Screen-Space Subsurface Scattering

Revision 1.8 February 2021

davidmiranda.me

Model sources provided by **https://ten24.info/**

Works on Built-In pipeline ONLY

Index

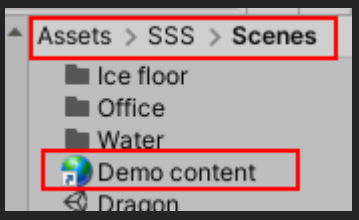
Abstract

Commonly known as SSS, this effect performs a special type of blur of the diffuse lighting pass in screen space to simulate how light is scattered behind the surface. The program is highly customizable to meet performance compromises easily. Version 1.7 introduces a new module for blurred transparencies. Working Platforms: PC & VIVE. Here you can find some demos.

Windows - Chess Demo (PC) - VR Head (VIVE - PC)

Installation instructions

1. Make sure your project is set to linear color space in player settings
2. Delete any previous folder of SSS
3. Import the package
4. For better quality of shadows set "Shadow distance" to something around 100
5. Install post process from package manager
6. Download and import the Demo content (Head, eyes and office art assets)



Gallery

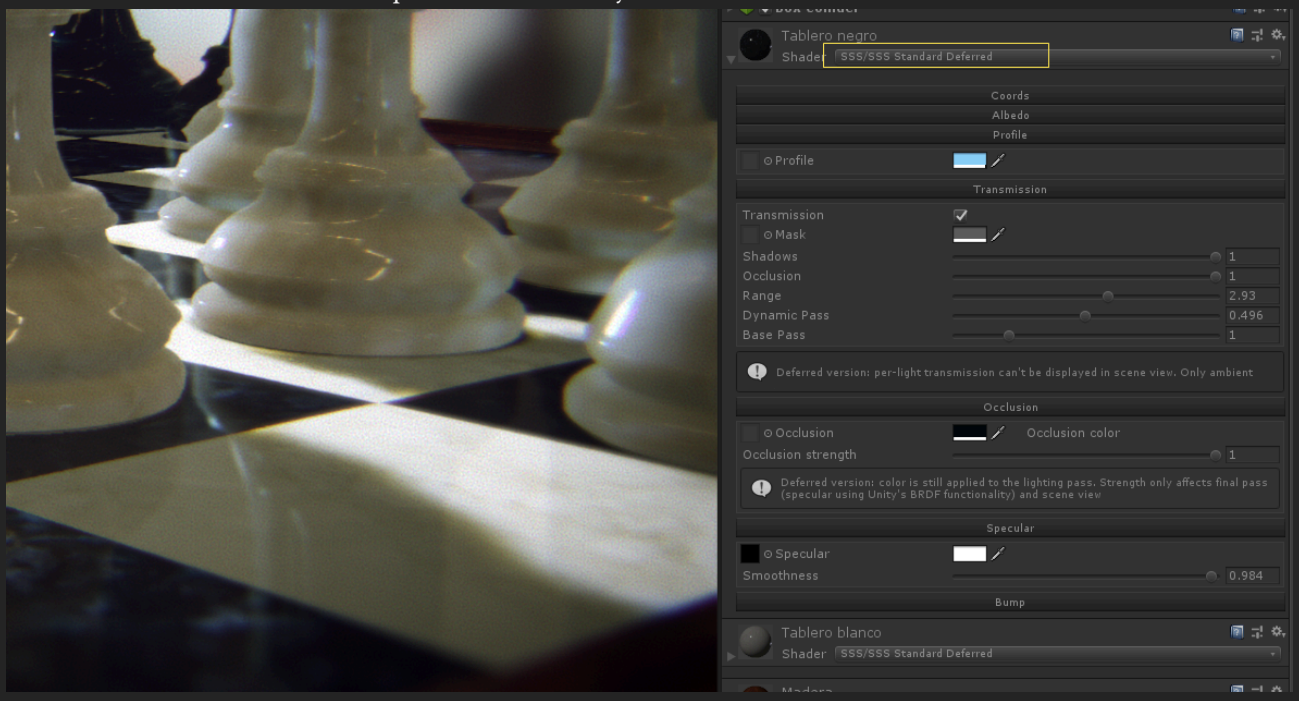
Full resolution image Gallery

Components

- Camera program
  Add this script to the game camera to start using it.

- Material
  Only materials using the shaders 'SSS/Standard', 'SSS/Standard Deferred' and 'SSS/Sample' are compatible with this effect.

The deferred version allows us to use post-effects that rely on GBuffers such as SSR:

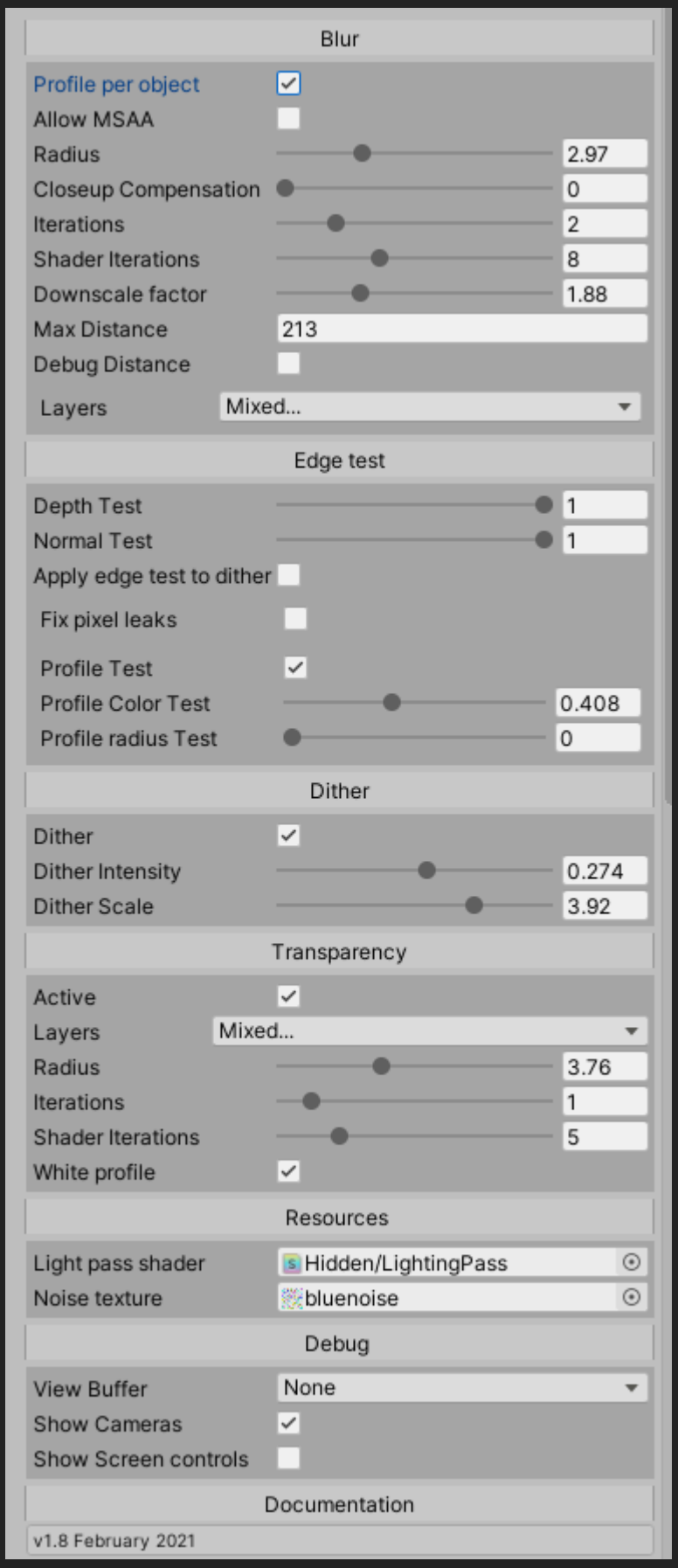


The effect is not applied to the scene view:


Scene
Asset Store
Shaded
2D
Gizmos
Persp
Game
Display 1
Free Aspect
Scale
Maximize On Play
Mute Audio
Stats
Gizmos

Usage

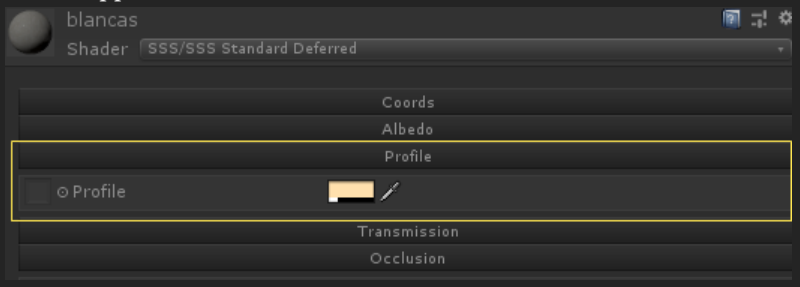
Camera



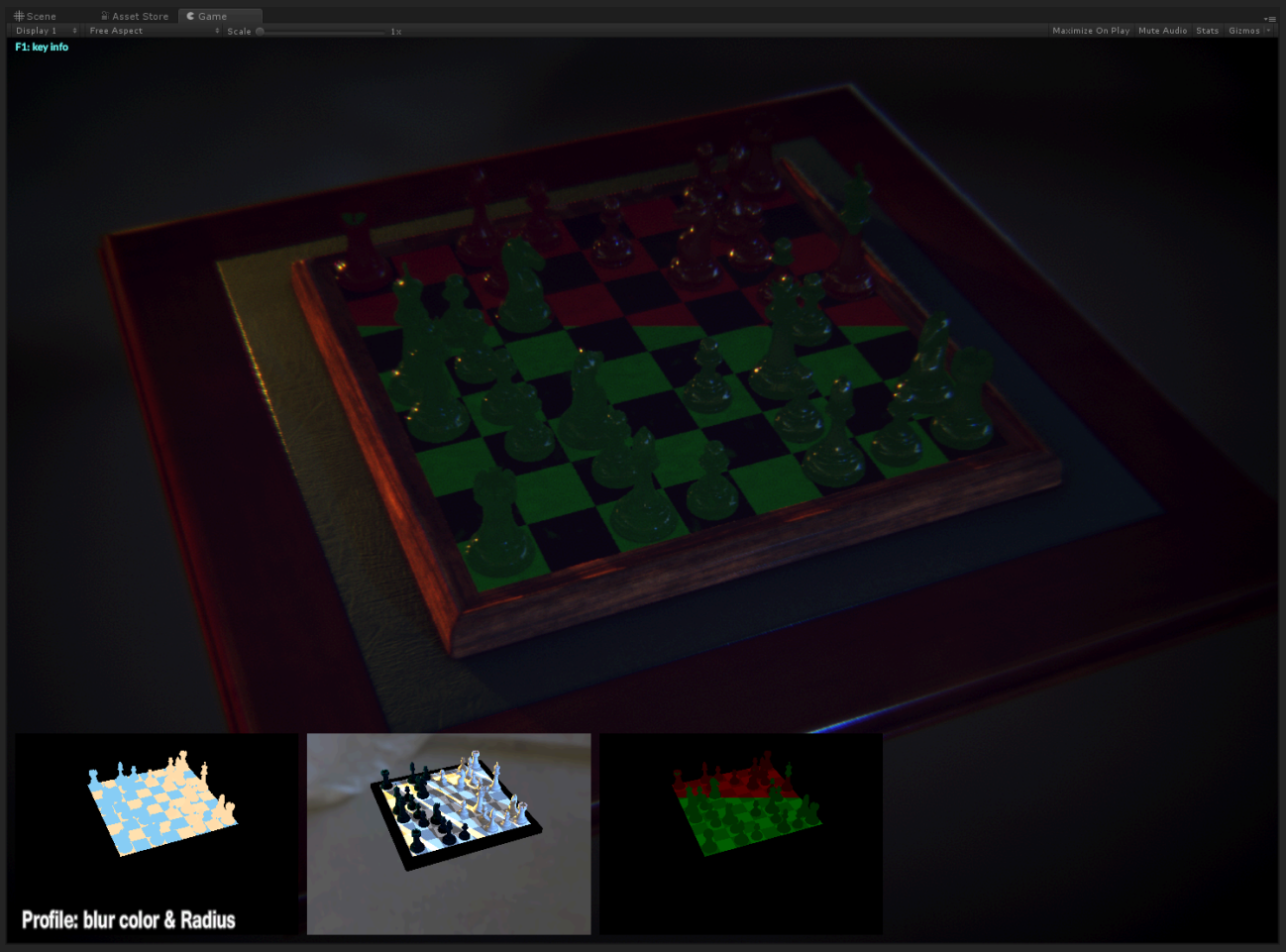
Blur tab

- **Scattering Color:** if profile is not enabled, the scattering value is global and controlled from this color.

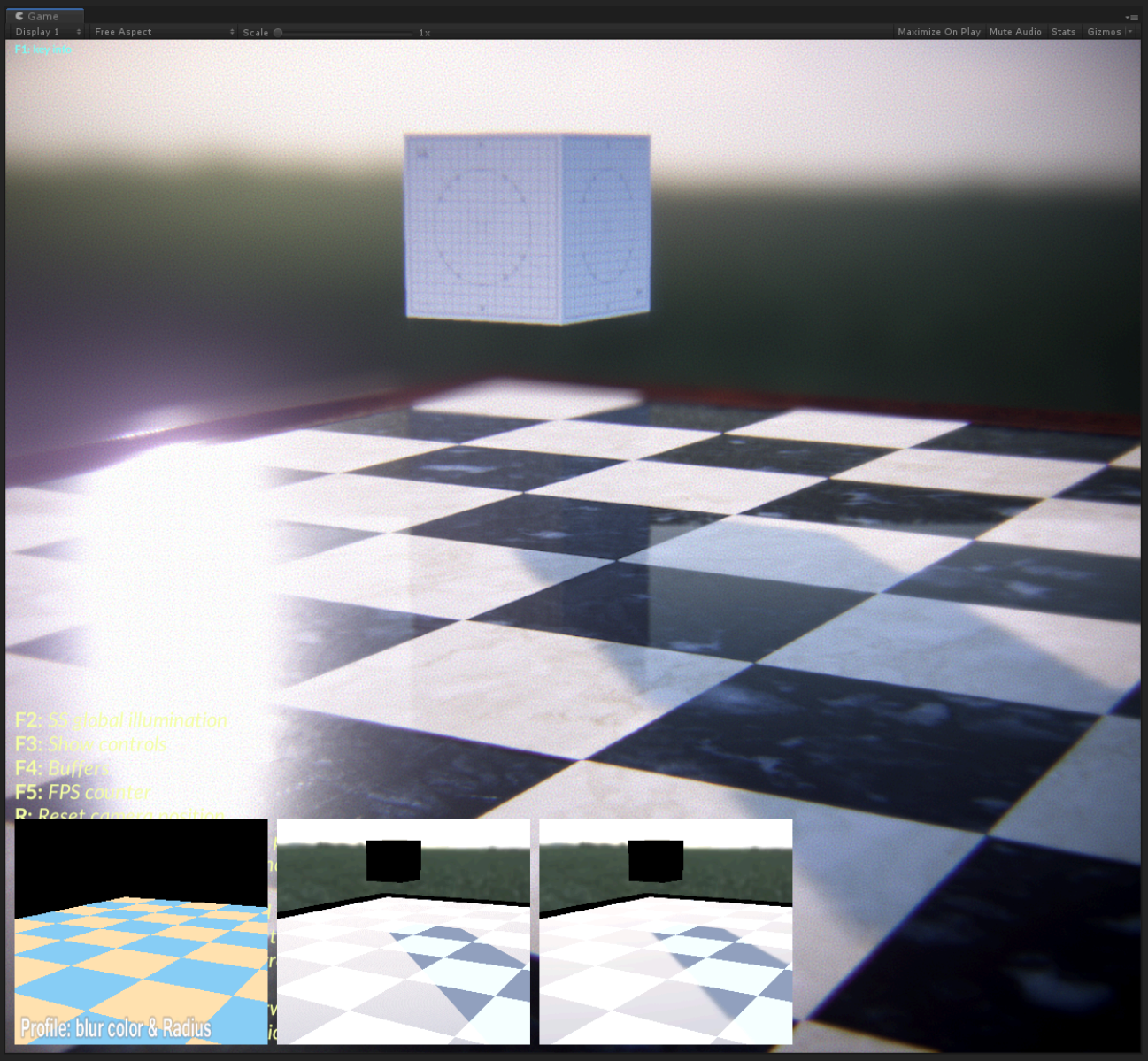
- **Profile per object**: allows to control the scattering behavior (color and radius) from the material. Note that this increases the cost
    - In this case, a new tab will appear in each material



- **Allow MSAA**: If MSAA is enabled in this camera, perform antialiasing in the generated buffers. Note that this increases the memory cost. Sadly, unity seems to not apply msaa in the depth buffer from a secondary camera, so this is not 100% correct yet.
- **Radius**: blur radius
- **Closeup Compensation**: blur radius is constantly adjusted by distance from camera to pixel to preserve the perceptual blur radius in every situation. When the object is very close to the camera, the radius increases considerably. The problem is that this situation has a high impact on performance, so this parameter is used to allow you to choose the amount of correction to apply. 0: no compensation (cheap), 1: max compensation (expensive)
- **Iterations:** number of iterations in the image effect blur
- **Shader Iterations:** number of iterations per pass.
- **Downscale factor:** allows to reduce the buffers size. Useful to improve performance
- **Max distance:** limit the computation of the effect by distance. Enable **Debug Distance** to see the range in the scene:



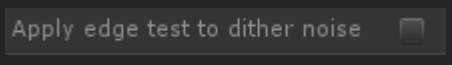
- **SSS Layer:** layers that will be rendered. To receive shadows from objects that are not sss they must be included. Example: this cube is not an sss object, but to get shadows from it on a sss surface it is included in the lighting pass by selecting its layer. During the lighting pass it is just black and casts the shadows that we need.



Edge test

Depth & Normal test: Thresholds required to detect the edges so that blur takes edges into account to avoid blur on them.

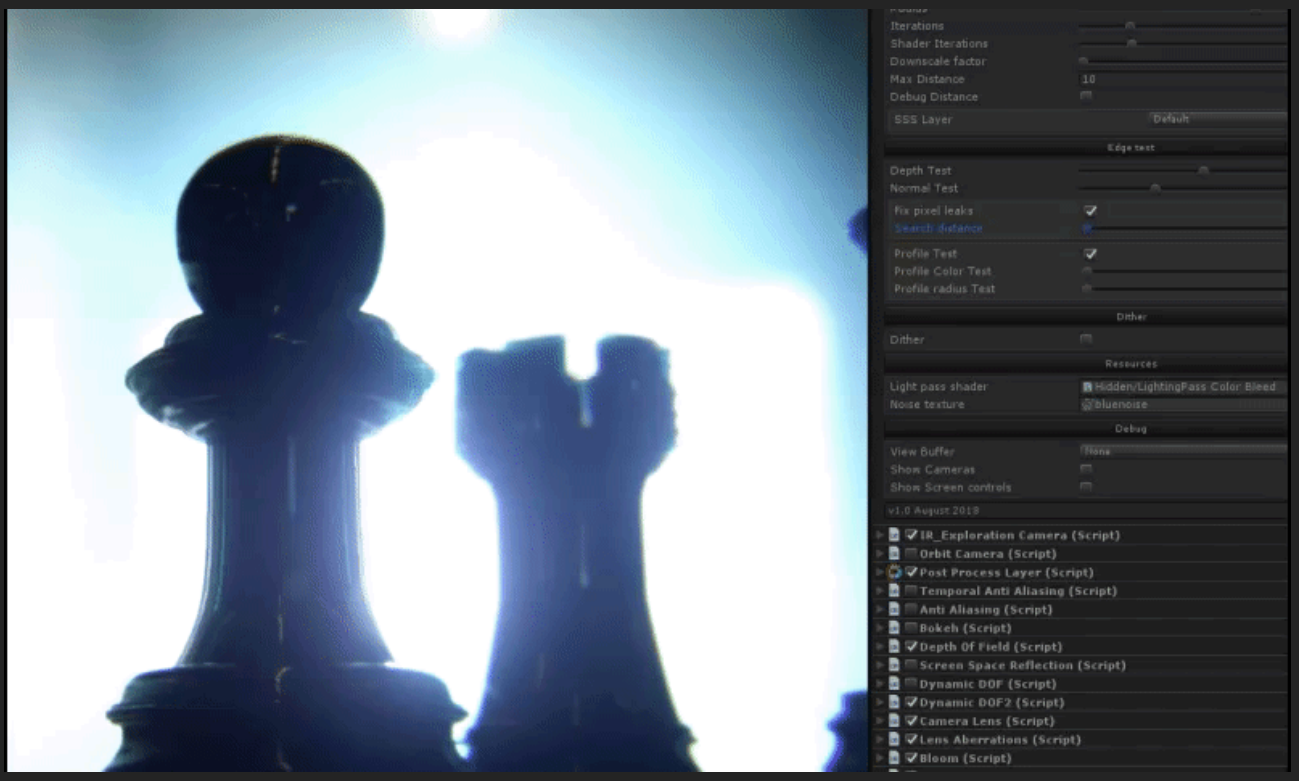
Depth & Normal test: Thresholds required to detect the edges so that blur takes edges into account to avoid blur on them.



When Dither is enabled, in some situations with high contrasts we will see pixel leaks like these:

Enable **Apply edge test to dither noise**
Situations with high contrast will reveal leaked pixels like here:

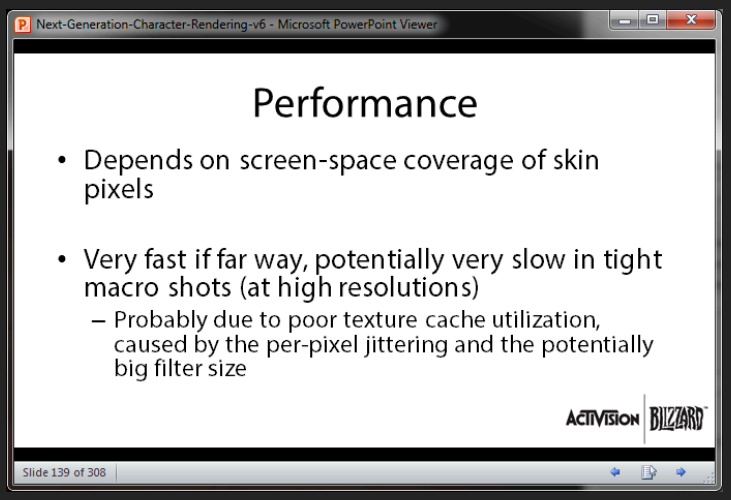
This situation can be solved with **Fix pixel leaks:**



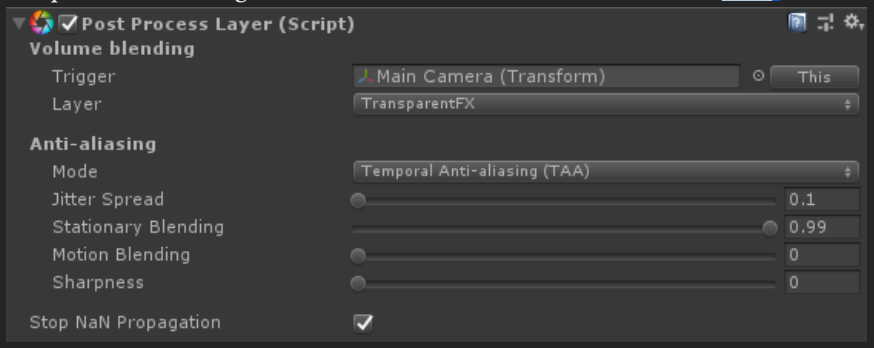
Dither

- Dither is the key to hide the visible seams that are inherent to this technique.

Dither can be very expensive in closeups. This is already a known hardware problem described by Jorge Jimenez.

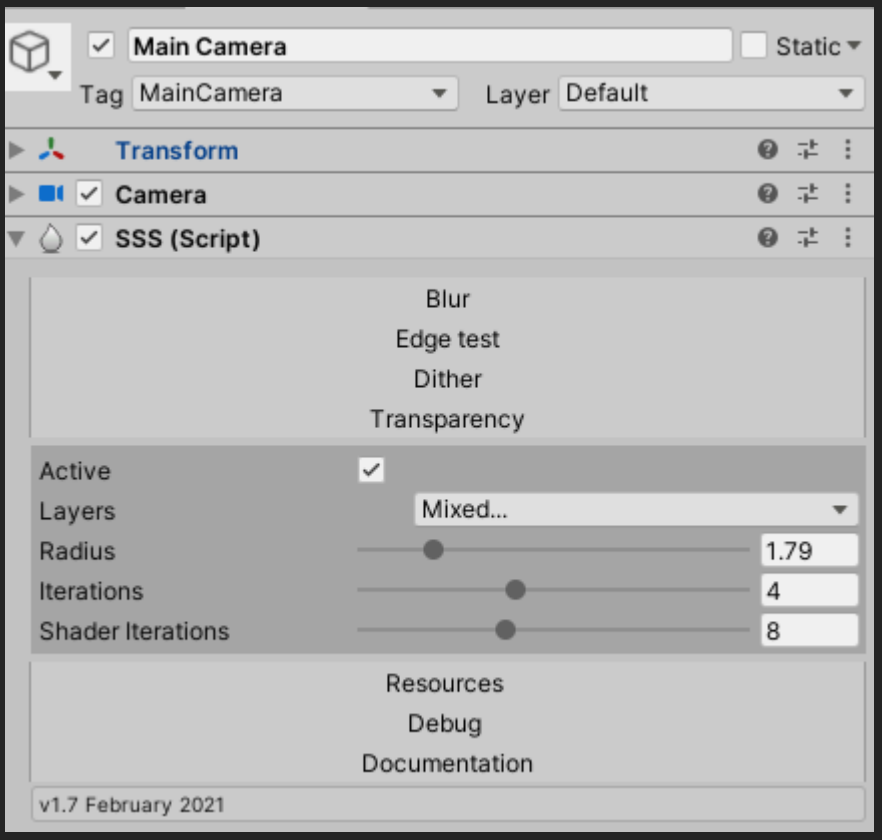


Use 'Postprocessing' temporal antialiasing to remove the noise. The TAA included here also works fine

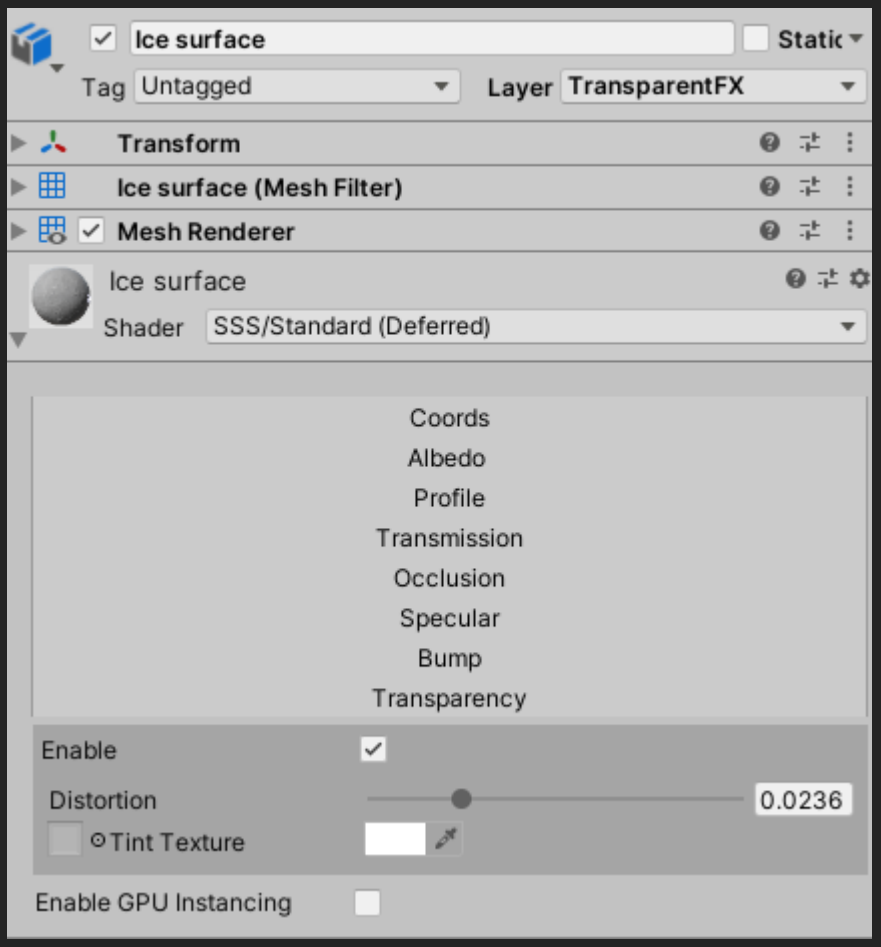


Transparency

Version 1.7 introduces a new module to allow blurred transparencies. Note that this could double the GPU work, so use it carefully.

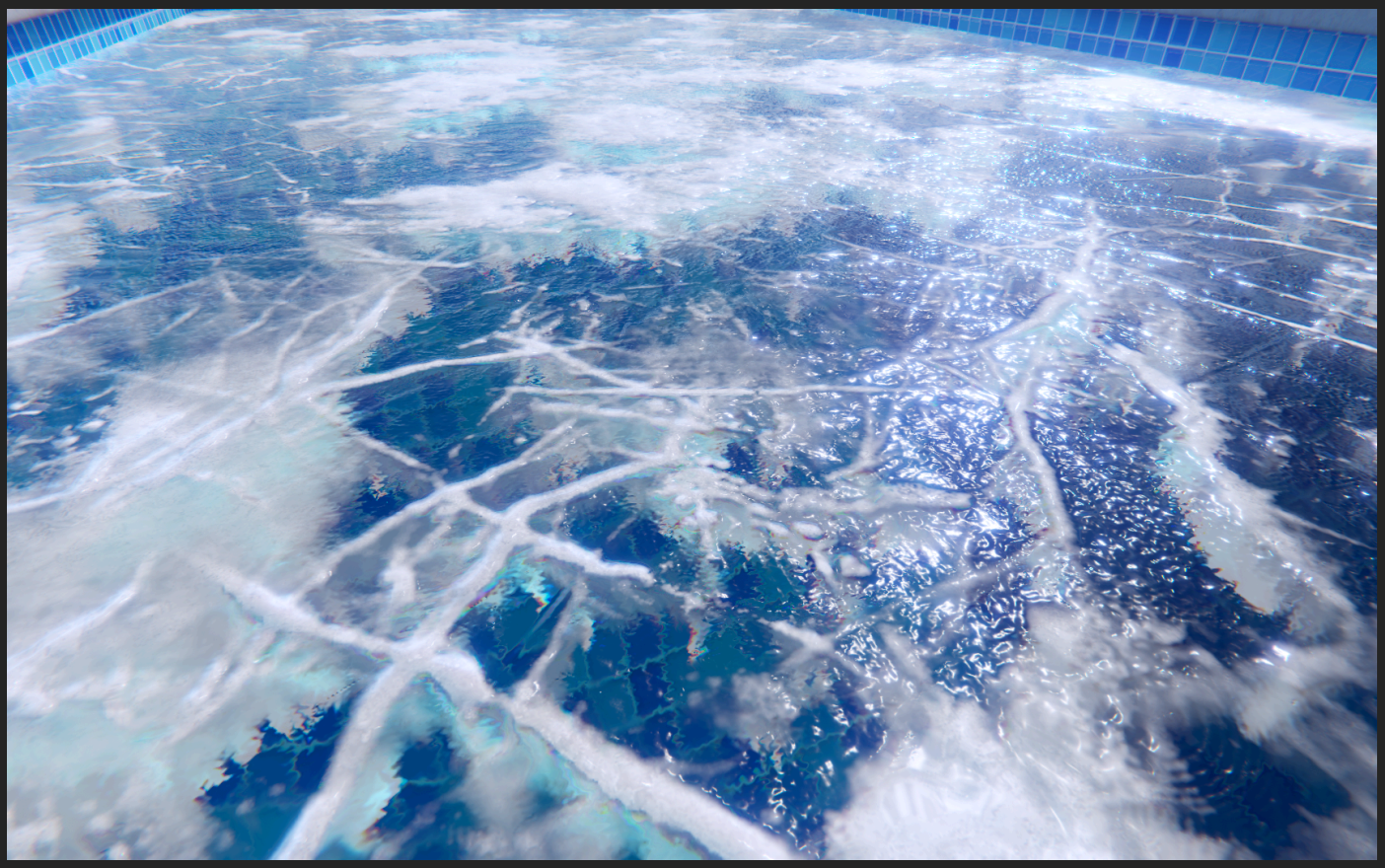


Functionality is the same as SSS. Select the layers to be rendered and adjust blur radius and iterations. When enabled, the materials will show a new tab with specific adjustments.

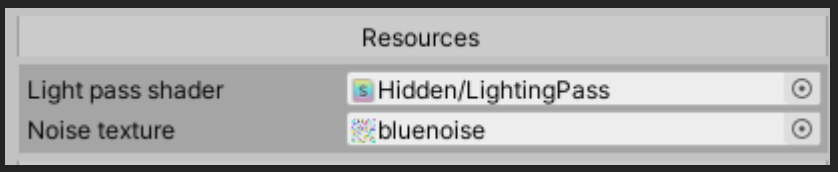


This new feature makes possible to simulate semi opaque materials. Opacity is controlled by Albedo alpha and blur radius by Profile texture. The provided glass examples don't use SSS, only transparency. SSS blur iterations is set to 0 and albedo is black.

Ice floor is using both surface scattering and transparency blur



Resources

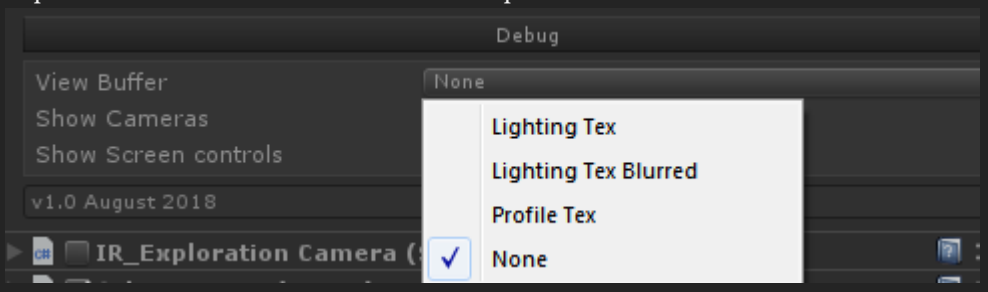


- **Light pass shader:** optionally, you can customize the light pass. The lighting pass can be seen here:

- **Noise texture:** noise texture used for dither

Debug

- **View Buffer:** output the chosen buffer to the screen. The prefab 'Buffer viewer' interacts with this

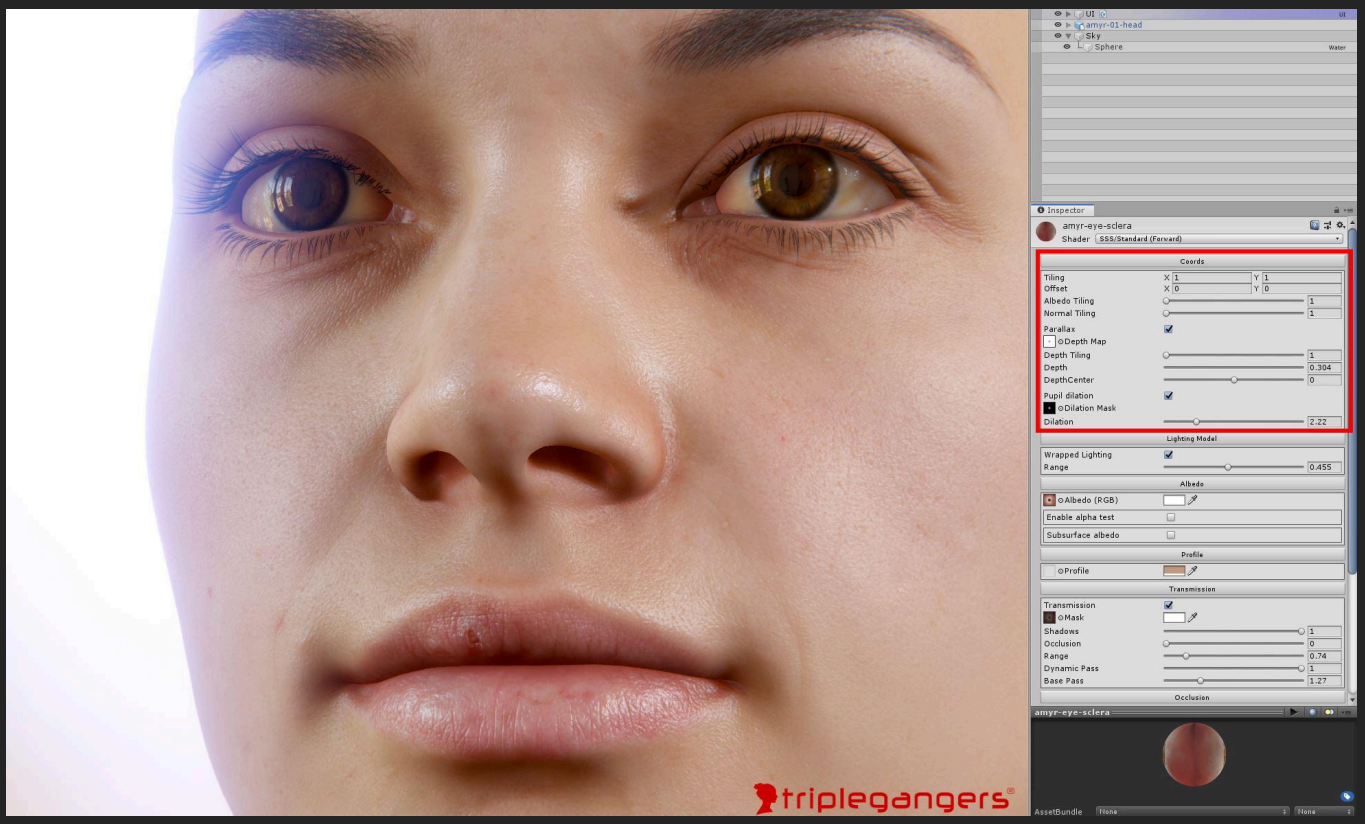


Model provided by triplegangers **NOT INCLUDED**

- **Show Cameras:** hide or unhide the secondary cameras
- **Show Screen Controls:** useful to test performance in final build.

Material

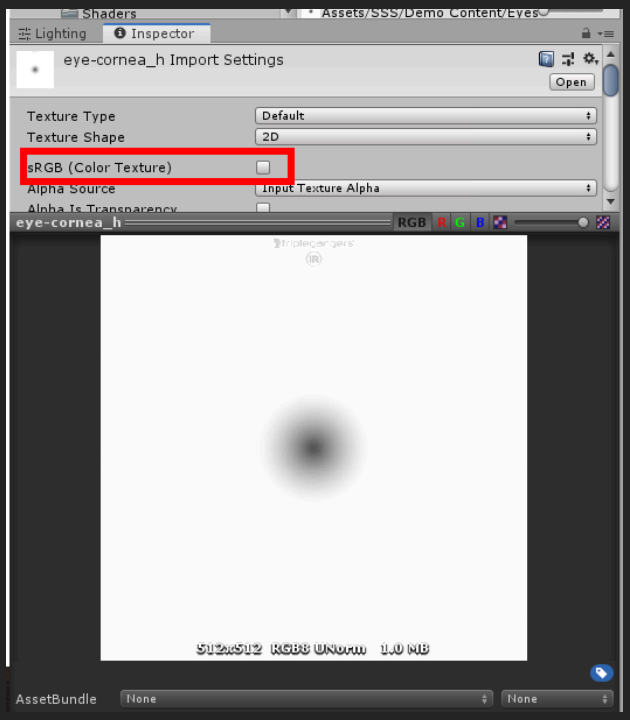
- **Coords Tab**
    - Version 1.6 introduces a new set of options to render eyes properly. These are:
        - Parallax
        - Pupil dilation



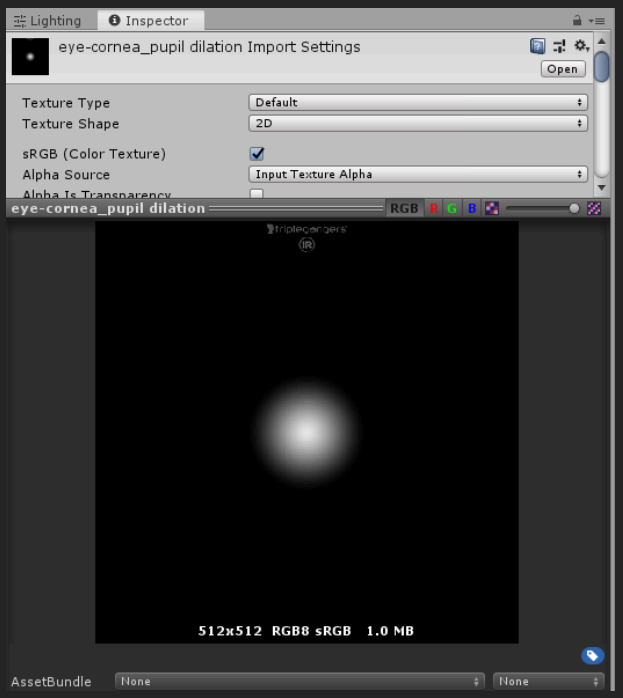
Model provided by triplegangers. All sources included in the package only for eye area.

Triplegangers kindly donated this model to us. For obvious commercial reasons we were only allowed to share a portion of their product. This is what you get with the package:

Depth map must be set to linear color space

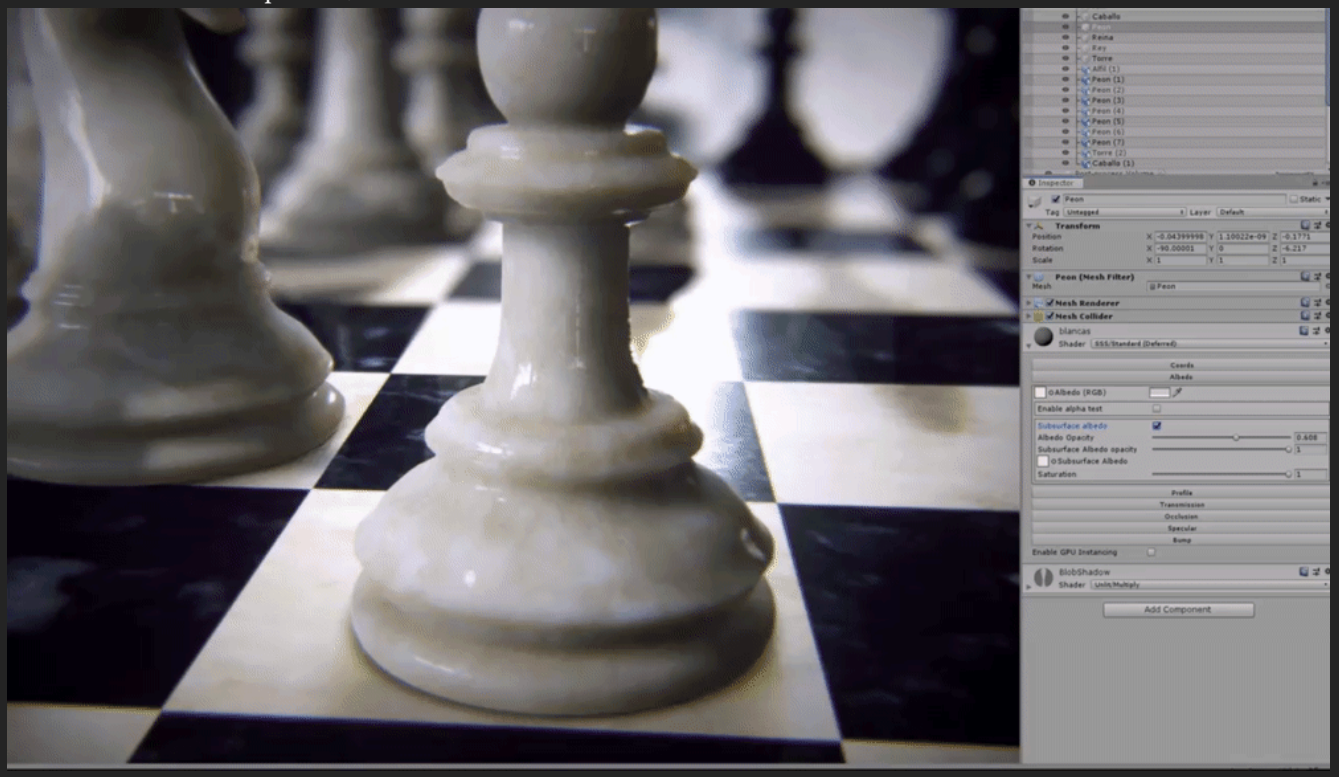


Eye dilation map masks the area where we want to scale the pupil area



- **Lighting Tab**
    - Version 1.6 introduces a new option to modify the diffuse model (only forward version):
        - Wrapped lighting: softens NdotL

- **Albedo Tab**
  - **Albedo (RGB):** standard color texture
  - **Subsurface Albedo:** renders albedo map in the lighting pass to simulate deeper features (tatoos, moles, marble deepness...)



- **Profile Tab**
  We can control scattering color (RGB) and radius (A) by texture. Download sample

- **Transmission Tab**
  Using geometry math, this attempts to emulate light transmission
  - **Mask:** channel R of this texture and color are used to multiply the effect.
  - **Shadows:** influence of shadows
  - **Occlusion:** influence of occlusion
  - **Range:** angle threshold between light direction and geometry normal
  - **Dynamic Pass:** intensity of the dynamic component of this effect.
  - **Base Pass:** intensity of the base pass. This effect has a dynamic component (per light) and this one is based on ambient (spherical harmonics with inverted normals)
- **Occlusion Tab**
  - **Occlusion:** used to multiply color and environment reflections
- **Specular Tab**
  - **Specular:** standard specular map and color as seen in Standard shader
  - **Fresnel effect:** sometimes, the standard shader, even being correct, outputs too strong reflection at the edges. With this slider we can attenuate reflection intensity at glancing angles.
  - **Smoothness:** standard smoothness as seen in standard shader

- **Cavity Map**

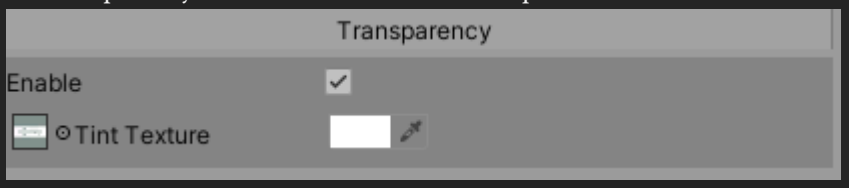
  Optionally, we can use a cavity map to attenuate specular and reflections as suggested by Jimenez

- **Bump Tab**

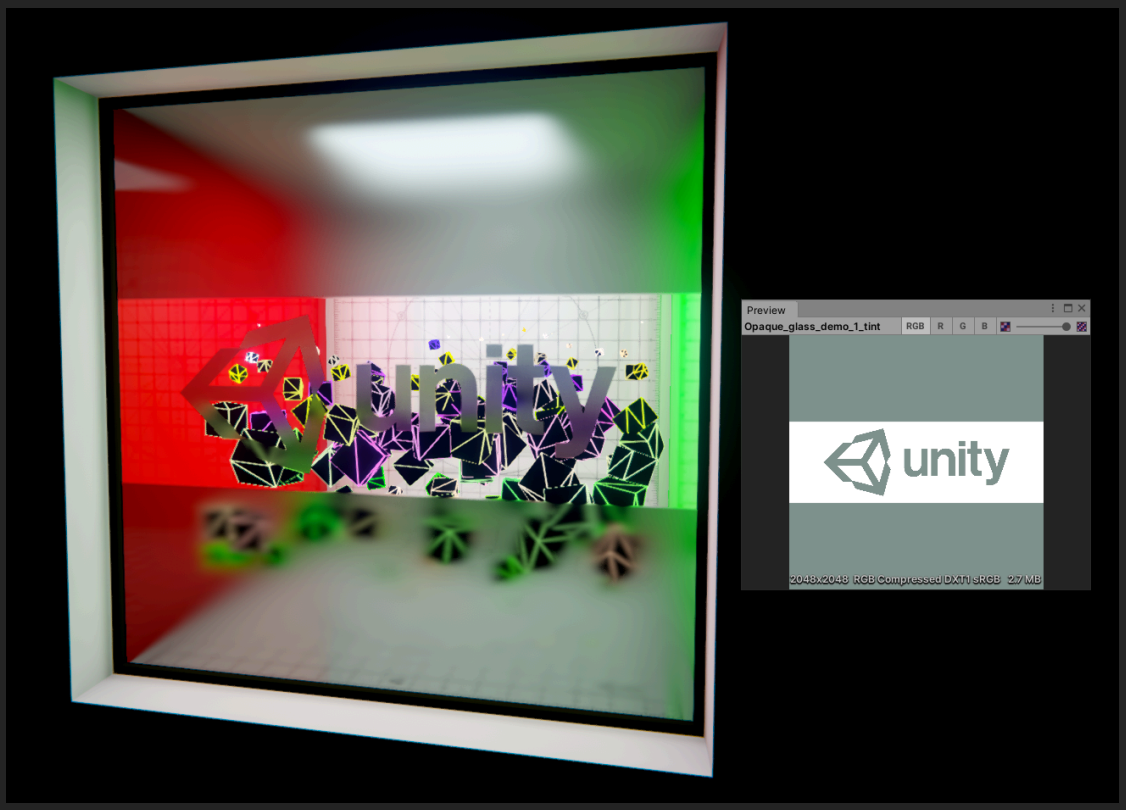
  Normal Map: standard normal map

- **Transparency**

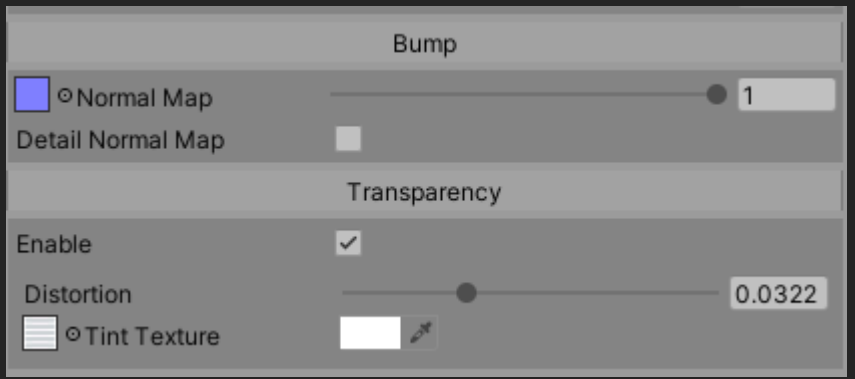
  Only available when transparency is enabled in the camera script.

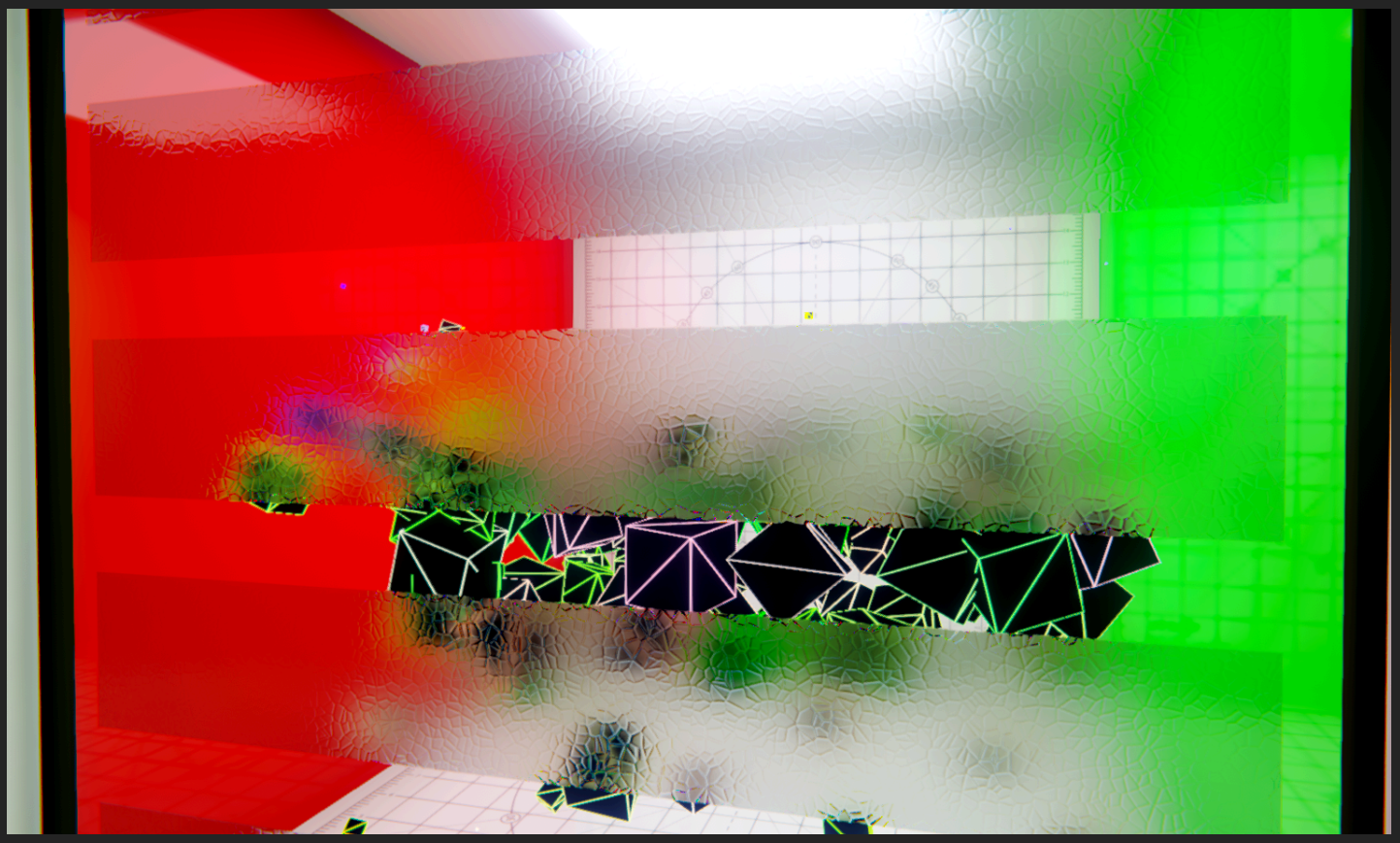


  - **Tint**: tints the transparency buffer with texture and color



  - **Distortion**: when a normal map is assigned it is possible to use it to distort the transparency buffer





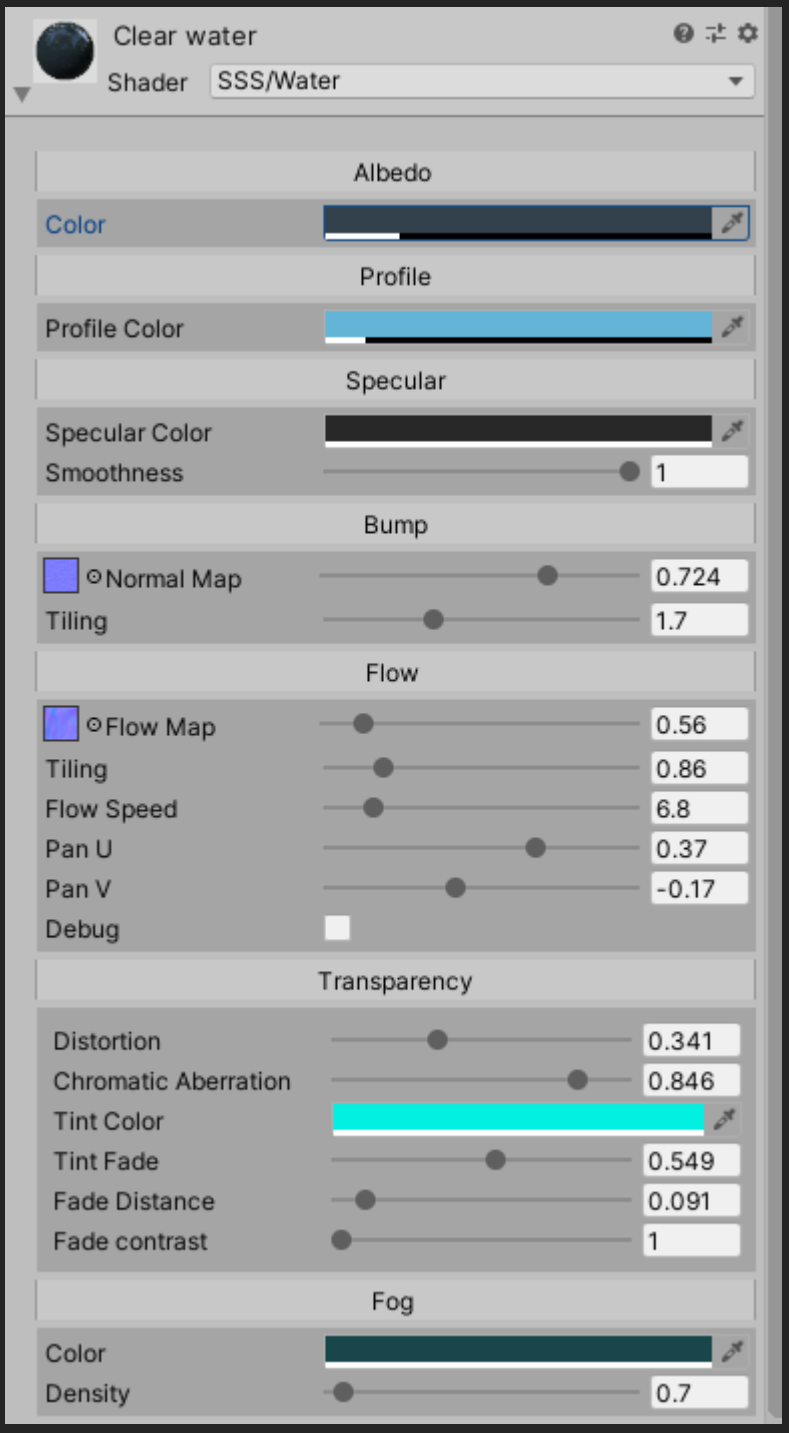
Water

Version 1.8 introduces a new shader to make water or liquids of any kind.

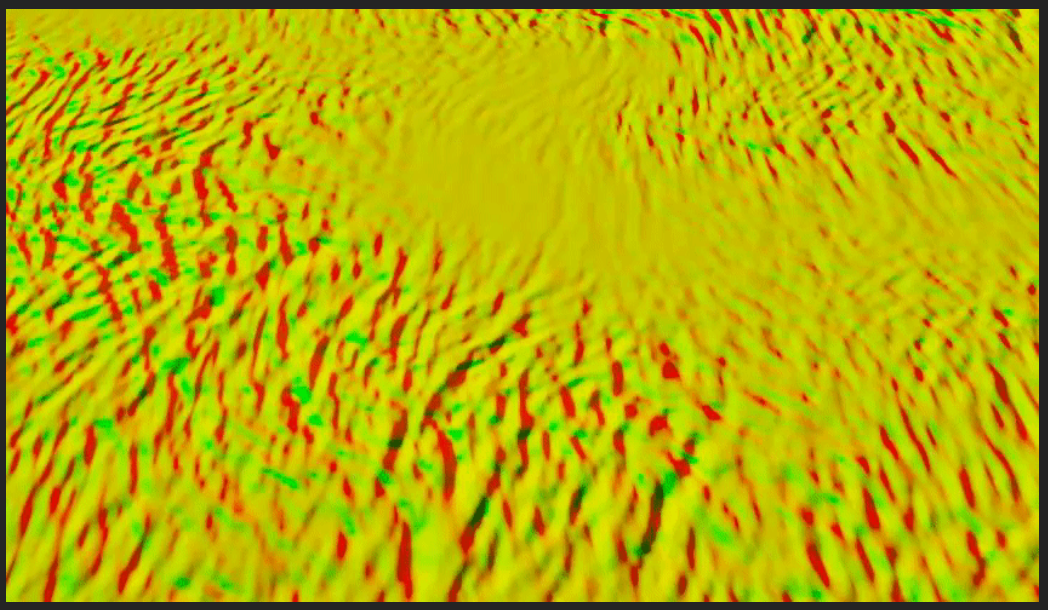
Features:

- Blurred transparency
- Chromatic aberration
- Normal animation by flow map
- Deferred path (compatible with SSR)
- Edge fade

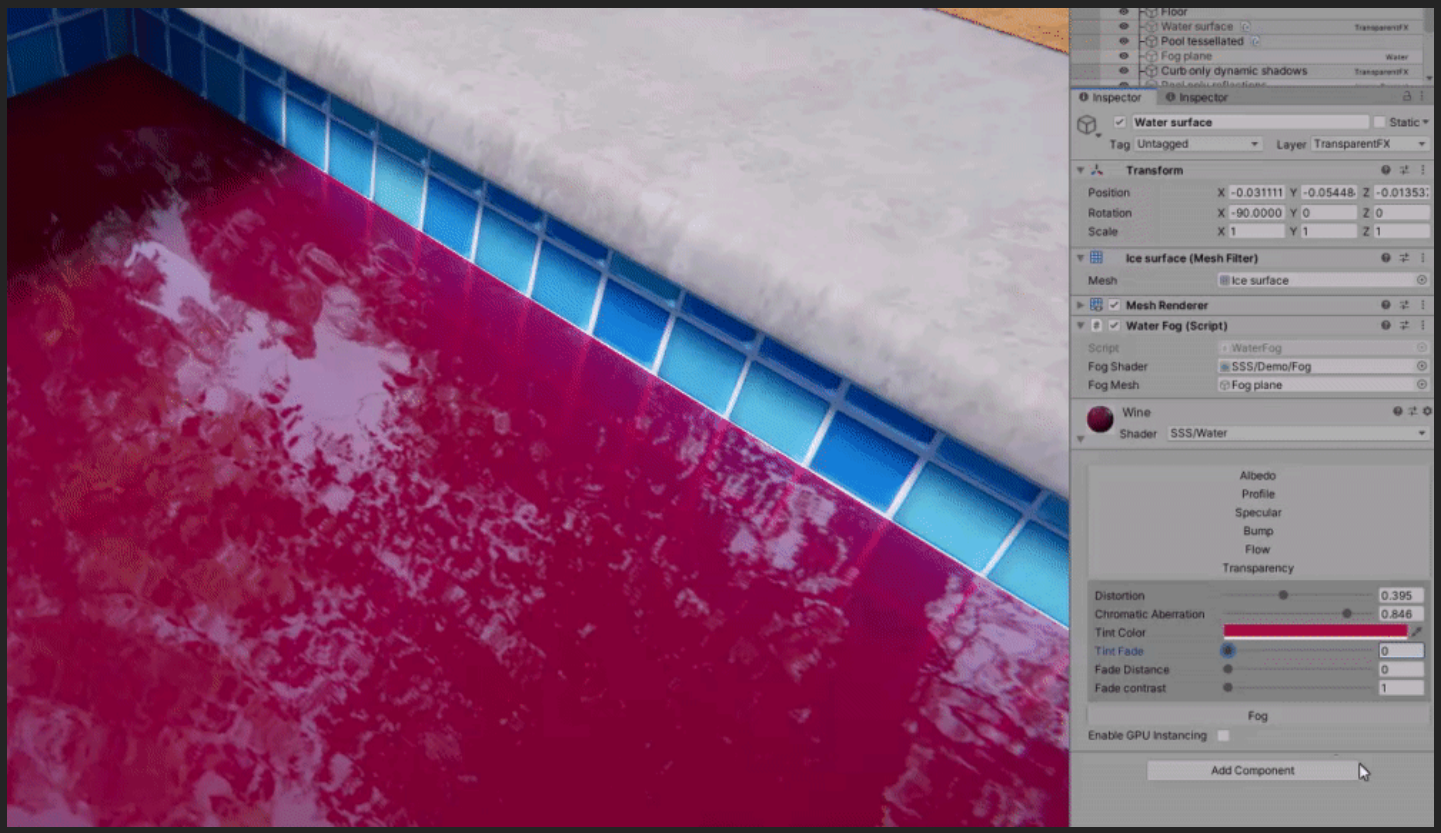
- Shader: SSS/Water



- **Albedo:** Surface color and opacity
- **Profile:** Blur color and radius
- **Specular:** Color and glossiness
- **Bump:** Normal map and tiling
- **Flow**
    - **Flow map:** normal map and intensity (texture format must be normal map)
    - **Tiling:** flow map tiling
    - **Flow speed:** animation speed
    - **Pan U and V:** adds a panning movement
    - **Debug:** shows only the resulting normal



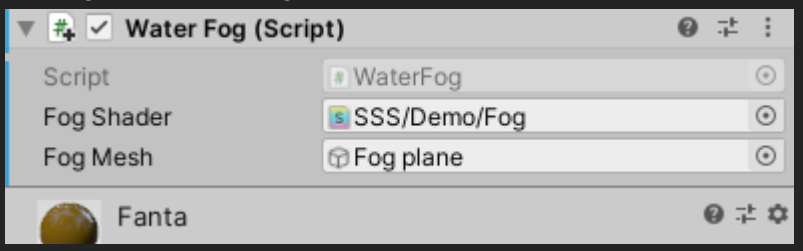
- **Transparency**
  - **Distortion:** amount of distortion
  - **Chromatic aberration:** amount of chromatic aberration
  - **Tint color:** multiply transparency buffer with this color
  - **Tint fade:** use depth to fade tint color(only affects tint color and the depth input is distorted)



  - **Fade distance:** edge transparency, affects all colors. Depth input not distorted. Usage: blend only contact edges

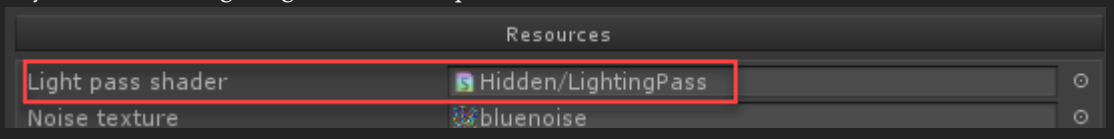
- **Fog**
  - **Color:** Fog Color
  - **Density:** fog opacity

Fog is computed in a separate geometry. Duplicate the water mesh and move it down in the Y axis a little. Now in the water mesh add this script and add the fog mesh to the “Fog Mesh” field

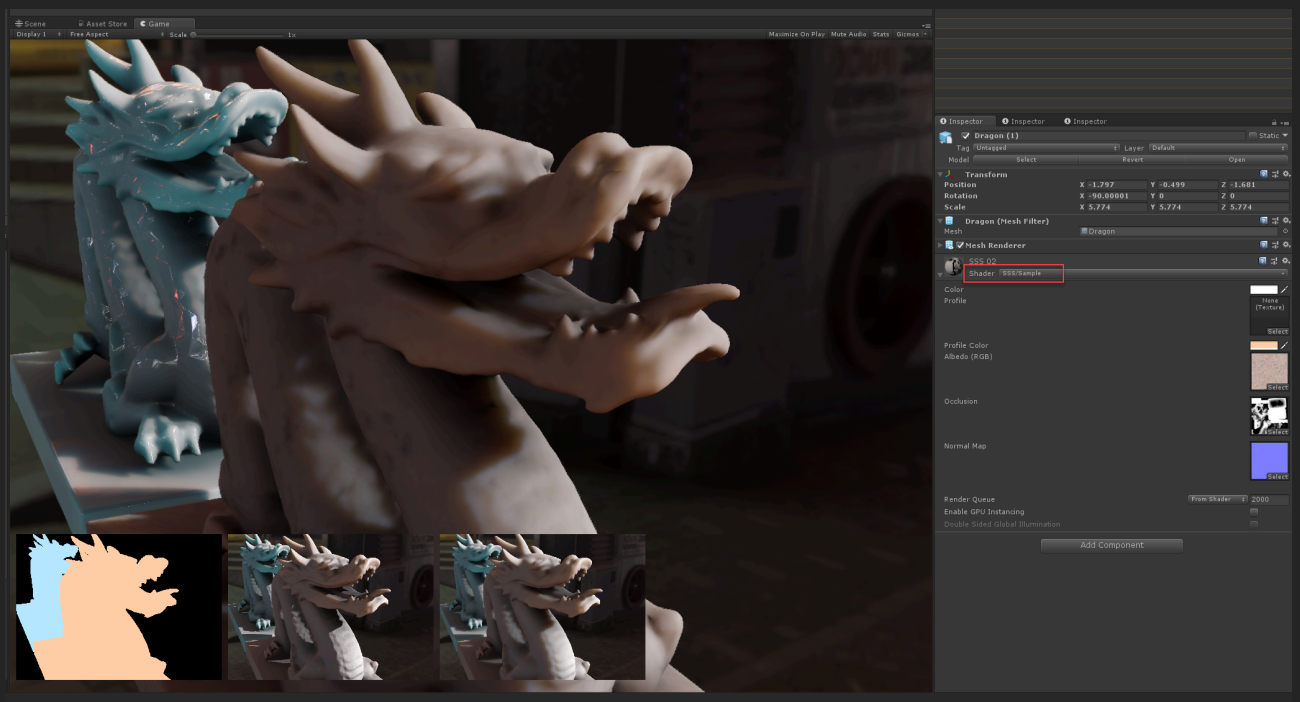


For shader artists

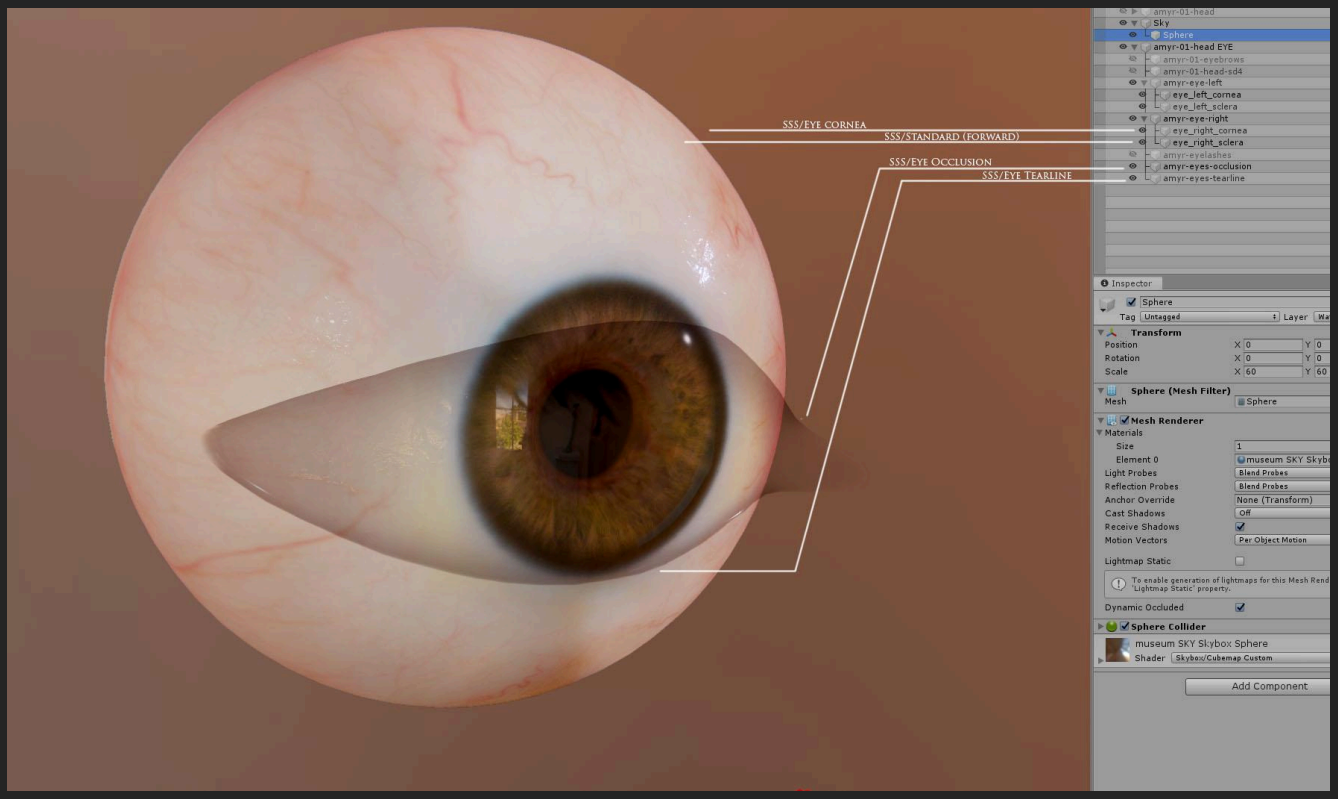
If you know how to work with shader replacement and write shaders, you can customize this plugin.
You can write your own base lighting shader and replace it here:



Then, you can make your own object shader to get anything you may need. A basic sample is included:



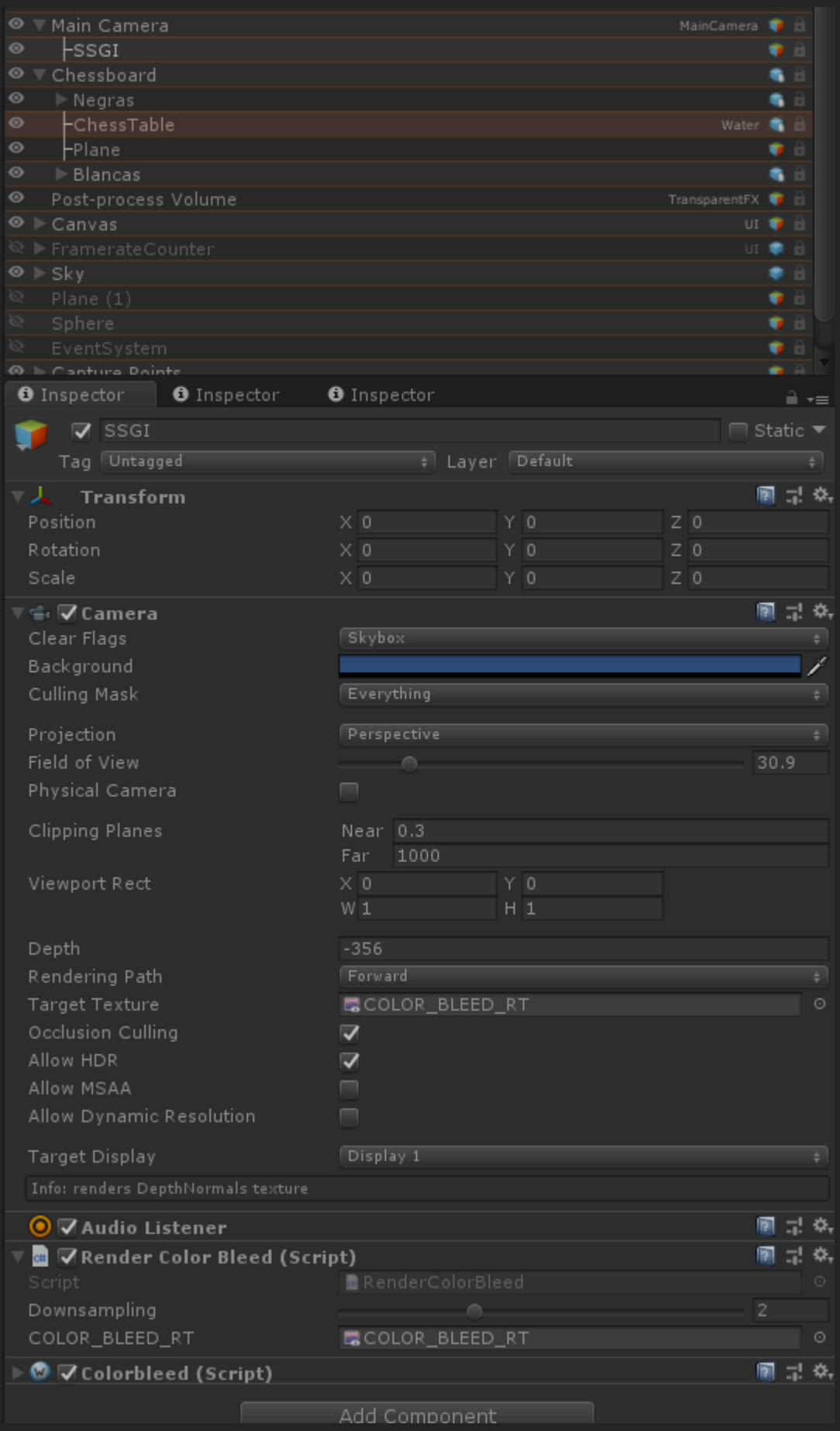
Eye parts

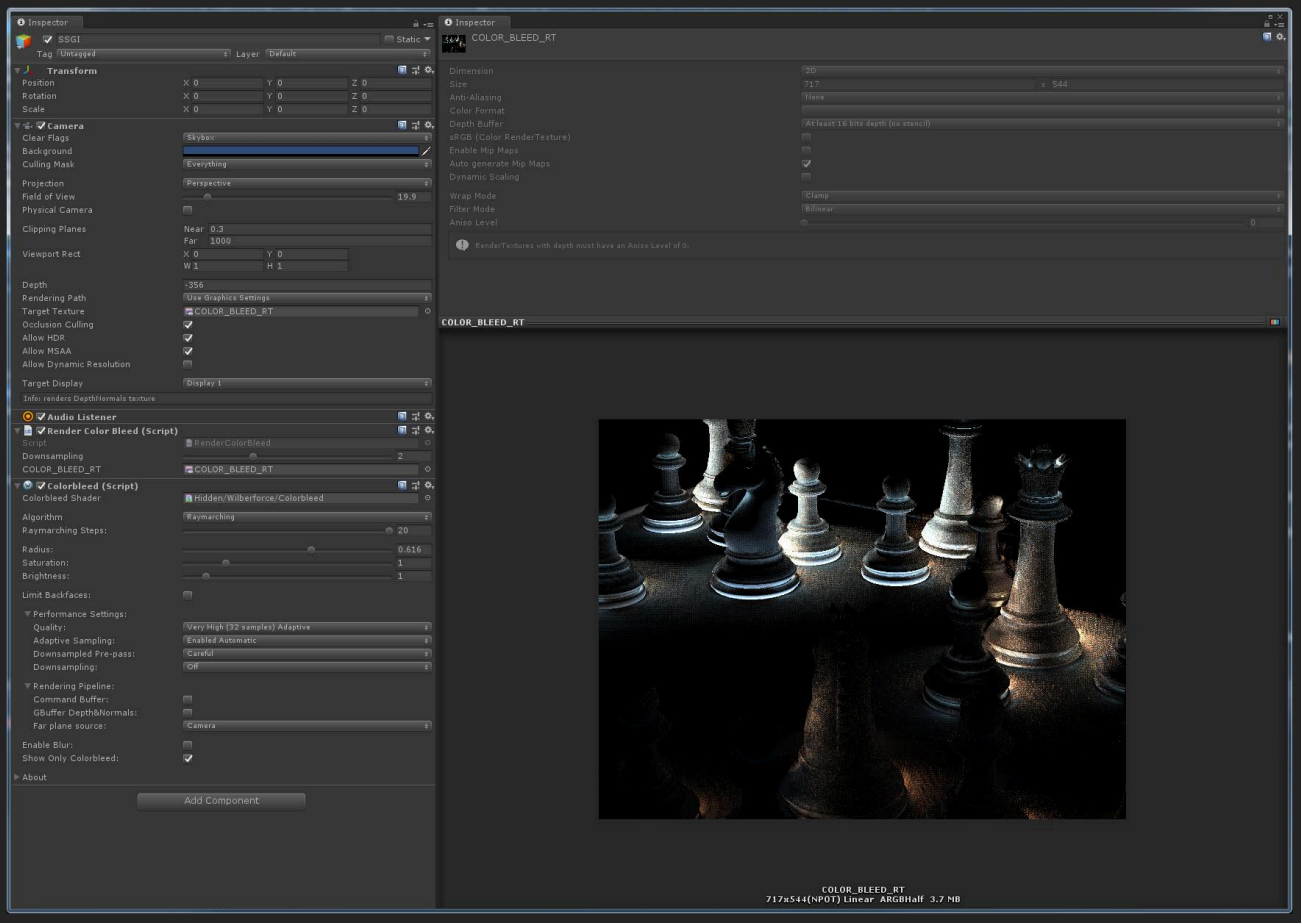


Customizing

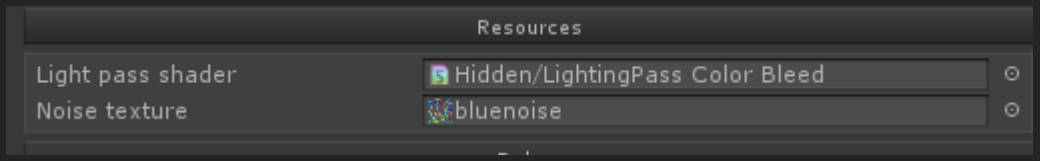
A good case that may require customization is to inject another buffer generated by another plugin. In my case with the Chess scene, I wanted to add a screen-space global illumination pass.

The plugin used is Colorbleed. First of all was to modify it to output a natural color. The author performed a bunch of color manipulations that in my opinion didn't make much sense. Then, to use it correctly , the buffer must be injected during the lighting pass, not on top of everything (as it is by default). So I had to inject it in the first pass of sss, so it gets also blurred with the rest of light contributions (diffuse & transmission).
The setup consists of a secondary camera attached to the main one, thus sharing the transform. ColorBleed.cs + my baker script.


Main Camera
MainCamera
SSGI
Chessboard
Negras
ChessTable
Water
Plane
Blancas
Post-process Volume
TransparentFX
Canvas
UI
FramerateCounter
UI
Sky
Plane (1)
Sphere
EventSystem
Inspector
Inspector
Inspector
SSGI
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
X 0
Y 0
Z 0
Camera
Clear Flags
Skybox
Background
Culling Mask
Everything
Projection
Perspective
Field of View
30.9
Physical Camera
Clipping Planes
Near 0.3
Far 1000
Viewport Rect
X 0
Y 0
W 1
H 1
Depth
-356
Rendering Path
Forward
Target Texture
COLOR_BLEED_RT
Occlusion Culling
Allow HDR
Allow MSAA
Allow Dynamic Resolution
Target Display
Display 1
Info: renders DepthNormals texture
Audio Listener
Render Color Bleed (Script)
Script
RenderColorBleed
Downsampling
2
COLOR_BLEED_RT
COLOR_BLEED_RT
Colorbleed (Script)
Add Component



RenderColorBleed.cs will store the result of ColorBleed in a new RT that is then sent to a modified version of LightingPass.shader:



Then, in the alternative shader *LightingPass* it is sampled like this:

```
#if defined(SS_COLOR_BLEED)
	float4 coords = 0;
	coords = UNITY_PROJ_COORD(IN.screenPos);
	coords.w += 1e-9f;
	float2 screenUV = coords.xy / coords.w;
	Emission +=tex2D(COLOR_BLEED_RT, screenUV).rgb * tex2D(_MainTex, uv * _AlbedoTile).rgb * _Color.rgb;
#endif
```

Changelog:

- 1.2
- 1.3
- 1.4
- 1.5
- 1.6
- 1.7
- 1.8