SpriteToParticles - Documentation

Current Version 1.2.2

Update v1.2.2

- Added Static Emission for UI
- Fixed Unity 5.5+ issues.
- Fixed UI some shader compiling errors (A “reimport all” in the project might be needed to fix this).

Update v1.2

- Added support for UI.
- Added 2 UI Demo Scenes.
- Added a new non UI Demo Scene.
- Added a few prefabs for faster development (to be used as blueprints).

UI specifications (be sure to read non UI documentation down below first).

Hi!

This small guide will walk you through the basics of using SpriteToParticles package also giving some insight on how the components are made. For complete variable and method documentation check here.

Important! The scripts need the default Unity Sprite Component as a base to get the texture information from. So basically, no sprite component, no particle emission from sprite.
Also for the scripts to work **Read/Write** option must be **enabled** in the texture Import Setting (Texture Type must be set to advance to modify this setting) **By doing this Unity duplicates the texture in memory**, adding a memory overhead. That said, if your program is short on memory I would think twice before using this for big images.

Important 2! Particles are particles, they cost memory and processing power. Don’t get too greedy.

How to Use

Create a GameObject, assign a SpriteComponent and a Shuryken ParticleSystem to it. Then add any of the 3 provided components. Press Play and voilá! Particles emitted from the sprite.
Be sure to check the provided demo scenes.

So, how does it work?

The components will emit particles from the assigned Sprite primarily based on sprite pixels’ position. It can also emit from a specific color in the sprite. Pixels with alpha channel set to 0 will be omitted.

Why is it separated in 3 scripts?

It's a matter of performance and memory consumption. If one is not going to change the sprite image all the time it's better to cache all possible data the system uses beforehand, although this would mean more memory consumption. That said, StaticEmitterContinuous is far better performant than DynamicEmitter but at the cost of... well, dynamicness.
Check **Components Description** for more information on this.

ParticleSystem's Setting overridden by the scripts

The scripts needs to override some Shuryken variables and modules:

Looping, Prewarm, Play On Awake.
The Emission and Shape modules are also overridden.
Start Color will also be overridden if the emitter component is set to "Use Pixel Source Color".

Components Description

The package contains 3 main component you can use: DynamicEmitter, StaticEmitterContinuous and StaticEmitterOneShot.

All components extends from **EmitterBase**, a MonoBehaviour class, which works as a base defining all common methods and variables.

StaticEmitterContinuous (extends from **SpriteParticleEmitter**)

Based on cached Sprite pixels positions, the component will emit particles continuously. Emission Rate works as the one in Shuryken Emission module.
It may be used for backgrounds or animations where you only move the GameObject around but it doesn't change the base sprite.
Using the Particle System Simulation space as Local will be better performant as less computations needs to be made by the script.

Memory consumption:
The script will cache as much information as possible for performance reasons:
Worst case allocation scenario:

```
sprite Width * sprite Height * ((Color memSize) + (Vector3 memSize))
```

Example: 256x256 fully color texture (Worst case scenario):

```
256*256 * (16 + 12) = 1.75mb
1024*1024 * (16 + 12) = 28mb
```

StaticEmitterOneShot (extends from **SpriteParticleEmitter**)

This component will emit particles based on all pixels in a Sprite. It's intended to use for large sprites. Bear in mind that 512x512 sprite will potentially emit 262.144 particles (which is a lot).
The component emits particle by particle with Shuryken ParticleSystem's method Emit() (https://docs.unity3d.com/ScriptReference/ParticleSystem.Emit.html)
Emitting this amount of particles in a single frame would take too much time meaning the frames per second in the game would drop. To circumvent this issue, particles are emitted silently over the course of needed frames.
The variable WantedFPSDuringSilentEmission defines the minimum of fps you want your application to drop to during the silent emission process. The component will ensure the game will not drop below that number.
The main reason of being for this component is to use https://docs.unity3d.com/ScriptReference/ParticleSystem.GetParticles.html after all the particles have been generated. In the Oneshot Demo Scene (HTML5 icon) is the Unity wind the one that moves them.

Memory consumption:
Same as **StaticEmitterContinuous**

DynamicEmitter

Works as the StaticEmitterContinuous but has the ability to change the Sprite Component image on the fly.
It's the component you should use if you are animating the Unity's Sprite Component thus changing the sprite completely
It enables the possibility to cache each of the texture's sprites data to minimize GC collection, finally it would be the developer's decision to deal with more memory consumption or GC collection.

SpriteParticleEmitter

It caches Sprite's pixel position and color for later emission. Works as a base for the components **StaticEmitterOneShot** and **StaticEmitterContinuous**.

Note: EmitterBase and **SpriteParticleEmitter** are not supposed to be directly assigned to a GameObject, only their derivatives.

UI Specifications

Currently only the DymamicEmitter has it's port to UI. The StaticEmitter will come later on.
Important! Sliced Images are not supported at the moment.

The new DymamicEmitterUI needs to be in a separate UI GameObject. This is because it needs its own renderer and there can not be 2 renderers in the same UI object. Which leads to,

ParticleSystem's Modules overridden by the scripts

Besides the original overridden modules DymamicEmitterUI overrides the ParticleSystem's Renderer component. It is replaced with a new renderer component based on Glenn Powell's (glennpow) script that can be found here (basically the same that UI extensions framework uses)
This component is named UIParticleRenderer and it's added automatically when DynamicEmitterUI is added to a UI GameObject.
Max Particles limit is 14.000 (check performance specs down below)

Shaders

The UI particles need specifics shaders to work properly. With small modifications almost any shader could be used.
The most common particle shaders are already added in a separate folder, these are:

- Add
- Add Multiply
- Add Smooth
- Alpha Blend
- Blend
- Multiply
- Multiply Double
- Premultiply Blend
- VertexLit Blended

Performance

The UnityUI is not designed to run the Shuryken particle system or any large amount of images. So there's a cap for the amount of particles it can run within an acceptable frame rate. Cool effects can be made as shown on the Demo Scene but don't expect having several particle systems with 10.000 particles each running at 60fps.
Mobile has a tighten cap. Beware.
Memory alloc is 128 bytes. This is due to using ParticleSystem.Simulate()
Whenever possible one should use the non UI components instead (example: menu backgrounds).
Info: The overridden renderer component (UIParticleRenderer) is built with the VertexHelper class. This utility class was created by Unity for this kind of situations. It works on a mesh modifying it every frame, so it's slow when dealing with large vertex assignations.

Information for components' variables and methods can be found in the code's Doxygen Documentation here.

Sprites used in the demos

Pixel art assets used in Dynamic Demos by
http://opengameart.org/users/calciumtrice
Using license:
https://creativecommons.org/licenses/by/3.0/

Gunner and Floor assets used in Static Shooter Demo by
http://opengameart.org/users/tatermand
Using license:
https://creativecommons.org/licenses/by-sa/3.0/

Have a question? Need help?
info@numbloq.com

Thanks for choosing **SpriteToParticles**. Happy particling!