MadGoat

Super Sampling

1. INTRODUCTION

Thank you for purchasing our asset!

MadGoat Super Sampling brings one of the highest quality forms of anti-aliasing to Unity3D.

What is SSAA?

SSAA, short from **super sampling anti-aliasing** (not to be confused with screen space anti-aliasing) is one of the highest quality antialiasing methods, since it works by rendering the whole image at a higher resolution and down sampling it to the screen resolution.

Then why don't you see it in many games?

SSAA works by directly changing the internal render resolution of your game, therefore the higher the setting is, the higher the performance hit is. Because of that, it uses more resources and needs more processing power than other anti-aliasing options, but also looks better. **This AA solution is great for high end GPUs where there is performance headroom left, and also for high resolution, sharp screenshots.**

2. SETUP

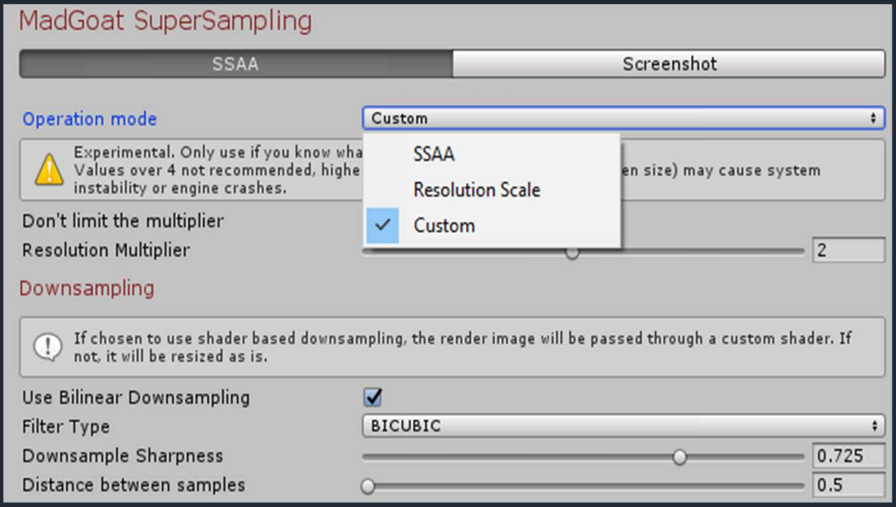
MadGoat Super Sampling is easy to integrate in your projects. Just drag and drop the main script on your camera and you are good to go. If you have multiple cameras, you can add the script to each one of them.

NOTE: For the best results, the SSAA can be used along with the Unity's SMAA, without a huge performance loss

2.1 THE SETTINGS

There are 3 operation modes that can be chosen in the inspector:

- SSAA: Works by using the typical SSAA options (x2, x4 and x0.5)
- Resolution scale: Scales the current screen resolution based on a percent (from 50% to 200%)
- Custom setting: Allows to setup a custom resolution scaling setting.


MadGoat SuperSampling

Operation mode: Custom

Experimental. Only use if you know wha… Values over 4 not recommended, highe… en size) may cause system instability or engine crashes.

Don't limit the multiplier
Resolution Multiplier: 2

Downsampling

If chosen to use shader based downsampling, the render image will be passed through a custom shader. If not, it will be resized as is.

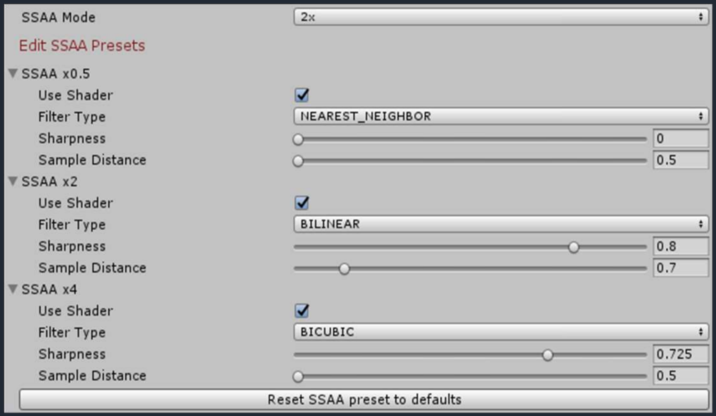
Use Bilinear Downsampling: ✓
Filter Type: BICUBIC
Downsample Sharpness: 0.725
Distance between samples: 0.5

2.2 CUSTOMIZING EACH MODE

Every operation mode has its own set of settings. These settings can be very important in achieving the best quality image.

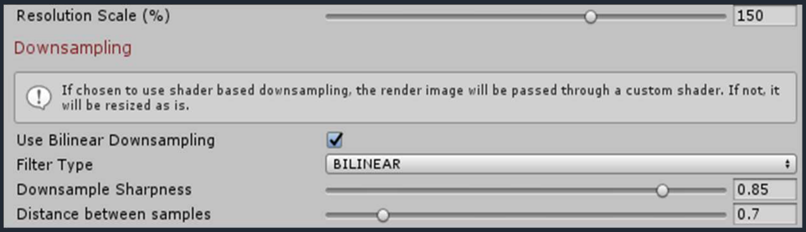
For SSAA mode, each profile can be configured independently.

- Use Shader: If enabled the final rendered image will be passed through a custom down sampler.
- Filter Type: The type of filter used to down sample the image.
- Sharpness: The sharpness of the down sampled image. (Only for Bilinear & Bicubic*)
- Sample Distance: Distance between the samples. (Only for Bilinear*)



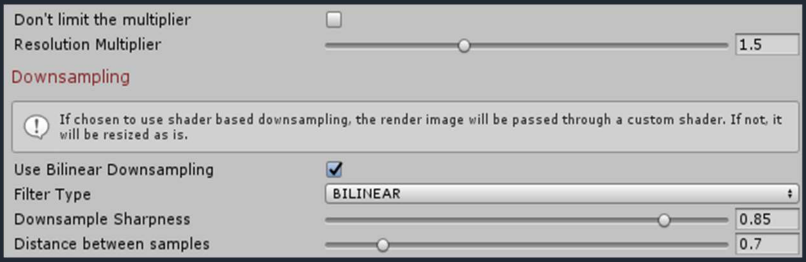
SSAA Mode 2x
Edit SSAA Presets
SSAA x0.5
Use Shader ✓
Filter Type NEAREST_NEIGHBOR
Sharpness 0
Sample Distance 0.5
SSAA x2
Use Shader ✓
Filter Type BILINEAR
Sharpness 0.8
Sample Distance 0.7
SSAA x4
Use Shader ✓
Filter Type BICUBIC
Sharpness 0.725
Sample Distance 0.5
Reset SSAA preset to defaults

The Resolution Scaling mode and the Custom Setting mode also have these settings, but in addition they have their own specific settings:



Resolution Scale (%) 150
Downsampling
If chosen to use shader based downsampling, the render image will be passed through a custom shader. If not, it will be resized as is.
Use Bilinear Downsampling ✓
Filter Type BILINEAR
Downsample Sharpness 0.85
Distance between samples 0.7

The resolution scale slider is used to set the size of the rendered image relative to the size of the screen. Options from 50% to 200% are available.



Don't limit the multiplier ☐
Resolution Multiplier 1.5
Downsampling
If chosen to use shader based downsampling, the render image will be passed through a custom shader. If not, it will be resized as is.
Use Bilinear Downsampling ✓
Filter Type BILINEAR
Downsample Sharpness 0.85
Distance between samples 0.7

On the Custom Setting mode, the Resolution Multiplier is the value with which the screen resolution is multiplied (From 0.2f to 4f). If you choose “Don’t limit the multiplier”, you can set any value. Be aware that values over 2 are not recommended.

2.3 THE FILTER TYPES

Nearest Neighbor

The simplest and crudest filtering method — it simply uses the color of the texel closest to the pixel center for the pixel color. While simple, this results in a large number of artifacts

When to use Nearest Neighbor: When looking to get a sharp image while rendering at a smaller resolution. This can also be used to achieve a pixelated effect on low resolution renders.

Bilinear Filtering

In this method four texels in the neighborhood of the center pixel are sampled, and their colors are combined by weighted average. This removes the artifacts seen while rendering at higher resolution, as there is now a smooth gradient of color change from one texel to the next, instead of an abrupt jump between the texels.

When to use Bilinear Filtering: When looking to get good down sampling quality without a big performance hit. It can also be used when rendering at lower resolutions to create a "smoother" image.

Bicubic Filtering

Bicubic filtering is often resulting in higher down sampling quality than bilinear filtering or nearest neighbor, but with the cost of slower computation. In contrast to bilinear filtering, which only takes 4 pixels into account, bicubic uses more samples in order to create a smooth curve based interpolation between the texels colors. Bicubic filtering also produces less image artifacts than the bilinear filtering.

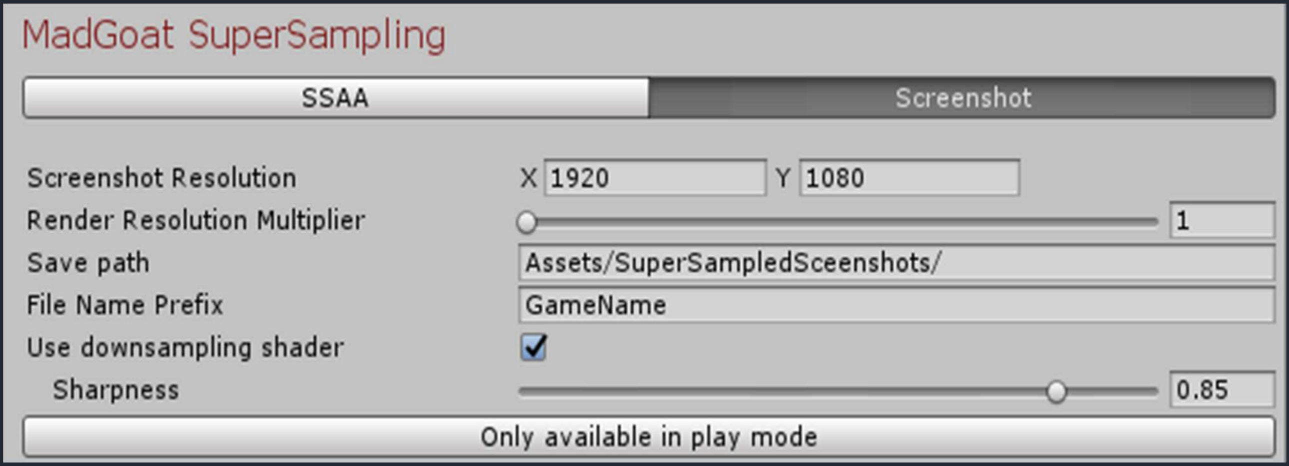
When to use Bicubic Filtering: When the best quality is desired and the performance is not an issue.

***The Sharpness shader setting is only available to Bilinear and Bicubic filtering, and the Sample Distance only to the Bilinear filtering, but due to the way in which the component's inspector's UI is implemented, they are shown for all three filtering methods. They just don't do anything when a filter mode that don't support them is used.**

3. THE SCREENSHOT MODULE

The screenshot Module can be added separately on the camera gameobject. It can take in-game screenshots at custom resolutions without the need to change the actual screen resolution and without affecting the aspect ratio.

*The screenshot module currently suffers from memory leaks at high resolutions or high multipliers, but it seems **its only affecting AMD video cards**. We are currently investigating the issue.


MadGoat SuperSampling

Screenshot Resolution	X 1920 Y 1080
Render Resolution Multiplier	1
Save path	Assets/SuperSampledSceenshots/
File Name Prefix	GameName
Use downsampling shader	✓
Sharpness	0.85

Screenshot specific settings can be seen in the picture above. These settings are only used when taking screenshot from the editor (by pressing the button on the module).

- Screenshot Resolution: The resolution of the taken screenshot.
- Render Resolution Multiplier: Gives the internal resolution at which the screenshot is taken
- Save Path: The path at which the screenshots are saved
- File Name Prefix: The prefix of the file
- And the shader sharpness. By default, the screenshot module is using bicubic filtering.

The screenshot module is also available in-game, and screenshots can be easily taken by calling just one function which we'll discuss in the API section.

4. CODE API

This section describes the several functions that can be used to change the parameters of the SSAA component from your own code. Those functions can be used in-game, for example for your settings menu code.

In order to gain access to those functions, you first have to reference the SSAA component from the camera you want to change. For example:

```
MadGoat_SSAA.MadGoatSSAA ssaa = GetComponent<MadGoat_SSAA.MadGoatSSAA>();
```

Take note that all of the asset's scripts are contained in the MadGoat_SSAA namespace.

4.1 The functions

1) `SetAsSSAA(SSAAMode mode)`

Sets the rendering mode to a given SSAA mode. Available SSAA modes as defined in the enum are SSAA_OFF, SSAA_HALF, SSAA_X2 and SSAA_X4.

2) `SetAsScale(int percent)`

Set the rendering mode to scale, and the resolution scale to a given percent. When using this function, the down sampling shader is disabled!

3) `SetAsScale(int percent, Filter FilterType, float sharpnessfactor, float sampledist)`

Set the rendering mode to scale, and the resolution scale to a given percent. When using this function, the down sampling shader is enabled. The **FilterType** parameter can be NEAREST_NEIGHBOR, BILINEAR and BICUBIC. The **sharpnessfactor** and the **sampledist** parameters are FilterType dependent as stated before.

WARNING: While it is possible to set any value to the percent, avoid values greater than 400%, as the renderer can cause engine instability at high resolutions and even crashes.

4) SetAsCustom(float Multiplier)

Set the rendering mode to custom, and the resolution multiplier to a given number. When using this function, the down sampling shader is disabled!

5) SetAsCustom(float Multiplier, Filter FilterType, float sharpnessfactor, float sampledist)

Set the rendering mode to custom, and the resolution multiplier to a given number. When using this function, the down sampling shader is enabled. As stated before, the SetAsScale function, the **sharpnessfactor** and **sampledist** parameters FilterType dependent, and they are clamped to correct values automatically.

WARNING: While it is possible to set any value to the Multiplier, avoid values greater than 4, as the renderer can cause engine instability at high resolutions and even crashes.

6) SetDownsamplingSettings(bool use)

Set the downsampling shader parameters. The use parameter specifies whether or not the down sampling shader should be used. If use = true, default settings will be given to the shader. It is important to know that this function should be always called after calling any of the functions above, since they always override the downsampler settings.

7) SetDownsamplingSettings(Filter FilterType, float sharpnessfactor, float sampledist)

Set the down sampling shader parameters. In this case the down sampling shader is always enabled. Values for the **FilterType**, **sharpnessfactor** and **sampledist** have to be given. As for the function above this function should be always called after calling any of the functions above.

4.2 The functions of Screenshot Module

There are 2 similar functions that can be used for taking screenshots:

1) TakeScreenshot(string path, Vector2 Size, int multiplier)

Take a screenshot of resolution Size (x is width; y is height) rendered at a higher resolution given by the multiplier. The screenshot is saved at the given path in PNG format.

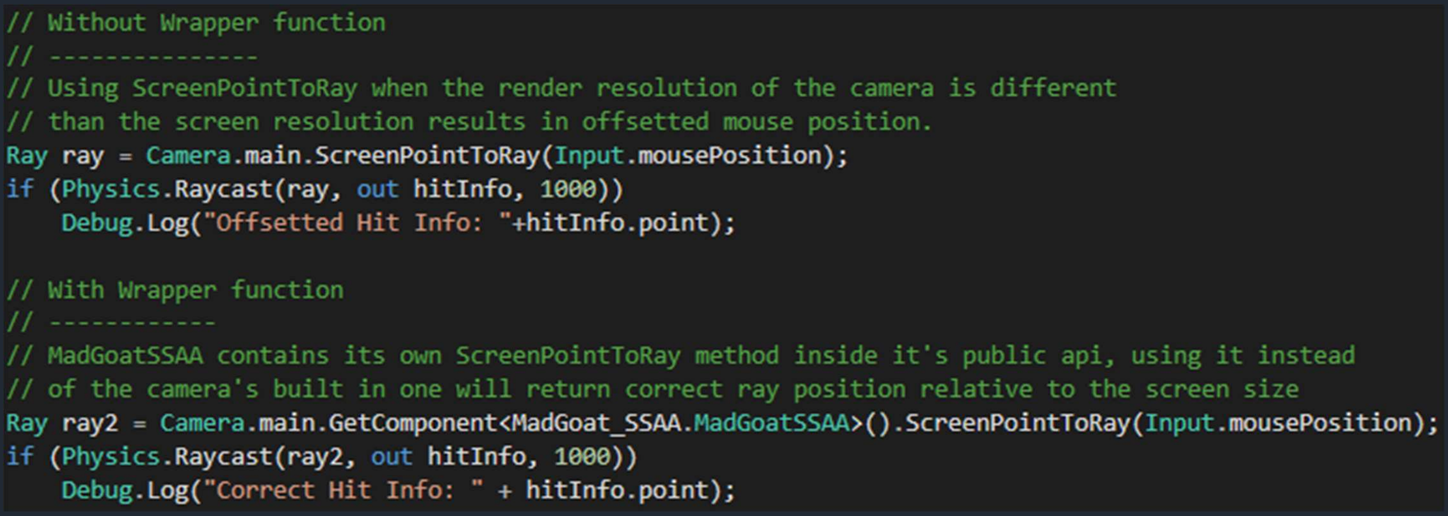
2) TakeScreenshot(string path, Vector2 Size, int multiplier, float sharpness)

Take a screenshot of resolution Size (x is width, y is height) rendered at a higher resolution given by the multiplier using bicubic filtering for down sampling, with a given sharpness. The screenshot is saved at the given path in PNG format.

4.3 Misc API functions

1) ScreenPointToRay(Vector3 Position)

This has to be used instead of Camera.ScreenPointToRay when SSAA is ON. Usage example and explanation:



```
// Without Wrapper function
// ---------------
// Using ScreenPointToRay when the render resolution of the camera is different
// than the screen resolution results in offsetted mouse position.
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
if (Physics.Raycast(ray, out hitInfo, 1000))
    Debug.Log("Offsetted Hit Info: "+hitInfo.point);

// With Wrapper function
// ------------
// MadGoatSSAA contains its own ScreenPointToRay method inside it's public api, using it instead
// of the camera's built in one will return correct ray position relative to the screen size
Ray ray2 = Camera.main.GetComponent<MadGoat_SSAA.MadGoatSSAA>().ScreenPointToRay(Input.mousePosition);
if (Physics.Raycast(ray2, out hitInfo, 1000))
    Debug.Log("Correct Hit Info: " + hitInfo.point);
```

5. CONTACT

For any issue or bug report regarding the asset, please contact us at:

Support Email: madgoatstudio@gmail.com

Other contact methods:

Website: www.MadGoat-Studio.com

Facebook: www.facebook.com/madgoatstudio/

Twitter: twitter.com/MadGoatGames