Fábio Carvalho
fabio.dscar@gmail.com

CDP Unity Test

This is a very simple implementation of a *blurred glass* material. The background is blurred into four global textures and the glass blur is calculated by interpolating between those four textures. The glass material includes optional parameters such as tint, blur and distortion maps. The frequency at which those blur textures are updated can also be adjusted.

1 Key Submitted Files

- **BlurGen.cs**: This file implements the demo and its interface.
- **BlurFilters.cs**: This file implements the client filter logic that handles the `Blit` calls and `RenderTextures`.
- **BlurFilters.cginc**: Declaration of functions that perform Gauss and Kawase filtering - included by others shaders.
- **BlurryGlass.shader**: Shader for the material being tested.

2 The Filters

Gaussian blur is implemented as a 7 tap separable filter. In the CG code, it receives precomputed weights and offsets as input for efficiency. This also enables the use of linear sampling filtering [1].

In order to produce high blurs, one generally needs a filter with a large radius and an high number of taps to make its quality acceptable. Here, I blur downsized textures so that the blur gets spread when upscaled and filter `Blit`s are faster.

3 Limitations

This simple implementation has several limitations:

- **Front objects bleeding into the blur**: The global blurred textures have all the opaque objects rendered into them. When applying high amounts of blur and/or distortion, front objects bleed into the blur. In order to solve this, a technique such as [2] needed to be implemented.
- **Inability of refracting stacked glasses**: Blurred textures only include opaque objects and information about other glasses is not included.
- **Inaccurate refraction**: This implementation can only simulate very thin glass. Not only are distortions not physically based, when the material is applied to an object that has depth (ex: a sphere), it will appear 2D when rendered. It also displays refractions at full intensity even at grazing angles. To solve this, we would need to, at least, use a Fresnel function to accurately scale refraction intensities through angles. In that case we would also need to add reflections (e.g.: using the cubemap) to account for the energy loss that would show up at those angles.

References

[1] Daniel Rákos. Efficient gaussian blur with linear sampling, 2010.

[2] Sousa, Tiago. *Generic Refraction Simulation.* GPU Gems 2, 2004.