F2U's Sprite Batching

Wednesday, September 04, 2013

Table of Contents

Abstract

This document explains the details regarding F2U's Sprite batching functionality, the document starts with an explanation of the batching systems provided by Unity3D and then explains the custom Sprite batching provided by F2U.

Introduction

F2U has a custom implementation of sprites (F2USprite).
F2USprite is a script that renders a texture sprite in the scene, the rendering performance of F2USprites can be improved using batching.
Batching is a system to improve the rendering performance, please read the documentation of the Built-in batching provided by Unity3D (“http://docs.Unity3D3d.com/Documentation/Manual/DrawCallBatching.html”).
As documented in previous link Unity3D provides two modes of Built-in drawcall batching “Dynamic Batching” and “Static Batching”.
Note that Unity3D Built-in drawcall dynamic batching can be applied to F2USprites with the some limitations described in the Unity3D documentation.

General limitations of the Built-in Drawcall Batching provided by Unity3D:

- Only objects sharing the same material can be batched together.
- Combining geometry in the modeling tool, prevents efficient culling and results in much higher amount of geometry being rendered.
- Generally, objects should be using the same transform scale.
- Using different material instances - even if they are essentially the same - will make objects not batched together.
- Generally dynamic lightmapped objects should point to exactly the same lightmap location to be batched.
- Multi-pass shaders will break batching. Almost all Unity3D shaders support several lights in forward rendering, effectively doing additional pass for them. The draw calls for "additional per-pixel lights" will not be batched.
- Objects that receive real-time shadows will not be batched.
- Semitransparent shaders most often require objects to be rendered in back-to-front order for transparency to work. Unity3D first orders objects in this order, and then tries to batch them - but because the order must be strictly satisfied, this often means less batching can be achieved than with opaque objects.

Limitations of “Dynamic Batching” provided by Unity3D:

- Batching dynamic objects has certain overhead **per vertex.**
- Batching is applied only to meshes containing less than **900** vertex attributes in total. The exception is non-uniform scaled objects; if several objects all have different non-uniform scale then they can still be batched.

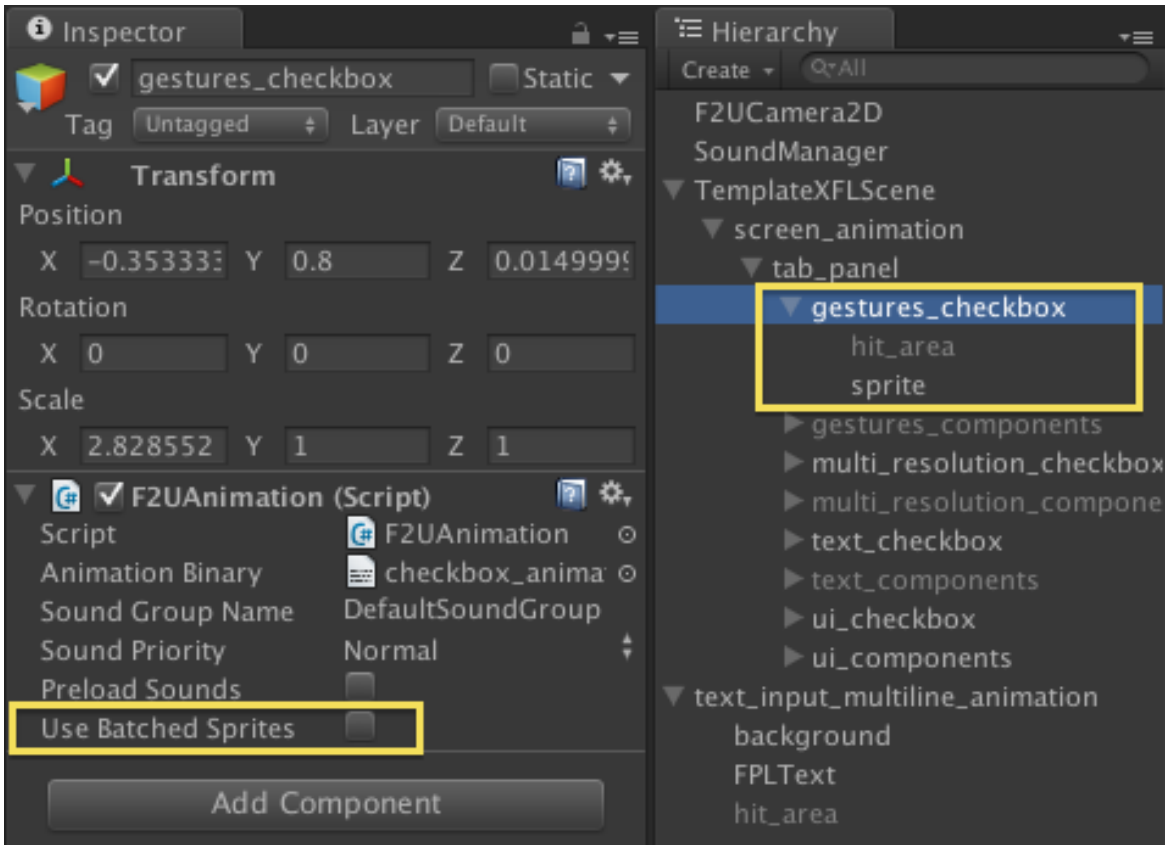
F2U’s Custom Sprite Batching

F2U has a custom implementation of batching to batch Animation’s Sprites, by default F2UAnimation has the “Use Batched Sprites” property disabled, so every child

F2USprite of the animation will be a game object that renders its Sprite (F2USpriteMesh).

F2USpriteMesh is a logic class “Not a Unity script” (not visible in the scene) that holds the information to render a Texture’s Sprite.
F2USprite is a Unity script that renders a F2USpriteMesh.
An F2UAnimation with “Use Batched Sprites” disabled during Unity play mode.
The “gestures_checkbox” F2UAnimation has two F2USprites “hit_area” and “sprite”, both are game objects that render two different F2USpriteMesh.

F2USpriteBatchingManager is a Unity script that manages the F2USpriteMesh batching,



it is dynamically created when Unity enters to play mode, it creates children F2USpriteBatchers in order to render a list of batched F2USpriteMesh.

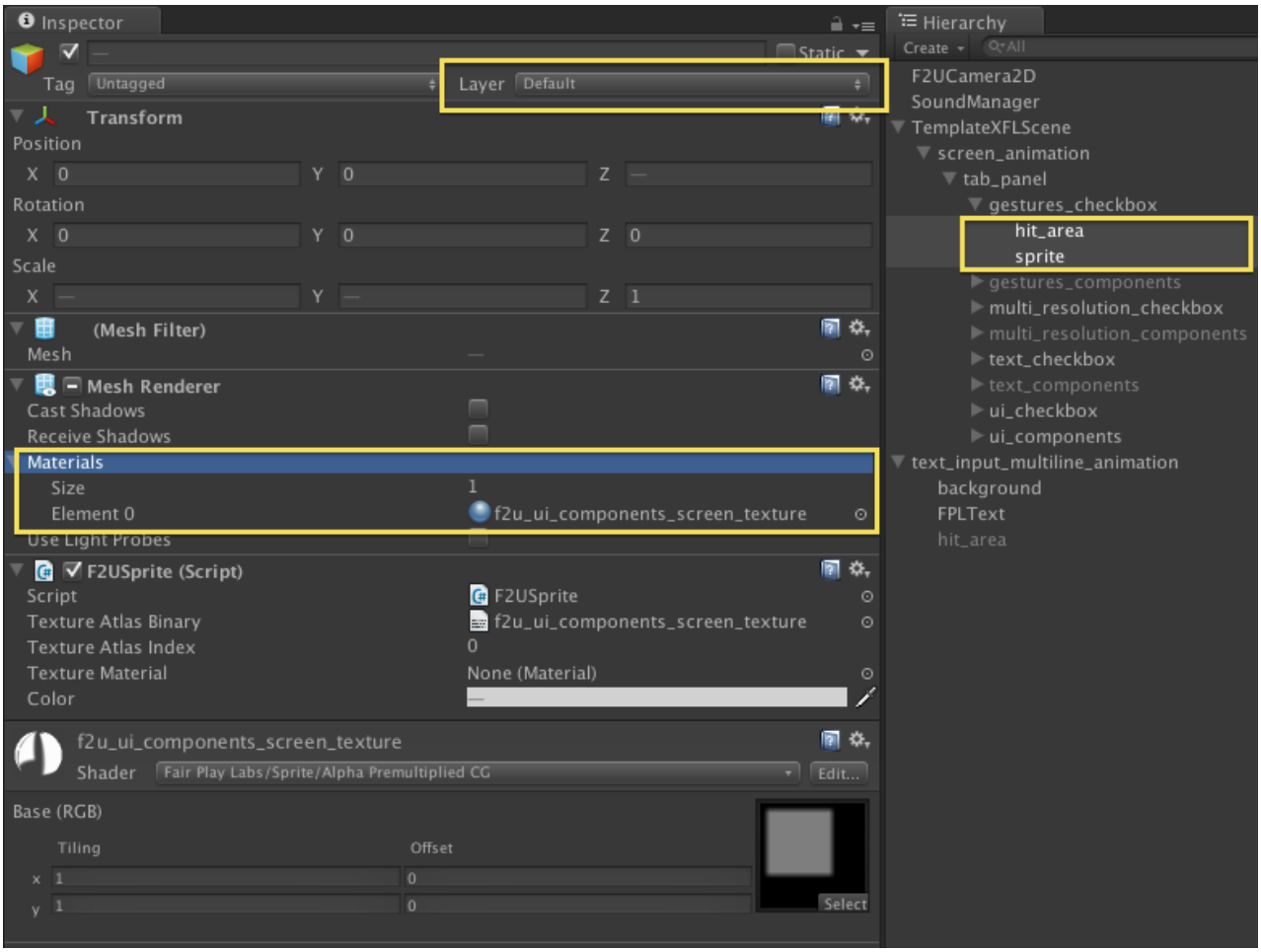
F2USpriteBatchingManager uses two criteria to batch the F2USprites of the F2UAnimations:

Material of the F2USprites.

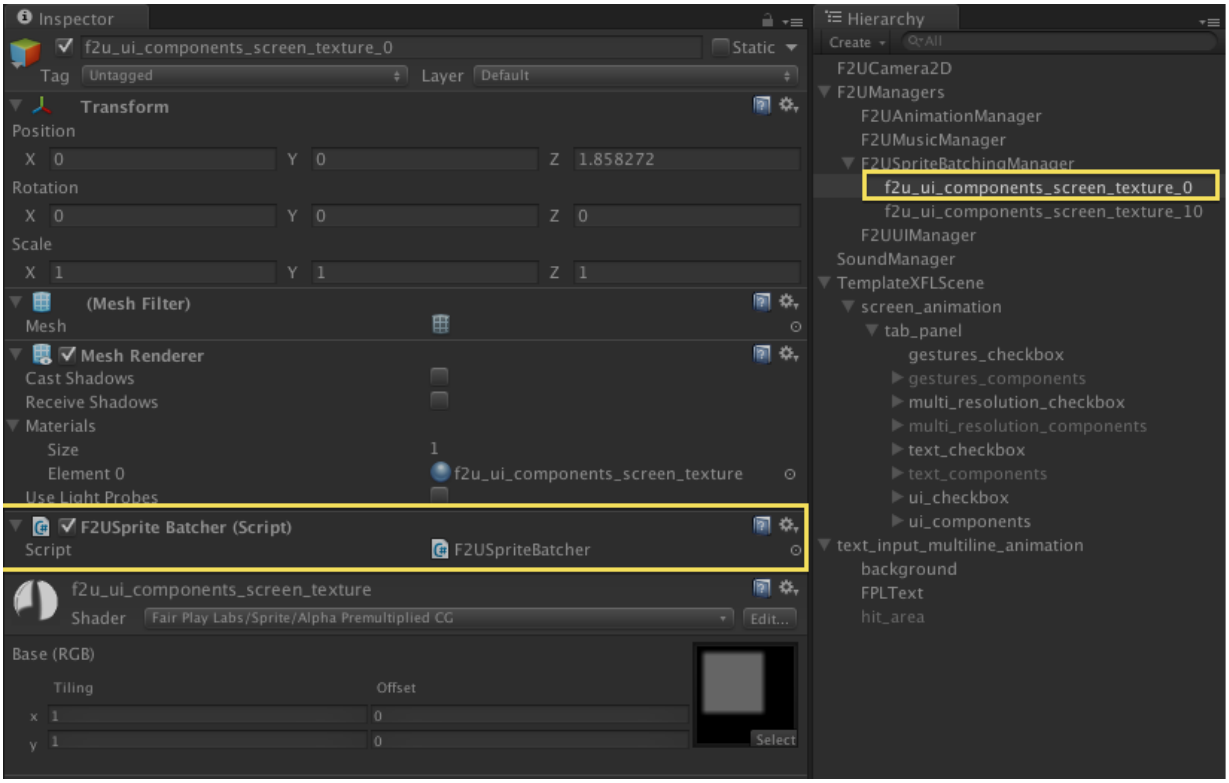
Layer of the F2UAnimation.

Two selected F2USprites “hit_area” and “sprite”, the batching criteria are marked with yellow.

When the “Use Batched Sprites” is set enabled the F2USprites of a F2UAnimation will exist during the Unity3D edit mode, once Unity3D enters to play mode the F2USprites of a F2UAnimation will be destroyed and the F2USpriteMeshs will be rendered by a



F2USpriteBatchingManager dynamically created at runtime.



A F2UAnimation with “Use Batched Sprites” enabled during Unity play mode, the F2USpriteBatchingManager draws the “hit_area” and “sprite” children of “gestures_checkbox” with the “f2u_ui_components_screen_texture_0” F2USpriteBatcher.

F2USpriteBatcher renders F2USpriteMeshs of the F2UAnimations in a single draw call.

A F2USpriteBatcher selected during Unity play mode.
The Unity3D “Default” layer is represented in code by the integer “0” so that’s the reason why the F2USpriteBatchingManager has a game object with a F2USpriteBatcher named “f2u_ui_components_screen_texture_0”.

Limitations of F2U's Custom Sprite Batching:

- All batched objects in a single F2USpriteBatcher are rendered in the same z from camera (single draw call), so what F2USpriteBatcher does is calculate the average z of the batched sprites to render all the batched sprites in the average z.
- Any change in the hierarchy of transformations made from code is not updated automatically, that's because Unity3D doesn't report any event to detect that hierarchy transformations of a Transform have been changed (and detecting it by pooling is expensive), so in order to refresh any hierarchy change on the sprites it's necessary to call "F2UUIUtils.UpdateHierarchyAttributeChangedRecursively".

Examples:

1- HierarchyAttribute.Visibility:

```
transform.gameObject.SetActive(false);

F2UUIUtils.UpdateHierarchyAttributeChangedRecursively(transform,
F2UUIUtils.HierarchyAttribute.Visibility);

//Any visibity (gameobject.activeSelf) change must be updated using
HierarchyAttribute.Transform
```

2- HierarchyAttribute.Transform:

```
Vector3 rotation      = Vector2.zero;
transform.localRotation = Quaternion.Euler(rotation);
```

```
F2UUIUtils.UpdateHierarchyAttributeChangedRecursively(transform,
F2UUIUtils.HierarchyAttribute.Transform);

//Any change of: transform.localRotation, transform.localScale,
transform.localPosition, transform.position, transform.rotation must be
updated using HierarchyAttribute.Transform.
```

3- HierarchyAttribute.Z:

```
transform.localPosition = new Vector3(transform.localPosition.x,
transform.localPosition.y, 3);

F2UUIUtils.UpdateHierarchyAttributeChangedRecursively(transform,
F2UUIUtils.HierarchyAttribute.Transform);

//Any change of transform.localPositon.z or transform.position.z must be
updated using HierarchyAttribute.Z
```

F2U Custom Sprite Batching Tips:

- F2U Sprite Batching has a better performance for animations that are not transformed (moved, scaled, and rotated), so for example to implement a scrolling 2D environment it's better to move the camera instead of move the environment animations.

- The game objects "Layer" (Layering system provided by Unity3D) can be used to separate the batching in rendering layers.

- Avoid changing the z (HierarchyAttribute.Z), because transparency shaders can't write to z buffer (write to z buffer is only for opaque shaders) the batching system need to calculate the z order of the sprites when a z position has changed, so it's an expensive calculation. The z of the objects (parents of sprites) can be changed but changing the z each frame will slow down the framerate, so it must be used carefully.

Dynamic Batching Tips:

In order to use dynamic batching, the user is needed to place the objects they want batched in the same Z position. Because Flash to Unity automatically chooses different Z values for every object as a way to emulate layering, though, this process must be done manually by the developer.