iOS 3D Touch (Force Touch) & Touch Radius Input Plugin

CHANGES IN 1.2:

New

- Added support for touch radius. Touch data now contains properties radius and radiusTolerance.
- Added method ForceTouchPlugin.SupportsTouchRadius() to query touch radius support. Only iOS 8 and above support radius.
- Added support for Unity 4.6.2p2 and above.

Changed

- ForceTouchState.OldiOS is changed to ForceTouchState.IncompatibleOS.
- Improved code comments throughout the project.
- Native side of the plugin code was optimized removing some unnecessary method calls.

Fixed

- Fixed a bug causing app to crash on iOS 7 and below.

Foreword

First of all, I would like to personally thank You for purchasing this plugin and supporting its development. I will do my best to make sure You will not be disappointed with Your decision. Asset Store does not provide any information about customers but feel free to send me an email with your thoughts and ideas to raimis@pinstudios.com.

About Plugin

This plugin is a way to integrate iOS 3D touch (pressure touch) and touch radius into your game before Unity officially supports this feature.

Few allocations and fast performance thanks to binary serialization of touch data on iOS side. No strings allocations.

Be first to deploy games for iPhone 6s / iPhone 6s Plus and iPad Pro (with iPencil).

Features

- Full support of iOS 3D Touch API
- Full support of touch radius API

- Full source code provided
- No need to understand objective-c - simply add plugin files to the project and you are ready to go.
- No extra views added to handle touch on native side.
- Multitouch support
- Easy-to-use callback for event of user disabling 3d touch in the middle of the game - you can run plugin only on devices that support 3d touch, use Unity on others.
- iPencil pressure support
- Backwards compatibility to work well with older versions of iOS (iOS 5 and above)
- Easy to understand example project with InputManager that can handle mouse, unity touches and iOS native touches at the same time (multitouch).

How things work under the hood

When requested, plugin captures raw iOS input data using objective-c code, converts it to binary data and sends it back to Unity managed code.

Deserialized data contains the following information for each touch:
1) Finger Id
2) Position on screen
3) Delta position
4) force and maxForce
5) radius and radiusTolerance

Important notes

On devices that don’t have pressure sensitive screens the maxForce value is always set to -1.0f (force is 1.0f). For devices that support different pressure levels maxForce provides the max possible force. Apple claims that force value of 1.0f is regular touch. Anything below that is light touch, and above that is pressure touch. Value of maxForce may be different for different devices (iPad Pro with iPencil will have different value than iPhone 6s), therefore using normalized value might cause problems when more 3d touch enabled devices are available in the market. **Value of maxForce for iPhone 6s is 6.666667.**

On devices that don’t support touch radius, value for radius will be -1.0f and radiusTolerance will be 0f. On devices that support radius touch, radiusTolerance will return the float representing half the increments in which radius will change. For example if radiusTolerance is 5.5, possible radius values are 11, 22, 33 etc. The very light touch is 4 x radiusTolerance (I was unable to achieve touch as light as 2 x radiusTolerance neither on iPhone 5, nor on iPad AIR2); the regular touch is 6x or 8x touchRadius and “big” touch is usually in the region of 10x radiusTolerance. Each device has different radiusTolerance.

Running the example

1) Open ForceTouchExample scene in Example/Scenes folder.
2) Build iOS platform project
3) Make sure to open exported project with XCode 7.0 or above (BaseSDK has to be set to 9.0)
4) Make sure BitCode is set to No in Build Settings (default value for Unity bellow 5.2 is Yes which causes compiler errors)
5) Use iPhone 6s/6s+ to test force touch. Use iOS 8 or above device to test radius touch. Project will work fine on other devices with older iOS too (iOS 5 and above).

How to use the plugin:

Use **InputManager** as the base class for your project. You can remove the code related to touch state (Began, Ended, Moved) if that's not required for your project to function properly.

Use **ForceTouchPlugin.GetForceTouchState()** to determine the state of force touch:
1) If you call this method to early (in awake of the first scene this information is not yet available) it will return **Unknown**
2) If it's iOS device supporting force touch it will return **Available** (it has to be enabled in accessibility settings too)
3) If it's iOS device running iOS 9 or above but there’s no hardware support or 3d touch is disabled in accessibility settings it will return **Unavailable**
4) If it's iOS device running iOS prior to iOS 9 it will return **IncompatibleOS**.

You can subscribe to an event if user changes his accessibility settings. To do that call method **ForceTouchPlugin.SetCallbackMethod(string gameObjectName, string methodName)**. It is important that GameObject is available in the hierarchy and method has the signature of function(string message). (See http://docs.unity3d.com/Manual/PluginsForIOS.html for more details). It is safe to call this method multiple times with different parameters. Only last callback will be executed. To unsubscribe from callbacks completely call method **ForceTouchPlugin.removeCallbackMethod()**.

Call **ForceTouchPlugin.SupportsTouchRadius()** to determine if device supports radius touch. True will be returned for iOS 8.0 and above devices. On every device below iOS 8 it will return False.

ForceTouchPlugin.GetScaleFactor() is the helper method. Though plugin returns touch data in screenspace, iOS operates in lowDPI mode. With the help of scale factor you can transform data back to iOS space.

To start tracking native touches call **ForceTouchPlugin.StartTracking()**. It is safe to call it multiple times. To stop tracking call **ForceTouchPlugin.StopTracking()**. Though plugin works fine with older iOS devices, it is best to only track native touches when **ForceTouchState** is **Available** or **SupportsTouchRadius()** is true. The reason for that is that Managed-to-Unmanaged calls on iOS are rather CPU intensive (according to Unity documentation) which adds some battery drain and slightly reduces the performance. To reduce battery drain it’s better to use plugin only on devices where plugin offers added value over Unity’s default input. Use

ForceTouchPlugin.IsTracking() to determine if plugin is currently tracking native touches.

Use **ForceTouchPlugin.GetNativeTouches()** to query the list of current touches. All touches are returned without their current state (Started/Moved/Ended). This has to be handled in managed code (InputManager class does that). This is due to the fact that you are not forced to track native touches whole time. You can only enable native touch tracking when the feature is actually needed to save performance. Though it is best to call this method on every frame, you are not forced to do it. In that scenario touch state provided by native code might be outdated, therefore it's better to track it in managed code.

Requirements

- XCode 7.0 or above (project needs to be compiled using iOS 9.0 SDK as base SDK version; plugin code is backwards compatible therefore it works fine with iOS 5 and above)
- Unity 4.6.2p2 (Pro ONLY) or Unity 5 (Personal or Pro) and above.
- iOS 8 or above device to test touch radius
- iPhone 6s / iPhone 6s / iPad PRO with iPencil to test force touch (iOS simulator doesn't provide means to test 3D Touch).

Considerations

Unity documentation states that managed-to-unmanaged calls are CPU intensive on iOS and should be limited to one call per frame. For best performance this plugin uses 1 call of this type per frame. It is recommended to use plugin only on devices that support 3d touch or touch radius. This will slightly reduce battery drain on devices where this plugin cannot offer any added value over Unity's default input.

Final notes

Store rating is the the only visible difference between **good** and **bad** assets on the Asset Store. If you found this plugin useful, I would really appreciate if you could spend a minute **rating this plugin** on the store. This will help me to better understand your needs and improve plugin to better match your needs. This will also help possible future customers to choose the best plugin between plugins of the same type.

Support

- Asset Store Link
- Forum Support Thread Link
- Email support