A scripting tool for real-time effect programming

Calle Lejdfors, Lennart Ohlsson
Department of Computer Science
University of Lund
Box 118, SE-221 00 Lund, SWEDEN
{calle.lejdfors, lennart.ohlsson}@cs.lth.se

ABSTRACT

Writing real-time visual effects for graphics hardware is made difficult by the high degree of dependence between GPU-level shaders and CPU-level orchestration of pipeline settings parameter bindings. This paper presents PyFX, an effect framework embedded in the programming language Python. Compared to existing existing frameworks this language embedding gives the effect programmer greater expressive power. These benefits, together with some imrpoved functional features of the framework, are demonstrated through some illustrative examples.

Keywords
Effect programming, CPU/GPU interaction, embedded languages

1 INTRODUCTION

With the introduction of programmable real-time hardware, procedural techniques previously used for cinematic effects have been made available for use in real-time graphics as well. However, writing real-time effects is made difficult because the CPU and GPU code is typically written in separate languages, while still having strong inter-dependencies.

One solution to this problem are so called *effect frameworks* which enable the unification of shader programs with the necessary pipeline states required for correct operation. Current effects frameworks such as DirectX Effects (DXFX) by Microsoft [2] and CgFX by NVIDIA [1] provide a number of features which simplify programming real-time visual effects. Both frameworks rely on effect specifications consisting of parameters, shaders, techniques, and passes, which are stored in text files and loaded at run-time.

However, the format used in current effect frameworks is lacking all but the most basic forms of abstractions, data-hiding, or sharing, making effect development unnecessarily complicated. Also there is lack of integrated support for some functional features such as RTT (render-to-texture) commonly used in todays graphics applications.

Posters proceedings ISBN 80-903100-8-7
WSCG'2005, January 31–February 4, 2005
Plzen, Czech Republic

2 THE PYFX FRAMEWORK

Our effect framework, PyFX, implements a complete effect programming language embedded in the object-oriented scripting language Python. The fact that PyFX is embedded in a fully fledged programming language enables high-level language features to be used for expressing effects. By using constructs such as function definitions, loops, conditionals, and modules to express and share common parts, an effect description becomes shorter and more clear. And the choice of a very high level language like Python makes it possible to achieve these benefits and still retain the declarative style of existing frameworks.

The functionality of PyFX includes all the features found in the DXFX and CgFX frameworks. In addition it provides:

- *Render-to-texture* – Render to an off-screen area which can be used as a texture.
- *Image processing support* – GPU based image processing operations can be applied to any texture or off-screen area.
- *Support for shader interfaces* – PyFX enables easy use of Cg's interfaces allowing run-time construction and composition of shader programs.

To accommodate these functional additions the PyFX language extends the passes of other effect frameworks to:

- *RenderGeometry* – the usual passes of other frameworks. Sets up the appropriate states and then instructs the application to submit geometry
- *ProcessImage* – performs 2D image processing between any number of images (which may reside in textures, off-screen areas or the current screen buffer). It provides support floating point targets and sources enabling HDR image processing.

PyFX is written in Python and built on top of OpenGL and Cg. The implementation in Python has made it possible to build a very flexible interface towards the application using the framework. This is described in more detail in [4].

3 EXAMPLES

As an example of using PyFX we present a glow effect [3], used to simulate the nimbus due to atmospheric scattering which appear around brightly lit surfaces. This effect is implemented by rendering an object to the screen, rendering the glowing parts of the object to an off-screen buffer, blurring the off-screen buffer and then additively blending the result to the screen. This can be expressed in PyFX by:

```
def RenderGlowRegions(target):
    return RenderGeometry(
              Target=target,
              VertexShader=glowMask.vs(),
              FragmentShader=glowMask.fs())

def GaussianBlur(source):
    ...

def AdditiveBlend(source, target):
    return ProcessImage(Source=source,
                        Target=target,
                        SrcBlend = SRCALPHA,
                        DestBlend = ONE)
```

The technique which performs blurring can now be written simply as

```
technique = [RenderGeometry(),
             RenderGlowRegions(blurBuffer),
             GaussianBlur(blurBuffer),
             AdditiveBlend(blurBuffer, Screen)]
```

By making use of PyFX's language and functional features the resulting effect is a readable specification of what the effect does and how it does it.

Examples of effect is simplified by introducing high-level languages features are numerous. For instance, fur typically makes use of multiple layers of decreasing opacity which are additively blended over the solid object [5]. This can easily be expressed in PyFX by using a function definitions to express the drawing of a single fur shell:

```
def RenderFurShell(s):
    shell = s/FurThickness
    return RenderGeometry(
                AlphaBlendEnable = True,
                SrcBlend = SRCALPHA,
                DestBlend = ONE,
                VertexShader = furVS(Shell=shell),
                FragmentShader = furFS(Shell=shell))
```

The entire effect, drawing the solid object followed by a number of shells, can then be expressed as

```
technique = [RenderGeometry()] + \
            [RenderFurShell(i)
               for i in range(1,NumberOfShells)]
```

again providing a clearly legible description of the operation of the effect.

4 CONCLUSIONS

We have presented an effect framework which improves on current frameworks in two respects; it enables to use of high-level language features for effects descriptions while retaining the declarative style of current effect frameworks, and, it provides an extended set of integrated functional features making a larger set of effect possible.

The main purpose of PyFX is to be a tool for investigating which features and characteristics are useful and desirable for effect programming. As such it provided a flexible environment for experimenting with effect frameworks and effect programming.

References

[1] CgFX 1.2 Overview. `http://developer.nvidia.com/`.

[2] DirectX SDK Documentation. `http://msdn.microsoft.com/`.

[3] Greg James and John O'Rorke. *GPU Gems*, chapter Real-Time Glow, page 816. Addison Wesley Professional, 1 edition, March 2004.

[4] Calle Lejdfors and Lennart Ohlsson. Pyfx - an active effect framework. In Stefan Seipel, editor, *SIGRAD 2004*, number 13 in Linköping Electronic Conference Proceedings, 2004.

[5] Jerome Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Real-time fur over arbitrary surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 227–232. ACM Press, 2001.