GPU Image Processing
—
The `cg_scotopic` Demo

Frank Jargstorff
NVIDIA

March 15, 2004

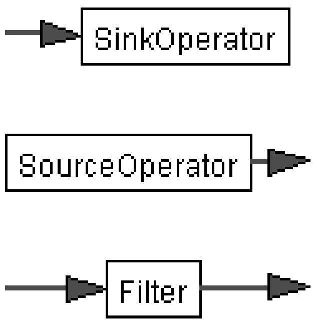
1 Introduction

The fully programmable fragment shaders and long shader programs, as supported by NVIDIA's FX class of GPUs, enable simple implementation of highly efficient GPU image processing.

The `cg_scotopic` demo showcases the underlying `nv_image_processing` framework (a C++ library part of the NVIDIA SDK 7.0) and how to extend it.

2 Design

The framework is based on the concept of an image-filter graph where the graph's nodes are image filters. The images flow from source nodes through filter nodes where they are processed to output nodes (e.g. display or save-to-file).



The `SourceOperator`, `SinkOperator`, `ImageFilter`, and `Image` classes all function as proxies to objects living on the graphics card so that the image once uploaded actually never leaves the card during processing.

The actual operator base classes are designed such that a derived class needs to specify only minimal information in order to implement a new filter:

- A Cg fragment shader implementing the actual processing operation, and
- some C++ code providing the fragment shader with the necessary parameters (e.g. blur factor, ...).

The image class is based on 16bit floating point numbers for each of the red, green, blue, and alpha channels (i.e. compatible to the OpenEXR format [2]). Images are reference counted for automatic memory management which allows image creation in source operators and deletion in sink operators without having to keep track of additional references to the image used for buffering results.

3 Implementation

The framework is implemented using the OpenGL API and NVIDIA's Cg programming language [1]. For optimum performance a variety of OpenGL extensions were used:

WGL_ARB_pbuffer which allows rendering images into background buffers.

WGL_ARB_render_texture which allows to use pBuffers as textures. For larger images this is a huge performance win over the alternative method of copying the pBuffer content into a texture.

NV_texture_rectangle which allows textures with non-power-of-two sizes. Traditionally OpenGL textures must conform to the number of pixels along the edges begin a power of two (i.e. $w, h \in \{2^n | n = 5, 6, 7, \ldots\}$).

NV_half_float for loading and storing 16bit OpenEXR images.

WGL_ARB_pixel_format for querying the correct 16bit floating point format.

4 Summary

For a more complete description of the design and implementation of the framework and a variety of other articles on GPU image processing as well as GPU use in general see [3]

References

[1] Fernando, R. & M.J. Kilgard, *The Cg Tutorial,* Addison-Wesley, 2003.

[2] `www.OpenEXR.org`, Industrial Light & Magic.

[3] Fernando, R. (ed.), *GPU Gems*, Addison-Wesley, 2004.