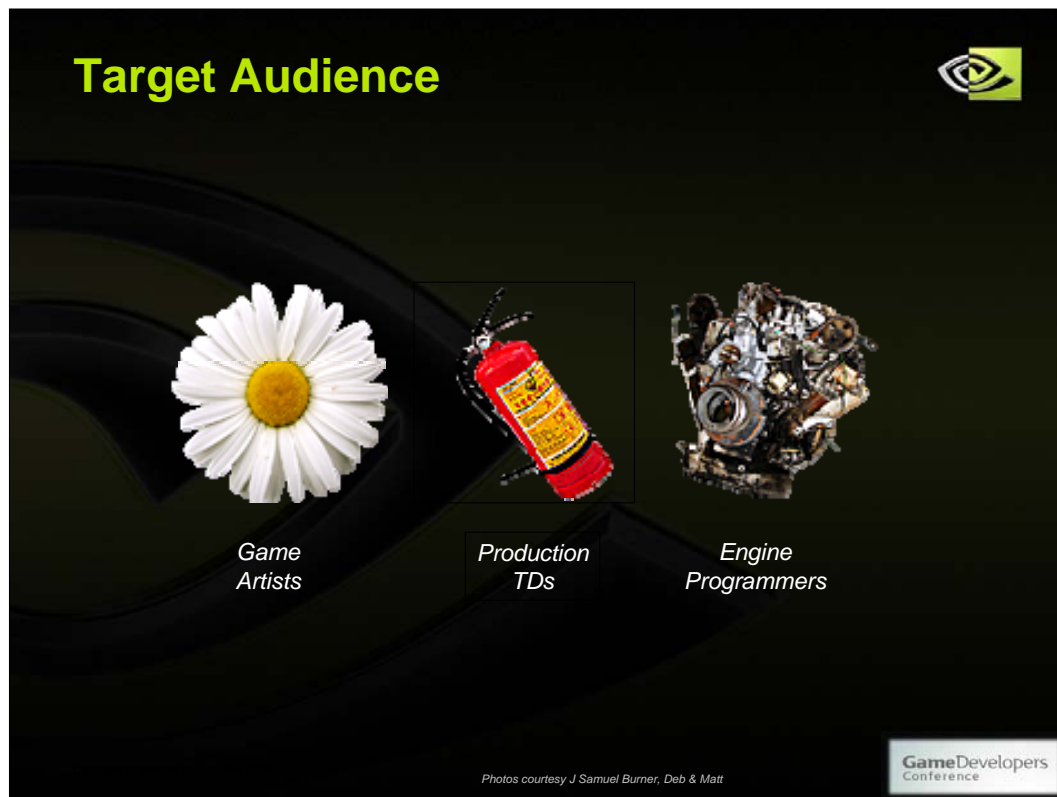
My name is Chris Maughan & I'll be telling you about some of the principal features and improvements we've made to FX Composer for version 2



Here's an example image showing our mike character. You can see that there are actually a couple of different render-to-texture full-screen effects being used here – one window is using an corona effect, while another window is displaying an explosion effect.

Check out what's going on in the texture panel too – the various stages of the corona effect are being displayed and updated in real time and the effect animates, letting you really get a clear look right down into the heart of how your render works

You can see – project explorer, textures, python scripting, the shader editor and the properties.




FX Composer 2 is designed from the outset as a PRODUCTION TOOL.

We see three different sorts of users – you yourself may fit into one, two, or all three categories.



Those categories are game artists, production TDs, and core game-engine programmers.

Importantly, FX Composer 2 doesn't just provide them with tools to do their own work, it provides a common platform for all of these people to work together.

## Why FX Composer 2 ?



- **Draws things a DCC app can't**
  - Shadows, full-screen effects
  - New hardware features
- **Fully customizable**
  - Plug-ins, Scripting, devices, and GUI Layout
  - Engine integration
- **Shader profiling**
  - Using our own compiler technology



FX Composer is designed with features for every kind of user.

First, it can make pictures that artists just can't get when they're in a DCC app. DCC apps are very capable but they're designed to be fast at their core strengths and they each have a fixed real-time rendering architecture. FX Composer is designed specifically for managing shading, for handling arbitrary render architectures. So that means that artists can directly visualize and tweak attributes of their scenes that just aren't visible in a DCC app. We'll expand on this further in a little while.

Second, FX Composer is customizable, scriptable, and .NET compliant. Whatever the production environment, FX Composer is readily adapted to it. If your studio has custom data formats, monolithic control applications, or any other tools based on ET or on Python or C# or other CLR-based languages, those tools can interoperate easy with FX Composer.

Finally, FX Composer has an integrated shader profiler that lets game-engine programmers get right in at the heart of efficiency and debugging issues, so they can squeeze every last nanosecond of performance out of their titles.

# FX Composer Introduction



- Major update from 1.x
  - Reworked from the ground up
  - Generalized rendering engine
  - User interface improvements
  - Plugin IO – e.g. COLLADA, OBJ, ..
  - Multiple device support - Cg, PS3, GLSL, etc...
  - Scripting with IronPython.NET

GameDevelopers  
Conference

We rebuilt it from scratch, to address the needs of developers using the current version. After listening to feedback developers wanted tighter integration with their pipelines, more customizability and cross platform support.

Rendering engine is one of the big changes. A generic, graph based, architecture. Nodes and pins and links. Very general for different cases.

Massively improved UI. Undo/Redo means less annoying dialogs. Cleaner approach to the whole thing.

IO is pluggable. We support what FXC1 did, but with collada as well

Devices. GL, etc.

Scripting through an .NET compliant language. We use IronPython.

## Reworked design



- **Now written in C#/.NET**
  - Easy to extend and integrate
- **A hierarchical plugin system**
  - Completely extensible – plugins define layers of behaviour
  - SDK examples
- **A graph system**
  - Manages component dependencies
  - Used in the rendering engine

GameDevelopers  
Conference

Almost 100% C#/.NET Add plugin system diagram.

Plugins define hierarchies of functionality. Entirely new features can be implemented. For example, the scene plugin defines a scene graph, the scene rendering plugin defines a scene graph renderer & idea of a device plugin.

Graph system is used to manage dependencies between engine components, and extensively in the rendering

# Rendering Engine



- **Extensible**
  - Plugin graph nodes can change rendering behaviour
  - Custom graphs to match your game engine
  - Several default graph nodes supplied – draw, clear, etc.
- **COLLADA FX & SAS supported through the graph**
  - Layered effects just build bigger graphs...

GameDevelopers  
Conference

The renderer is one of the major changes. The previous version of FX Composer was basically a SAS renderer. This version uses a graph to evaluate the scene. This graph is flexible enough that developers can extend it and it can support existing concepts, such as SAS. You can build graphs through script to match your rendering engine, for example.



## DXSAS Sample – Edge Detect



Technique Main

```
<
    string Script =
        RenderColorTarget0=SceneTexture;
        RenderDepthStencilTarget=DepthBuffer;
        ClearSetColor=ClearColor;
        ClearSetDepth=ClearDepth;
        Clear=Color;
        Clear=Depth;
        ScriptExternal=color;
        Pass=ImageProc;
```

>
Pass ImageProc

```
<
    string Script = RenderColorTarget0=;
        RenderDepthStencilTarget=;
        Draw=Buffer;
```

>

Game Developers  
Conference

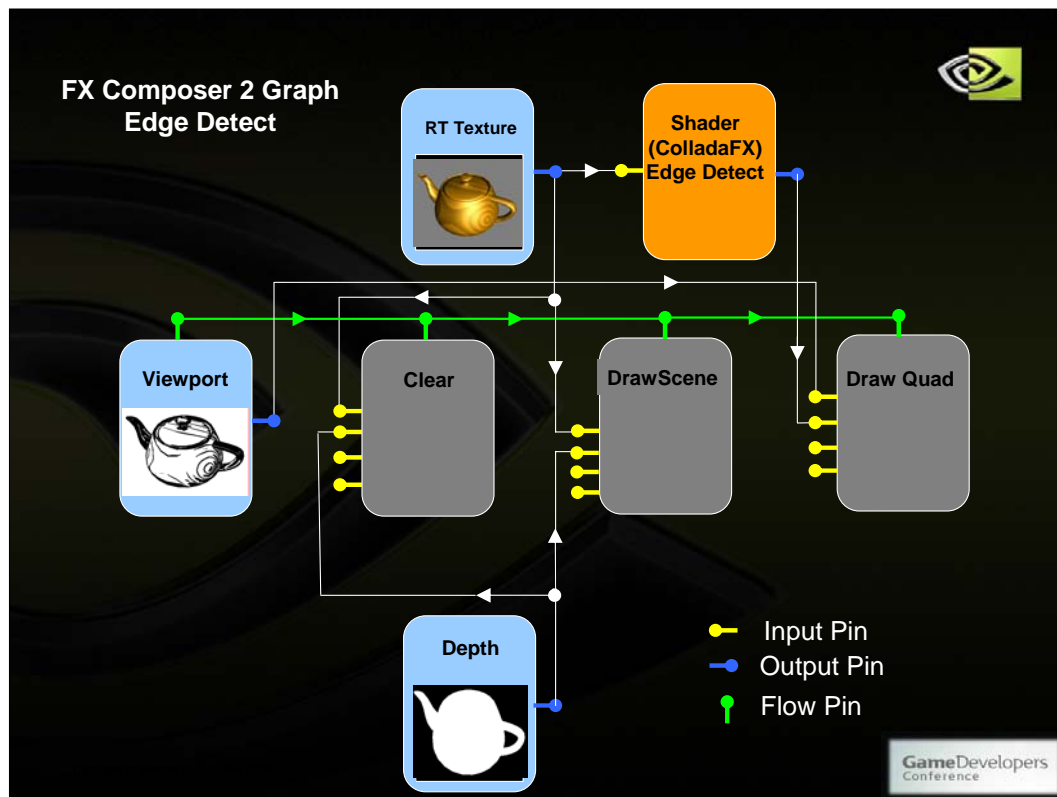
Don't worry about the syntax/details

Sets a couple targets

Clears them

ImageProc does the quad (Draw="Buffer")

This is how a SAS full screen effect looks.



Use the graph to do any sort of rendering. Extensible.

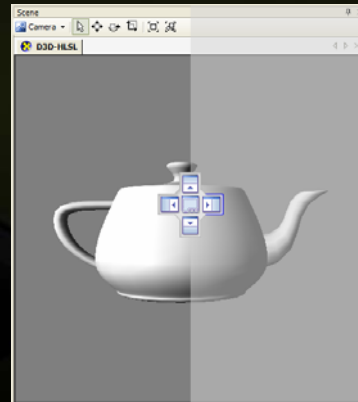


Top right is a combo. The others are individual effects.

## User Interface

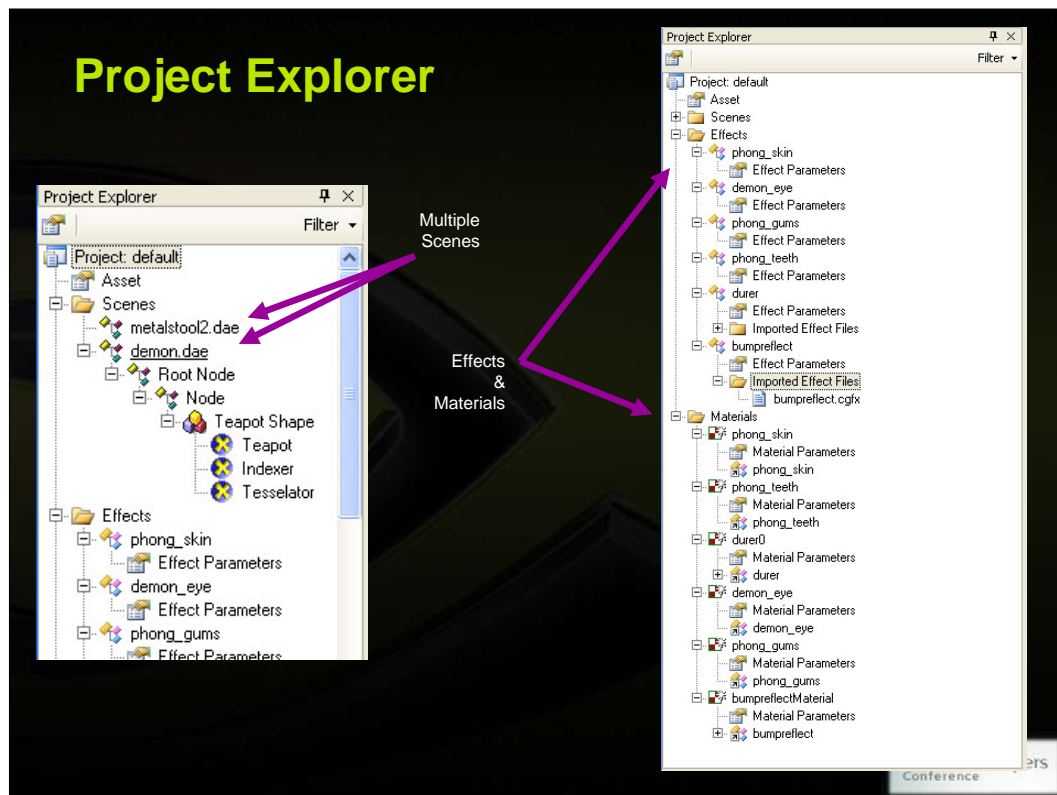


- All docking windows are plugins
  - Can add menu items/toolbars to the application window
  - VC2005 docking style
- Scripts can create menus & toolbars
- Many new and enhanced controls
- Full Undo/Redo



GameDevelopers  
Conference

Note the nice 2005 style dock hints. Much easier to use with this and other improvements such as undo/redo



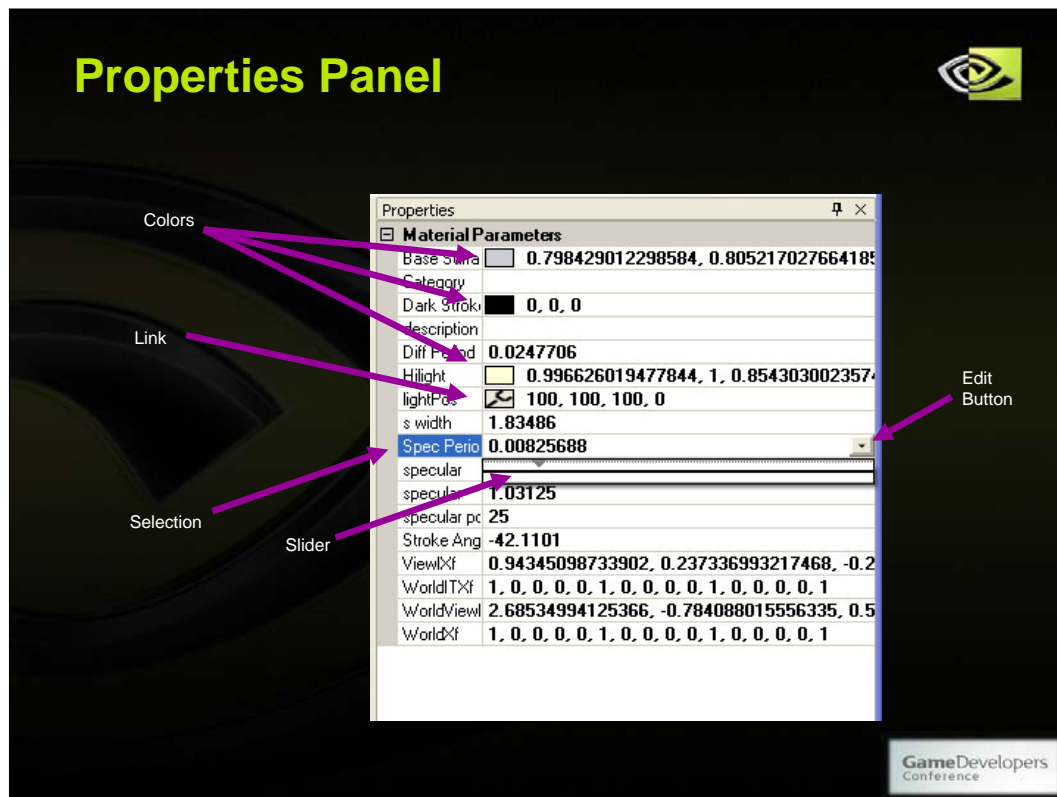
The project explorer is a hierarchical look at your entire Collada project area....

A project can contain multiple scenes, which is good if you're working on different parts of the same game – say, different levels, or collections of different characters, vehicles, and so forth. This project has two scenes

In the larger view, note the distinctions COLLADA has between effects and materials...a material USES an effect. If I have a model that uses both blue plastic and red plastic, I only need ONE “plastic” effect – but I can have two different materials, one red and one blue. Both of those materials call on the same underlying effect, but each material has its own properties – color, obviously, but any other properties too, such as choice of textures, choice of techniques, and so forth.

In FX Composer 2 we don't bind effects to models, we bind materials to models.

So, looking at these nodes, each one has its properties. We can see or edit the properties of these nodes in the FX Composer Properties Pane.



You can see the properties of any node by selecting it in the project explorer and right-clicking “Properties.” Materials also have a shortcut node called “parameters” that you can just double-click, since editing parameters is the most-common operation for an FX Composer user.

For material parameters, as we see here, there are a few things I’d like you to notice.

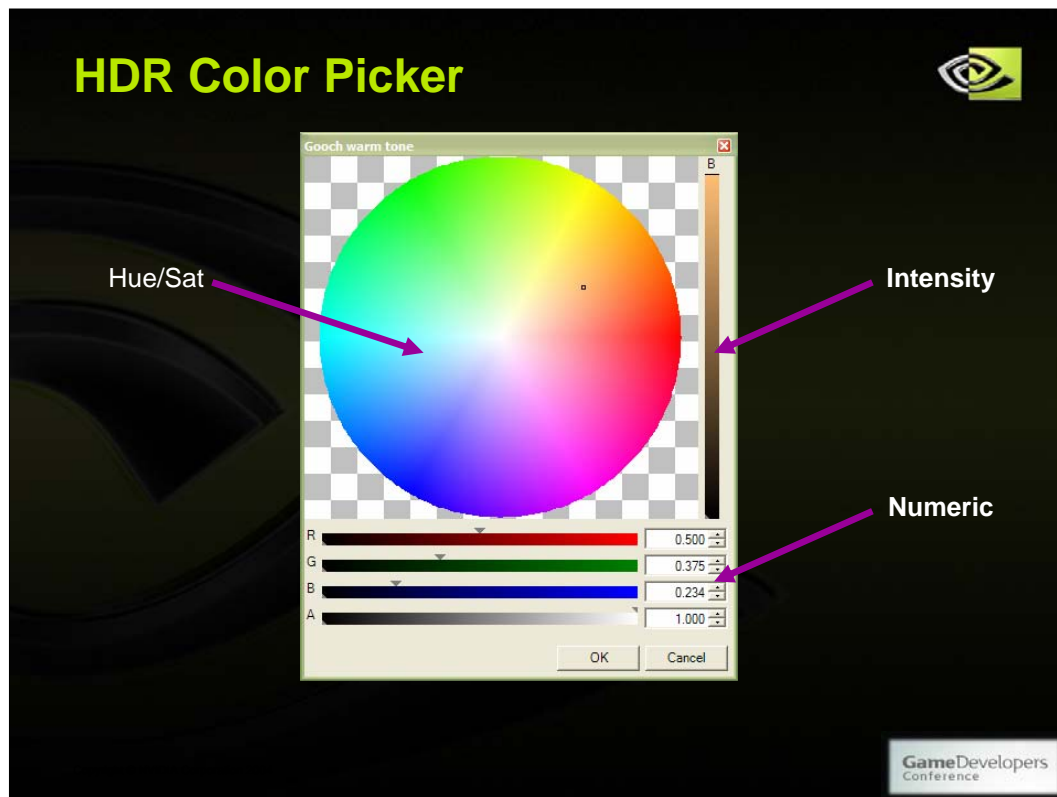
First, one property is selected, because I’ve clicked on its name

Second, I’ve also clicked the little “edit” button that appears when some properties are selected, and that’s caused...

...this slider to appear. The slider ranges and values can be set directly by the shader source itself. We can slide-around the little gray pointer with the mouse to assign a value, or type-in a specific number.

This little “Y” symbol means that this particular parameter is linkable – we can click on this and link it appropriately to the values of some other node. In this case we can see it’s a light position, and we can link the value of this property to any light that exists in the current scene. Then we’re free to drag-around the light or the shaded model and everything will just update auto-magically.

Finally, these little chips indicate a color. When we try to edit these, we’ll get the FX Composer color picker....



The color picker looks a lot like typical color pickers.

We have a Hue and Saturation wheel, and

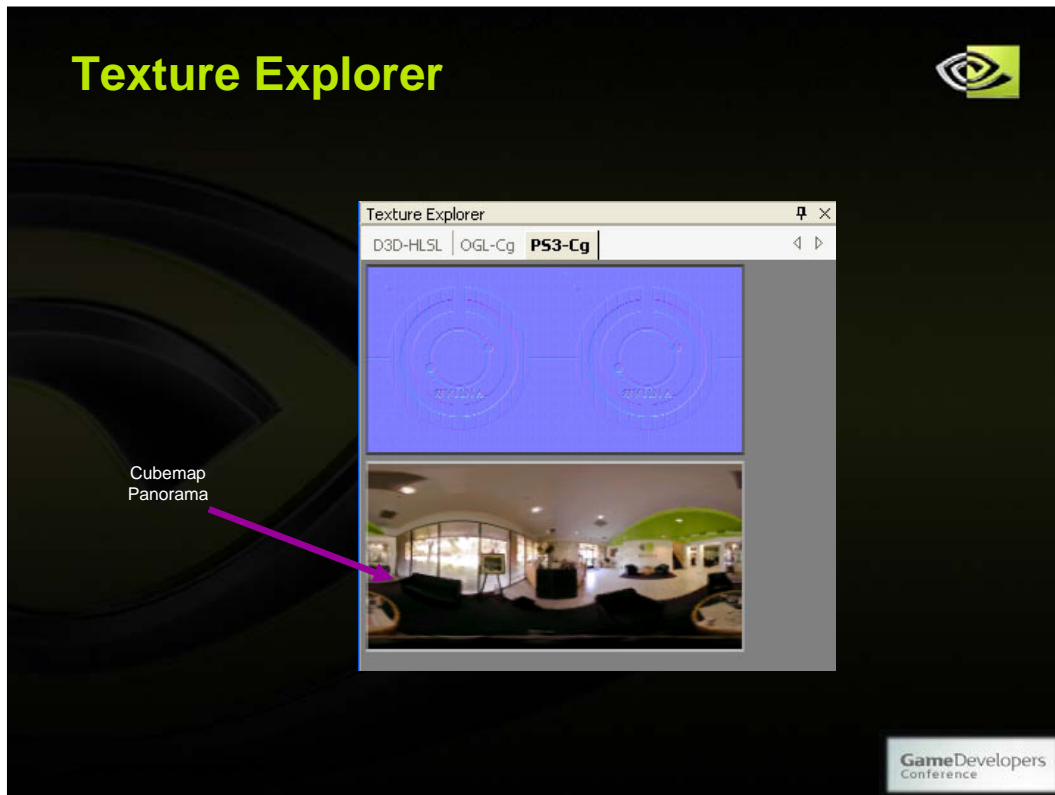
An Intensity slider.

But notice that the intensity slider can go ABOVE ONE, because this is an HDR color picker – that is, it has high dynamic range. So you can have colors brighter than one.

If you know what that's about, great. If not, I'll show you an example later. Just consider over-bright colors as "over-exposed" and you'll be on your way. But as I said we'll do that in a little while.

Finally, for greatest precision we can always just enter text in the numeric fields

# Texture Explorer



Let's move to a different part of the UI, the texture explorer. This window shows you the textures currently in use – it will even do live updates for complex texture effects.

I'll just mention one innovation here, because I think it's cool

The lower window is showing us a cube environment map – the display automatically unwraps the display as a panorama, making it a lot easier to see.

And on that, um, slightly geeky note let's look at some programmer parts to the UI:



# Material Editor

The screenshot displays the Material Editor interface with a CgFX script. The script defines two samplers, `sampler2D` and `samplerCUBE`, and a struct `a2v`. The interface features syntax highlighting (e.g., keywords in blue, strings in red), line numbers on the left, and a collapsible structure indicated by a minus sign in the left margin. Three purple arrows point from labels to these features: 'Syntax Highlighting' points to the keyword `sampler_state`, 'Line #'s points to the line number 62, and 'Collapser' points to the minus sign in the left margin.

```
46 >;
47
48 =sampler2D NormalSampler = sampler_state {
49     Texture = <normalMap>;
50     MinFilter = LinearMipMapLinear;
51     MagFilter = Linear;
52 };
53
54 ////
55
56 texture envMap : Environment <
57     string ResourceName = "Default_reflection.dds";
58     string ResourceType = "Cube";
59 >;|
60
61 =samplerCUBE EnvSampler = sampler_state {
62     Texture = <envMap>;
63     MinFilter = LinearMipMapLinear;
64     MagFilter = Linear;
65     WrapS = ClampToEdge;
66     WrapT = ClampToEdge;
67 };
68
69 ///////////////////////////////////////////////////////////////////
70 // structures and shaders ////////////////////////////////////////
71 ///////////////////////////////////////////////////////////////////
72
73 struct a2v
74 = {
75     float4 Position : POSITION; //in object space
76     float2 TexCoord : TEXCOORD0;
77     float3 Tangent : TEXCOORD6; // TANGENT; // ATTR14; // in object
78     float3 Binormal : TEXCOORD7; // BINORMAL; // ATTR15; // in object
79     float3 Normal : NORMAL; //in object space
80 };
81
82 struct v2f
83 = {
84     float4 Position : POSITION; //in projection space
85     float2 TexCoord : TEXCOORD0;
86     float4 TexCoord1 : TEXCOORD1; //first row of the 3x3 transform f
87     float4 TexCoord2 : TEXCOORD2; //second row of the 3x3 transform f
88     float4 TexCoord3 : TEXCOORD3; //third row of the 3x3 transform f
89     float4 TexCoord4 : TEXCOORD4; //fourth row of the 3x3 transform f
90     float4 TexCoord5 : TEXCOORD5; //fifth row of the 3x3 transform f
91     float4 TexCoord6 : TEXCOORD6; //sixth row of the 3x3 transform f
92     float4 TexCoord7 : TEXCOORD7; //seventh row of the 3x3 transform f
93     float4 TexCoord8 : TEXCOORD8; //eighth row of the 3x3 transform f
94     float4 TexCoord9 : TEXCOORD9; //ninth row of the 3x3 transform f
95     float4 TexCoord10 : TEXCOORD10; //tenth row of the 3x3 transform f
96     float4 TexCoord11 : TEXCOORD11; //eleventh row of the 3x3 transform f
97     float4 TexCoord12 : TEXCOORD12; //twelfth row of the 3x3 transform f
98     float4 TexCoord13 : TEXCOORD13; //thirteenth row of the 3x3 transform f
99     float4 TexCoord14 : TEXCOORD14; //fourteenth row of the 3x3 transform f
100     float4 TexCoord15 : TEXCOORD15; //fifteenth row of the 3x3 transform f
101     float4 TexCoord16 : TEXCOORD16; //sixteenth row of the 3x3 transform f
102     float4 TexCoord17 : TEXCOORD17; //seventeenth row of the 3x3 transform f
103     float4 TexCoord18 : TEXCOORD18; //eighteenth row of the 3x3 transform f
104     float4 TexCoord19 : TEXCOORD19; //nineteenth row of the 3x3 transform f
105     float4 TexCoord20 : TEXCOORD20; //twentieth row of the 3x3 transform f
106     float4 TexCoord21 : TEXCOORD21; //twenty-first row of the 3x3 transform f
107     float4 TexCoord22 : TEXCOORD22; //twenty-second row of the 3x3 transform f
108     float4 TexCoord23 : TEXCOORD23; //twenty-third row of the 3x3 transform f
109     float4 TexCoord24 : TEXCOORD24; //twenty-fourth row of the 3x3 transform f
110     float4 TexCoord25 : TEXCOORD25; //twenty-fifth row of the 3x3 transform f
111     float4 TexCoord26 : TEXCOORD26; //twenty-sixth row of the 3x3 transform f
112     float4 TexCoord27 : TEXCOORD27; //twenty-seventh row of the 3x3 transform f
113     float4 TexCoord28 : TEXCOORD28; //twenty-eighth row of the 3x3 transform f
114     float4 TexCoord29 : TEXCOORD29; //twenty-ninth row of the 3x3 transform f
115     float4 TexCoord30 : TEXCOORD30; //thirtieth row of the 3x3 transform f
116     float4 TexCoord31 : TEXCOORD31; //thirty-first row of the 3x3 transform f
117     float4 TexCoord32 : TEXCOORD32; //thirty-second row of the 3x3 transform f
118     float4 TexCoord33 : TEXCOORD33; //thirty-third row of the 3x3 transform f
119     float4 TexCoord34 : TEXCOORD34; //thirty-fourth row of the 3x3 transform f
120     float4 TexCoord35 : TEXCOORD35; //thirty-fifth row of the 3x3 transform f
121     float4 TexCoord36 : TEXCOORD36; //thirty-sixth row of the 3x3 transform f
122     float4 TexCoord37 : TEXCOORD37; //thirty-seventh row of the 3x3 transform f
123     float4 TexCoord38 : TEXCOORD38; //thirty-eighth row of the 3x3 transform f
124     float4 TexCoord39 : TEXCOORD39; //thirty-ninth row of the 3x3 transform f
125     float4 TexCoord40 : TEXCOORD40; //fourtieth row of the 3x3 transform f
126     float4 TexCoord41 : TEXCOORD41; //forty-first row of the 3x3 transform f
127     float4 TexCoord42 : TEXCOORD42; //forty-second row of the 3x3 transform f
128     float4 TexCoord43 : TEXCOORD43; //forty-third row of the 3x3 transform f
129     float4 TexCoord44 : TEXCOORD44; //forty-fourth row of the 3x3 transform f
130     float4 TexCoord45 : TEXCOORD45; //forty-fifth row of the 3x3 transform f
131     float4 TexCoord46 : TEXCOORD46; //forty-sixth row of the 3x3 transform f
132     float4 TexCoord47 : TEXCOORD47; //forty-seventh row of the 3x3 transform f
133     float4 TexCoord48 : TEXCOORD48; //forty-eighth row of the 3x3 transform f
134     float4 TexCoord49 : TEXCOORD49; //forty-ninth row of the 3x3 transform f
135     float4 TexCoord50 : TEXCOORD50; //fiftieth row of the 3x3 transform f
136     float4 TexCoord51 : TEXCOORD51; //fifty-first row of the 3x3 transform f
137     float4 TexCoord52 : TEXCOORD52; //fifty-second row of the 3x3 transform f
138     float4 TexCoord53 : TEXCOORD53; //fifty-third row of the 3x3 transform f
139     float4 TexCoord54 : TEXCOORD54; //fifty-fourth row of the 3x3 transform f
140     float4 TexCoord55 : TEXCOORD55; //fifty-fifth row of the 3x3 transform f
141     float4 TexCoord56 : TEXCOORD56; //fifty-sixth row of the 3x3 transform f
142     float4 TexCoord57 : TEXCOORD57; //fifty-seventh row of the 3x3 transform f
143     float4 TexCoord58 : TEXCOORD58; //fifty-eighth row of the 3x3 transform f
144     float4 TexCoord59 : TEXCOORD59; //fifty-ninth row of the 3x3 transform f
145     float4 TexCoord60 : TEXCOORD60; //sixtieth row of the 3x3 transform f
146     float4 TexCoord61 : TEXCOORD61; //sixty-first row of the 3x3 transform f
147     float4 TexCoord62 : TEXCOORD62; //sixty-second row of the 3x3 transform f
148     float4 TexCoord63 : TEXCOORD63; //sixty-third row of the 3x3 transform f
149     float4 TexCoord64 : TEXCOORD64; //sixty-fourth row of the 3x3 transform f
150     float4 TexCoord65 : TEXCOORD65; //sixty-fifth row of the 3x3 transform f
151     float4 TexCoord66 : TEXCOORD66; //sixty-sixth row of the 3x3 transform f
152     float4 TexCoord67 : TEXCOORD67; //sixty-seventh row of the 3x3 transform f
153     float4 TexCoord68 : TEXCOORD68; //sixty-eighth row of the 3x3 transform f
154     float4 TexCoord69 : TEXCOORD69; //sixty-ninth row of the 3x3 transform f
155     float4 TexCoord70 : TEXCOORD70; //seventieth row of the 3x3 transform f
156     float4 TexCoord71 : TEXCOORD71; //seventy-first row of the 3x3 transform f
157     float4 TexCoord72 : TEXCOORD72; //seventy-second row of the 3x3 transform f
158     float4 TexCoord73 : TEXCOORD73; //seventy-third row of the 3x3 transform f
159     float4 TexCoord74 : TEXCOORD74; //seventy-fourth row of the 3x3 transform f
160     float4 TexCoord75 : TEXCOORD75; //seventy-fifth row of the 3x3 transform f
161     float4 TexCoord76 : TEXCOORD76; //seventy-sixth row of the 3x3 transform f
162     float4 TexCoord77 : TEXCOORD77; //seventy-seventh row of the 3x3 transform f
163     float4 TexCoord78 : TEXCOORD78; //seventy-eighth row of the 3x3 transform f
164     float4 TexCoord79 : TEXCOORD79; //seventy-ninth row of the 3x3 transform f
165     float4 TexCoord80 : TEXCOORD80; //eightieth row of the 3x3 transform f
166     float4 TexCoord81 : TEXCOORD81; //eighty-first row of the 3x3 transform f
167     float4 TexCoord82 : TEXCOORD82; //eighty-second row of the 3x3 transform f
168     float4 TexCoord83 : TEXCOORD83; //eighty-third row of the 3x3 transform f
169     float4 TexCoord84 : TEXCOORD84; //eighty-fourth row of the 3x3 transform f
170     float4 TexCoord85 : TEXCOORD85; //eighty-fifth row of the 3x3 transform f
171     float4 TexCoord86 : TEXCOORD86; //eighty-sixth row of the 3x3 transform f
172     float4 TexCoord87 : TEXCOORD87; //eighty-seventh row of the 3x3 transform f
173     float4 TexCoord88 : TEXCOORD88; //eighty-eighth row of the 3x3 transform f
174     float4 TexCoord89 : TEXCOORD89; //eighty-ninth row of the 3x3 transform f
175     float4 TexCoord90 : TEXCOORD90; //ninetieth row of the 3x3 transform f
176     float4 TexCoord91 : TEXCOORD91; //ninety-first row of the 3x3 transform f
177     float4 TexCoord92 : TEXCOORD92; //ninety-second row of the 3x3 transform f
178     float4 TexCoord93 : TEXCOORD93; //ninety-third row of the 3x3 transform f
179     float4 TexCoord94 : TEXCOORD94; //ninety-fourth row of the 3x3 transform f
180     float4 TexCoord95 : TEXCOORD95; //ninety-fifth row of the 3x3 transform f
181     float4 TexCoord96 : TEXCOORD96; //ninety-sixth row of the 3x3 transform f
182     float4 TexCoord97 : TEXCOORD97; //ninety-seventh row of the 3x3 transform f
183     float4 TexCoord98 : TEXCOORD98; //ninety-eighth row of the 3x3 transform f
184     float4 TexCoord99 : TEXCOORD99; //ninety-ninth row of the 3x3 transform f
185     float4 TexCoord100 : TEXCOORD100; //hundredth row of the 3x3 transform f
186     float4 TexCoord101 : TEXCOORD101; //hundred-first row of the 3x3 transform f
187     float4 TexCoord102 : TEXCOORD102; //hundred-second row of the 3x3 transform f
188     float4 TexCoord103 : TEXCOORD103; //hundred-third row of the 3x3 transform f
189     float4 TexCoord104 : TEXCOORD104; //hundred-fourth row of the 3x3 transform f
190     float4 TexCoord105 : TEXCOORD105; //hundred-fifth row of the 3x3 transform f
191     float4 TexCoord106 : TEXCOORD106; //hundred-sixth row of the 3x3 transform f
192     float4 TexCoord107 : TEXCOORD107; //hundred-seventh row of the 3x3 transform f
193     float4 TexCoord108 : TEXCOORD108; //hundred-eighth row of the 3x3 transform f
194     float4 TexCoord109 : TEXCOORD109; //hundred-ninth row of the 3x3 transform f
195     float4 TexCoord110 : TEXCOORD110; //hundred-tenth row of the 3x3 transform f
196     float4 TexCoord111 : TEXCOORD111; //hundred-eleventh row of the 3x3 transform f
197     float4 TexCoord112 : TEXCOORD112; //hundred-twelfth row of the 3x3 transform f
198     float4 TexCoord113 : TEXCOORD113; //hundred-thirteenth row of the 3x3 transform f
199     float4 TexCoord114 : TEXCOORD114; //hundred-fourteenth row of the 3x3 transform f
200     float4 TexCoord115 : TEXCOORD115; //hundred-fifteenth row of the 3x3 transform f
201     float4 TexCoord116 : TEXCOORD116; //hundred-sixteenth row of the 3x3 transform f
202     float4 TexCoord1
```

Each language definition is stored in a little XML file within the FX Composer install tree, so new languages are easy to add, or you can tweak the existing ones around if you're so inclined.

# Scripting and Debugging Panels



Log

Debug

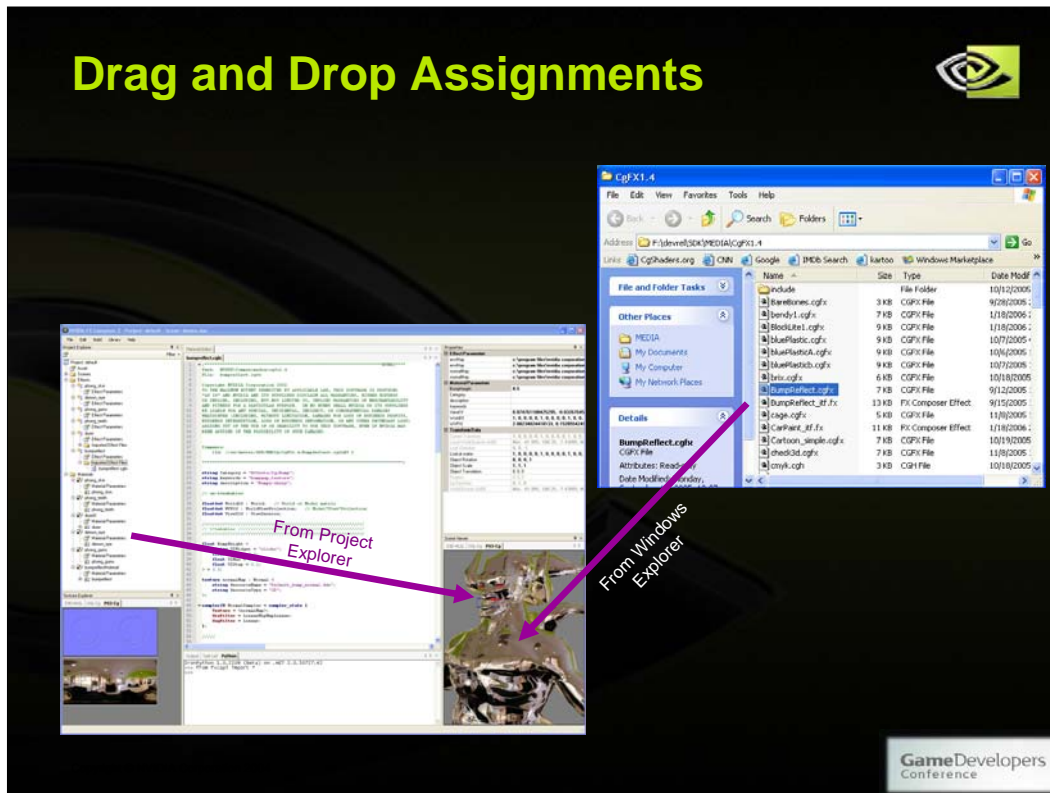
Script

```
Output | Task List | Python
IronPython 1.0.2208 (Beta) on .NET
2.0.50727.42
>>> from fxcapi import *
>>> Reset()
>>> tp = Teapot()
>>> tp.set_Name("thingie")
>>> Translate(0,-1,0)
<FXNodeCollection object at
0x000000000000002B>
>>> Undo()
>>>
```

GameDevelopers  
Conference

By default, the log, debugging, and scripting panels appear by default together at the bottom of the window. All windows are drag and dockable, so if you debug a lot, or script a lot, you can rearrange them to your fancy.

# Drag and Drop Assignments

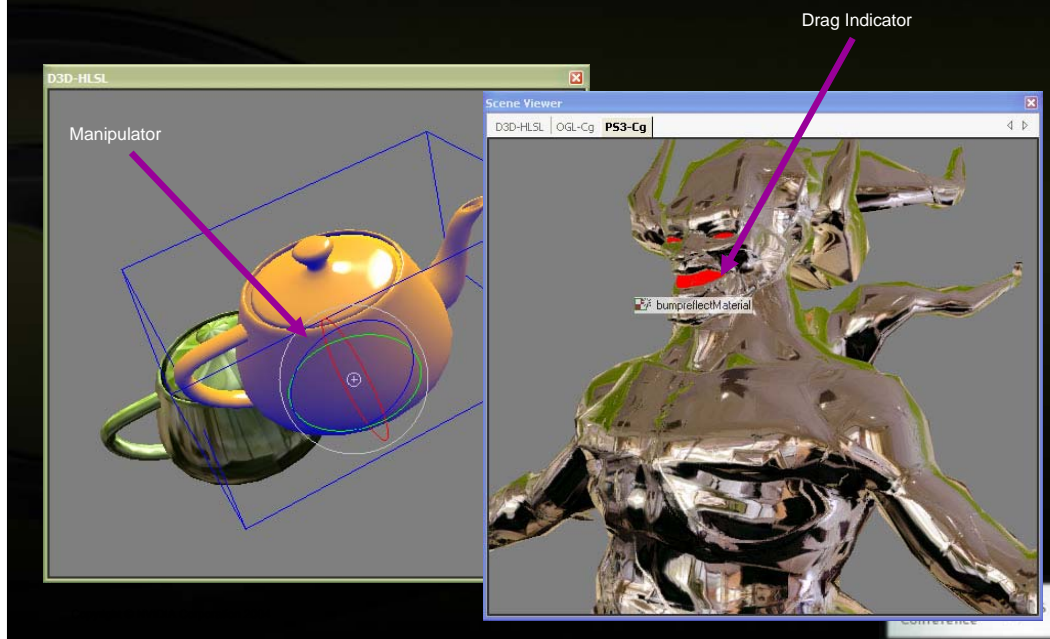


FX Composer lets us drag and drop both models and materials. We can drag effect files directly from Windows Explorer – either cgfx format or COLLADA FX files – directly onto any particular scene object. FX Composer will add these new effects to the current project, and is smart enough to recurse through any sort of indirect #include directives and so forth too.

The result will render and you'll see the new effect and a new matching material appear in the project explorer window.

OR, if you already have a material defined within your COLLADA project, you can drag it from the project explorer into the scene view

## Scene Viewer



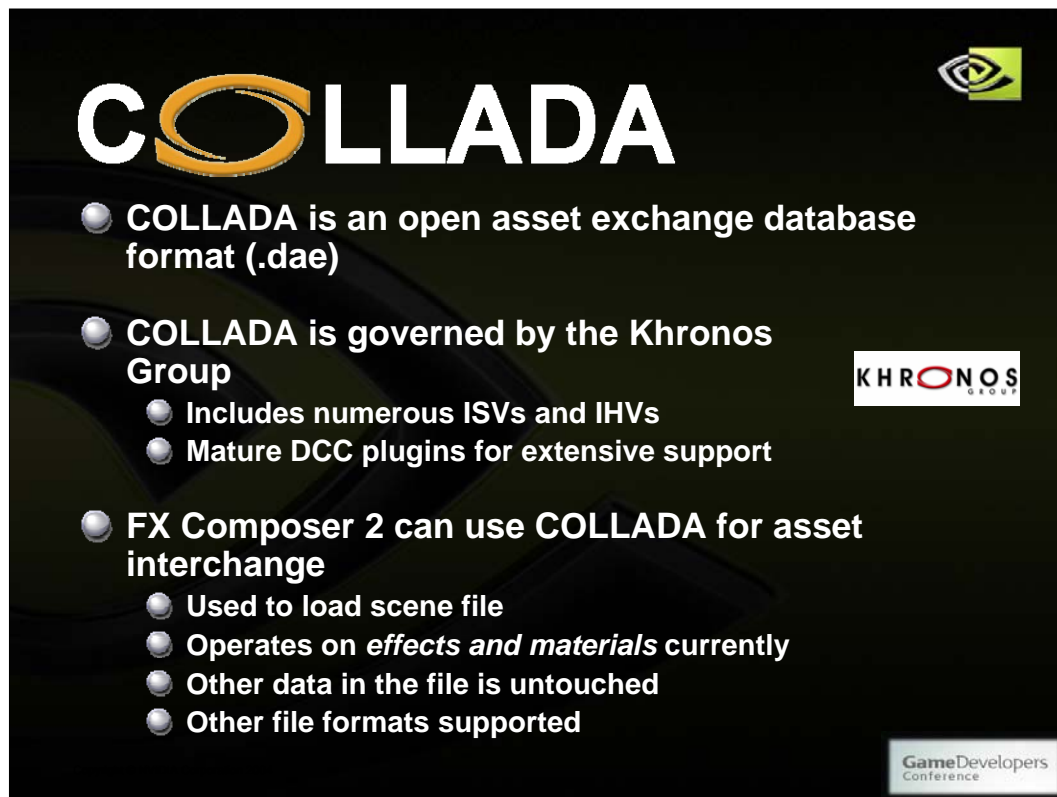
The scene window can display one or more views simultaneously, each with a different camera or even a different rendering method.

We can tumble and drag the camera by pressing the ALT or CTRL keys and zoom by rolling the mouse wheel.

We can also select objects and move them around by using manipulators – here's a rotate manipulator, it has three colored axes and in this picture one of them is yellow, which means it's being rotated right when the screenshot was snapped.

The manipulators have shortcuts, QWER across the top of the keyboard, Q for select, then W E R for translate rotate and scale, just like Maya. This will be in the notes for this talk too.

This second screenshot shows the name of a material – what we're seeing is a snapshot of FX Composer's drag-and-drop process.



# COLLADA

- COLLADA is an open asset exchange database format (.dae)
- COLLADA is governed by the Khronos Group
  - Includes numerous ISVs and IHVs
  - Mature DCC plugins for extensive support
- FX Composer 2 can use COLLADA for asset interchange
  - Used to load scene file
  - Operates on *effects and materials* currently
  - Other data in the file is untouched
  - Other file formats supported

Game Developers Conference

COLLADA....

FX Composer 2 supports the COLLADA file format. What is it? We had our own XML project format, nobody liked it.

“COLLADA is an open asset exchange database format” okay that’s a mouthful. What it means is that COLLADA can be used by multiple programs to exchange assets – models, shaders, scenes, textures, even animation and physics.

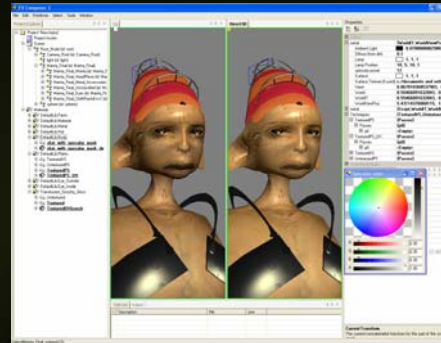
COLLADA is a shared standard, belonging to no one company. The Khronos Group defines COLLADA, manages and licenses it, and no individual software or hardware vendor can arbitrarily tweak it or change it. This is the way you want your standards to work.

FX Composer 2 can load a COLLAA file from most any source, ad edit it – but FX Composer 2 ONLY edits the shading-related parts of that COLLADA file – everything else passes through without change. FX Composer 2 isn’t an animation program or a paint tool, it’s a real-time shading tool.

## Devices



- Support Cg, Direct3D, PS3, GL-ES, GLSL, etc...
- Simultaneous rendering on the same model
  - COLLADA file contains different 'profiles' in the same effect
- Scene shown was imported from XSI, then Direct3D added



GameDevelopers  
Conference

Big new feature for FXC2 – Not just D3D anymore.

# Scripting



- **Plugin provided that enables scripting with IronPython.NET**
  - Any .NET language could be used though
- **Scripting is integrated completely into the engine**
  - ...because it talks to the engine the same way as any other plugin
- **Complete control**
  - You can shoot yourself in the foot if you want to...
  - ...with extreme prejudice

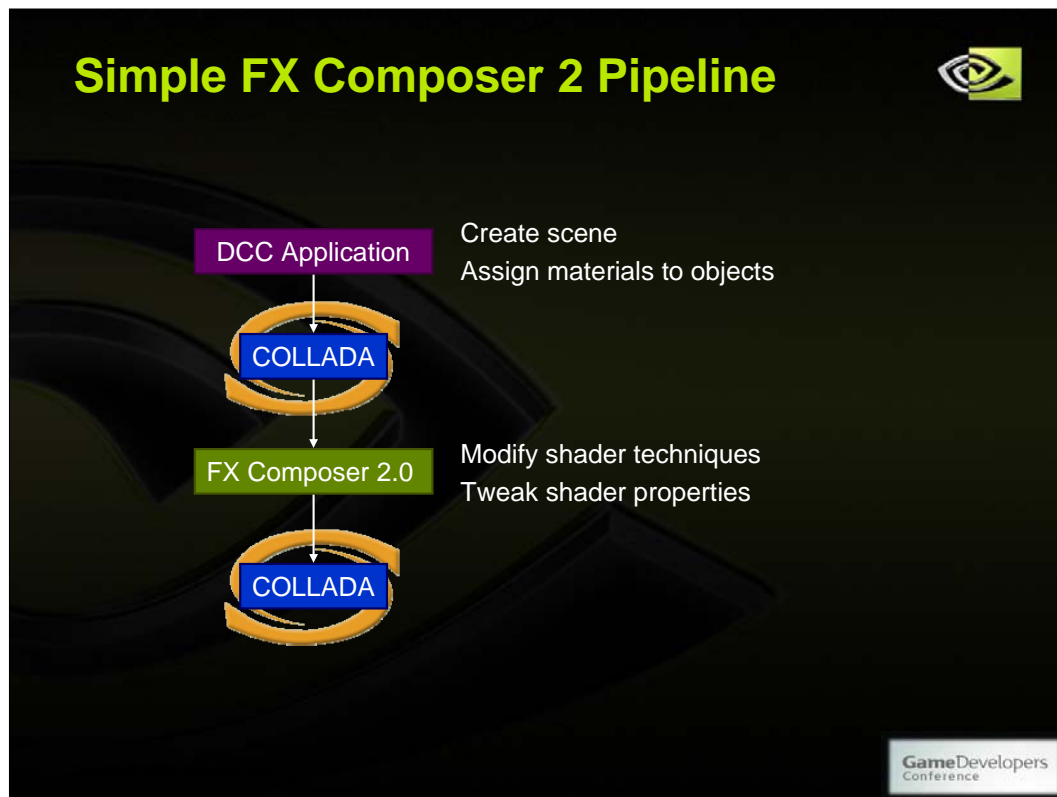
GameDevelopers  
Conference

Scripting through Python is powerful enough to get things done, with care.



Okay, hello – my name is Kevin Bjorke & I'm here to talk a little about FX Composer and its place in production processes, both in the abstract and also in a minute we'll show you the program in action.

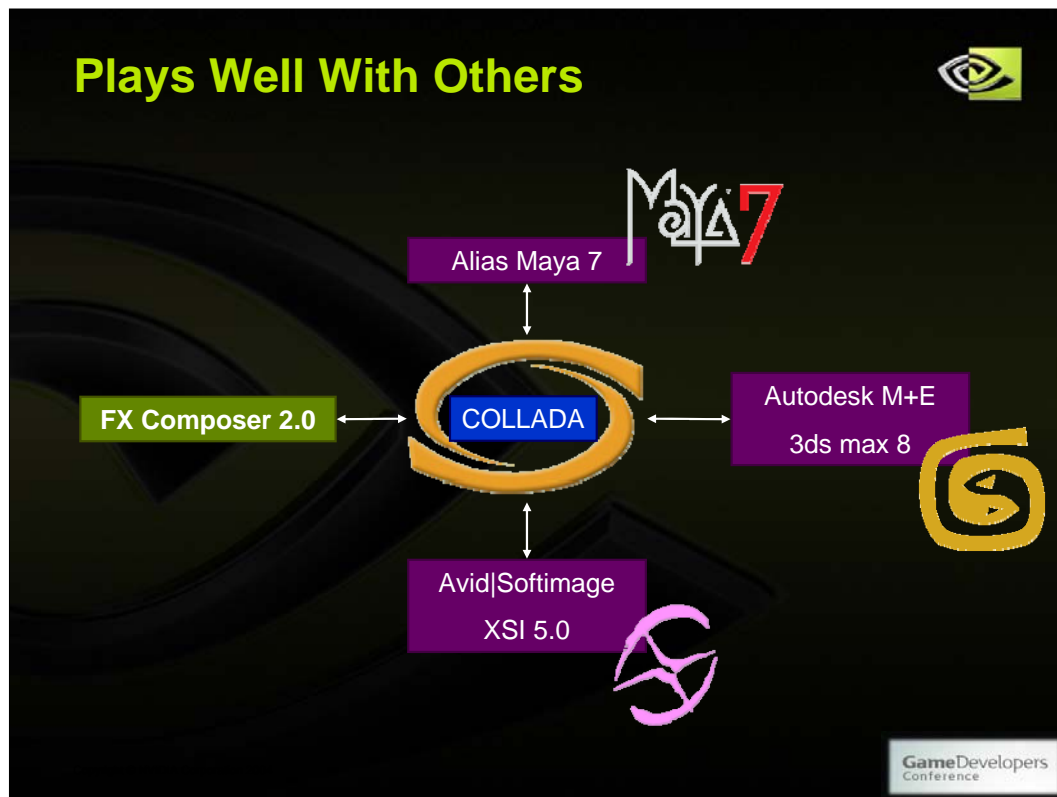




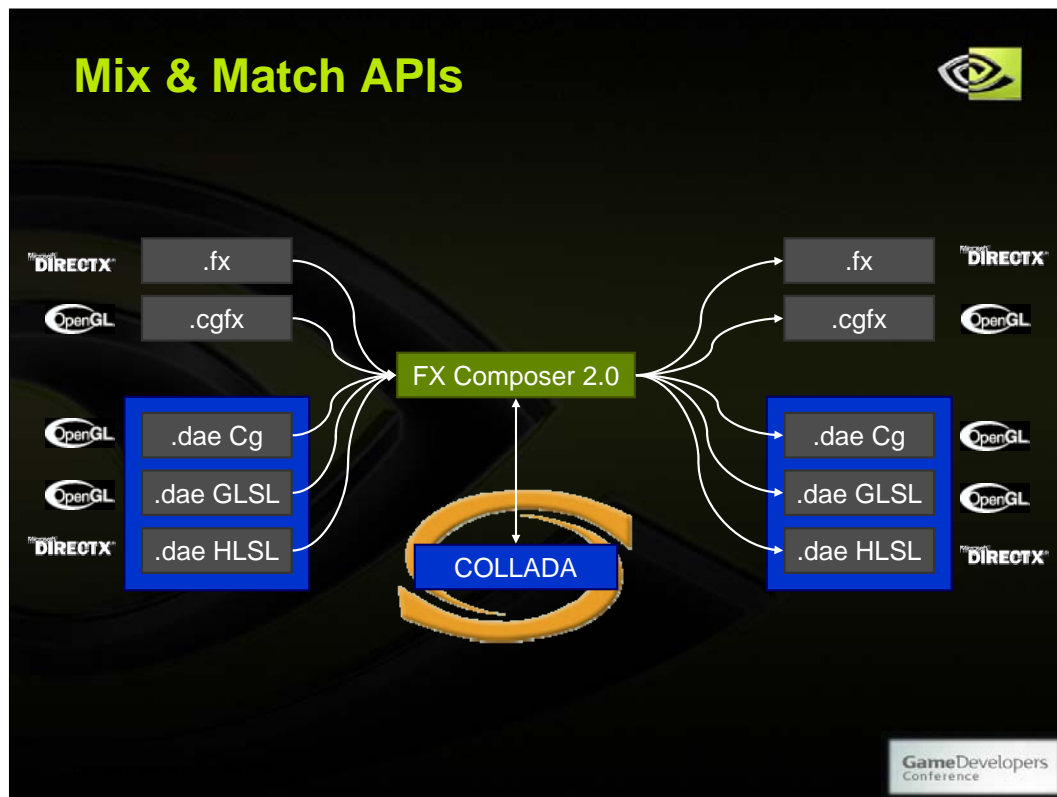
Here's a quick slide showing the simplest sort of use of FX Composer, where data has come from a DCC app like XSI as a Collada file, it's tweaked by FX Composer and a modified COLLADA file moves on downstream.

FX Composer isn't actually fully constrained to COLLADA usage – you can import models in other standard formats like OBJ or .X, or roll your own reader quite easily – FX Composer *will* write-out a new COLLADA file as a result but the portions of your scene that are stored in other formats will just be mentioned as references, to keep the core elements of your production pipeline consistent.

But real production rarely travels perfectly 100% downstream, right?



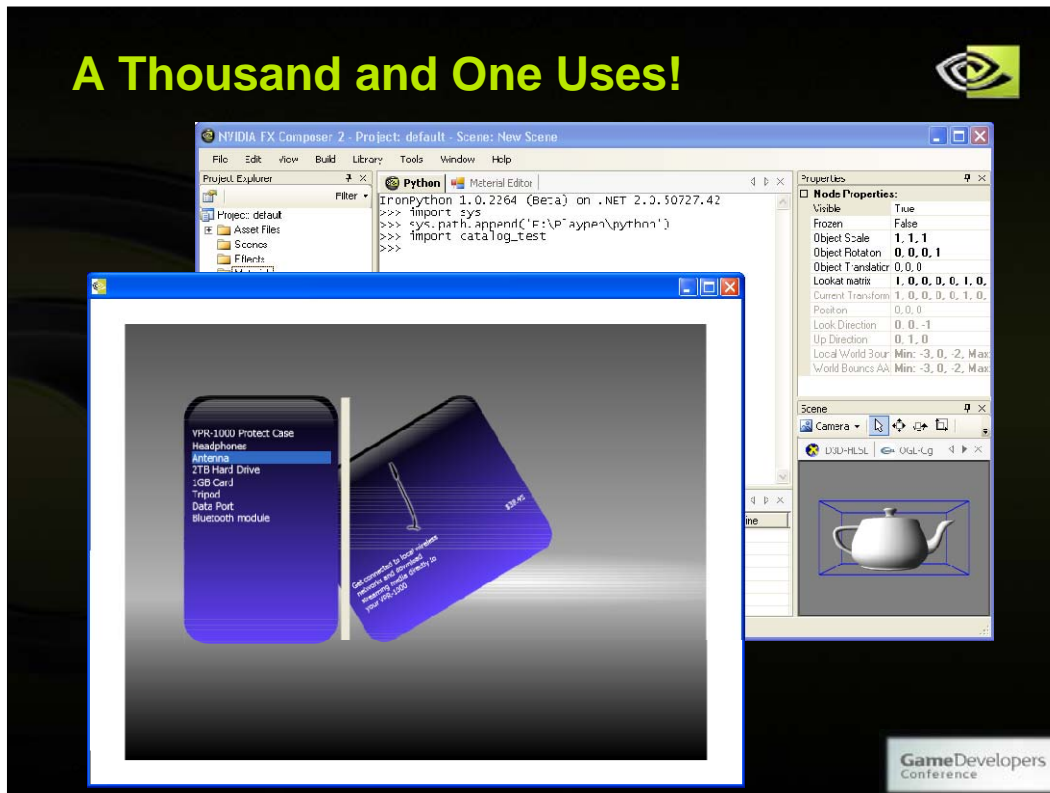
COLLADA *can* move models, rigging, animation, and more freely between applications, and FX Composer can play very well in that field – this is important because it lets the program be part of whatever sorts of production approval loops, stages, and production stages your studio may have in place. And it's really quite agnostic about which tools you use external to itself.



Here's just a quick chart, we won't dwell on it, but it shows how the program is happy in production environments of any sort because it can deal with any API or combination of APIs.

Our goal isn't to make a cool tool for creating NVIDIA demos, but to give developers and artist a genuinely useful channel to create, view, and manage shading in real, shipping games.

## A Thousand and One Uses!



So here's FX Composer in a role you probably haven't seen before!

The background window looks pretty typical – the teapot means it's an empty scene – but the foreground window is running a complex, XAML-based, Windows Vista-ready, database-connected animated Point-of-Sale kiosk UI, created and driven entirely by the Python scripting engine.

Yes, you can edit and view shaders from your favorite DCC application while shopping.

Okay, a LITTLE disingenuous, but the truth is that FX Composer *is* indeed running this additional sales-kiosk application via the IronPython scripting extensions. The point here isn't to sell MP3 players or SD cards, but to show off the ability of IronPython and FX Composer to integrate and extend across a variety of different parts of your workspace, and to have the ability to provide custom UI elements and operations as needed by whatever your own studio environment might need. You can use .NET, you can open sockets and talk to your asset database, the works.

As for Point of Sale displays, I'm not saying that you SHOULD....



**IronPython**

● “Iron” = “I Run On .Net”

<http://workspace.gotdotnet.com/ironpython>



```

Output | Task List | Python
IronPython 1.0.2208 (Beta) on .NET
2.0.50727.42
>>> from fxcomp1 import *
>>> Reset()
>>> tp = Teapot()
>>> tp.setName("thingie")
>>> Translate(0,-1,0)
<FxCNodeCollection object at
0x0000000000000026>
>>> Undo()
>>>
  
```

This acronym is NOT MY FAULT

But it is a pretty swell version of Python, very fast and full-featured and well-connected. The picture we just looked at was based on a XAML file created in Microsoft Expression Interactive Designer – you can use any XAML-based editor to create UIs, or code them up arbitrarily, or just use the Python window in a simple text-based form.

The Python editor itself is pretty sweet too, with keyword and class-member completion and so forth, making it relatively easy to navigate through FX Composer’s hierarchies. The installer includes the standard Python 2.4 libraries, and you’ll find that with VERY few exceptions, you can use any of them, or the .NET equivalents, pretty interchangeably.

If you’re already a Python hacker, or you’d like to know moer about python, definitely check out the IronPython web site, you can get a standalone edition as well. We’ve already been coding up ideas for tools and extensions to FX Composer for running in Pythin, for example, I translated the CgFX shader creator, previously written in Mel for use in Maya, into a Python class for use in both XSI and FX Composer – took just a couple of hours and now it’s useful and general for ever.

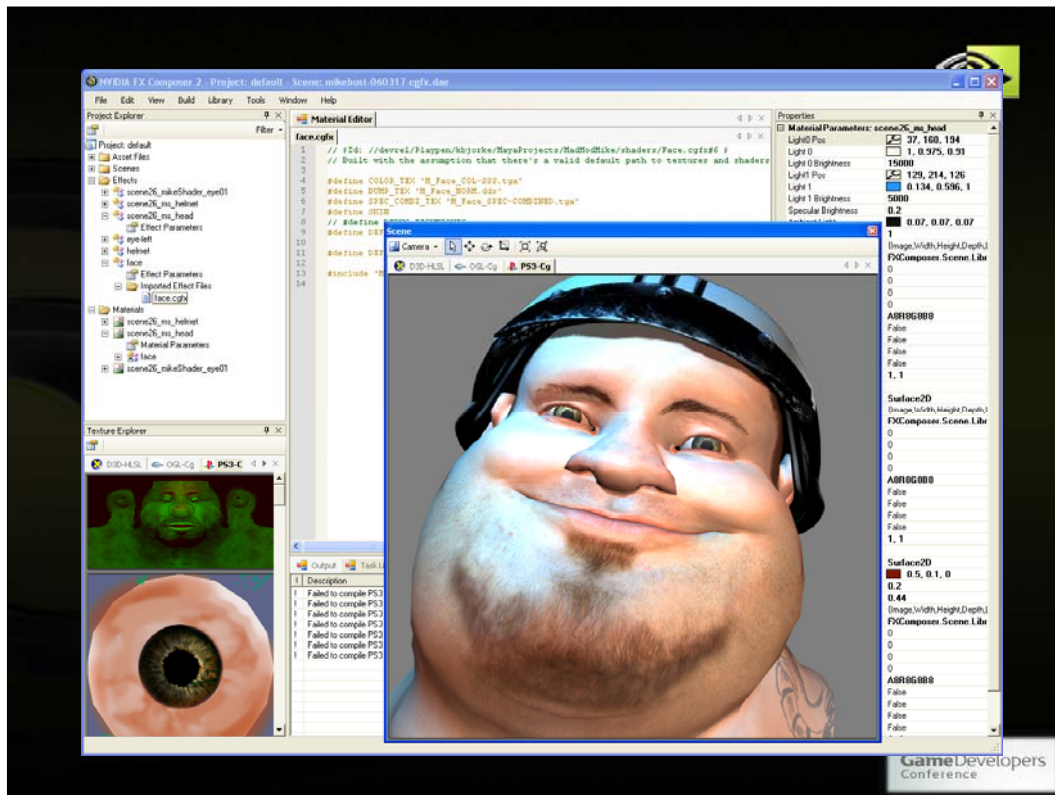
## Integrated Shader Profiling



- Convenient tweak-and-profile workflow to tune shaders
- Integrated NVShaderPerf 2.0 gives access to:
  - Performance across multiple GPUs and drivers
  - Assembly output
  - Vertex and pixel throughput
  - Cycle count
  - Register usage
- Coming Soon

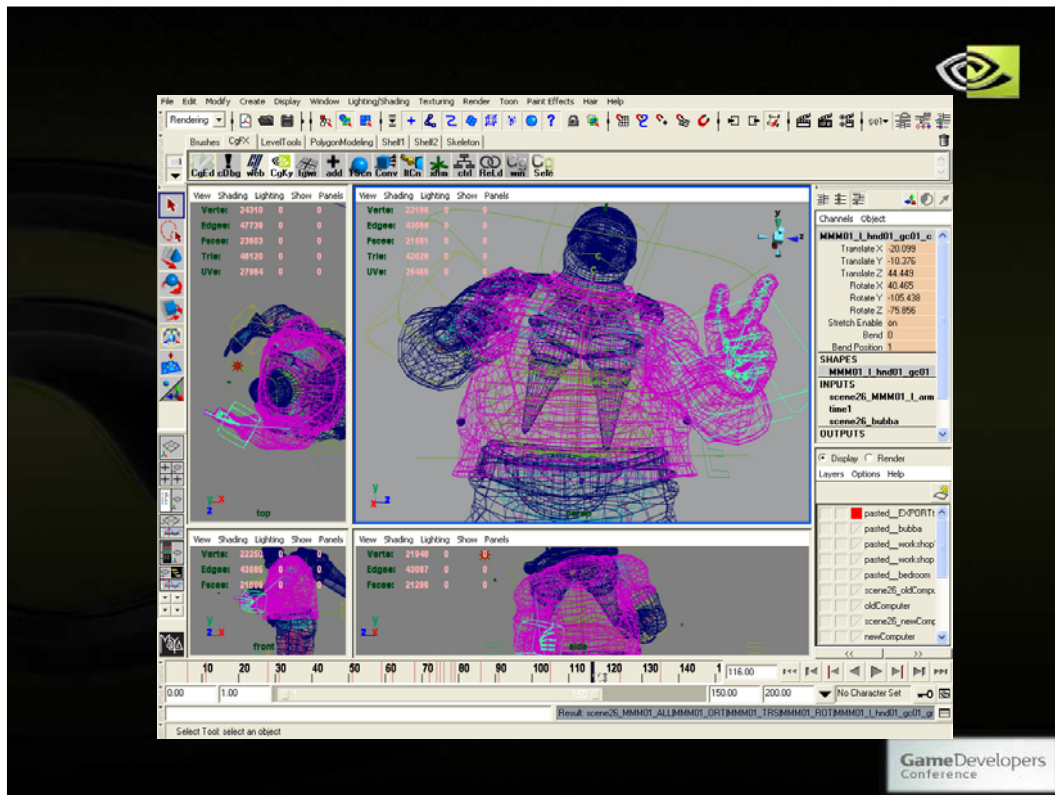
GameDevelopers  
Conference

This feature is crucial for production TDs and engine programmers and if you're used to it from FX Composer 1.x you know what I mean. Again, I won't dwell on this slide since Jeff Kiel gave a complete talk on NVShaderPerf just a few minutes ago (also available on the NVIDIA website! <http://developer.nvidia.com/>), but an integrated panel for shader perf analysis is a key part of why FX Composer is useful, and it's something that is a unique part of pipelines that include FX Composer 2



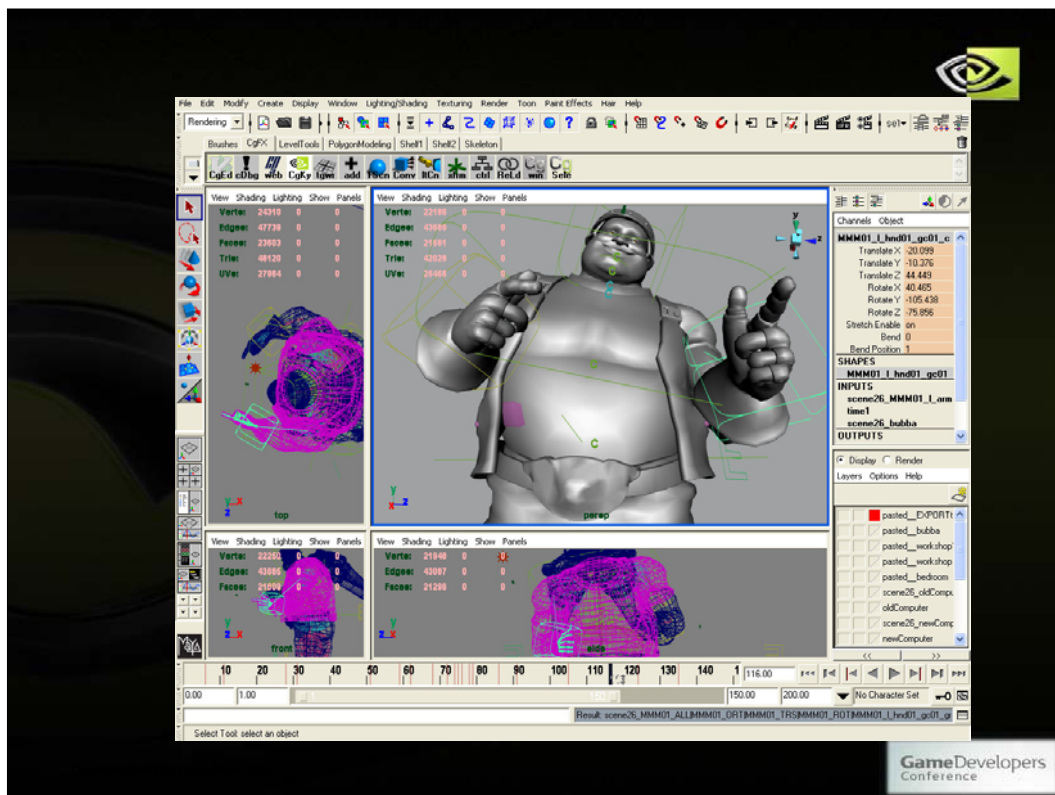
Let's take a look at bringing up Mike in Maya and FX Composer 2



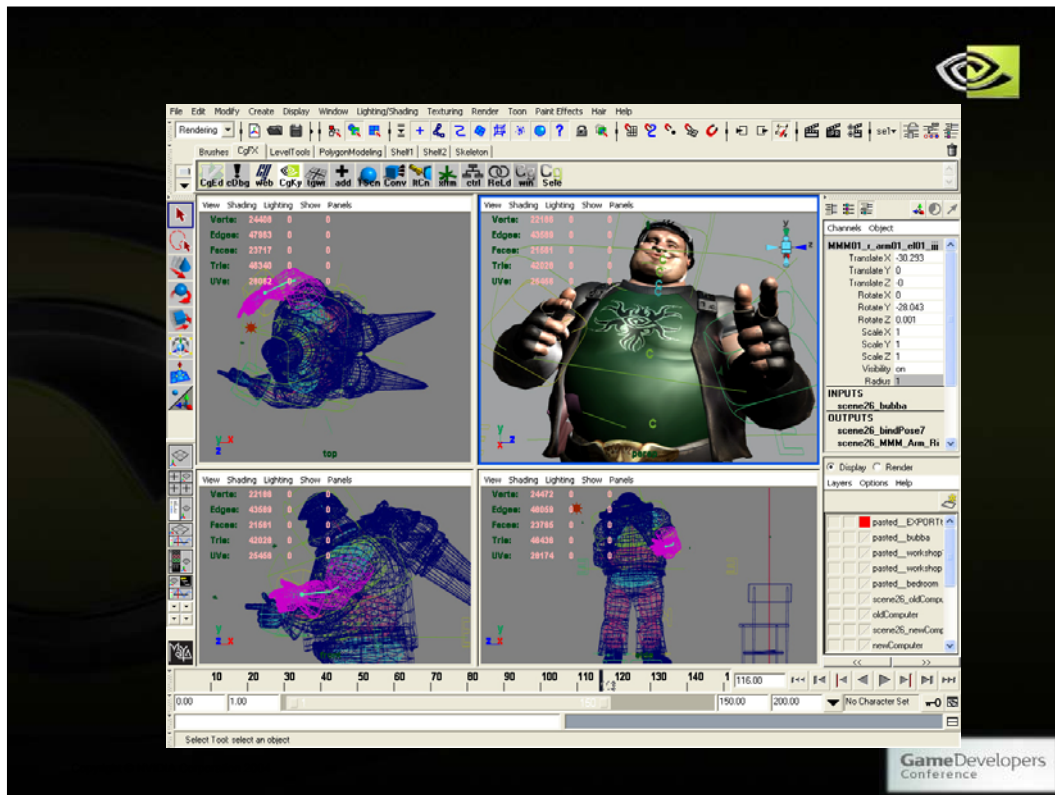


Mike was animated in Maya originally, but the steps here would be much the same in any other DCC application



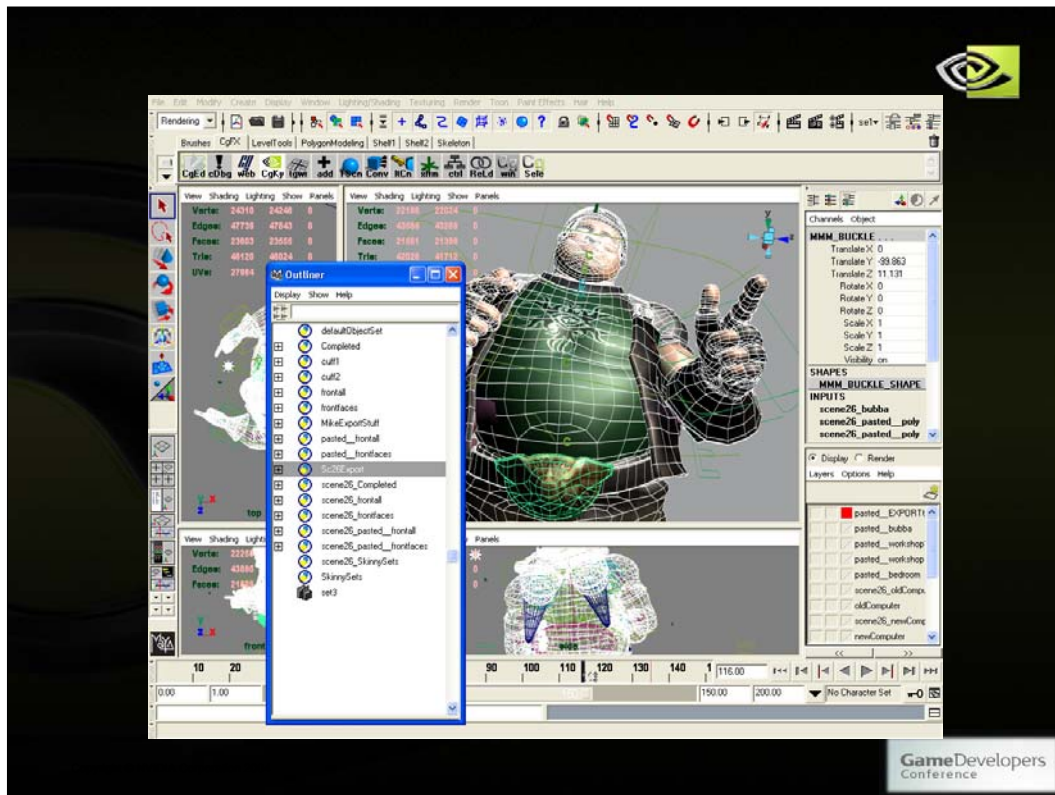


...shaded...

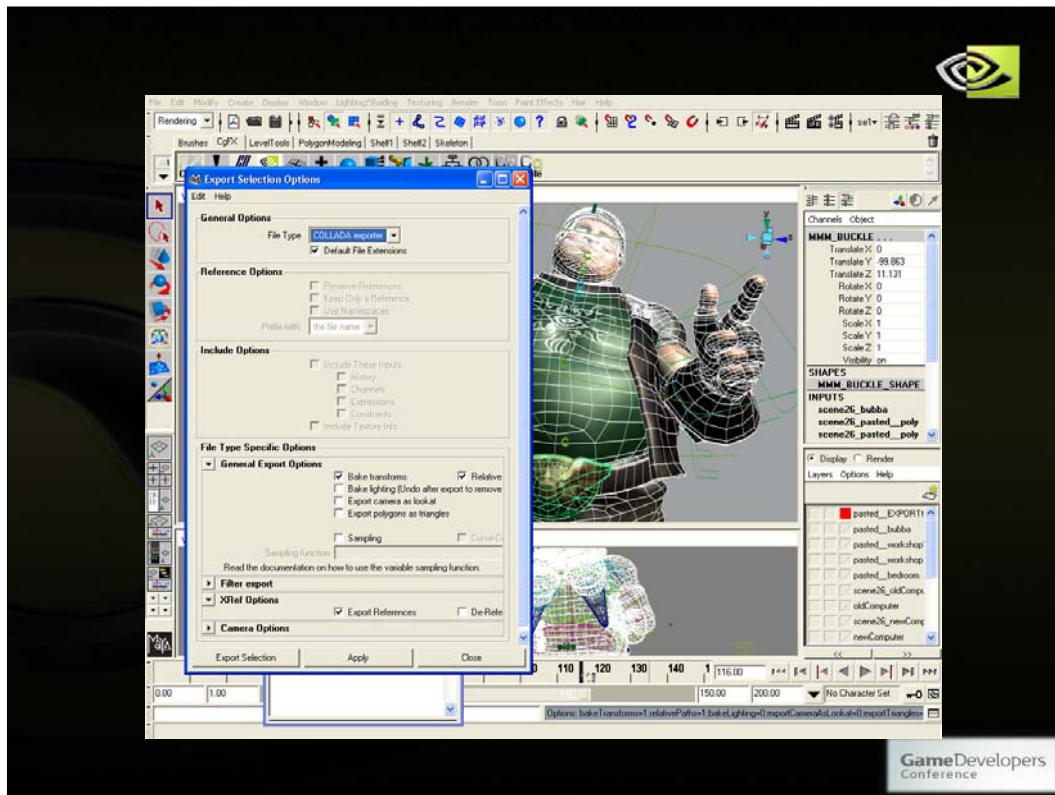


Mike was originally made and textured within Maya, so he has Maya materials but no realtime shading. So we're going to export him and assign new materials in FX Composer

(Quick aside: the shaders we will use in the sample were actually generated to match Maya's! The Mel scripts for CgFX, available for some time in the CgFX toolbar, have been ported to Python & were used to generate the appropriate templates, before being further tweaked by hand.)

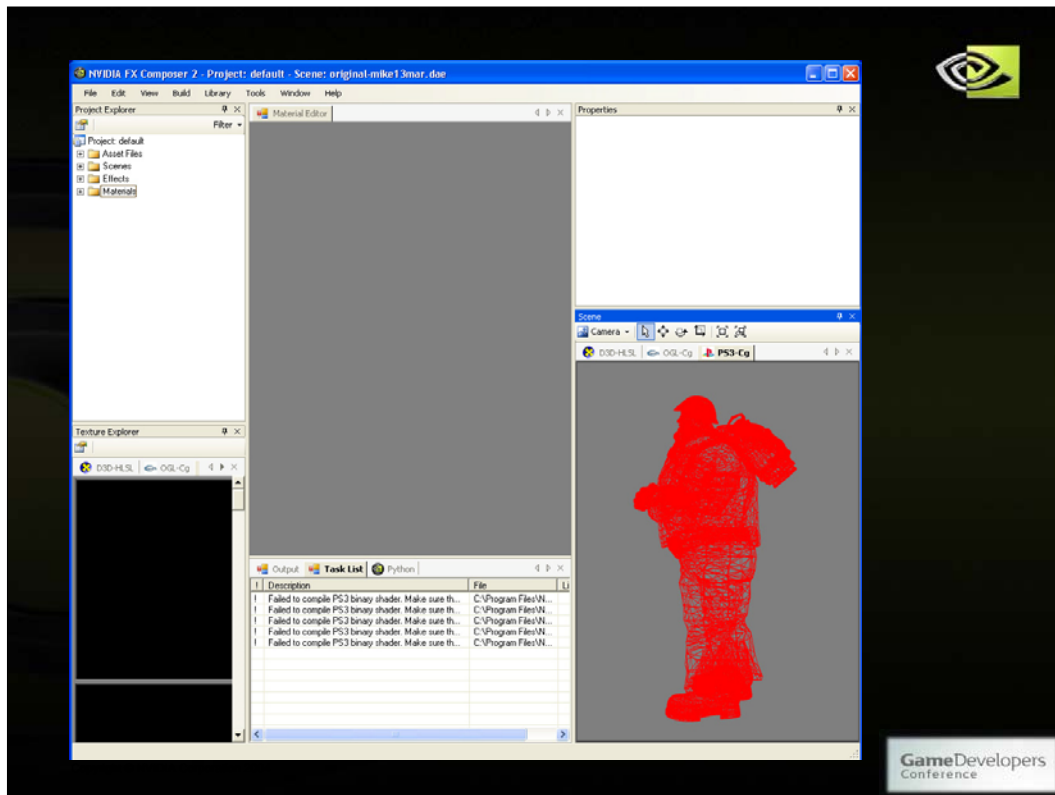


I just want to shade Mike so I've made a set containing just the parts I want to export



...and I “export” to COLLADA the same as I might to any other format.

So let's switch to FX Composer



I can just drag and drop my new .DAE COLLADA file onto the FX Composer scene window and Mike will load.

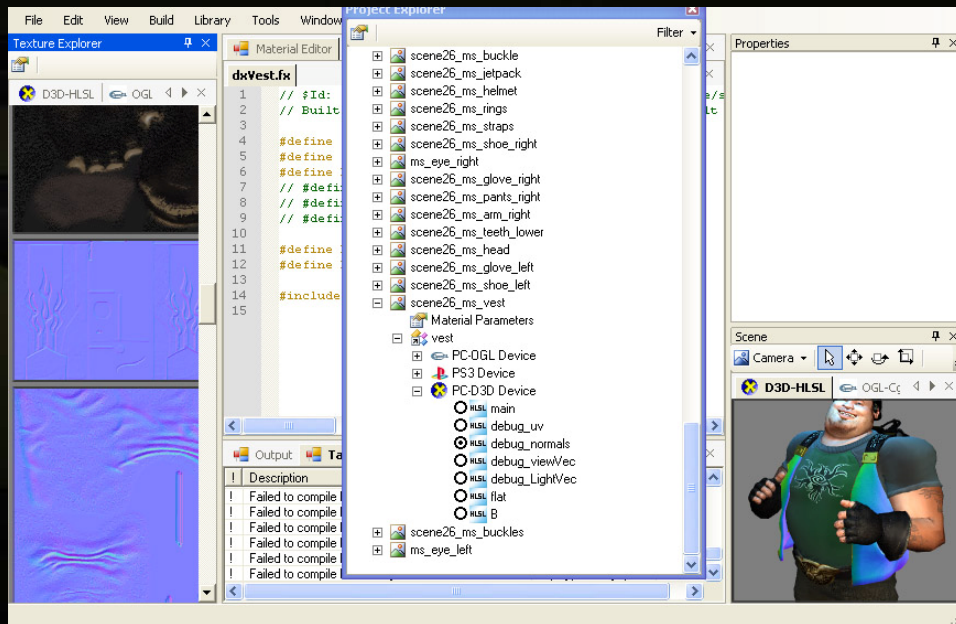
Wireframe means materials are assigned to those surfaces, but those materials have no valid effects for the current profile (which in this illustration is "PS3-Cg")

We can similarly drag effect file from the Windows Explorer onto surfaces, where they'll be automatically assigned, or into the blank areas or the Project Explorer, where they'll be defined as effects (and then we can assign them to materials as a second step – select the material and right-click to see "Assign Effect....")



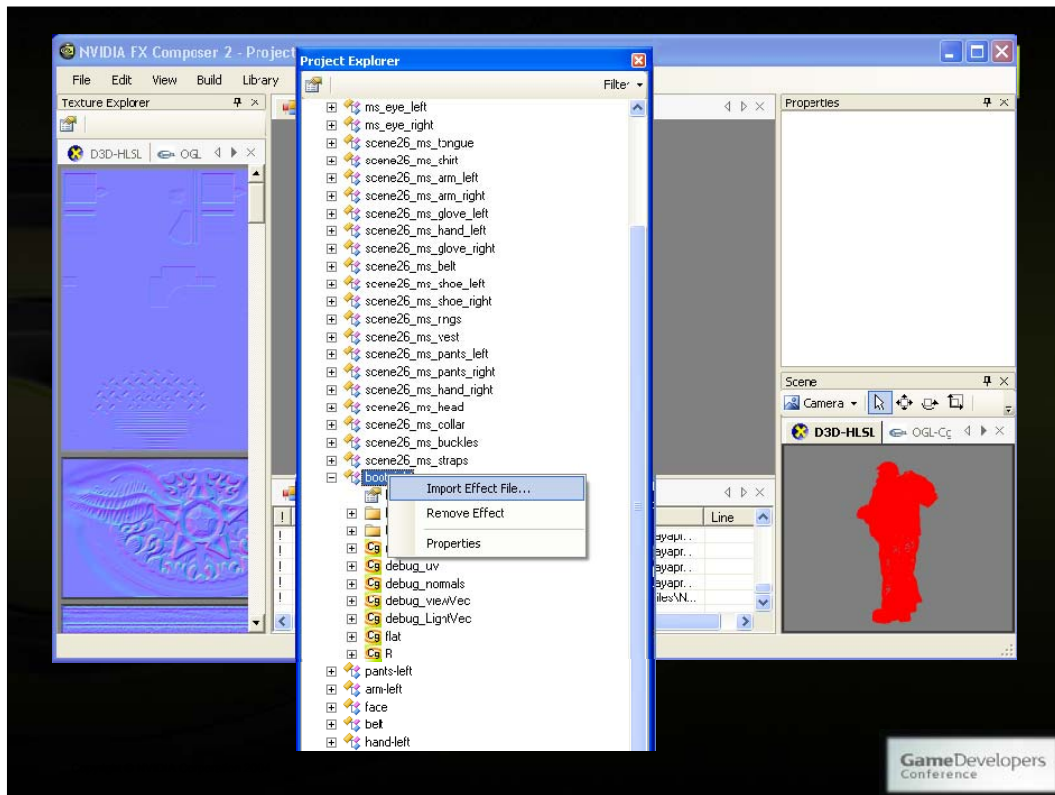


## All Devices Have Technique Lists



GameDevelopers  
Conference

Note the debug techniques... optional but handy



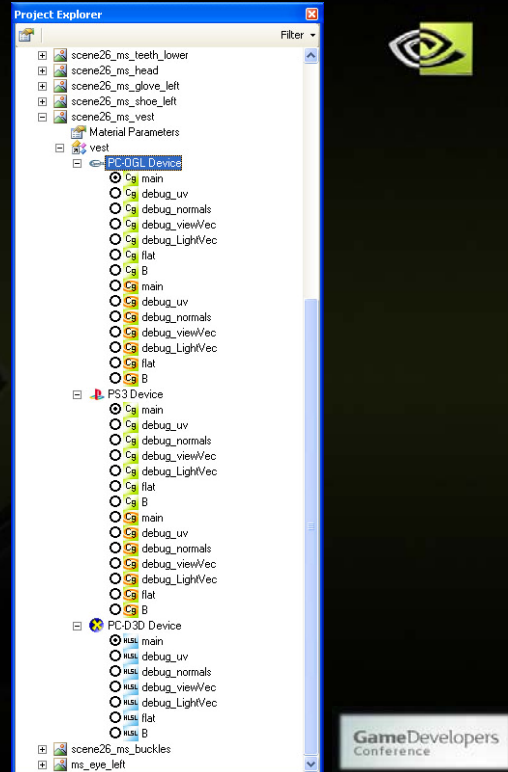
Assigning extra effects – switching to DirectX view, we can see that the model is still in red wireframe – no effect techniques are defined for THIS profile. So let's add some.

I've popped-free the Project Explorer pane so I have more working space, and I can add additional effect files (CgFX or HLSL FX) so that the effect can be valid in multiple profiles.

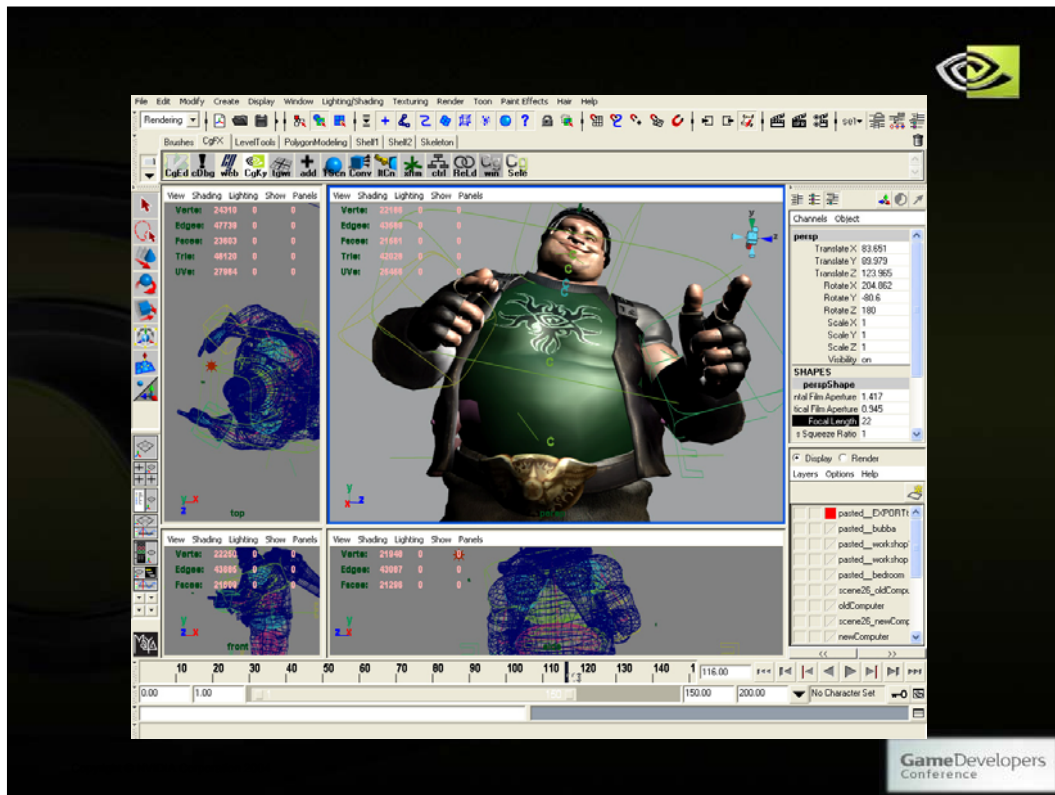


## Technique-a-Rama

- Each material will have a list of techniques for each render device (here they are all the same, but it's not required!) and all will share the material parameters



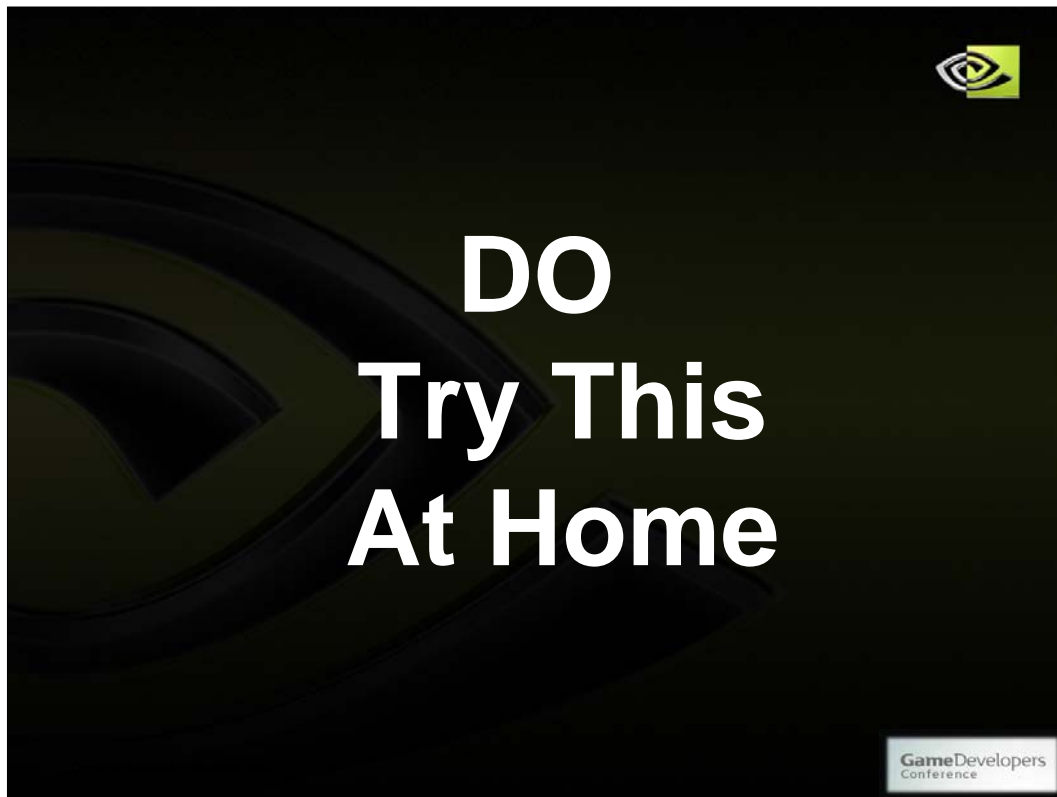
Here's a closeup of a fully populated scene valid for all current profiles.



But instead of looking at slides, let's do this again right here!

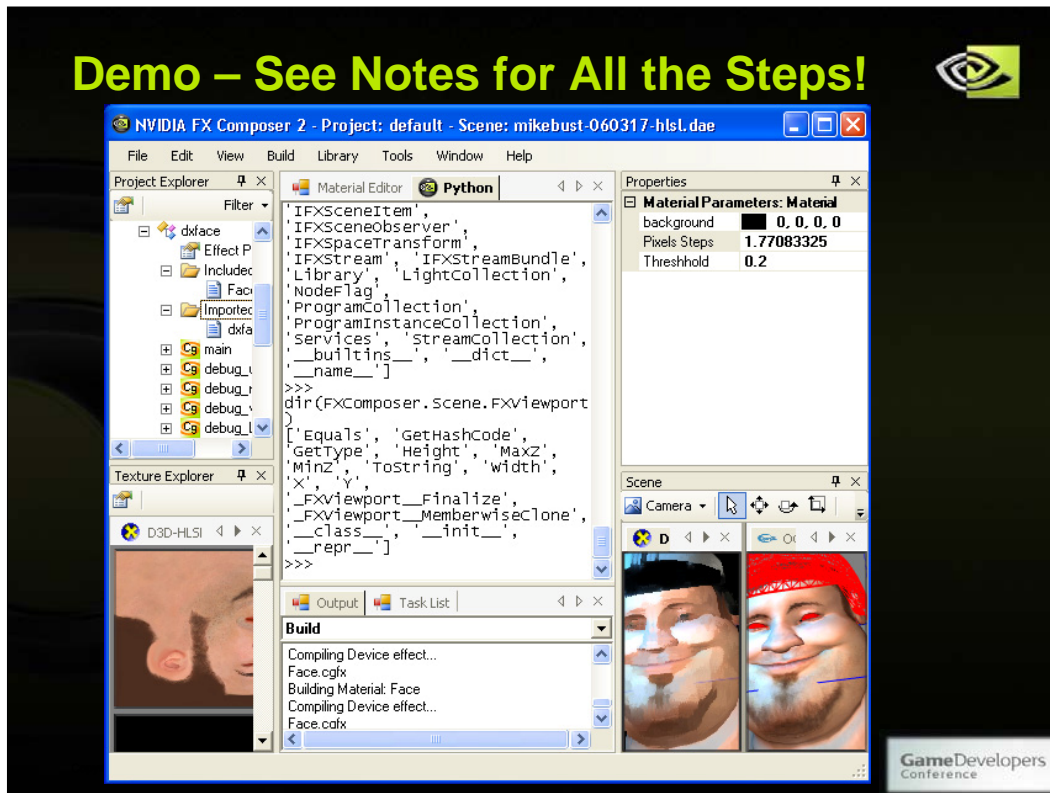
Mike has lots of parts, so for the sake of a quick demo I'm going to pick a frame where his mouth is closed (so I can ignore his teeth and tongue) and just export his head, eyes, and helmet.

Let's load that result and try it out!



Let's start with a fresh FX Composer 2, our fresh Mike head model ("MikeBust") and begin....

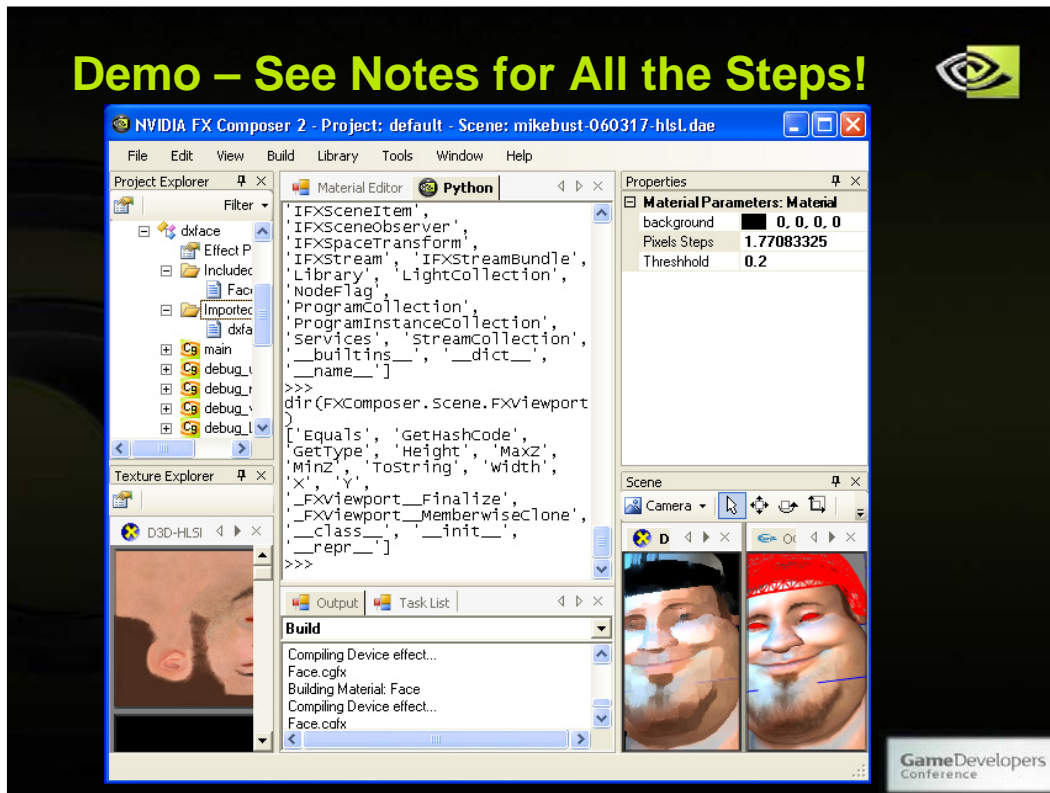
## Demo – See Notes for All the Steps!



### PART ONE

1. Start FX Composer 2
2. Resize so that we can see Windows Explorer windows, too
3. Drag-n-drop mikebust.dae into TEXT window – see syntax highlights etc – close it (just looking)
4. Drag MikeBust dae again – onto the scene pane. It will load & show red wireframes
5. Select different render devices – settle on DirectX
6. Find fx Shaders in the Windows explorer – dxFace fdxEye-Left dxHelmet
7. Drag dxHelmet onto helmet
8. Drag dxFace onto face
9. Eyes are tiny, so drag onto blank area or into the Project Explorer pane
10. Look at proj explorer – see the effects and materials
11. Select eyes material, and Assign “dxEye-left” effect
12. Rotate around in the scene view by dragging while hold-down “Alt”
13. In and out zoom with the mouse scroll wheel
14. Make scene pane big... double-click on the title bar and then resize.  
Double0click agaion to put it back & vice versa
15. Zoom extents on eyes – click on an eye to select, then press zoom extents on the scen pane toolbar
16. Zoom extents on the whole scene

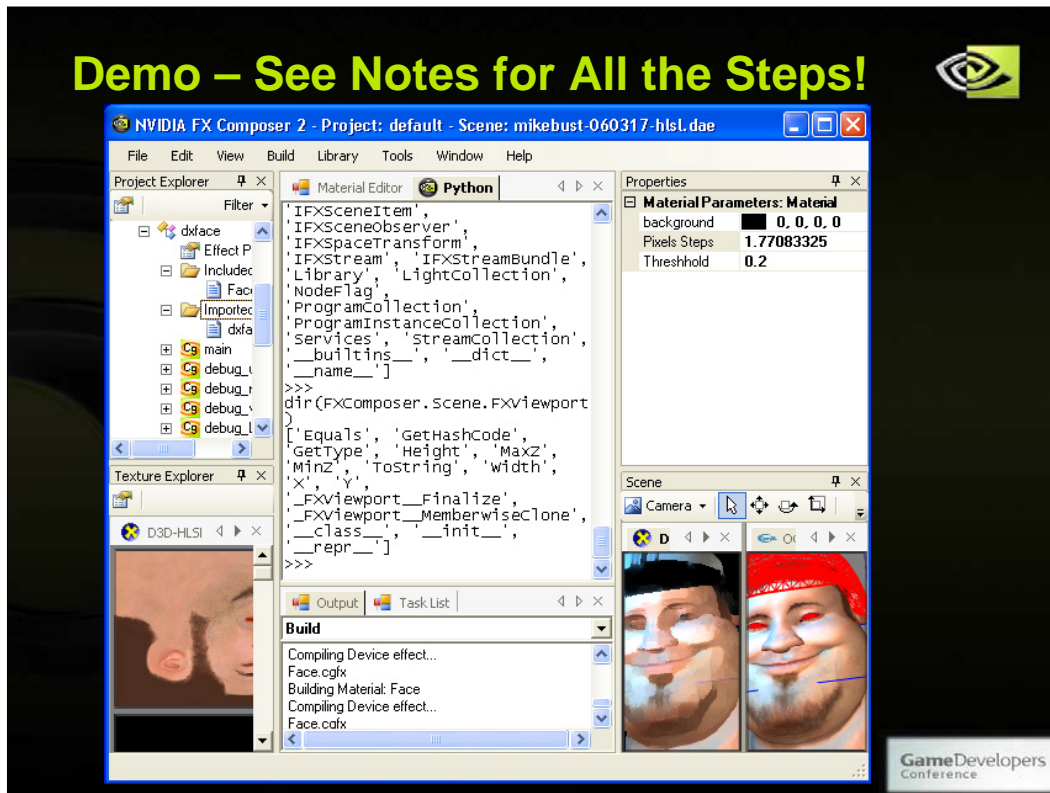
## Demo – See Notes for All the Steps!



## PART TWO

1. Select the face in the scene pane, and zoom on it
2. Look at assigned techniques for this in the Project Explorer
3. Show debug techniques by clicking the radio buttons.
4. Go back to "main" technique
5. Select material parameters
6. Collapse textures in the properties pane
7. Roll bump and sub rolloff sliders
8. Change subcutaneous color by clicking the color and then the edit button
9. Zoom back out in the scene pane. WAY out.
10. Tools->create point light. New light is already the selected object
11. Drag it around near mike – press "w" to get the move manipulator
12. Materials properties on "Face" – click the "light0 pos" edit button and assign to "point light"
13. Connect each material to this light
14. Wow bright! Move light around until it's a good brightness

## Demo – See Notes for All the Steps!



## PART THREE

1. Select dxface effect in the project explorer, and right-click for "add effect file"
2. Import face.cgfx – give it a second or two to compile
3. Look at new list of files under this effect
4. Double-click file – text editor appears – note highlighting etc
5. Switch to the head material – hey we have new techniques! Note properties are already set!
6. Switch to PC-OpenGL and PS3 views to see what's defined and what's not
7. Back to face effect
8. Convert face.cgfx to colladafx – select the cgfx and right click "convert to COLLADA FX..."
9. Note that imported effect go smaller, imported shader grew, and effects are now managed by collada
10. Peel-off ps3 pane as its own window, note dx is still rendering!
11. Assign other ogl effects just as we did for "face" – each window can have a different technique selected
12. Select the helmet
13. Drag python window to main pane - watch how pressing [tab] can help you when you type:
 

```
from fxapi import *
Translate(0,2,0)
Undo()
import FX Composer.Scene
dir(FXComposer.Scene)
```
14. Drag-on post\_kuwahara.fx (or any other HLSL full-screen effect) onto DX window – cool huh? Note the new material has a checkbox to enable/disable it
15. Keep screwing around! Go crazy layering effects and trying things.





**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.

Thanks for coming to see this new tool!

Be sure to contact us for any ideas, schedules, and needs you might have.