nvFX : A New Scene and Material Effect Framework for OpenGL and DirectX

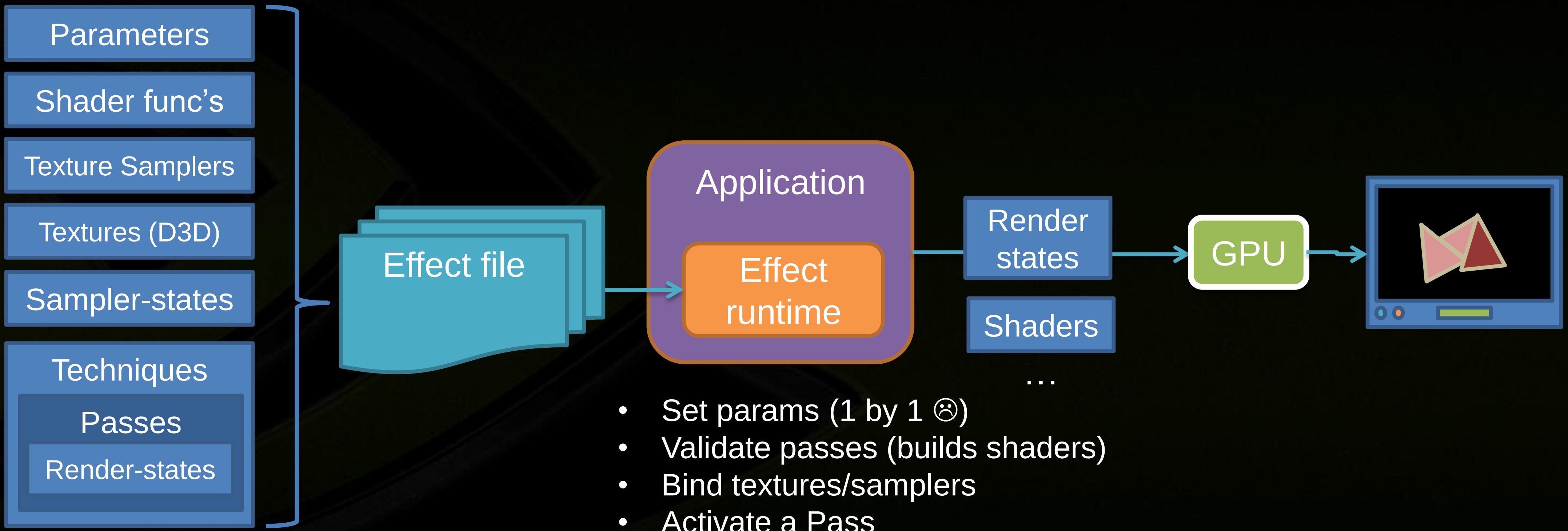
Tristan Lorach
tlorach@nvidia.com

NVIDIA

What is an effect ?

- Higher level management
  - Of Shader Code
  - Parameters
  - Samplers, Textures and Sampler States
- Allows to package all in one “ecosystem”
- Concept of Techniques and Passes
  - A Techniques == a way to perform a specific setup for specific rendering
  - Pass : setup Shaders and render states for a rendering pass
- Important : an Effect file is not directly sent to the Driver/GPU
  - CPU work here to maintain the loaded effect

Standard Effect design



- Set params (1 by 1 ☹)
- Validate passes (builds shaders)
- Bind textures/samplers
- Activate a Pass

Issues with Existing Effect (CgFX or DX FX)

- Cg
  - CgFX part of Cg toolkit; written in Cg
  - Source code of CgFX not available
  - Specs never evolved since 2002
- Microsoft DirectX ®
  - HLSL Shaders Only
  - Features never evolved
  - Nobody using it, nowdays
- Khronos Groups's GLSL
  - Nothing available
- Let's make a Generic *Open-Source* solution !

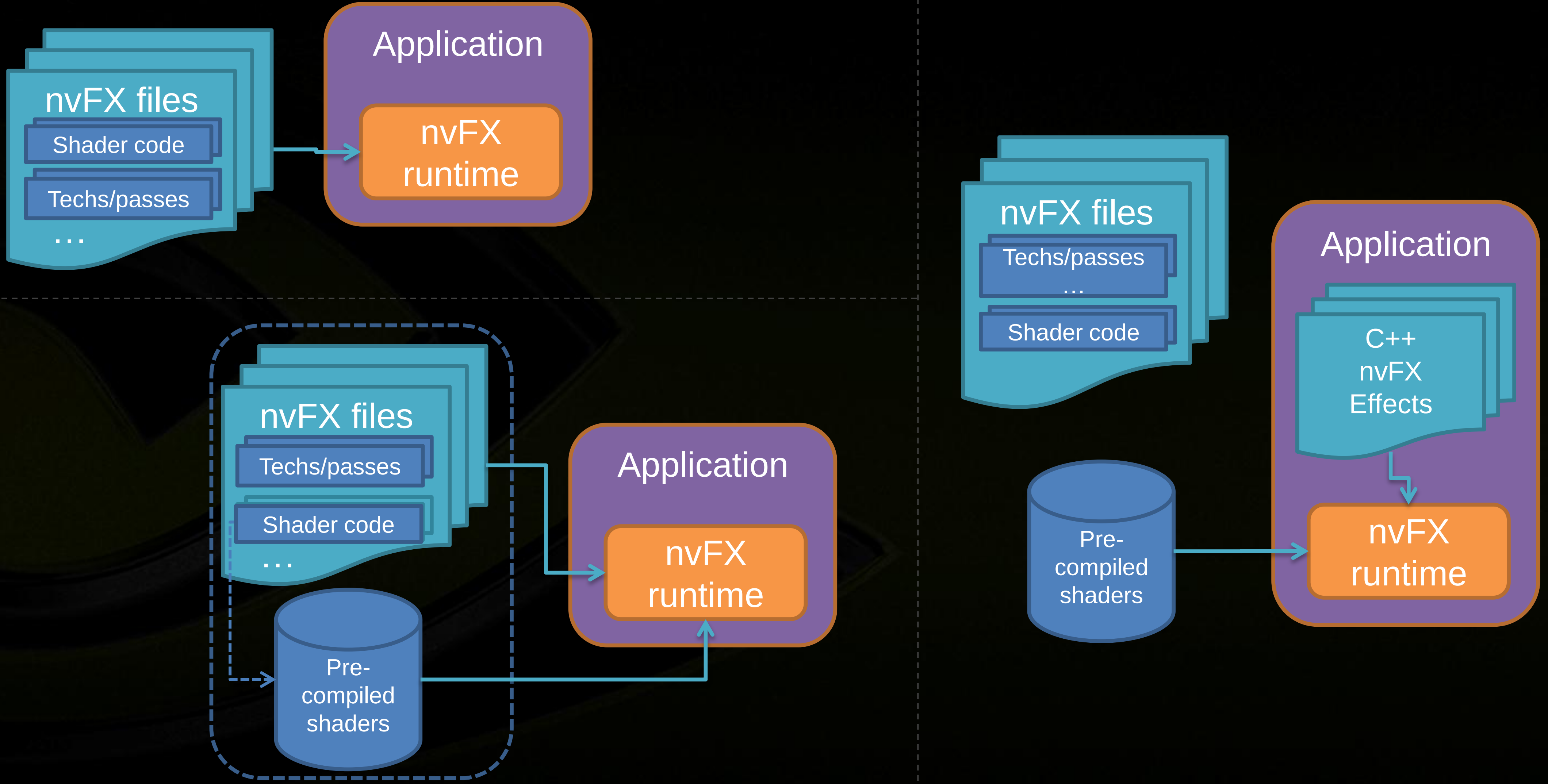
Expectation For A New Effect Design (nvFX)

- Host many Shading languages (GLSL, GLSLCompute, HLSL, CUDA…)
- Effect must be as self sufficient as possible
  - Very few special C++ implementation from the hosts application
- Simplify the code in the Application
  - Better maintenance & productivity
- consistency in Effect file and between Effect files
  - ➔Modularity for various Shadowing, Lighting (etc.) implementations
  - Post-processing of the scene ⇔Object materials consistent
- Self descriptive and easier to read
  - Spares us 100s of #ifdef #else #endif (Games do this a lot)

User Target

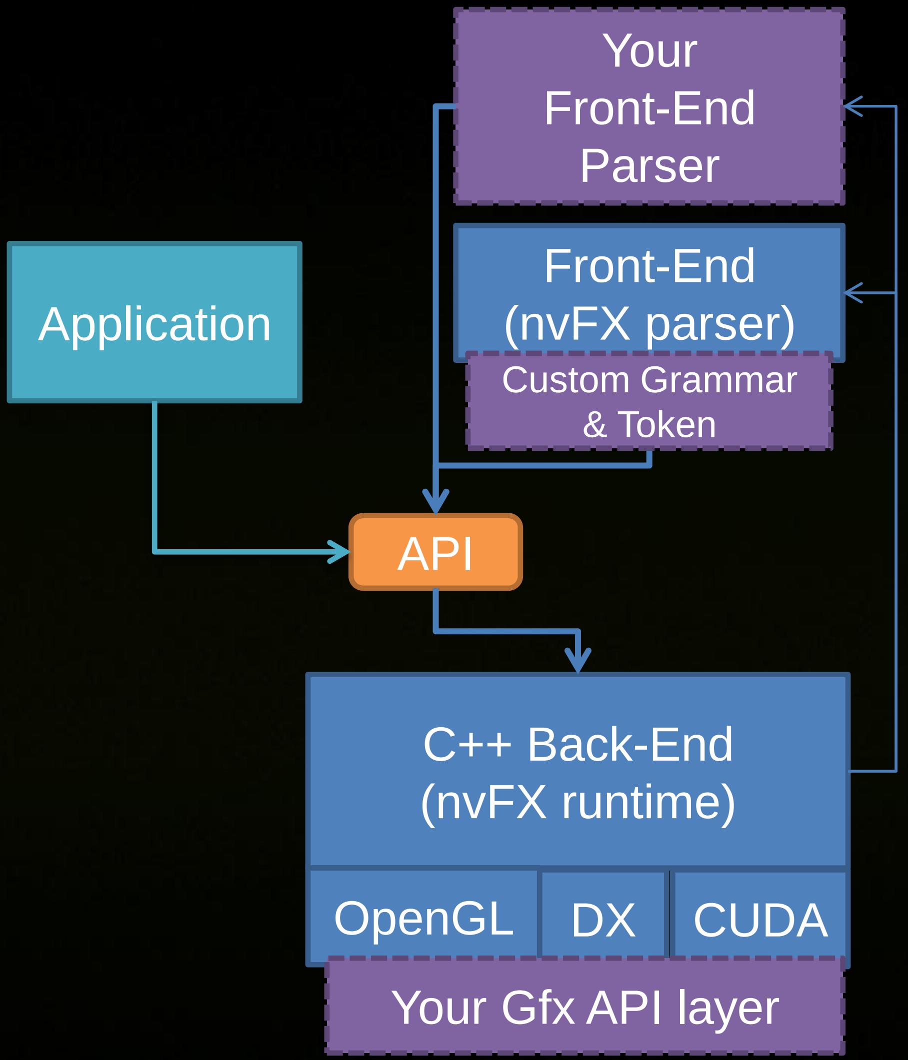
- Games
  - Helps highly combinatorial Shaders
  - Avoids heavy pre-processor code (#ifdef/#else/#endif everywhere)
  - Runtime optimizations of nvFX designed to be efficient
- Workstation CAD/DCC
  - Convenient to expose some programmability to the end-user
  - Helps for maintenance of heavy projects
- Labs / research (Prototype for a Siggraph paper !)
  - Helps to perform rapid and flexible prototyping
  - Convenient for Demos, Samples showcasing Shaders

nvFX Effect Integration

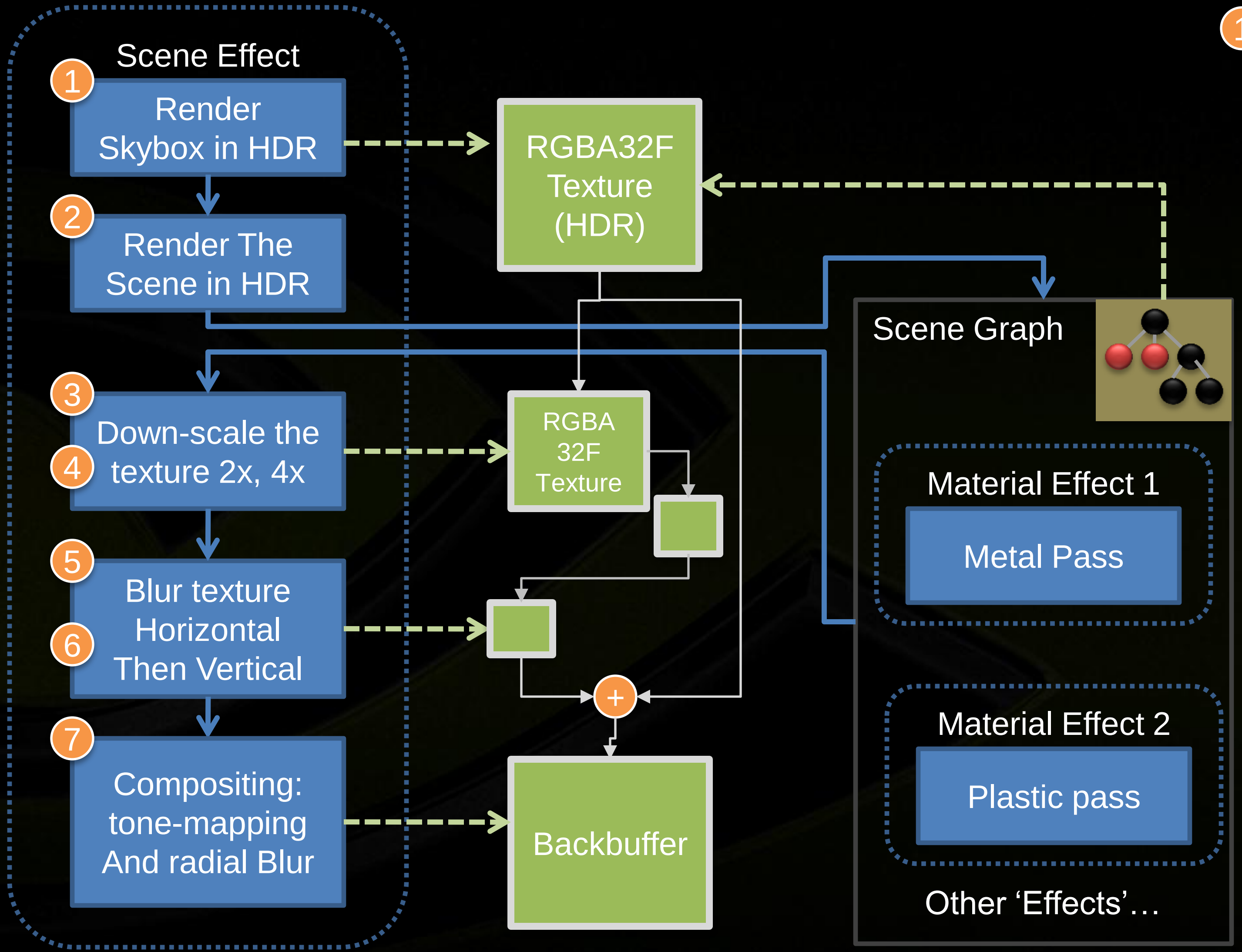


API Design

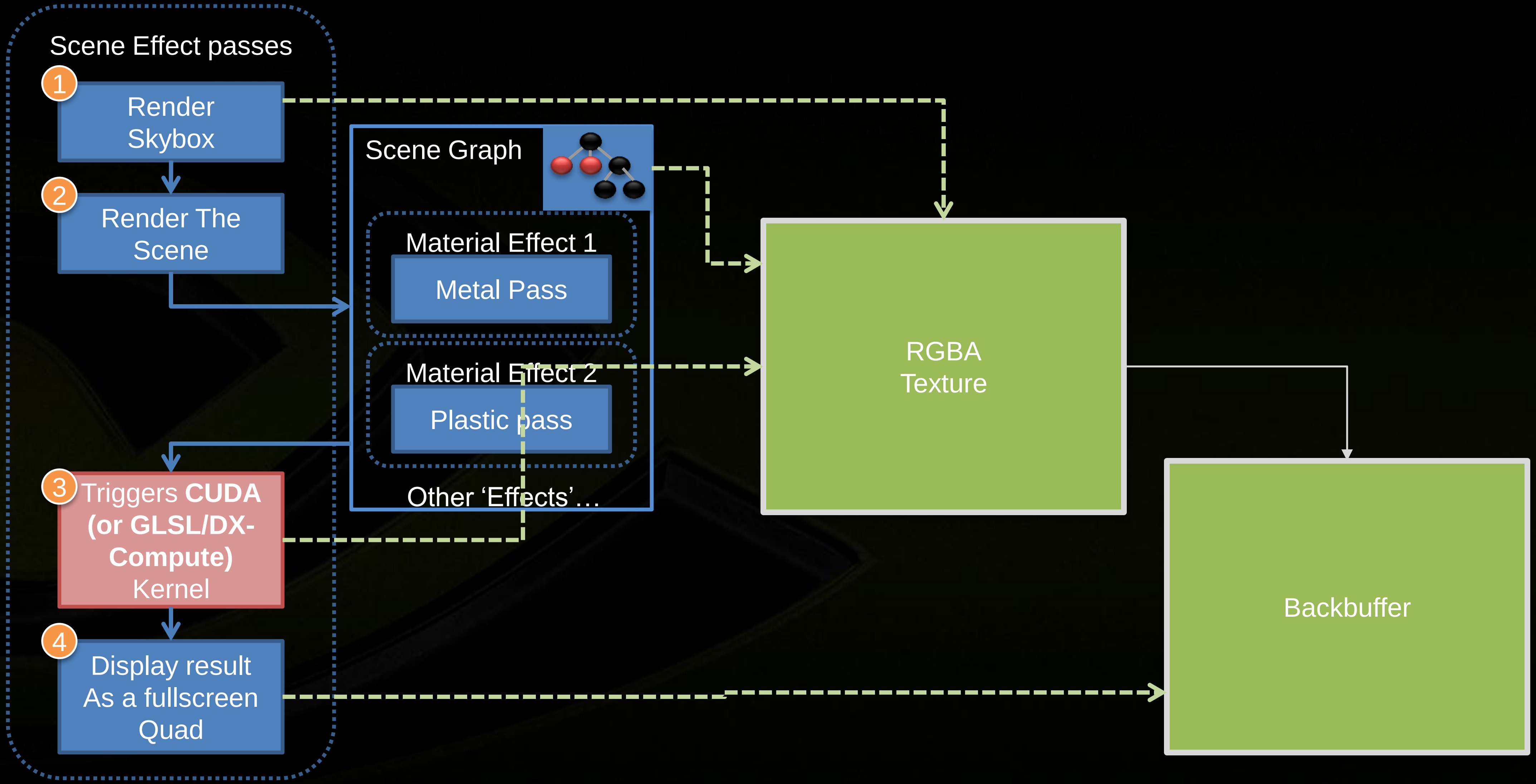
- Front-End : parser (*Bison*)
  - Parses the effect
  - Does not parse the shader/compute code that is inside !
- Back-End : the library to build the effect data
  - Used by the Front-End to create parsed data
  - Used by the application to drive the effects
- Works on PC, Unix (OSX/Linux), Android… even iOS



Example : HDR Rendering With Glow

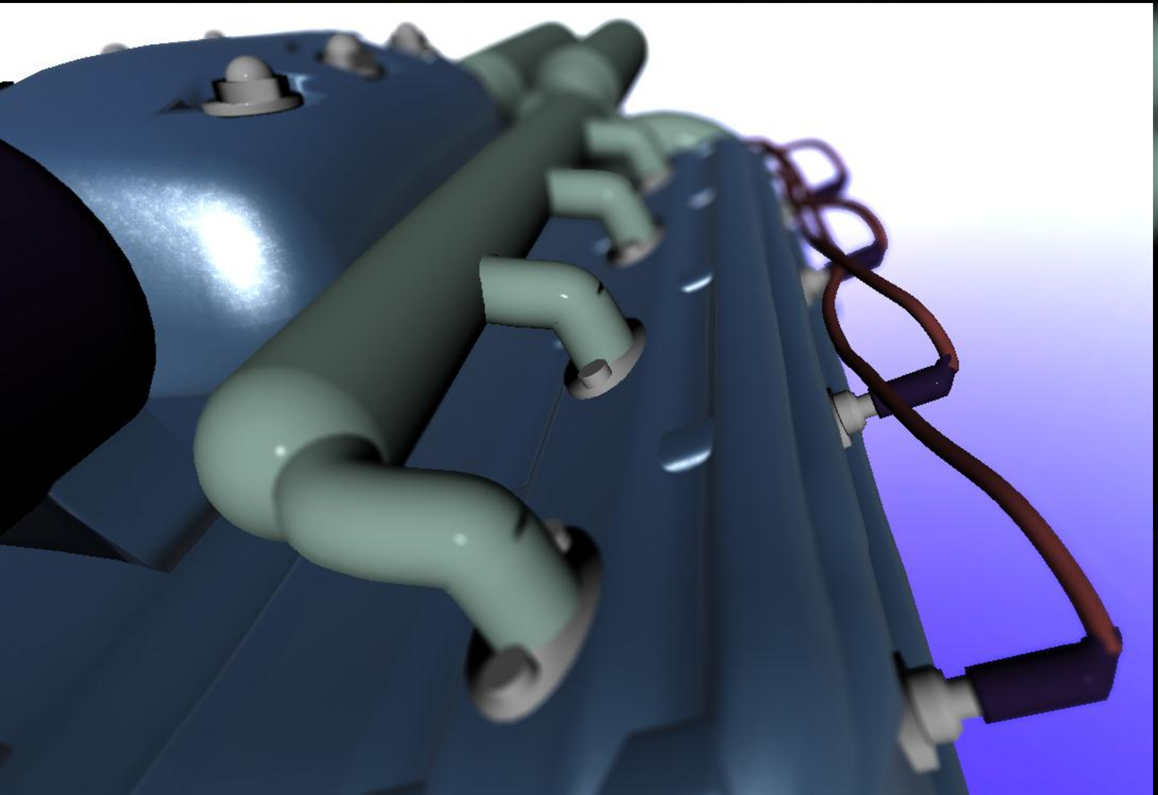


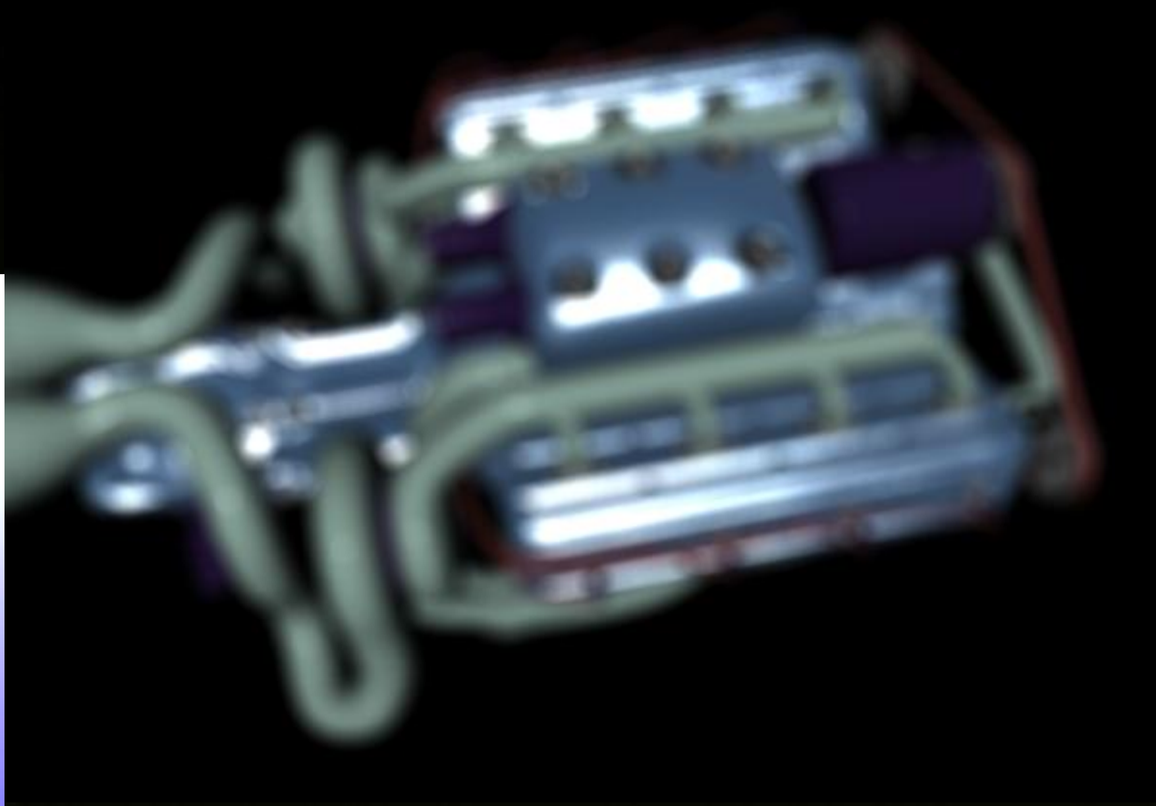
Example : Compute Post-Processing



Results with CUDA / GLSLCompute filtering

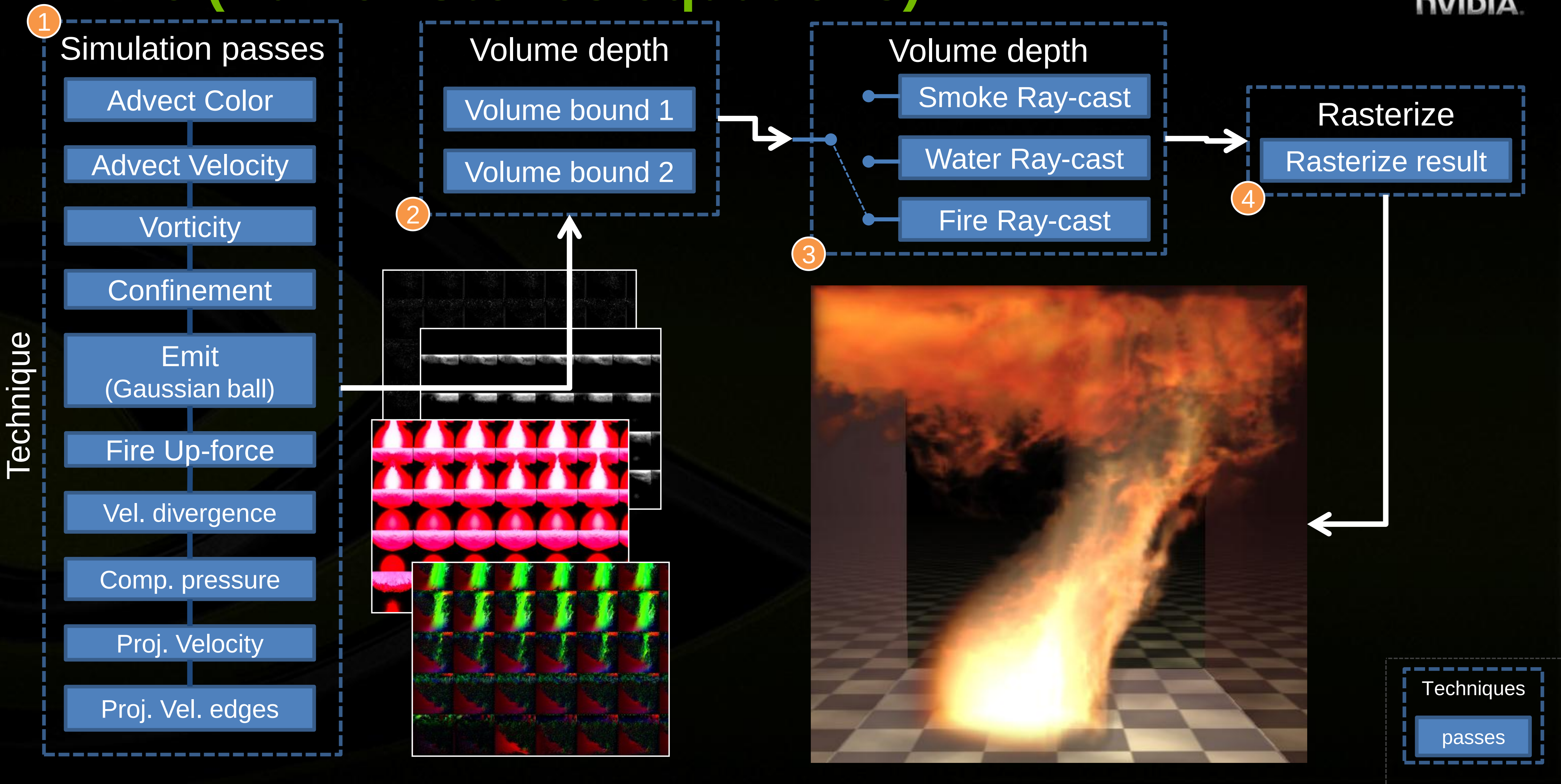
Bokeh Filter



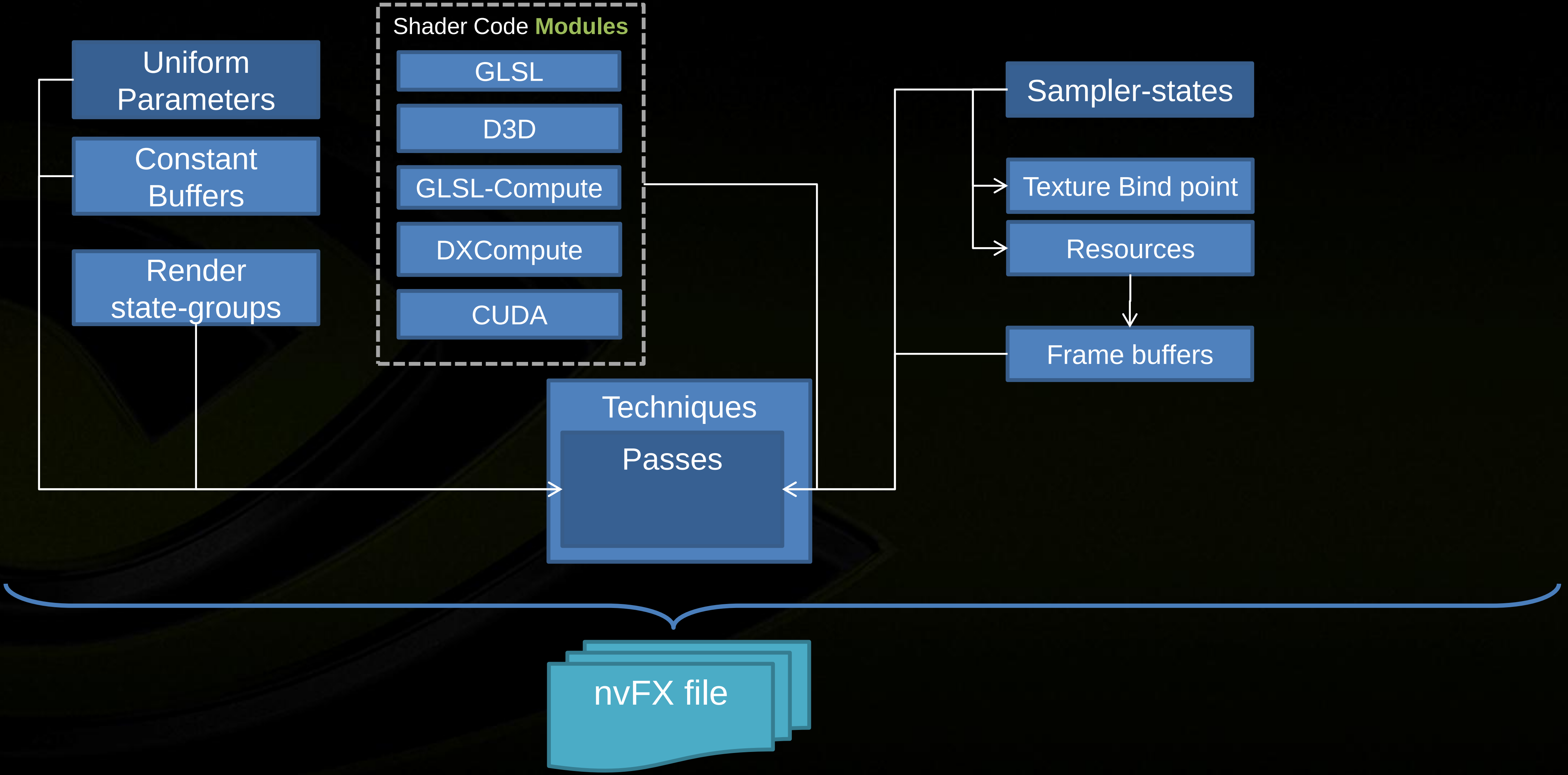


Convolution

Fire (Navier-Stokes equations)



Inside And nvFX Effect



Simple nvFX Example

```
GLSLShader {
 #version 410 compatibility
 #extension GL_ARB_separate_shader_objects : enable
… }
GLSLShader ObjectVS {
  layout(location=0) in vec4 Position;
  layout(location=0) out vec3 v2fWorldNormal;
  void main() { … }
}
GLSLShader ObjectPS {
  layout(location=0) in vec3 v2fWorldNormal;
  Main() { … }
}
rasterization_state myRStates {
    POLYGON_MODE = FILL;
… }
```

```
sampler_state defaultSamplerState
{
  TEXTURE_MIN_FILTER = LINEAR_MIPMAP_LINEAR;
  TEXTURE_MAG_FILTER = LINEAR;
}
Texture2D diffTex {
    samplerState = defaultSamplerState;
    defaultFile = "gargoyleMossyDiffuse.dds";
}
technique BasicTechnique {
    pass p1 {
       RasterizationState = myRStates;
       samplerResource[diffSampler] = { diffTex, 0 };
       VertexProgram = ObjectVS;
       FragmentProgram = ObjectPS;
       attenuation = 0.9;
}
```

nvFX On C++ Side : Simple Example

Initialization:

- Validate effect's passes (Checks errors, compile shaders…)
- Create/Gather any object we need for update (Uniforms to set etc.)

Rendering Loop:

- Loop in a Technique (taken from a material id, for example)
- Set some Uniform values (projection matrix…)
- Loop in the Passes
- For each pass : 'Execute' it
  - Optionally update Uniforms/Cst Buffers afterward
- Render your geometry

Shader Code And Effect Compiler

- GLSL, D3D, CUDA, GLSL-Compute, DX-Compute… Not Parsed
- We rely on existing compilers
  - D3D Driver
  - GLSL OpenGL driver
  - CUDA compiler
  - OpenCL from OpenGL driver
- nvFX → invokes APIs to compile shaders
  - Easy
  - No redundant work
  - But nvFX doesn’t know what is inside (did not parse the code)

Shader Code

- **Declared within a section :**

```
GLSLShader myShader {
        layout(location=0) in vec4 Position;
        void main(void) {…}
}
CUDAKernel Blur(unsigned int* data, int imgw,…) {
        …CUDA code…
}
D3D10Shader myD3DShader {
        …HLSL code…
}
```

Sampler States

- We don't add sampler state info to the existing shader code
  - GLSL Does not have Sampler-states
- Instead : create sampler states in nvFX
- Can be connected in a Pass or via Textures or Resources

```
GLSLShader myShader {
      uniform sampler2D diffuseColorSampler;
      …
}
sampler_state mySamplerState {
      MIN_FILTER = GL_LINEAR_MIPMAP_LINEAR;
      MAG_FILTER = GL_NEAREST;
};
```

State Groups

- The modern way to use renderstate : DX10/11 default way
- OpenGL could have one : NV_state_object
  - Rasterization States
  - Color Sample States
  - Depth-Stencil States
- Define many of them in the effect :

```
rasterization_state myRasterState1 { POINT_SIZE=1.2; …}
rasterization_state myRasterState2 { CULL_FACE=FALSE; …}
color_sample_state myCSState1 { BLEND=TRUE; ALPHA_TEST=FALSE;…}
dst_state myDSTState { DEPTH_TEST=TRUE; DEPTH_WRITEMASK=TRUE;…}
```

- State groups can then used in Passes

Techniques & Passes

- A technique hosts passes. Nothing new
- A Pass carries render-pipeline
  - References to State-Groups
  - Or direct References to render-s
  - References to many Shaders (Ve
  - Value assignment to uniform pa
    - GLSL sub-routine
    - ➔ each pass can setup a set of
  - Connection of samplers/textures
  - Connection of images (ARB_shad

- Clear mode (glClear mode…)
- Clear color
- Rendering Mode
- Render Group Id
- Blit action of a resource to a target
- Current Target for rendering
- Viewport Size
- Swap of 2 resources
- Loop count (to repeat passes)
- Active Pass On/Off
- CUDA Module; Shared Mem. Grid/Block…
- GLSL Compute Groups

  - Lots of other special states to drive the runtime behavior

Pass example

```
Pass myPass {
        RasterizationState = myRasterState;
        GL_POLYGON_MODE={GL_FRONT_AND_BACK, GL_FILL};
        VertexShader ={MainVtxProg, HelperFunctions, InputAttribFunc};
        FragmentShader = MainFragmentShader
        FragmentShader[LightShaders]= {LightSpotFunc, LightDirFunc,…};
        mySubroutineArray = {srFunc_spot, srFunc_point, srFunc_dir};
        myOtherSubroutineArray[0] = srFunc32;
        myOtherSubroutineArray[1] = srFunc6;
        mySimpleUniform = {1.3, 2.2, 5.2};
        samplerResource(quadSampler) = myRenderTexture;
        samplerTexUnit(quadSampler) = 0;
        samplerState(quadSampler) = nearestSampler;
        …
}
```

Concatenation of Shaders

- Literally allows you to “link” Shader Objects to a program Object
  - A Pass hosts a program

```
VertexShader = {ShaderMain, ShaderHelpers, ShaderA, ShaderB, …};
```

- We can group shaders by name :

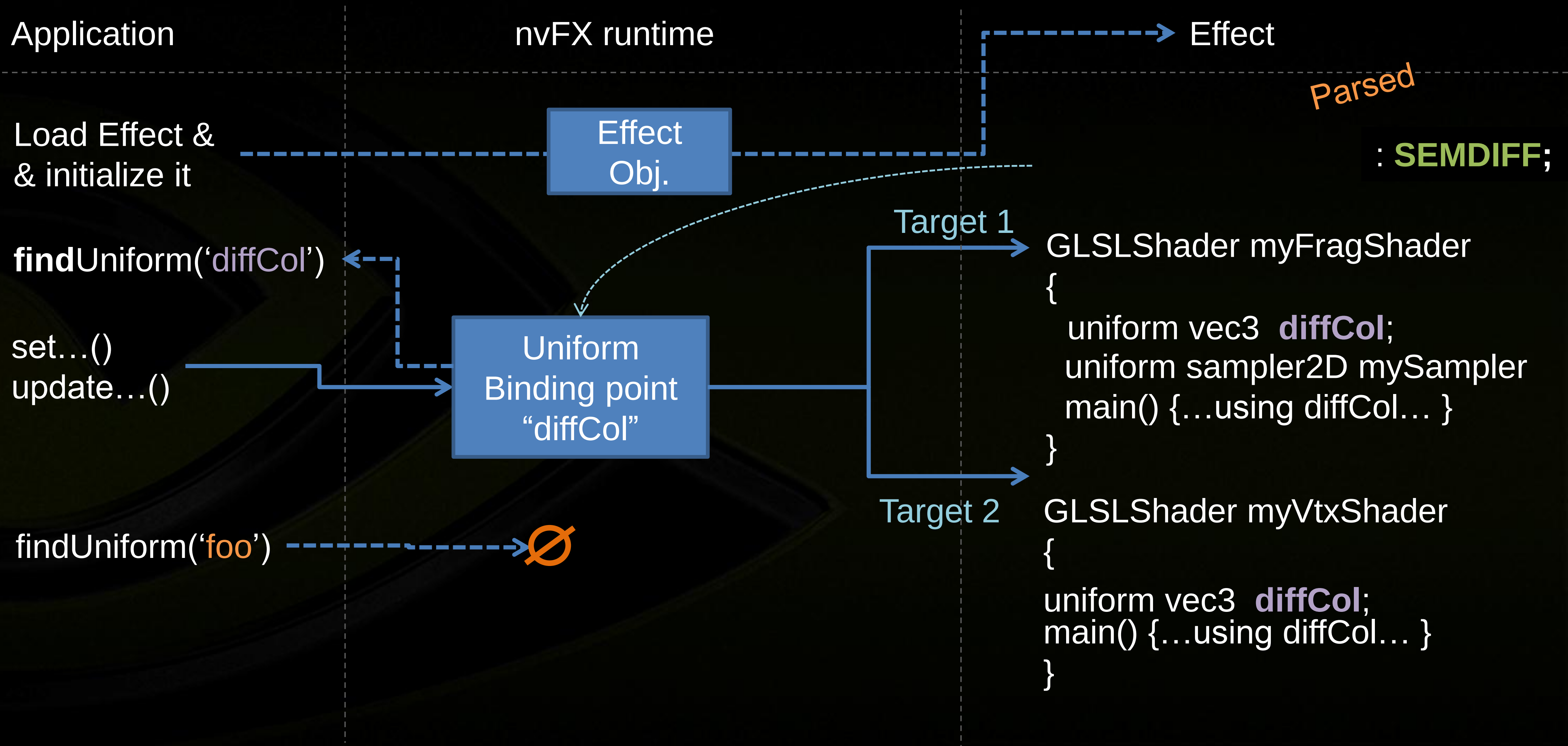
```
VertexShader = myVtxShaderMain;
VertexShader[Lighting] = {VtxLight0, VtxLight1, …}
```

- Groups allows to Change some behavior at *runtime*

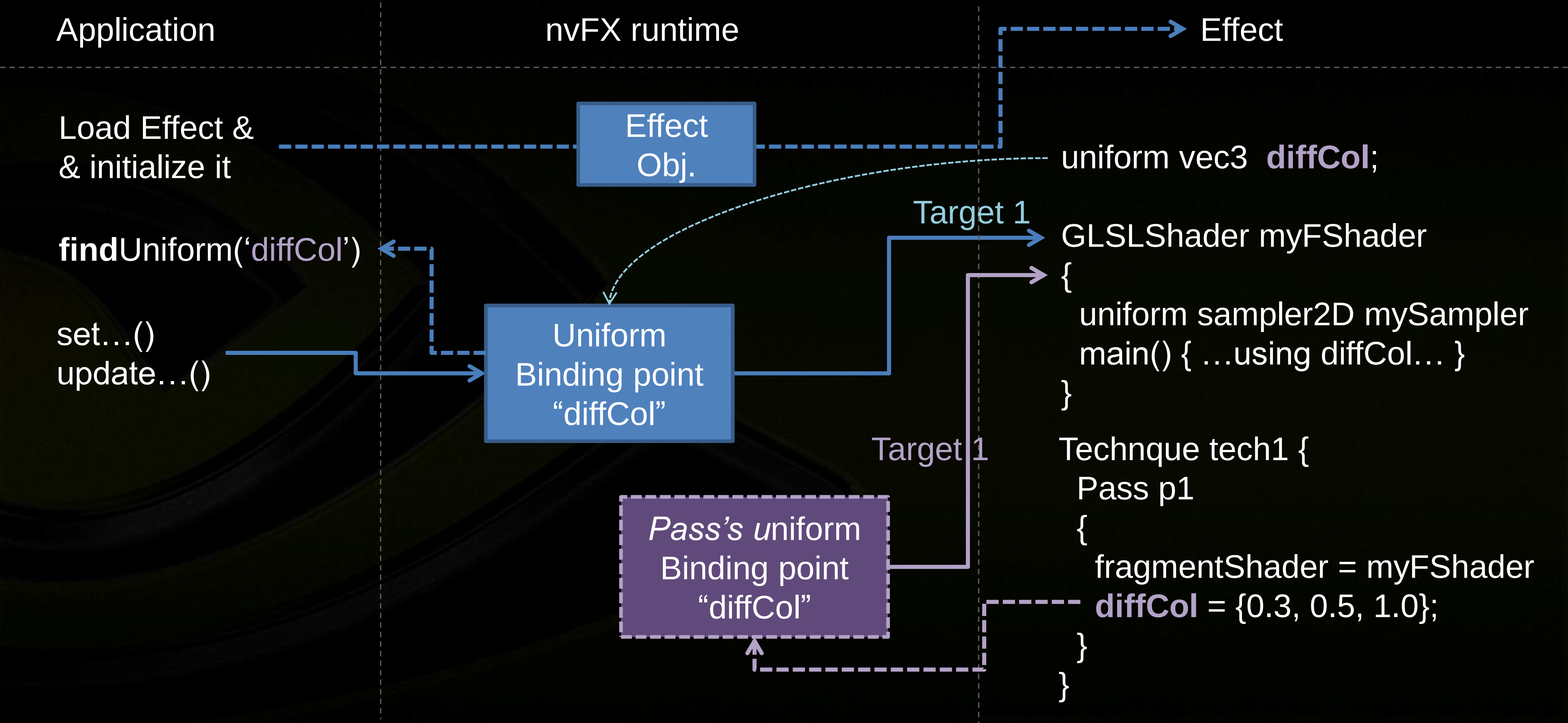
  Example:

  1. Gather the group of shaders named “Lighting”
  2. Remove these shaders from the Pass (Pass’s program)
  3. Add other shaders to this “Lighting” Group (for different lighting…)
  4. Link the program with new Shader Objects

Uniforms


Application
nvFX runtime
Effect
Parsed
Load Effect &
& initialize it
Effect
Obj.
: SEMDIFF;
Target 1
findUniform('diffCol')
GLSLShader myFragShader
{
uniform vec3 diffCol;
uniform sampler2D mySampler
main() {…using diffCol… }
}
set…()
update…()
Uniform
Binding point
"diffCol"
Target 2
GLSLShader myVtxShader
{
uniform vec3 diffCol;
main() {…using diffCol… }
}
findUniform('foo')
∅

Uniforms



Application
nvFX runtime
Effect
Load Effect &
& initialize it
Effect
Obj.
uniform vec3 diffCol;
findUniform('diffCol')
Target 1
GLSLShader myFShader
{
uniform sampler2D mySampler
main() { …using diffCol… }
}
set…()
update…()
Uniform
Binding point
"diffCol"
Target 1
Technque tech1 {
Pass p1
{
fragmentShader = myFShader
diffCol = {0.3, 0.5, 1.0};
}
}
Pass's uniform
Binding point
"diffCol"

Buffers of Uniforms (Buffer Objects)

- Direct mapping to
  - OpenGL Uniform Buffer Object (UBO + GLSL std140)
  - D3D10/11 Cst Buffers (*cbuffer* token in HLSL)
- Similar mechanism as explained for uniforms
- A constant Buffer made of uniforms
  - Can be *targeted* by a Uniform Object
- Can have default values specified by nvFX code
- Two ways for buffer's resource creation :
  - Create from your application and pass the handle to nvFX
  - Let nvFX create the buffer for you (and update it with default values)

Resources in nvFX

- Visual Effects ⇔resources : often inter-dependent
- Example : deferred shading
  - G-Buffer really depends on how the effect does deferred shading
- Furthermore : Compute ⇔ Graphics : interaction through resources
  - Compute reading from a rendered image and writing into a Textures…
  - Compute kernels sometimes need temporary storage…
- ➔ Idea of creation of resources *within* an effect

Resource Creation And Use

- Create resources :

```
RenderTexture myRTex1
{
    MSAA = {0,0};
    Size = ApplicationDefined;// or {800,600};
    Format = RGBA8;
}
RenderTexture myRTex2
{ … }
RenderBuffer myDST
{
    MSAA = {0,0};
    Size = ApplicationDefined;// or {800,600};
    Format = DEPTH24STENCIL8;
}
```

- Create Frame Buffer Object

```
FBO myFBO
{
    Color = { myRTex1, myRTex2 };
    DST = myDST;
}
```

- Use this in Passes

```
    CurrentTarget = myFBO;//(can be backbuffer)
    BlitFBOToActiveTarget = myFBOSrc;
    swapResources( mFBO1, myFBO2 );
    samplerResource(mySampler) = myRTex1;
```

- You can query all from your Application, too

Scene-Level / Multi-Level Effects

NVIDIA

- pre/post-processing are Effects, too : at scene level
- Scene-level Effects and material Effects must be consistent
  - Deferred shading : Material RT's must match the G-Buffer
  - Shadowing of the scene : must tell materials how to use Shadowing
  - Special scene lighting need to tell material Shaders how to do lighting
- nvFX Allows Effect (Scene-level) to override the final linkage of lower levels effects
  - lower level Effect shaders compiled for the needs of the higher one
  - ➔ instances of shader programs matching the scene-level requirements

Example of Scene-level override

```
GLSLShader mainEntry
{
  void main()
  {
    …
    lighting_compute(lightInfos, res);
    …
    finalColor(N, color, tc, p, matID);
  }
}
```

```
GLSLShader simpleOutput
{
  layout(location=0) out vec4 outColor;
  void finalColor(vec3 normal, vec4 colorSrc,
                  vec3 tc, vec3 p, int matID)
  {
      outColor = colorSrc;
  }
}
```

```
GLSLShader forGBuff
{
  layout(location=0) out vec4 outColor;
  layout(location=1) out vec4 outNormal;
  void finalColor(vec3 normal, vec4 colorSrc,
                  vec3 tc, vec3 p, int matID)
  {
    outNormal = …
    outColor …
    …
  }
}
```

```
GLSLShader noLight
{
 void lighting_compute(LIGHTINFOS infos,
   inout LIGHTRES res) {/*empty*/}
}
```

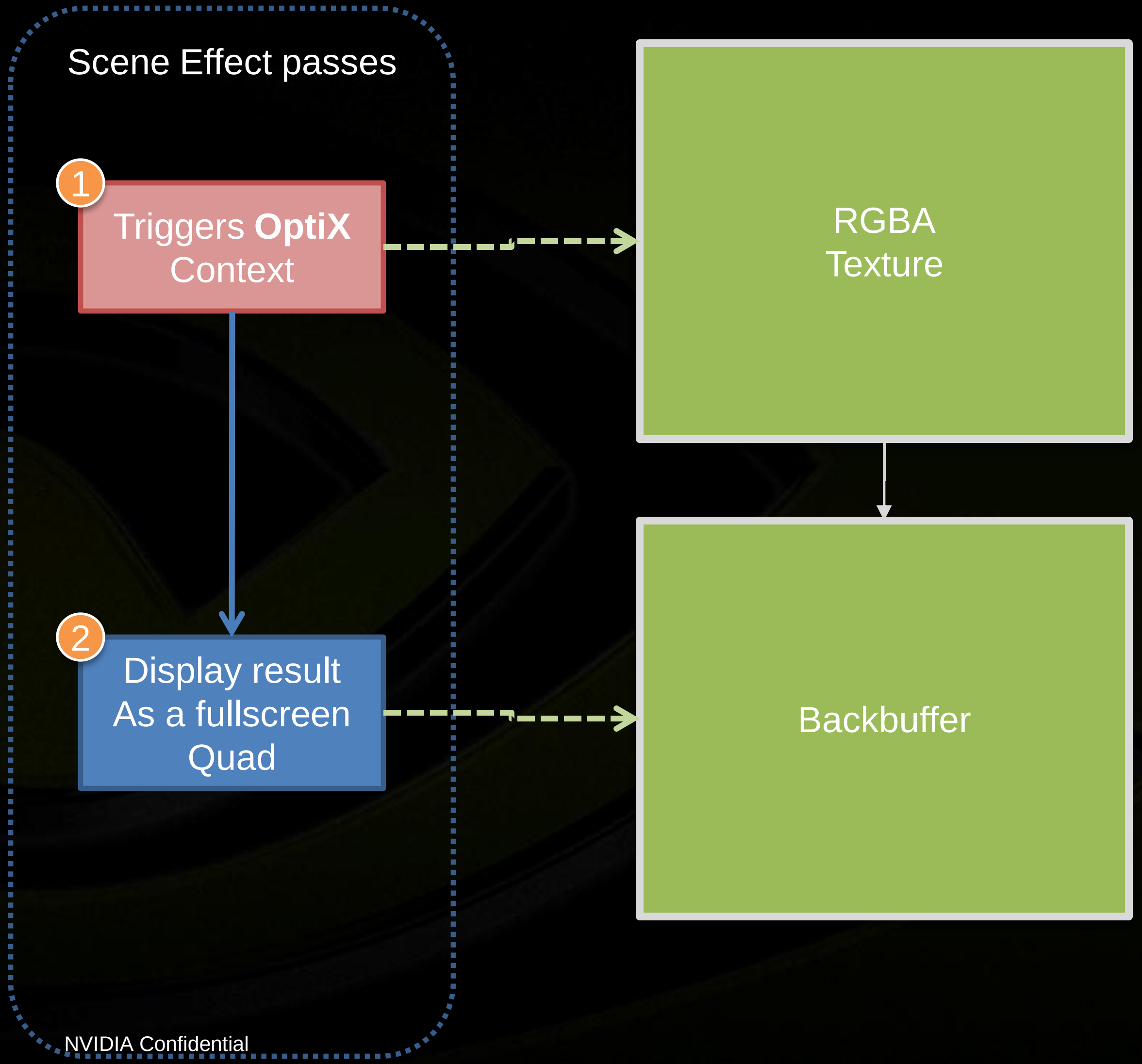
Conclusion

- Less code in Application
- More flexibility
- Consistency of Effect code. Helps for maintenance and creativity
- Updated use of modern APIs good for performance
- Open-Source approach to allow developers to
  - Easily debug it
  - Improve it
  - Customize it

Available soon on http://developer.nvidia.com

Feedback welcome : tlorach@nvidia.com

Example : Pure Ray Tracing With OptiX

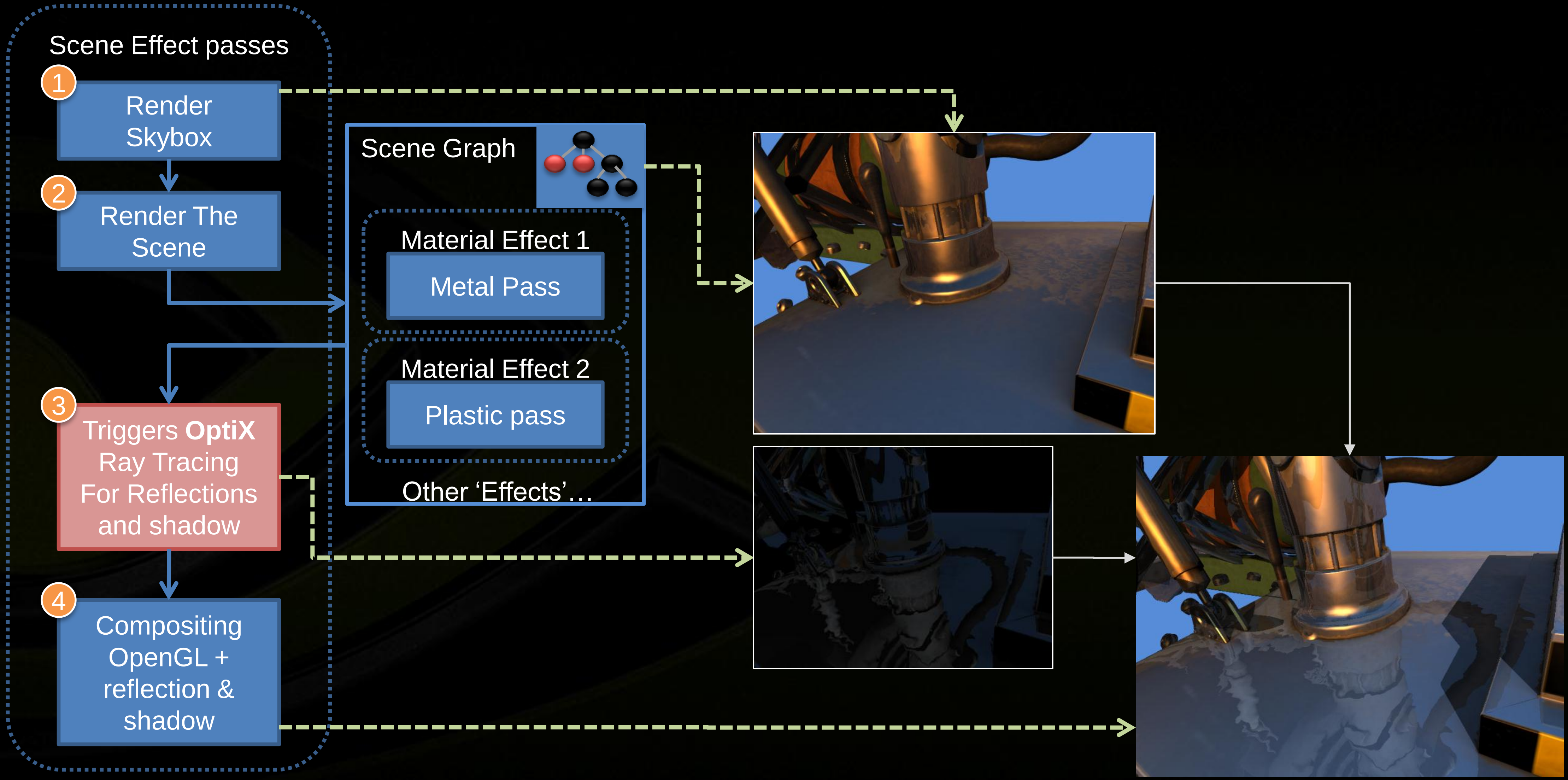


- Rendering at Interactive Framerate
- Generic use of Optix
  - No specialization on specific rendering methods
- 90% of the OptiX code defined outside of the application
  - In CgFX files
  - In CUDA/PTX files

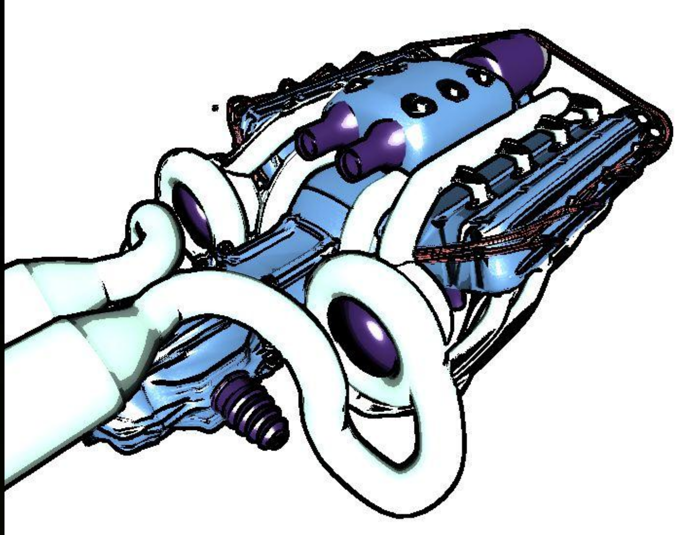
Pure Ray Tracing Examples

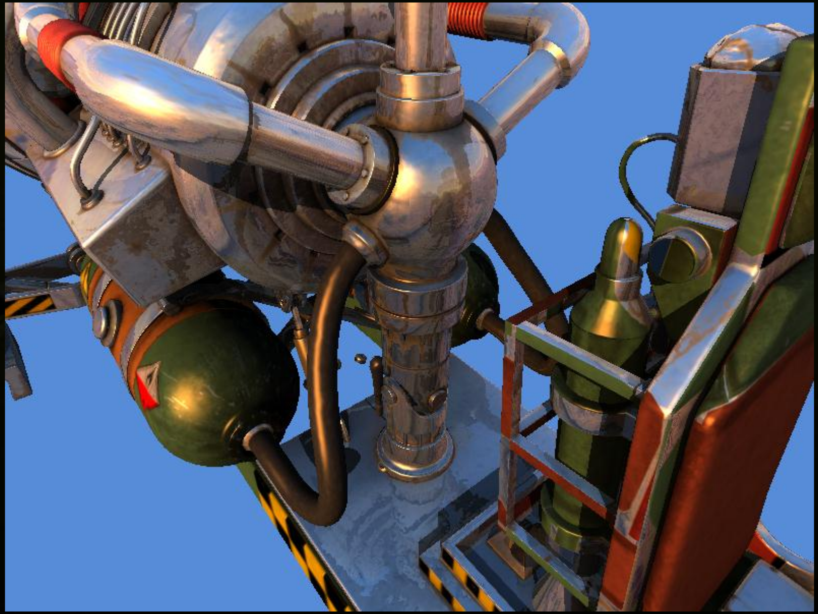
(Courtesy of *Watershot Digital Imaging*)

Hybrid Rendering : Mixing OpenGL & OptiX



More results

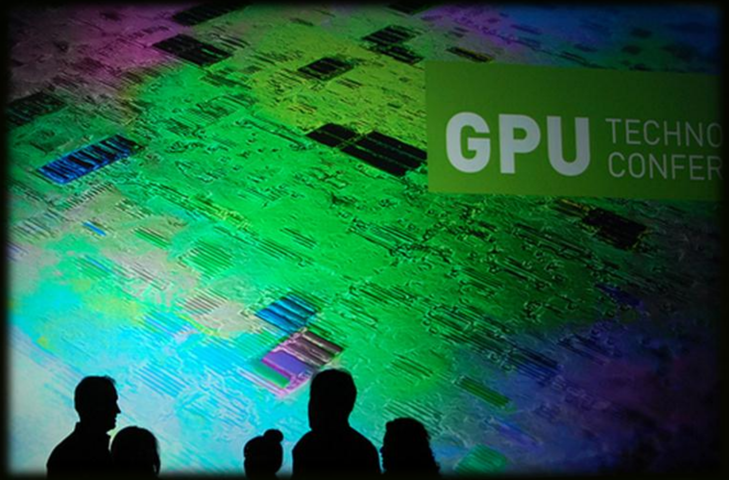




GTC 2013 | March 18-21 | San Jose, CA

The Smartest People. The Best Ideas. The Biggest Opportunities.

Showcase Your Work Among the Elite of GPU Computing!



Call for Submissions
- Call for Sessions: August 2012
- Call for Posters: October 2012

As a speaker you'll receive many benefits that include:

- A complimentary Full Conference pass
- Discounted passes for your colleagues
- Access to exclusive speaker-only on-site amenities
- Audio capture and hosting of your session
- A unique opportunity to promote your name, your expertise, and your company

Learn more at www.gputechconf.com