Simple Glow

Author
tlorach@nvidia.com

Month 2007

Document Change History

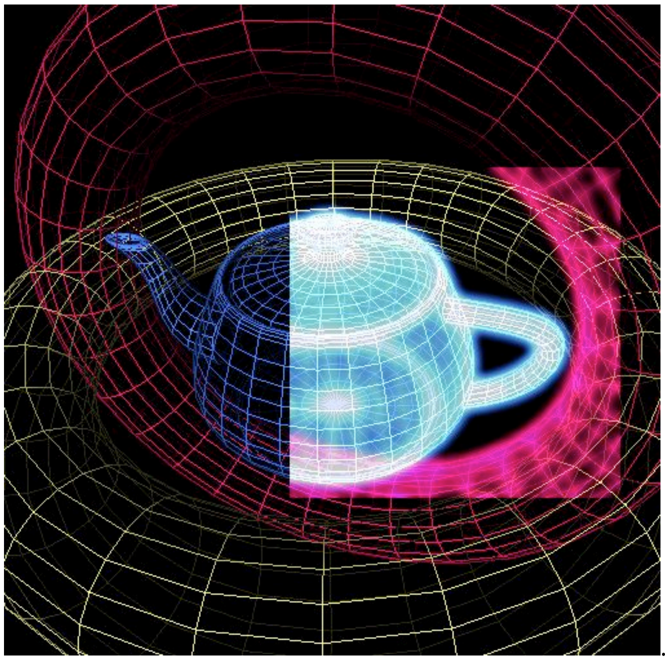
Version	Date	Responsible	Reason for Change
1	1/12/2007		Initial release

Abstract

This sample is a simple example on how to perform a glow effect by rendering into an arbitrary size Frame Buffer Object (FBO).

The Glow effect is performed on a specific part of the screen and can be done only on specific objects of the scene.

You can imagine using such a postprocessing effect in CAD/DCC to emphasize some items from a selection or picking for example.

.

Description

This sample is a simple example on how to use FBO and CgFX for a simple post-processing effect.

We are using a Gaussian filter in order to do the glow effect. The fact that this filter is *separability* will allow us to divide the glow effect in two passes: one vertical and one horizontal. We will use FBO for intermediate render targets. Those buffers will also be available as textures.

The basic idea is to

- render the scene in the back buffer
- activate the effect
- render some elements of the scene in a FBO
- compute the Glow effect
- bind the final FBO as a texture and blend this effect with the previously rendered scene in the backbuffer

Note that this sample is showing a different approach than what we are used to see :

Most of the time, the scene is being rendered in the back buffer or in an off-screen buffer. Then a down-sampled version of it will be used to do the post-processing effect.

Here, choose to draw for a second time some elements of the scene especially for the post-processing stage. In some situation this can be interesting while in some others this can be overkill. Good things are :

- When the effect must be locally applied : no need to render more than what's needed for the glow effect. And this could be a small part of the whole scene.
- This effect doesn't require to change the whole rendering strategy of the application where you want to add this effect : we only added some more rendering commands for intermediate render targets and merged the final result on the top of the original scene.

Running the Sample

The sample allows you to change the size of the window you are moving with the mouse.

You can also change the resolution of the offscreen buffers that are used for the Glow effect.

Move the mouse onto the scene to see the effect…

Implementation Details

Please go to chapter "**Step by step description and Integration**" for more details on the step-by-step process.

The sample is made of the following files :

- `Simple_glow.cpp`: This file contains the main layout for the sample to run. This is where you will find the main rendering loop and how the effect is being integrated to this loop.
- `FilterBox.[cpp|h]`: this file contains the class FilterBox : this is where everything is performed.

Step by step description and Integration

The integration of this effect requires Cg and GL_EXT_framebuffer_object extension.

The sample isolated the post-processing effect in a class located in *FilterBox[.cpp/.h]*.

No matter if you want to use this class or if you want to code everything from scratch, the integration will be made of these different stages :

Initialization

Initialization process is happening in the "*Initialize()*" method of *FilterBox*

- Check for the FBO extension
- Creation of 2 textures : one for horizontal pass, one for the vertical pass
- Creation of a render buffer for the depth buffer
- Creation of the CgFX effect
- Gather misc Cg parameters and set some of them

Rendering loop

The rendering loop is made of

- The typical rendering from your scene
- The rendering dedicated to the Glow effect : some elements from you scene

You may want to add at the end of your draw calls an additional set of draw calls encapsulated by *Activate()* and *Deactivate()*.

Activate() Method will

- bind the framebuffer (color buffer and depth buffer)
- setup the Viewport to the window area where you want the glow effect to work

Anything that will be rendered will of course end up in the frame buffer instead of ending to the Back buffer.

Then *Deactivate()* Method will :

- unbind the frame buffer to come back to the back buffer rendering
- restore the previous Viewport related to back buffer rendering

The final step through *Draw()* method will compute the Glow effect and render it on top of the scene, in the back buffer by using Blending operation :

- validate the technique
- bind the 2nd framebuffer so we render into it
- use the 1st framebuffer as a texture so we can use it as the source

- compute the horizontal Gaussian blur by rendering a fullscreen quad. Note that this quad will be fullscreen in the Viewport, not in the window of our demo…
- bind the 1st framebuffer previously used as a texture so we will render in it
- use the recent 2nd framebuffer as a texture so we can use it as the source
- compute the vertical Gaussian blur with a fullscreen quad rendering
- unbind framebuffer so we can render back to the backbuffer for the final pass
- Use the 1st framebuffer again as a texture
- Use a simple Additive blending and render a quad size and located at the specific place in the screen where we wanted the effect to be applied.
- Reset any pass state from CgFX

As shown in this sample, all of these operations are pretty independent of the code you wrote before.

Performance, known issues

The main issue with such a method is that some of the elements in the scene will have to be rendered twice. In fact it is even possible that you will have to render a second time the whole scene.

The bottleneck would be then be in the amount of draw calls.

However, this sample isn't made to show that it is better than the alternate method (downsampling the already rendered scene and using it for the effect…). It's just an alternative that could be interesting in some situations :

- When the effect must be locally applied : no need to render more than what's needed for the glow
- This effect doesn't require to change the whole rendering strategy : we only added some more rendering commands for intermediate render targets and merged the final result on the top of the original scene.1

Note that if you want to keep objects being correctly occluded, you'll have to draw occluders as black objects.