PyFX
A Framework for Real-Time Graphics Effects

Lennart Ohlsson
Lund University

http://graphics.cs.lth.se/pyfx
Outline

- Programmable GPUs
- Effect frameworks
- Real-time effects in Python
- Features and benefits
- General purpose number crunching on the GPU
Shaders

- Surface details as algorithms, not images
- Established technique in animated movies
- Renderman Shading Language
- Cinematic graphics may take time

Leather
float specRolloffEdge = pow((1.0 - VdH), 3.0);
float rollOff = specRolloffEdge + (1.0 - specRolloffEdge) * specularRollOff;
color C = LightColor = Ece * Gain * rollOff;
Programmable GPUs

• Real-time graphics requires GPUs
• Modern GPUs more complex than CPUs
• Specialized and very high performance
• Programmatic control
• Real-time shaders
• Programmed in high-level languages, for example Cg
Integrating shaders into application

- To be done by application on the CPU
  - Compiling and loading shaders
  - Providing shader parameters
  - Setting up rendering pipeline
  - Associating textures to shaders
  - Running multiple rendering passes
  - Using intermediate render targets
- Rather low level APIs
- Creates unnecessary dependencies
Effect frameworks

• “Effects”- units of encapsulation
• Existing effect frameworks
  • CgFX, DirectX Effects
  • Effects specified in declarative mini-languages
• Still insufficient
  • Encapsulation incomplete
  • Lack important features
  • Not extensible
PyFX

- A Python based effect framework
- Effects are specified in Python
- Framework implemented in Python
- Designed to be independent of graphics platform and shader runtime system
- Current implementation on PyOpenGL and Cg runtime (via SWIG)
A simple effect – Refraction

from FX import *

environmentTexture = Texture(...)
environmentMap = Sampler(environmentTexture, ...)

theta = 1.1    # index of refraction

refract = Cg( "" ... Cg code ... """)

passes =
[Render(
    VertexShader = refract.vertex(target=arbvp1),
    FragmentShader = refract.fragment(target=arbfp1)
)]

•  almost like CgFX so far ...
Generic application interface

```python
e = Effect("refract", theta=1.2)
...

while e.hasMorePasses():
    render(geometry)
```

- shader parameters bound by name matching
  - instance or class variables on effect
  - standard variables from the graphics API
  - vertex attributes on geometry
- complete encapsulation of effects
Another effect - Glow

```
passes =
    [Render(Target=blurBuffer, Color=GlowColor)] +
gaussian2DBlur +
[ImageProcessing(Source=blurTexture,
    AlphaBlendEnable=True,
    SrcBlend=SRCALPHA,
    DestBlend=ONE)]
```

- Multiple passes
- Needs features not in CgFX
  - render to texture
  - passes without geometry
- were easily added to framework
Glow continued

from convolution import *

gaussian2DBlur =
    3*[gaussian1DBlur(1,0),
        gaussian1DBlur(0,1)]

def gaussian1DBlur(x,y):
    vs = convolve4x1D.vs(...)
    fs = convolve4x1D.fs(...)
    return ImageProcessing(
        Target=blurBuffer,
        VertexShader=vs,
        FragmentShader=fs)

• Using ordinary programming language features make effects easier to write
GPGPU

- Using GPU as general purpose number cruncher
- Typical user scenario is interactive calculator
- Perfect match for a script language
- Example: Image processing