SpriteToParticles - Documentation

Current Version 2.0

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

Important!

The component uses Unity's Sprite Component or Unity's Image Component.
Toolkit 2D and similar renderers are not supported at the moment.
Sprite Component's draw modes Sliced and Tiled are not supported at the moment.
Image Component's images type Sliced, Tiled and Filled are not supported at the moment.

Also, **Read/Write** option must be **enabled** in Texture Import Setting for StP to work (Texture Type must be set to advance to modify this setting) **By doing this Unity duplicates the texture in memory**, adding a memory overhead. More info here.

Version 2.0 has a complete code rework so If you are using previous please go to this section.

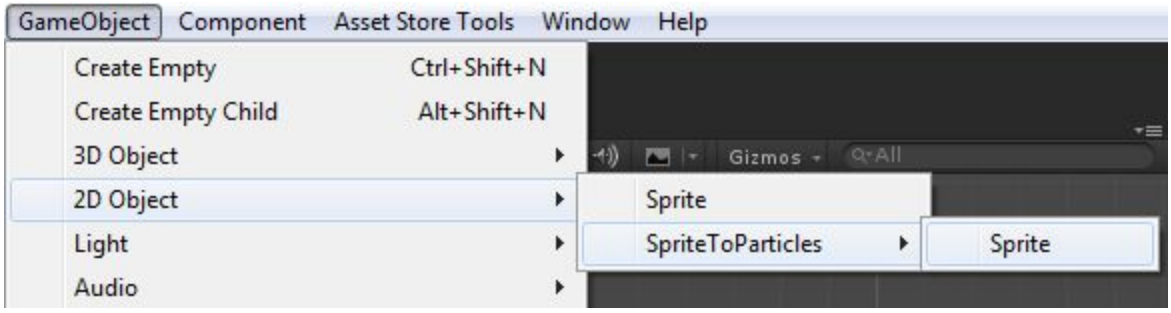
Version 2.1 and Unity 2017.1+

Effects made with StP in a 5.6 version **may** get somehow corrupted when updating to 2017.1+ version.
This **only happens** if the StP GameObject is a child of a complex hierarchy.
The **fix** for this is to remade the StP Gameobject from scratch. This means creating a new GO and adding the ParticleSystem and StP components and copying its setting by hand.
I believe this issue happens because of an internal position for the ParticleSystem is not properly updated but this is just a guess.
I'll try to make a reproducible project for the unity team to check that out.

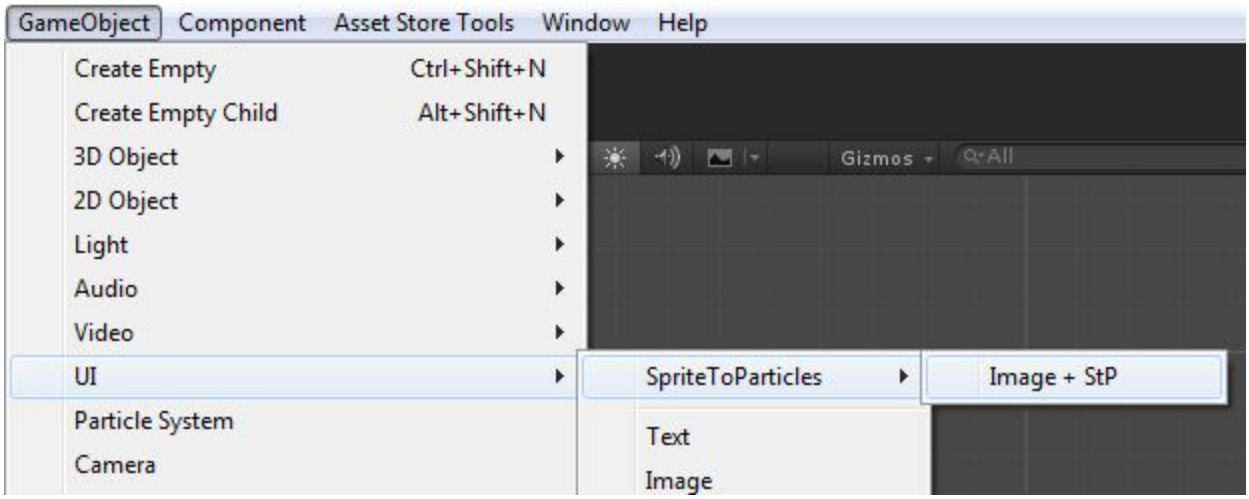
How to Use

Use Unity's menus to create a GameObject with all the components needed assigned.

For Sprites



For Images (UI)



Manually

For Sprites
Create a Sprite GameObject, then add a Particle System and the SpriteToParticles component.

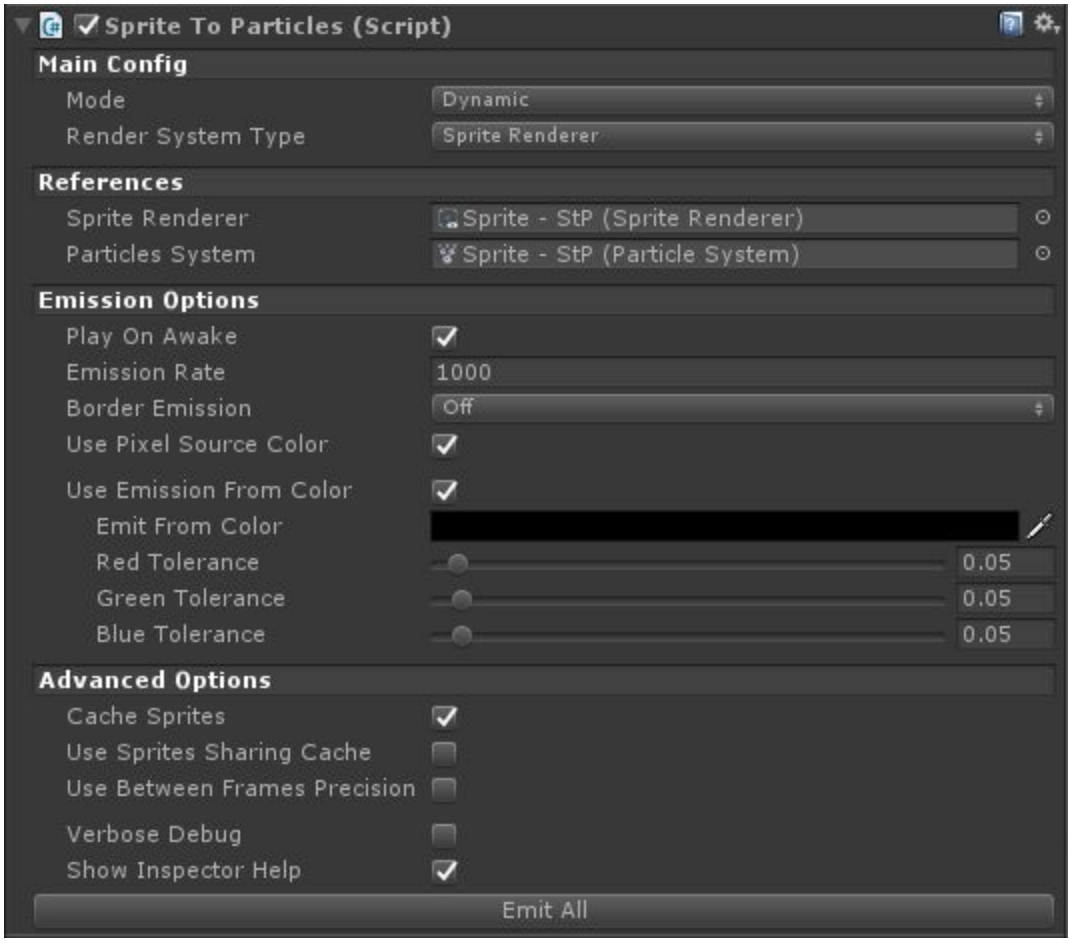
For UI
Create an empty GameObject under any Canvas, then add a Particle System, the UIParticleRenderer component and the SpriteToParticles component. Assign any GameObject in the Canvas with an Image component in the UI to the Image reference in the SpriteToParticles component.

Note: Be sure to check the provided demo scenes.

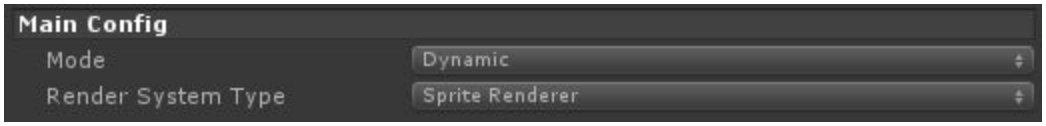
So, how does it work?

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

StP Inspector



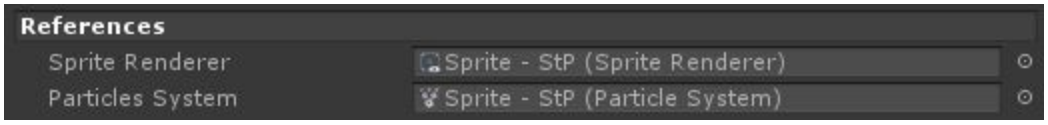
Main Config



Mode (Dynamic/Static)
Use Dynamic if the Sprite or Emission options will be changing during runtime. Example: using an Animator to change the sprite.
Otherwise use Static.

Render System Type (Sprite Renderer / Image Renderer)
Use Sprite Renderer when dealing with Sprites.
Use Image Renderer when dealing with Images, Buttons, etc on a Canvas UI.

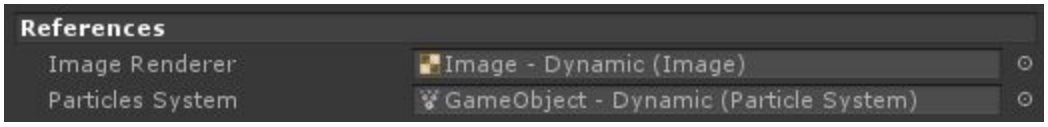
References



Sprite Renderer
Drag here the Sprite component to be used as source for emission. If none is set the system will try to find it the same GameObject the StP component is in.

Particle System
Drag here the Particle System component to be used in emission. If none is set the system will try to find it the same GameObject the StP component is in.

References (UI)

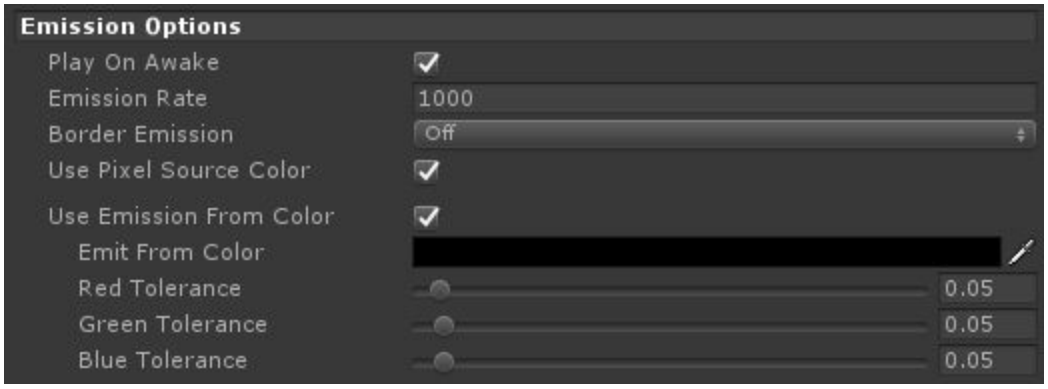


When using the Render System Type as Image this module will show instead of the above.

Image Renderer
Drag here the Image component to be used as source for emission. This could be any GameObject under a UI Canvas that has an Image Component attached (Image, Button, Panel, etc).
This Image component needs to be in a different GameObject than the one the StP component is in to work.

Particle System
Drag here the Particle System component to be used in emission. If none is set the system will try to find it the same GameObject the StP component is in.

Emission Options



Play On Awake
Whether the system will play on GameObject Awake.
Same as PaticleSystem's PlayOnAwake option.

Emission Rate
Particles amount emission per second.
Same as PaticleSystem's Emission Rate option.

Border Emission (Off / Fast / Precise)
Whether particles start position should be in the border of the Sprite.
Fast will only work on X axis, enough for rounded sprites.
Precise will work on both axis but is more performance expensive.
(Option not available in Static mode at the moment)

Use Pixel Source Color
Whether particles color should match the pixel color emitting from.

Use Emission From Color
Whether particles should only emit from specified color.

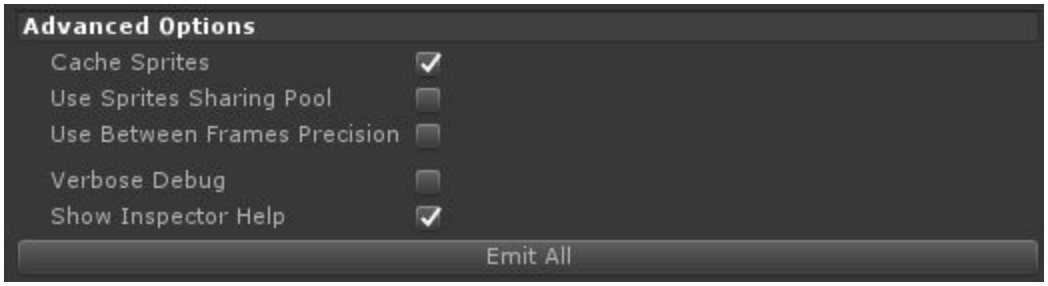
Renderer in different GameObject specifics

Only available when using the Renderer component is in a different GameObject than the StP component.

Match Target GO Position Data
For UI enabling this will set this RectTransform's position and rotation (the one StP component is on) as the same as the referenced Image component's Recttransform in the References module.
StP Object must have same parent as the target's Image component Transform.
For Sprites this will copy the target's transform's positions to the StP component position.

Match Image Render Scale
For UI enabling this will set this RectTransform's scale (the one StP component is on) as the same as the referenced Image component's Recttransform in the References module.
StP Object must have same parent as the target's Image component Transform.
For Sprites this will copy the target's transform's scale to the StP component scale.

Advanced Options



Cache Sprites (Only visible in dynamic mode)
Enabling this will grand better performance when animating a Sprite component but will increase memory consumption.

Cache On Awake (Only visible in Static mode)
Static mode works by caching the Sprite before hand. Enabling this will make that process run on the GameObject's Awake instance. If **Play On Awake** is enabled this option will be forced on.

Use Sprites Sharing Pool
Will share the sprites between all StP objects thus consuming less memory. Use this if there're a lot of same Sprites for StP components (example: 5 same enemies).
More info.

Use Between Frames Precision
Enabling this will allow particles to have a more precise position emission between frames by interpolating the last and current GameObject position.
Enable this if your GameObject is moving fast enough to see the undesired effect.

Verbose Debug
Enable this to get Console Debugs.

Show Inspector Help
Enabling this will show warning signs inside the StP inspector.

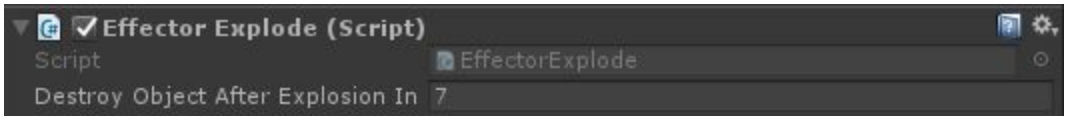
Emit All (Button)
Will trigger the EmitAll() in the selected StP. Useful for testing an effect.

Reset Cache (Button)
Will remove all stored data for the system. Useful when changing sprites on static mode.

Effectors

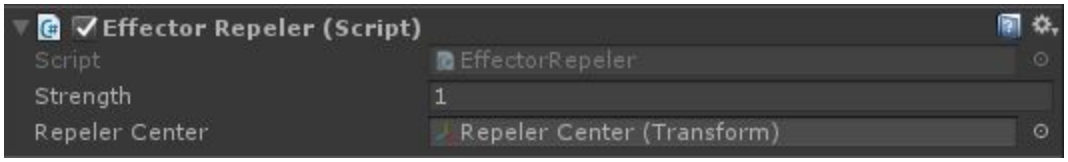
Effector are scripts that handle particles behaviour after StP has emitted them.
Currently there are 2 effectors.

Explode
A one time effect that will make a Sprite explode based on some parameters. Check code documentation to see how to launch it.
Check the **Effector - Explode** Scene.



Destroy Object After Explosion in
Defines how much time left to destroy the GameObject after the call to ExplodeAt() method.
A negative value will not destroy the GameObject.

Repeler
A constant effect over the emitted particles repelling them from a wanted Transform.
Check the **Effector - Repeler** Scene.



Strength
Defines the Repeler force intensity. A negative strength will attract particles instead of repelling them.

Repeler Center
Transform at which the particles will repel from. If none is set it will use the current Sprite position.

More effectors to come!

ParticleSystem's settings and modules overridden

The script needs to override some Shuryken variables and modules for the emission to work, these are:

For Sprite mode

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".

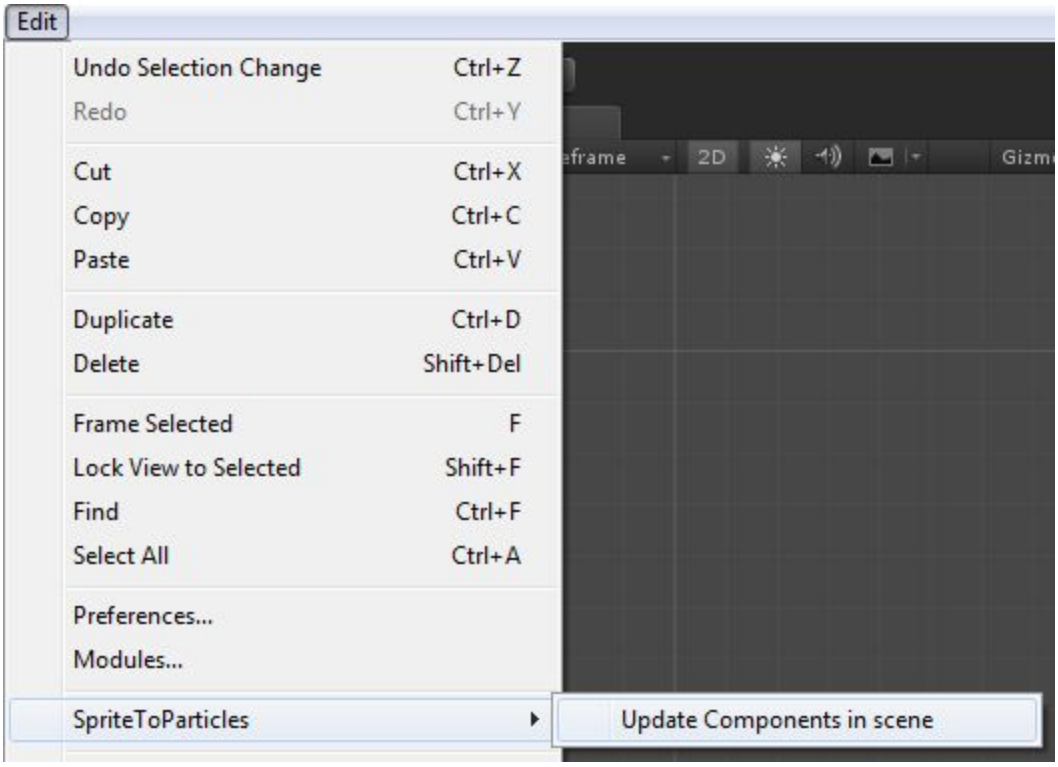
For Image mode (UI)

Looping, Prewarm, Play On Awake.
The Emission, Shape and Renderer modules.
The Renderer module 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 needs to be added for the UI version to work.
Max Particles limit is 14.000 (check notes on performance)

Updating to Version 2

A lot changed from last version 1.2.2 to current version.
All previous scripts, except for UIParticleRenderer, have been rendered obsolete.
For user's sake the asset now works on a single script, SpriteToParticles.cs.

Updating all scripts by hand would be a tedious task so there's Unity menu item to do this automatically.



The task will replace all old scripts in the current opened scene with the new script maintaining the old script setting for the effect.

Remember to always back up your projects before updating.

Notes on Performance and Memory Consumption

The StP code does two important things which, depending in the size of the texture, might be expensive:

1 - There's reading of the texture from unity, the call to GetPixels method on the texture (https://docs.unity3d.com/ScriptReference/Texture2D.GetPixels.html)
2 - There's the reading from the Color[] array returned by the above method.

Depending the mode used by the scripts the processing of this data will be hadled differently:

Static mode will do 1 and 2 only once. This is done in the CacheSprite() method.
For big textures one would do that on the Loading state of the game (using the LoadingTimeSpritesPool component), using the **Cache On Awake** on the Advanced module or calling it manually at a prefered moment.
Depending on the texture/sprite size the CacheSprite() method could take some milliseconds.
After 1 is done the static mode saves the info recollected in 2 and free the memory used in 1.

So, in worst case allocation scenario the script will save:

```
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
```

After CacheSprite() (there's an event **OnCacheEnded** to attach to) the performance will only depend on the amount of particles your are emitting. It doesn't matter if you have a 16x16 texture or a 4096x4096 texture.

Dynamic mode will do 1 only once if the **Cache Sprites** setting is enabled. This is per different texture referenced in the Sprite component or Image component.

If **Cache Sprites** setting is disabled it will do 1 every frame.

If **Use Sprites Sharing Pool** is enabled the caching of the sprites will be handled by the sharing pool object instead both in static and dynamic modes. Bear in mind 1 will remain in memory for static mode if this option is enabled.

Dynamic mode will always do 2 every frame; which is the cost of having the ability to change texture and/or the color source emission settings on the fly.

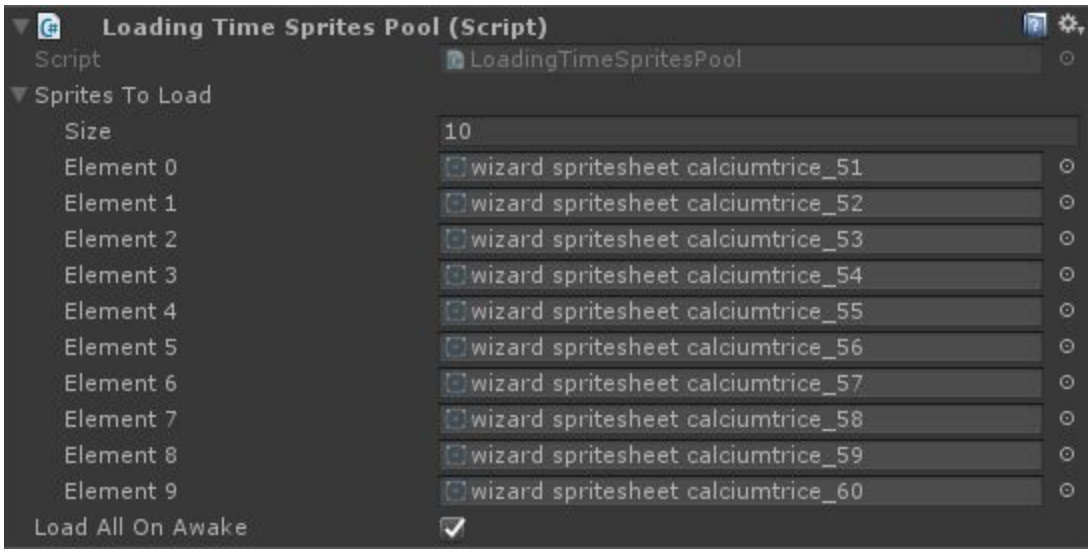
UI mode

The Unity UI 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 Scenes 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 mode 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 (this is large amounts of particles).

Loading Time Sprites Pool



Use this component if you want to load the sprites at a particular moment in the game. For the StP component to work in conjunction with this the **Use Sprites Sharing Pool** option must be enabled.

Sprites To Load
Drag here all the sprites to be loaded in the pool.

Load All On Awake
If enabled the load will be called on this GameObject's Awake method.
Otherwise it can be called by the method LoadAll().

Check the **LoadingTimeSpritePool Example** scene.

Versions History Changes

Update v2.1

- Fixed a bug that crashes the Editor on Unity versions 2017.x (more info: https://issuetracker.unity3d.com/issues/unity-crashes-when-particles-are-awaken-by-script)
- Fixed a bug that sets the particles system's initial position at (0,0,0) (Again a bug introduced in Unity 2017.x)
- Added a "Reset Cache" button for Editor use.

Update v2.0

- Completely reworked system.
- New Effectors post particle system's emission.
- New Inspector that helps configuration.
- New added precision between frames option.
- New Border emission option (for dynamic emission only atm).
- New contextual menu options for faster development.
- Added execute in edit mode. Now changes are visible in edit mode.
- Added more example scenes.
- Bugs fixing and warnings removal.

Update v1.2.2

- Added Static Emission for UI
- Fixed Unity 5.5+ issues.
- Fixed UI some shader compiling errors.

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).

StP's future

The next features will be delayed after Unity 2018 goes live because of the remarcable future changes in the editor.

- Add more Effectors.
- Add more Example scenes.
- Research TextMeshPro support.
- Research for adding Sliced and Tiled rendering modes.
- Add border emission for static mode.

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!