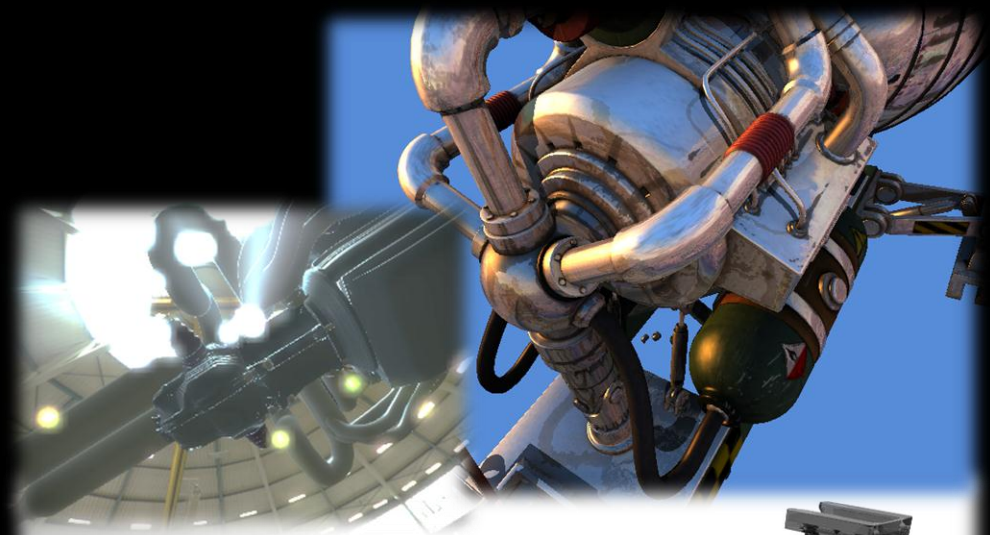
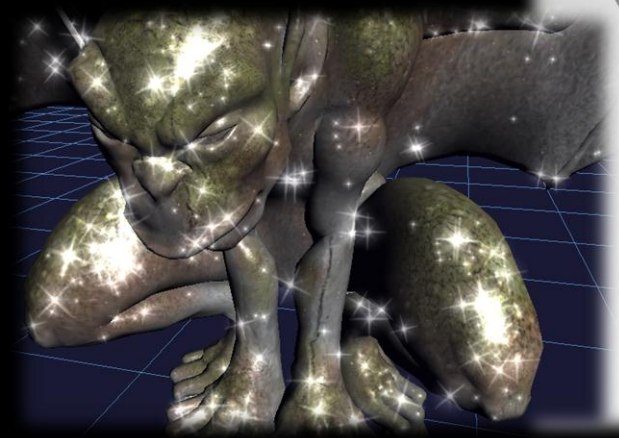
nvFX: A New Shader-Effect Framework for OpenGL, DirectX and Even Compute

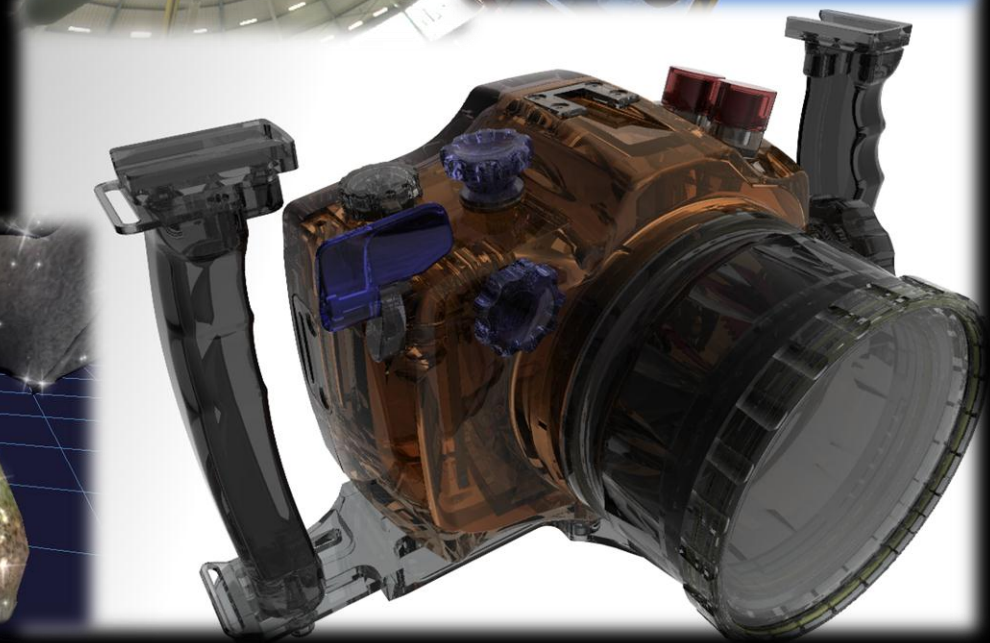
Tristan Lorach (tlorach@nvidia.com)

Plan

- What is an Effect
- How to redefine the “Effect” paradigm
- Examples
- More Details on concepts
- Conclusion

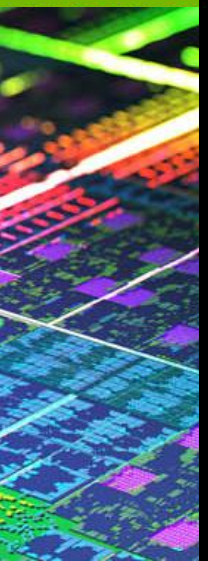






What Is An “Effect” ?

- Higher level management: packages things together
  - Of Shader Code
  - Uniforms / Parameters
  - Samplers, Textures and Sampler States
- Concept of Techniques and Passes
  - Passes : setup Shaders and render states for a rendering pass
  - Techniques : groups Passes
- NOTE : Effect is mostly CPU runtime work
  - Maintains the Effect Database and updates/binds GPU

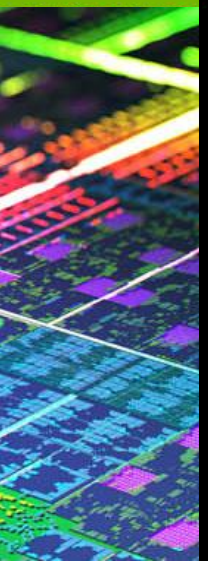


What Is An “Effect” ? (Cont’)

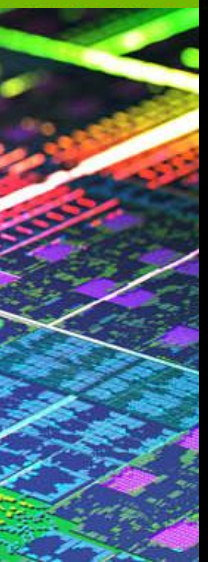
- In 2002
  - Cg and CgFX created by NVIDIA
  - Microsoft followed on the idea, for HLSL : HLSLFx

  Check out NVIDIA SDK Samples & Microsoft SDK Samples
- In 2013 :
  - Game Developers often have their own shader-builders
  - Cg 3.1 in maintenance mode… not expected to evolve

    Why ? Issues of maintenance…
  - Microsoft D3D Effect: No significant evolution



Anatomy Of An Effect



Uniforms / Parameters

```
someFunction(args…)
{
        Shader code
}
```

Samplers / Textures

…

```
someOtherFunction(args…)
{
        Shader code
}
```

Uniforms / Parameters

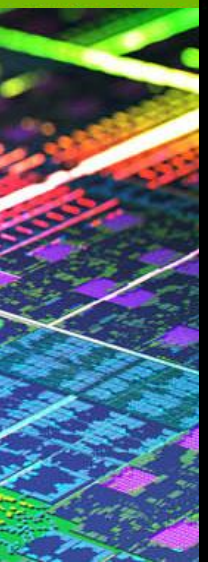
…

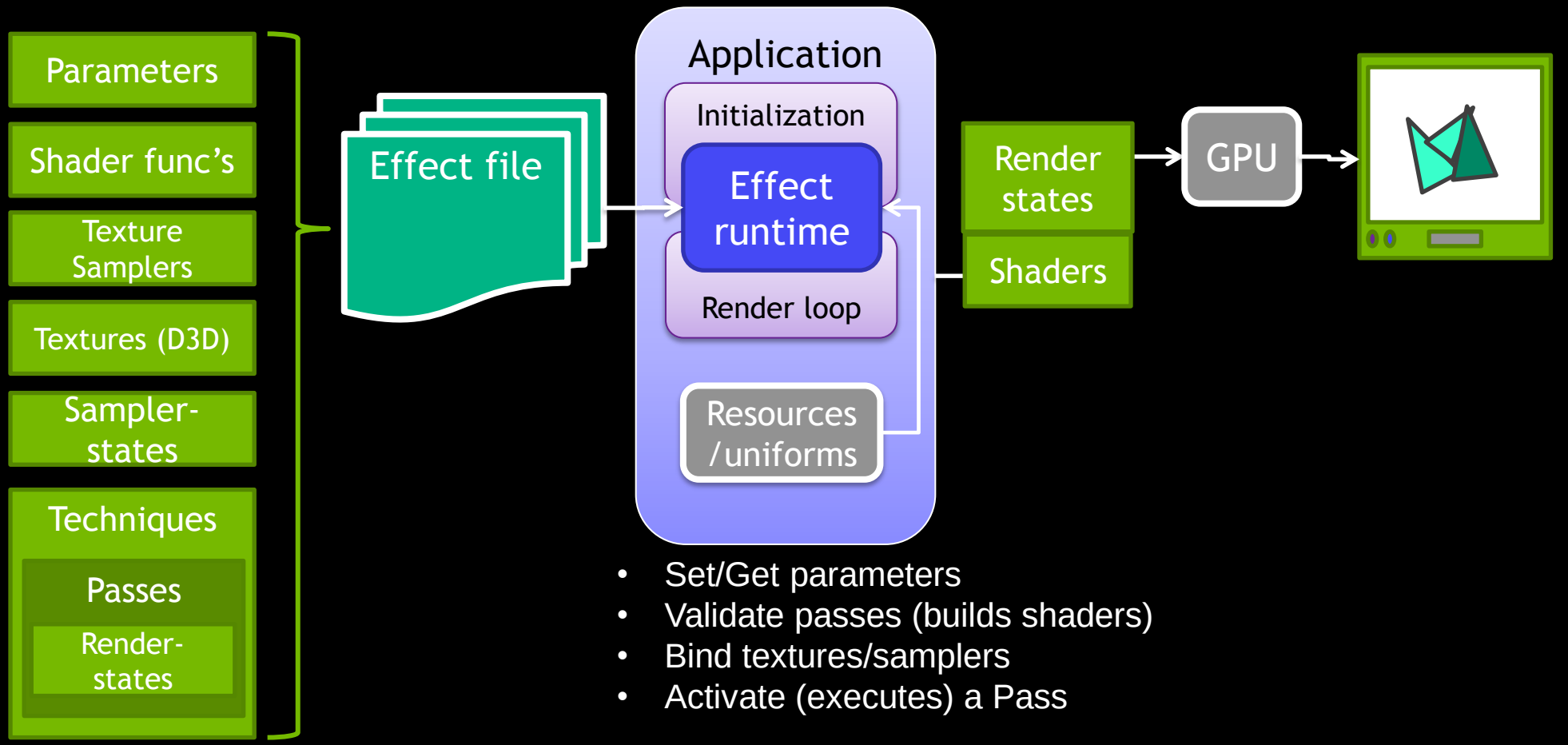
…

Technique *myTechName*
{
  Pass myPassName
  {
    Render / Shader States
  }
  Pass myPassName2
  {…}
}

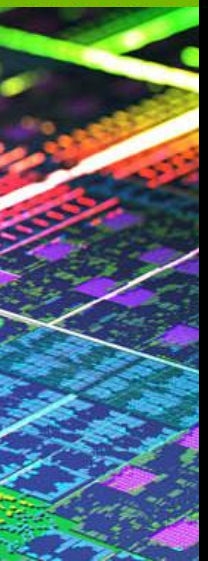
Technique *myTechName2*
{ … }

Standard Effect design



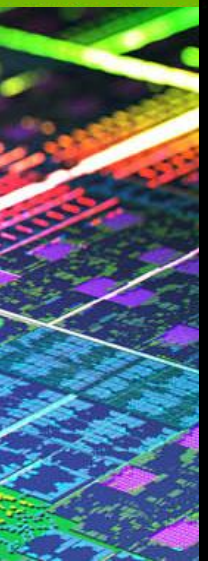


- Set/Get parameters
- Validate passes (builds shaders)
- Bind textures/samplers
- Activate (executes) a Pass



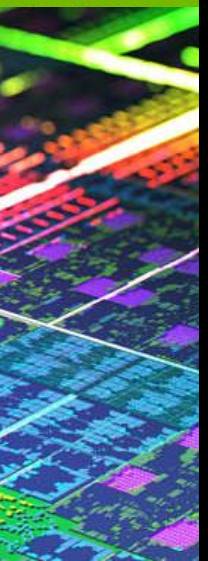
Generic Effect Container : Motivations

- Revive ‘Effect’ concept for today’s needs & HW
- Be a Generic ‘Container’ for GLSL, D3D, Compute, OptiX…
- Add new concepts and features
  (Add resource creation; etc.)
- Use latest 3D API features (OpenGL 4.3 or DX11.x and >)
- Help to simplify applications and C++ code
- Runtime efficiency
- Multi-Platform (Windows, Linux, OSX, iOS, Android)
- Make it Open-Source



What nvFx Can *Not* do

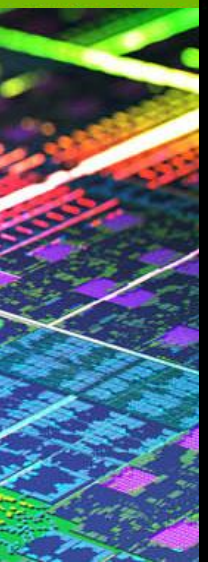
- Few cool CgFX features not in nvFx
  - One unified language (Cg language) for all (D3D, OpenGL…)
  - CgFx Interfaces (~== Abstract classes)
- nvFx can’t modify shading languages grammar
  - nvFx relies on GLSL, D3D, CUDA compilers
- nvFx shader code not unified
  - OpenGL != GL-ES != D3D != CUDA features
- Doesn’t sort and optimize Drawcalls for you
- “Regal” could help on OpenGL <> GL-ES

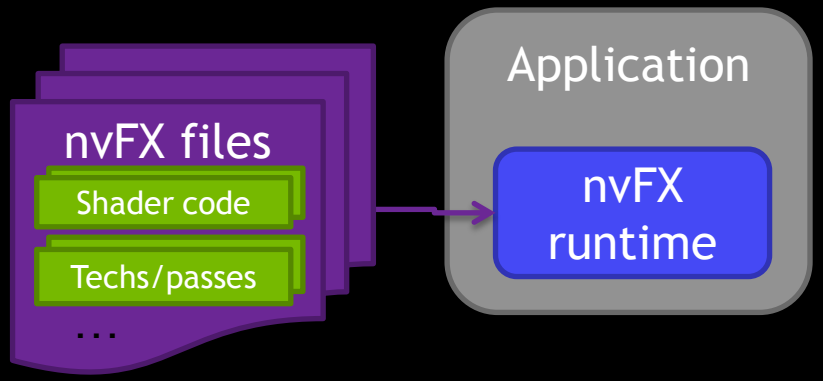


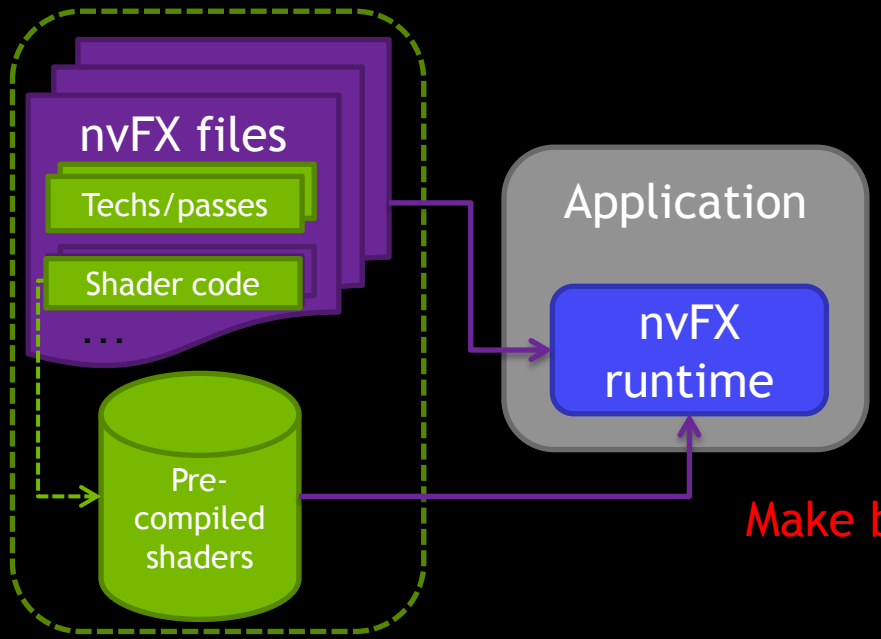
Potential User Target

- 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
- Games
  - Helps highly combinatorial Shaders
  - Avoids heavy pre-processor code (#ifdef/#else/#endif everywhere)

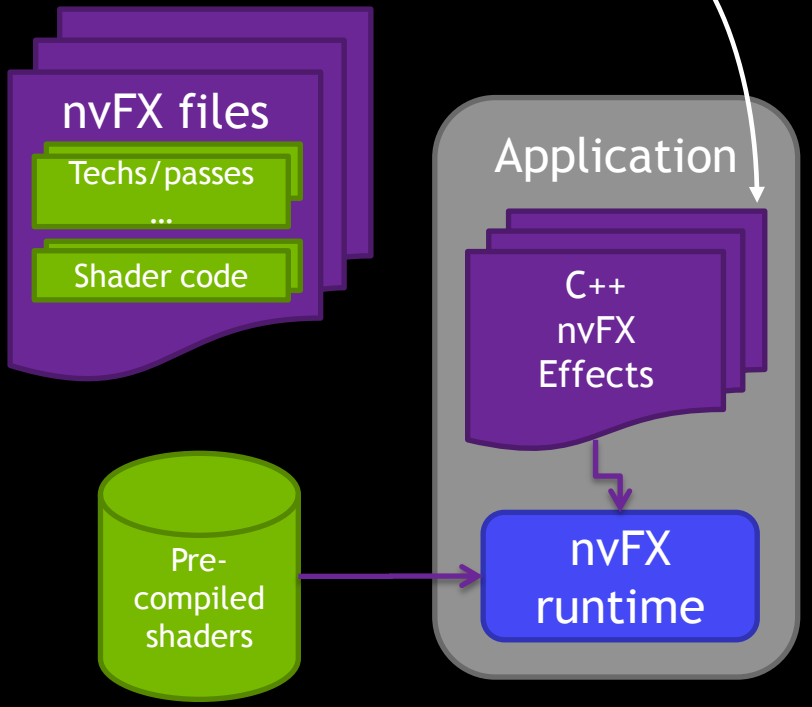
nvFX Effect Integration







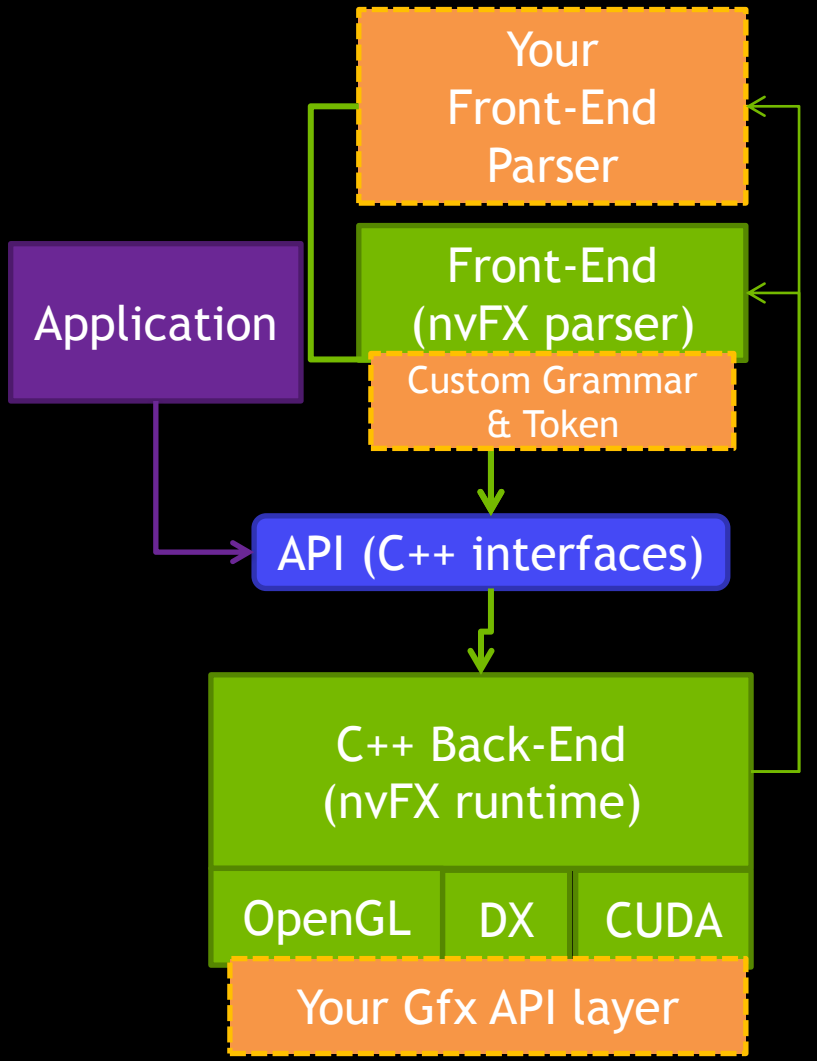
nvFxcc.exe -omyFx.cpp myFx.glslfx



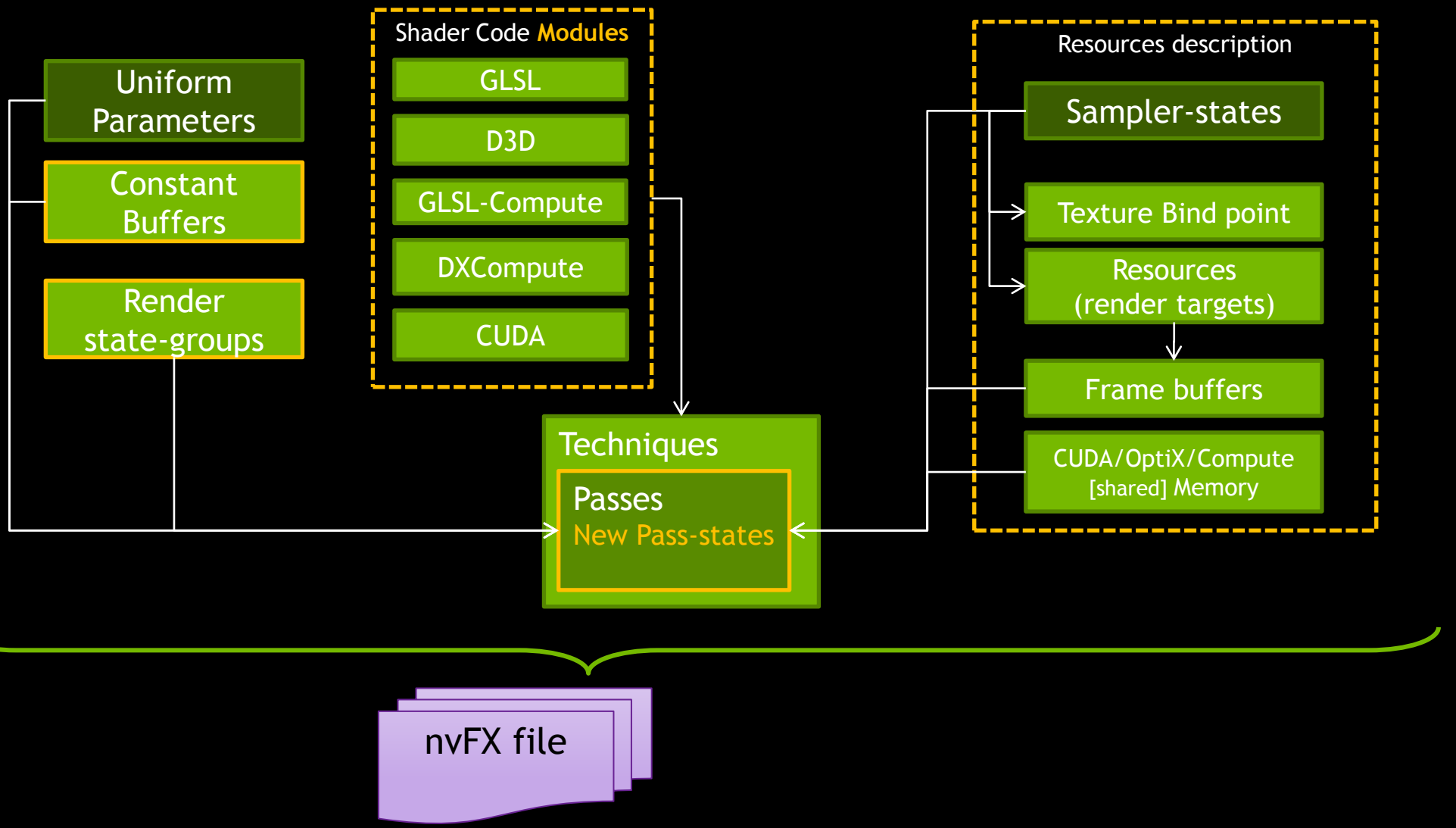
Make better diagram on details (loop, resource)

API Design

- Front-End : parser (*Bison/Flex*)
  - Parses the effect
  - Does not parse the shader/compute code in Modules !
- 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, Android… iOS etc.



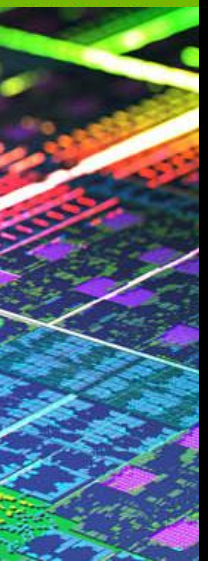
Inside An nvFX Effect

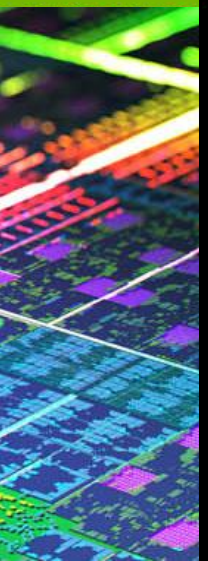


Simple nvFX Example

```
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;
}
Resource2D diffTex {
    samplerState = defaultSamplerState;
    defaultFile = "gargoyleMossyDiffuse.dds";
}
technique BasicTechnique {
    pass p1 {
       rasterization_state = myRStates
       samplerResource(diffSampler) = { diffTex, 0 };
       VertexProgram = ObjectVS;
       FragmentProgram = ObjectPS;
       Uniform(attenuation) = 0.9;
       CurrentTarget = backbuffer;
}
```



Simple nvFX Example : On C++ Side

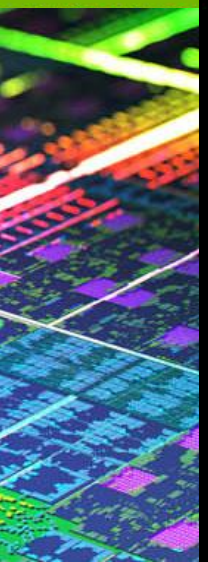
Initialization:

- 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

Example : HDR Rendering With Glow



Scene Effect

1. Render Skybox in HDR
2. Render The Scene in HDR
3. Down-scale the
4. texture 2x, 4x
5. Blur texture Horizontal
6. Then Vertical
7. Compositing: tone-mapping And radial Blur

nvFx-managed Resources

- RGBA32F Texture (HDR)
- RGBA 32F Texture
- +
- Backbuffer

Scene Graph

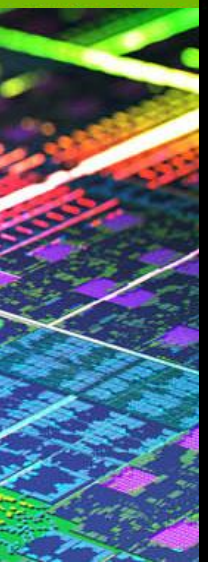
Material Effect 1

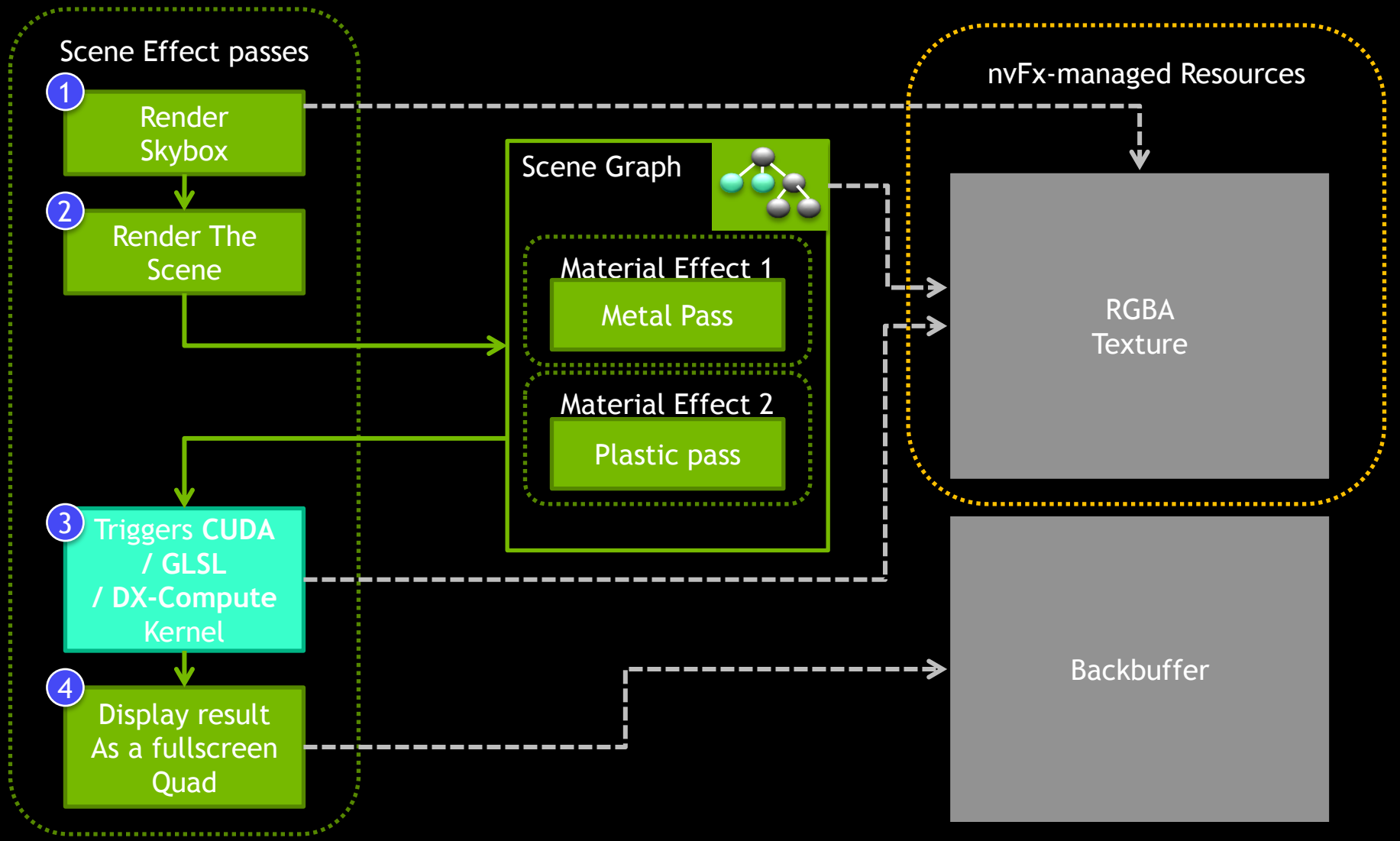
- Metal Pass

Material Effect 2

- Plastic pass

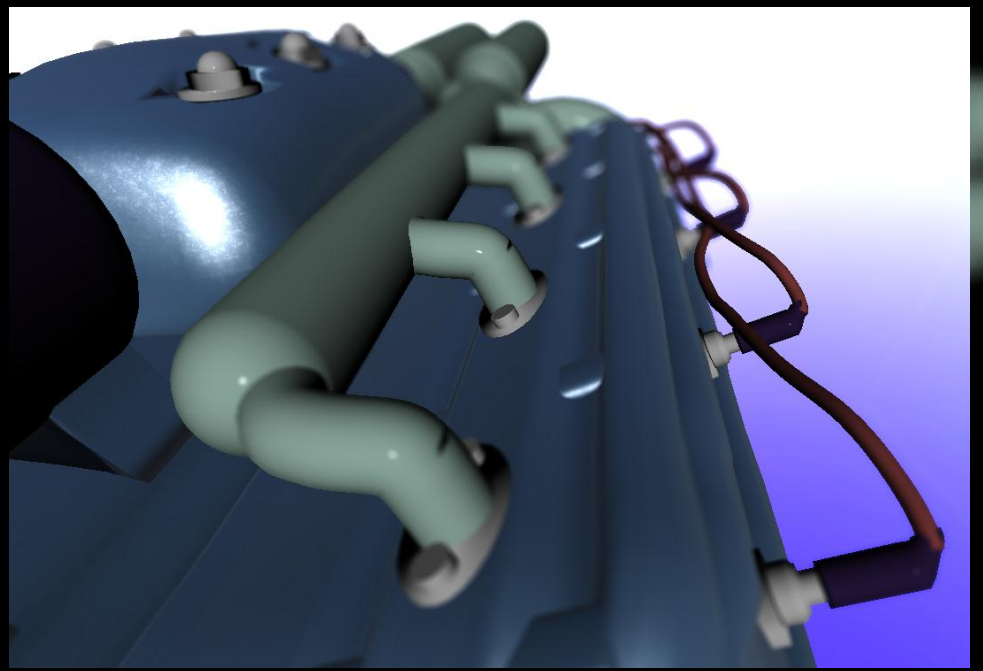
Example : Compute Post-Processing

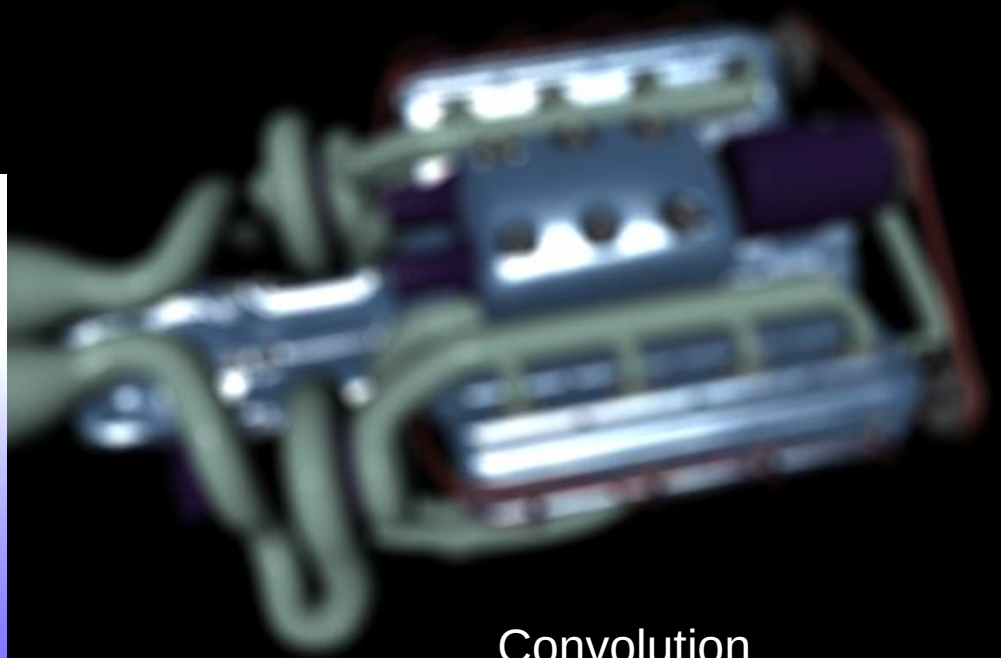




Demo

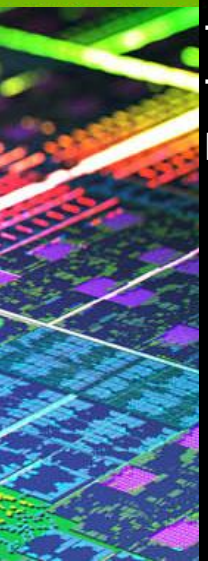
Bokeh Filter





Convolution

Fire (Navier-Stokes equations)



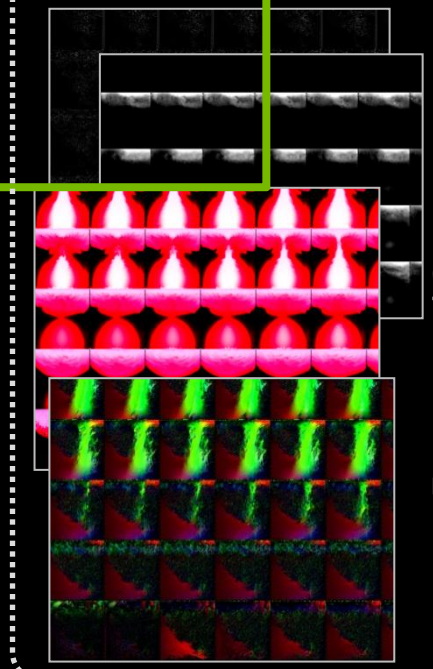
Technique

1 Simulation passes

- Advect Color
- Advect Velocity
- Vorticity
- Confinement
- Emit (Gaussian ball)
- Fire Up-force
- Vel. divergence
- Comp. pressure
- Proj. Velocity
- Proj. Vel. edges

2 Volume depth

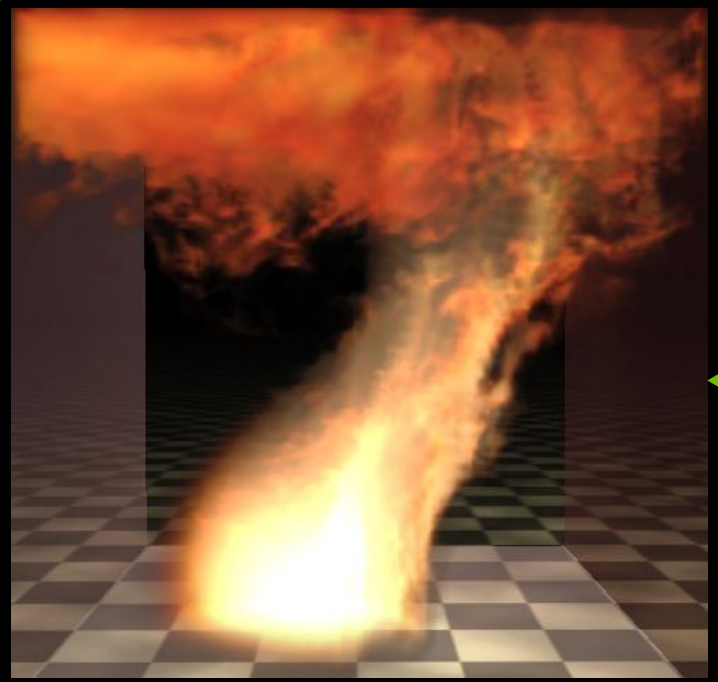
- Volume bound 1
- Volume bound 2



nvFx-managed Resources

3 Volume depth

- Smoke Ray-cast
- Water Ray-cast
- Fire Ray-cast



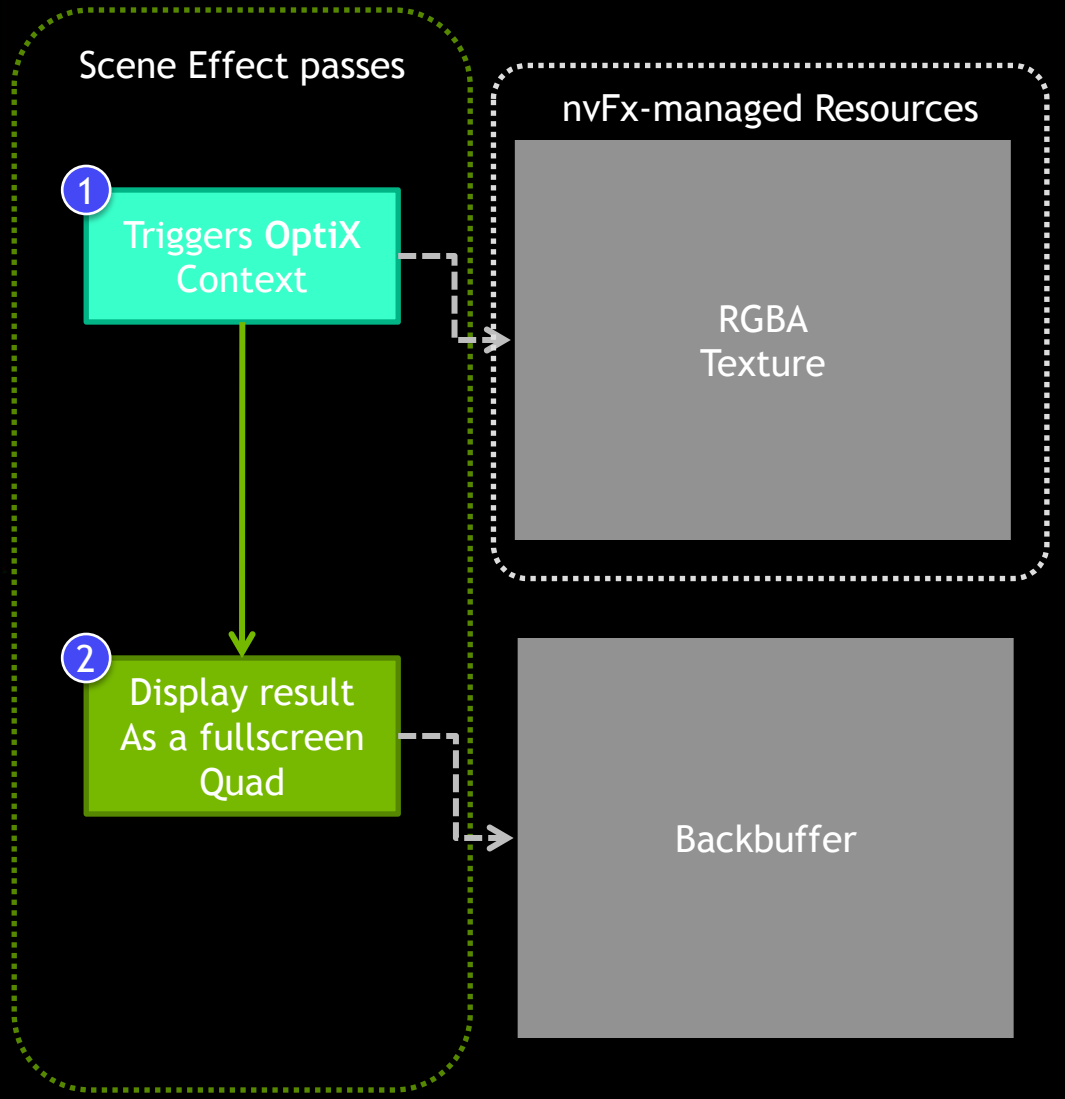
4 Rasterize

- Rasterize result

Techniques

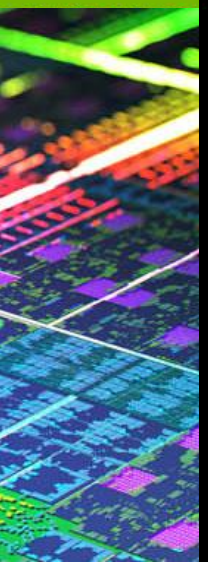
passes

Experiment : Pure Ray Tracing With OptiX



- Most OptiX runtime code
  - nvFx runtime
- Most OptiX Shader code
  - In nvFx files
  - In CUDA/PTX files
- nvFx Needs
  - OptiX NVIDIA SDK
  - Few OptiX setup from the application

Hybrid Rendering : Mixing OpenGL & OptiX



Scene Effect passes

1. Render Skybox
2. Render The Scene
3. Triggers **OptiX** Ray Tracing For Reflections and shadow
4. Compositing OpenGL + reflection & shadow

Scene Graph

Material Effect 1

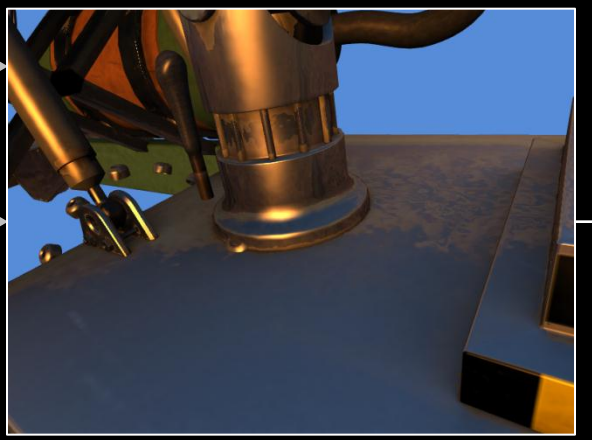
Metal Pass

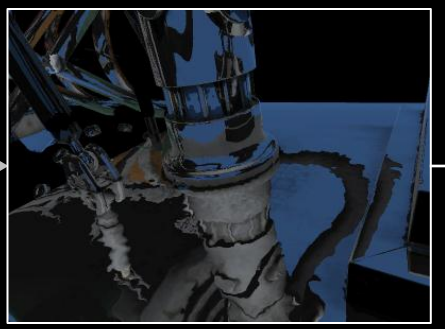
Material Effect 2

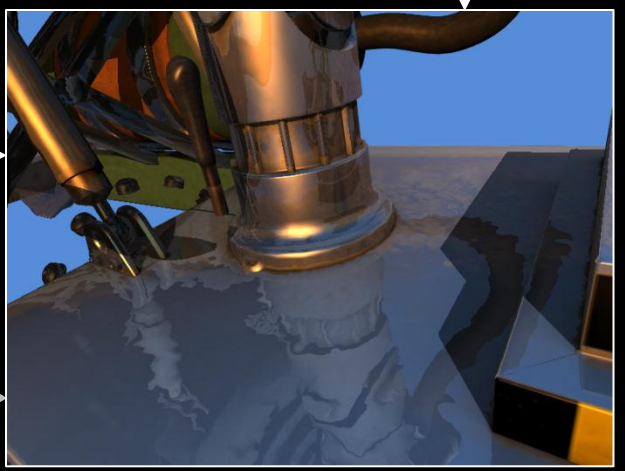
Plastic pass

Other ‘Effects’…

nvFx-managed resources

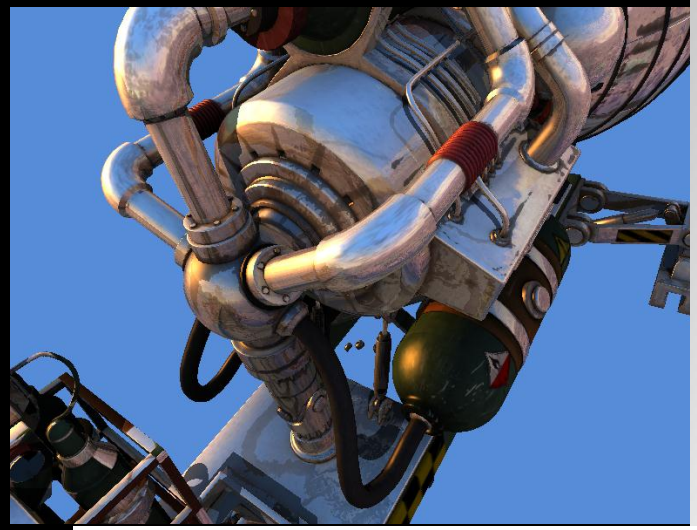


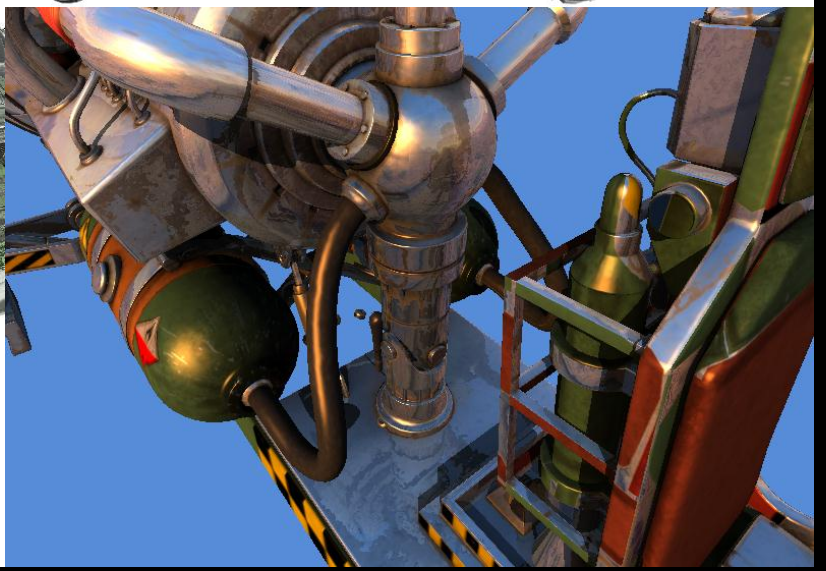




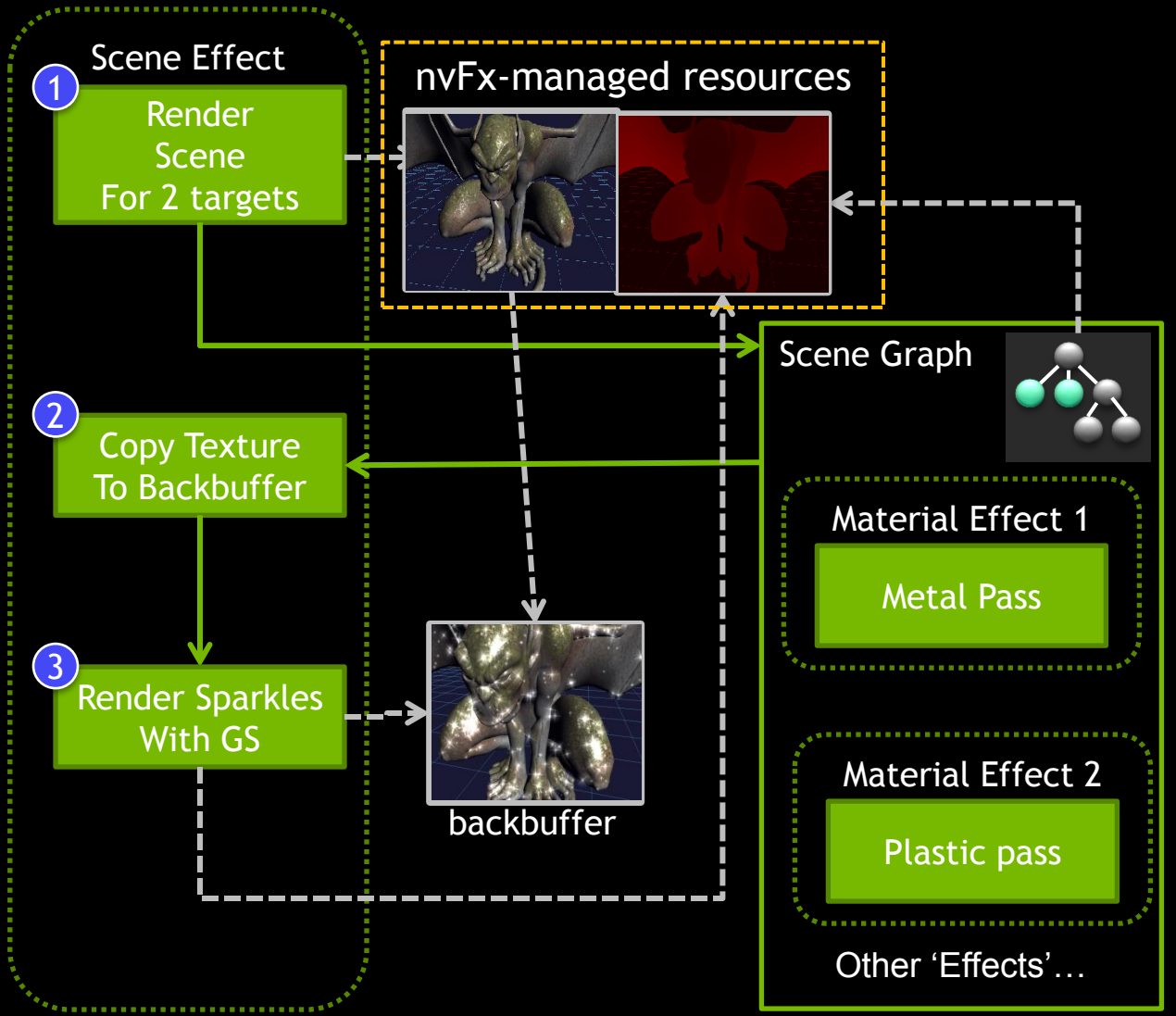
GPU TECHNOLOGY CONFERENCE

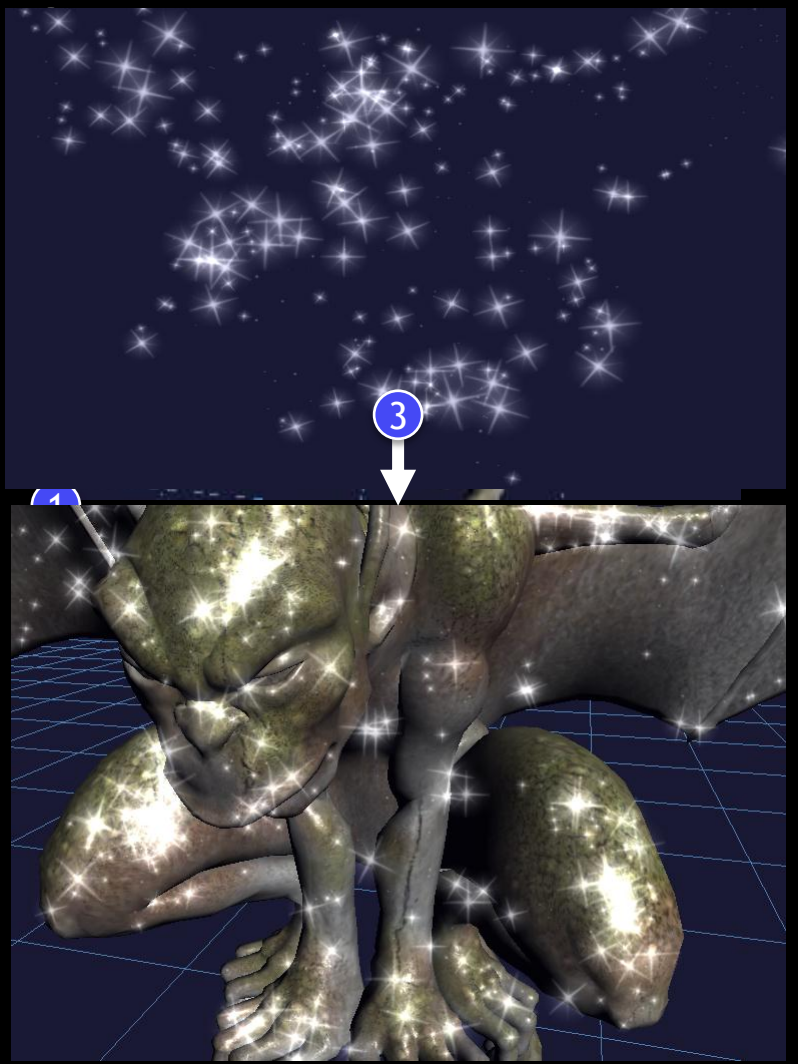
Demo

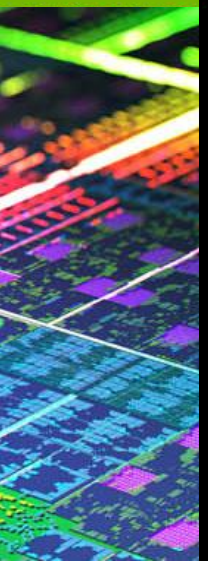




Example: GS Sparkling Effect







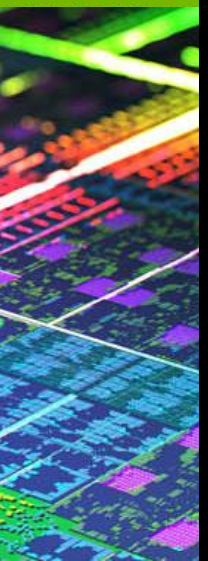
Shader Code In Effect

- GLSL, D3D, CUDA, GLSL-Compute, DX-Compute… Not Parsed
- We rely on existing compilers
  - D3D Driver : for Gfx Shaders and Compute
  - GLSL OpenGL driver : for Gfx Shaders and Compute
  - CUDA compiler
- nvFX invokes APIs to compile shaders
  - No redundant work
  - But nvFX doesn’t know what is inside (did not parse the code)
- nvFxcc.exe : check errors (and will generate C++ 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…
}
```



Shared Code sections

No name == is shared by all (implicit header)

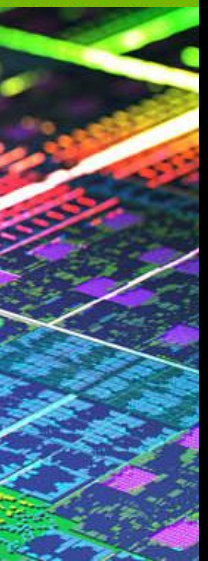
```
GLSLShader {
// used by myVShader and myFShader
#version 430 compatibility
vec3 someVectorMath(vec3 a, vec3 b)
{…}
    …
}
GLSLShader myVShader {
Void Main() { … }
}
GLSLShader myFShader {
Void Main() {…}
}
```

```
CUDACode {
__device__ float clamp(float x, float a, float b);
…
}
CUDACode CUDACode1
{ … }
CUDACode CUDACode2
{ … }
```

Techniques & Passes

- A technique hosts passes. Nothing new
- A Pass carries render-pipeline setup and actions
  - References to State-Groups
  - Or direct References to render-states (old style as CgFX)
  - References to many Shaders (Vertex, Fragment etc.)
  - Value assignment to uniform parameters
    - GLSL sub-routine
    - ➔ each pass can setup a set of default uniform values
  - Connection of samplers/textures with resources & Sampler-states
  - Connection of images (ARB_shader_image_load_store) with resources
  - Lots of other special states to drive the runtime behavior

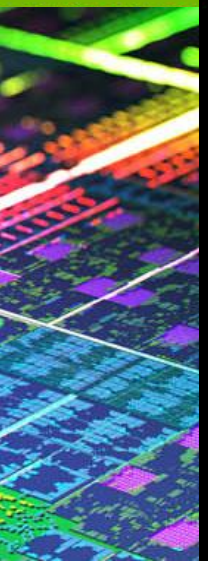
- Clear mode (glClear mode…)
- Clear color
- Rendering Mode
- Render Group Id
- Blit action of a resource to a target
- Current Target for rendering
- Viewport Size
- Swap of 2 resources
- Loop count (to repeat passes)
- Activate Pass On/Off
- CUDA Module; Shared Mem. Grid/Block…
- GLSL Compute Groups



Pass example

```
Pass myPass {
    RasterizationState = myRasterState;
    POLYGON_MODE = {GL_FRONT_AND_BACK, GL_FILL};
    VertexShader = {MainVtxProg, HelperFunctions, InputAttribFunc};
    FragmentShader = MainFragmentShader
    FragmentShader<'LightShaders'>= {LightSpotFunc, LightDirFunc,…};
    Uniform(mySubroutineArray) = {sr_spot, sr_point, sr_dir};
    Uniform(myOtherSubroutineArray[0]) = srFunc32;
    Uniform(myOtherSubroutineArray[1]) = srFunc6;
    Uniform(mySimpleUniform) = {1.3, 2.2, 5.2};
    SamplerResource(quadSampler) = myRenderTexture;
    SamplerTexUnit(quadSampler) = 0;
    SamplerState(quadSampler) = nearestSampler;
    …
}
```

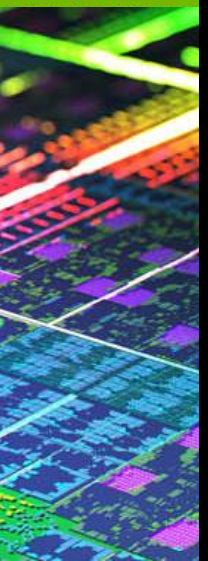
Clarify the “Pass” definition



Linkage of Shader Modules

- Pass : *Link* Shader Modules to a program Object
  - Done at Validation time
  - In Non-Separable Shader mode : 1 Pass hosts 1 program
  - In Separable Shader Mode: 1 Pass hosts many programs
- OpenGL 4.2/4.3 : nvFx uses GLSL linkage
- OpenGL ES: nvFx fakes linkage with concatenation

```
VertexProgram = {VtxMain, ShaderHelpers, ShaderA, ShaderB, …};
FragmentProgram = {FragMain, ShaderHelpers, …};
…
```



Linkage Of Shaders : Groups

- We can group shaders by name :

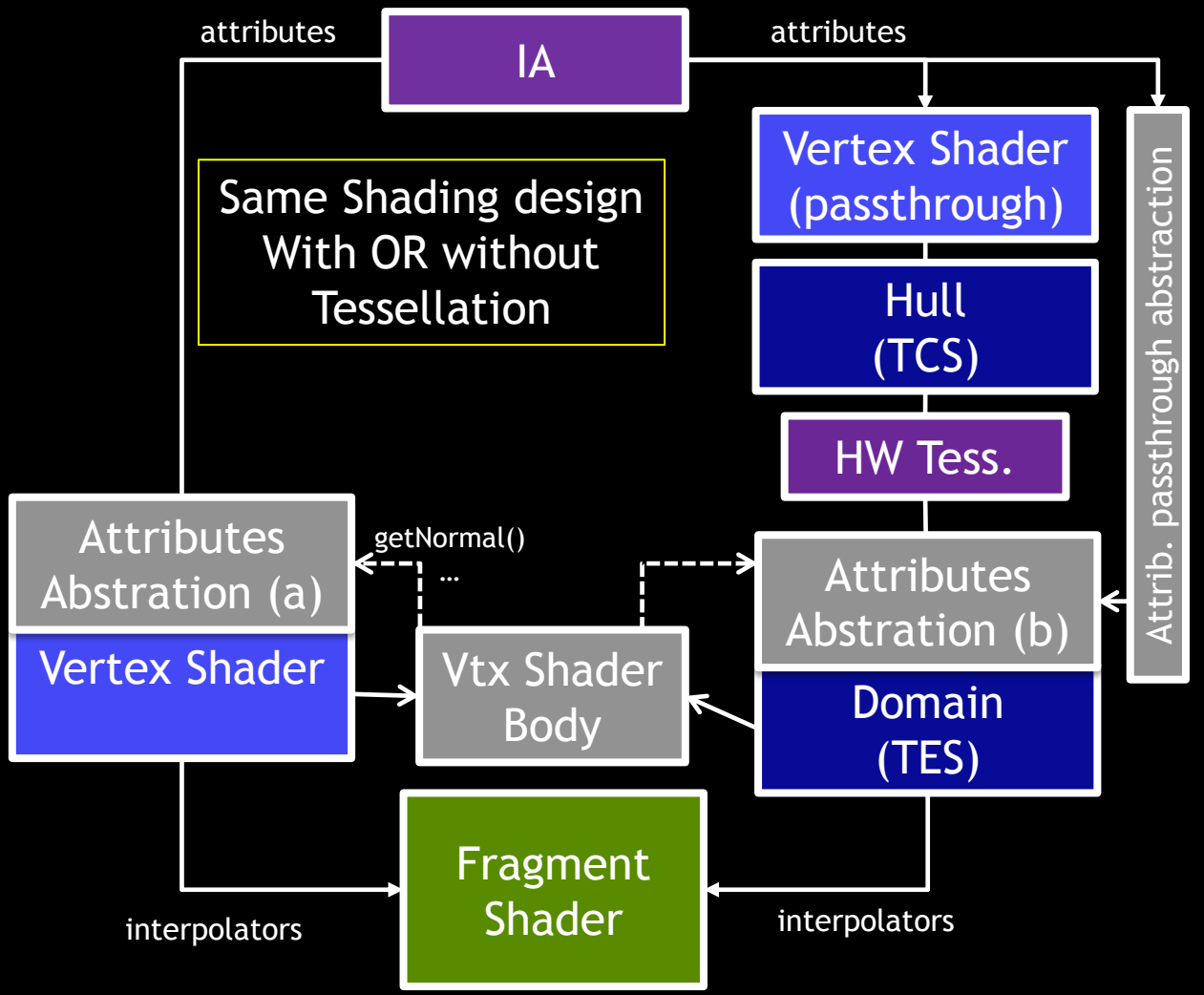
```
VertexShader = myVtxShaderMain;

VertexShader<"Lighting"> = {VtxLight0, VtxLight1, …}
```

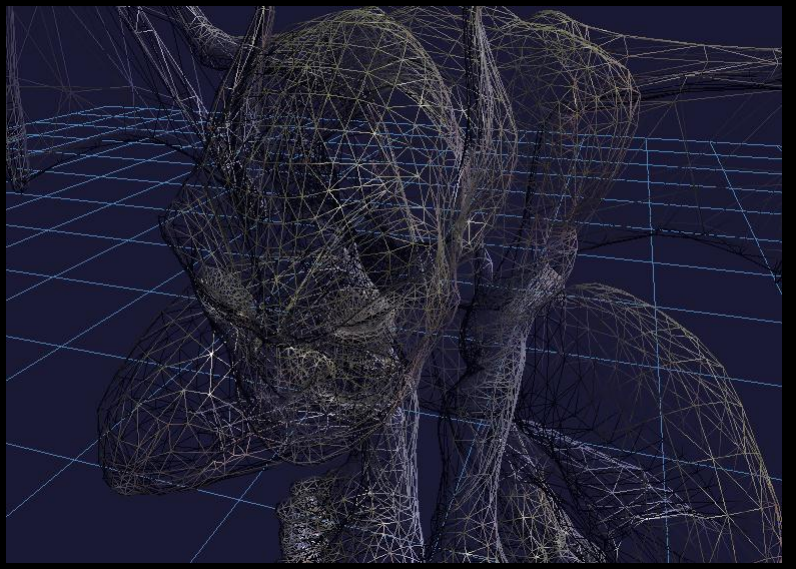
- Groups allows to Change some behavior at *runtime*

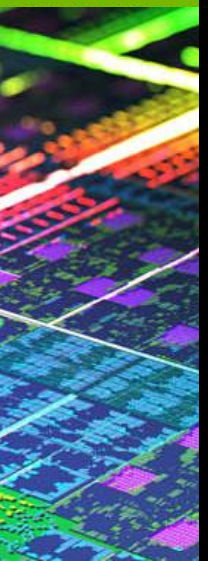
  Example:

  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

Linkage Use-Case Example







Resources in nvFX

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

➔ nvFx allows 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;
}
```

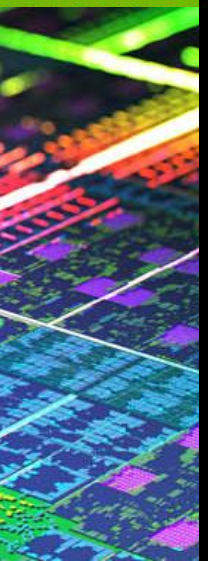
- Create Frame Buffer Object

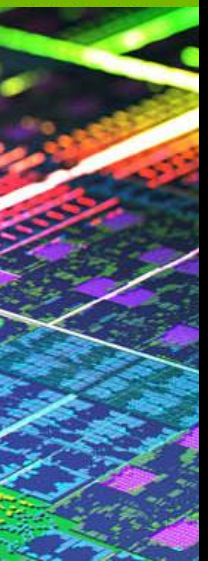
```
FBO myFBO
{
    Color = { myRTex1, myRTex2 };
    DST = myDST;
}
```

- 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

- pre/post-processing are Effects, too : at scene level
- Scene-level Effects and material Effects must be consistent
  - Deferred shading
  - Shadowing of the scene
  - Special scene lighting to tell material Shaders how to do lighting
- nvFX Allows Effect (Scene-level) to override some Shader Modules of lower levels effects
  - lower Effect's shaders code adapted for higher Effects

  ➔ Leads to instances of shader programs matching scene-level passes

Example of Scene-level override

- Scene-level Effect

```
…
Pass renderScene {
  ClearMode = all;
  FragmentProgramOverride<"out“> = forGBuff;
  FragmentProgramOverride<“light“> = noLight;
  CurrentTarget = myGBuffer;
  renderMode = render_scenegraph_shaded;
}
Pass deferredLighting {
    VertexProgram = deferredLightingVS;
    FragmentProgram = deferredLightingPS;
    renderMode = render_fullscreen_quad;
    CurrentTarget = backbuffer;
}
```

- Material Effect in the scene

```
…
Pass myMatPass1 {
  VertexProgram = myVtxProg;
  FragmentProgram = {helpers, mainEntry};
  FragmentProgram<out> = simpleOutput;
  FragmentProgram<light> = defaultLighting;
  …
}
```

- New instance of myMatPass1

```
FragmentProgram = {helpers, mainEntry};
FragmentProgram<out> = forGBuff;
FragmentProgram<light> = noLight;
```

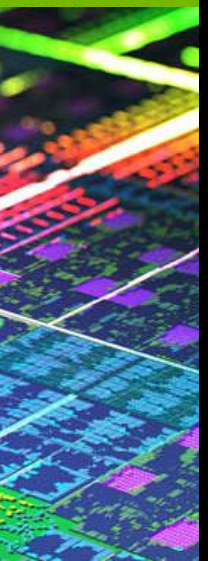
```
GLSLShader mainEntry
{
  void main()
  {
    …
    lighting_compute(lightInfos, res);
    …
    finalColor(N, color, tc, p, matID);
  }
}
```

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

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

…Some OpenGL-style lighting…



State Groups

- The modern way to use renderstate : DX10/11 default way
- OpenGL : maybe… extension
  - Rasterization States
  - Color Sample States
  - Depth-Stencil States

Remove?

- 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

Sampler States

- Sampler state is an nvFx Object
  - Maintained by nvFX and mapped to the API
  - Eventually translated as GLSL Samplers state (extension)
- Can be connected in a Pass or via Textures or Resources

```
GLSLShader myShader {
    uniform sampler2D colorSampler;
    …
}
```

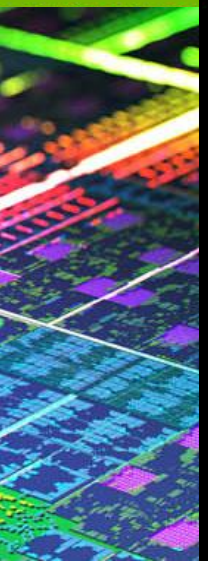
```
sampler_state mySamplerState {
    MIN_FILTER = GL_LINEAR_MIPMAP_LINEAR;
    MAG_FILTER = GL_NEAREST;
};
```

REMOVE?

```
Pass myPass {
    SamplerState(colorSampler) = mySamplerState ;
};
```

Uniforms

Make them 1 SLIDE



Application

nvFX runtime

Effect

Load Effect &
& initialize it

Effect
Obj.

createUniform()

set…()
update…()

Uniform
Binding point
“diffCol”

createUniform()

set…()
update…()

Uniform
Binding point
“foo”

Target 1

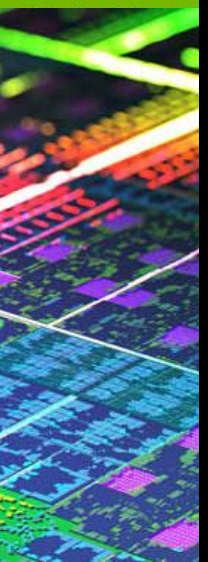
Target 2

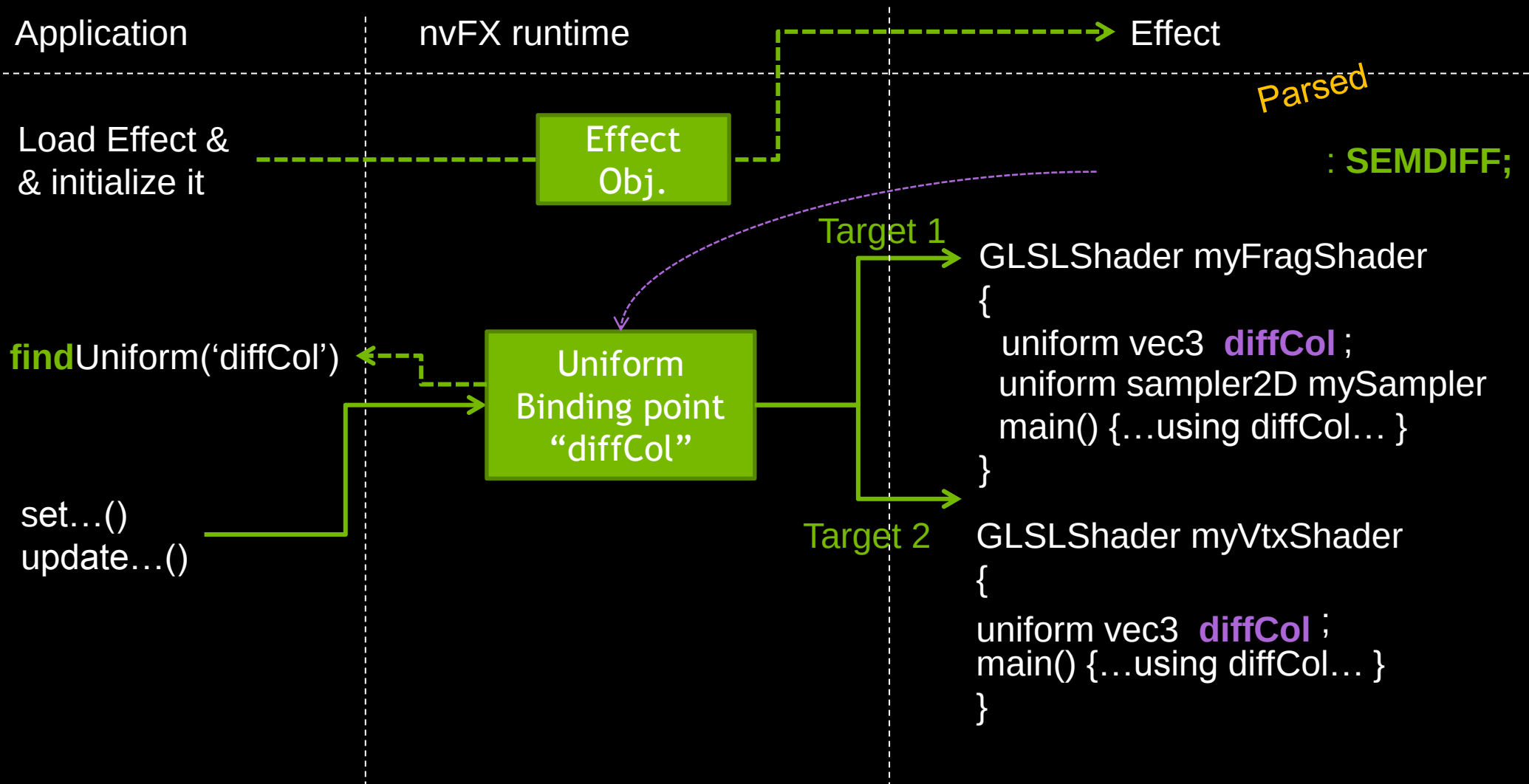
```
GLSLShader myFragShader
{
  uniform vec3  diffCol;
  uniform sampler2D mySampler
  main() {…using diffCol… }
}
```

```
GLSLShader myVtxShader
{
  uniform vec3  diffCol;
  main() {…using diffCol… }
}
```

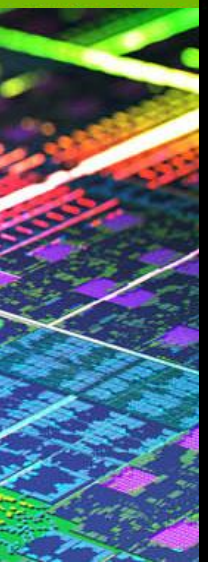
(Other targets)

Uniforms





Uniforms



Application

nvFX runtime

Effect

Load Effect &
& initialize it

Effect
Obj.

uniform vec3 **diffCol**;

Target 1

GLSLShader myFShader
{
  uniform sampler2D mySampler
  main() { …using diffCol… }
}

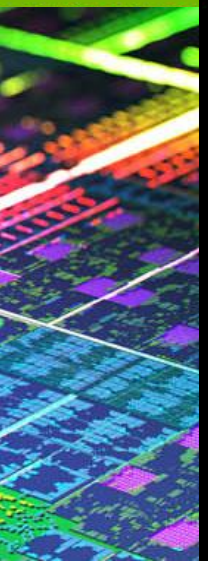
findUniform('diffCol')

Uniform
Binding point
"diffCol"

set…()
update…()

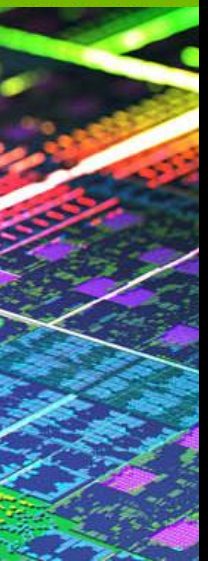
Technque tech1 {
  Pass p1
  {
    fragmentShader = myFShader
    diffCol = {0.3, 0.5, 1.0};
  }
}

*Pass's u*niform
Binding point
"diffCol"



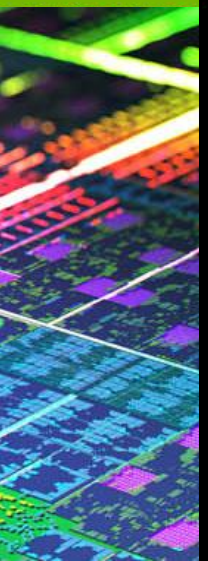
Buffers of Uniforms (Buffer Objects)

- Direct mapping to
  - OpenGL Uniform Buffer Object (UBO + GLSL std140)
  - D3D10/11 Cst Buffers (*cbuffer* token in HLSL)
- A constant Buffer is 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 :
  - application created : pass the handle to nvFX
  - nvFX creates the buffer for you



Performances

- Possible performance issues
  - Runtime implementation
  - Pass execution
  - Update of Uniform / Cst. Buffer / sampler / resource
- More CPU optimization can be done (Open-Source helps)
  - This first version prioritizes nvFx's proof of concept
- Users will always need to be careful
  - Avoid too many pass executions
  - Avoid too many uniform update
- NSight Custom Markers in nvFx



Conclusion

- Less code in Application
- More flexibility
- Consistency of Effect code. Helps for maintenance and creativity
- Updated Effect paradigm for modern API's
- Open-Source approach to allow developers to
  - Easily debug it
  - Improve it
  - Customize it

Questions ?

Feedback welcome : tlorach@nvidia.com

References:

- http://developer.nvidia.com
- https://github.com/p3/regal
- https://github.com/tlorach/nvFX
  (Soon available)