Adding Multi-Touch Support to Unity* Games on Microsoft Windows* 7 and Windows* 8 Desktop

Abstract

Currently, Unity does not process touch messages in applications running on Windows 7 and Windows 8 desktop. We present a solution so developers can access the available multi-touch inputs in Unity applications running on Windows. The solution is implemented as a plug-in utilizing various Windows APIs to intercept the touch messages that are sent to the application. We process the information from these touch messages and then create a simple data set that can be accessed via C# from a script in the Unity application.

1. Handling Touch Messages in Windows

The process for handling touch messages is well documented on MSDN and elsewhere, but we will mention a few points germane to our discussion here. The WNDPROC for the active window receives a WM_TOUCH message when a touch event occurs. Upon receipt of the message, the application calls GetTouchInputInfo() to populate an array of PTOUCHINPUT structures, one for each current touch point on the surface. The lower word of the wParam parameter passed to the WNDPROC contains the number of PTOUCHINPUT entries in the array.

The dwID member of the PTOUCHINPUT is particularly interesting in this case. This ID links the particular PTOUCHINPUT entry with entries provided to previous calls to GetTouchInputInfo(). This provides temporal continuity of the touch point. Many applications attempt to find the previous point closest to the new point and assume they are the same. This is not always the case. Examples are when multiple touches are clustered close together or cross over each other on the screen.

The touch data can be linked to touch point information once it is obtained from the previous calls to GetTouchInputInfo(). Then the data is processed by the application accordingly. In Unity applications, additional steps are required to derive touch information for use in the C# scripts that drive the application.

2. Using Plugins in Unity

Unity plugins are compiled as DLLs, and the export mechanism is well documented. “Exported functions” are the same in Unity plugins as they are in any other DLL. The plugin can be written in any language so long as the compiler is capable of building a DLL output file and the exported functions can always be called from the C# script in Unity.

Exported functions can be called directly from C# script once a Unity plugin has been loaded. In C#, the exported functions from the DLL can be accessed simply by declaring them extern in the C# script. For example, suppose an exported function from a C++ DLL named “MyPlugin” has the header:

```
__declspec(dllexport)  int __cdecl Init(int p);
```

This can be accessed from C# by declaring an extern reference to it.

```
[dllimport (“MyPlugin”)]  private static extern int Init(int p);
```

Then calling the function Init(some integer) in the script.

3. Implementation of the Plugin Touch Interface

To successfully capture the Windows touch messages directed to the Unity application, we must go through several steps. First, we will outline the steps, then the source code comments will complete the process.

The setup procedure for the plugin involves attaching a hook using SetWindowsHookEx() to receive the WM_TOUCH messages for the Unity application. You need two things to do this: the handle to the DLL module and the thread id**entifier** that created the Unity window that has focus. To obtain the thread identifier of the thread that created the Unity window, first find its handle. The DLLMain() entry in the DLL will show you the handle of the module. The easiest way to locate the Unity window handle is to pass the name of the Unity window from C# to the plugin and then enumerate all the desktop windows in the plugin using EnumDesktopWindows() to find the handle. Once the window is found, its handle can be obtained. With the window handle, call GetWindowThreadProcessID() to obtain the thread identifier required to call SetWindowsHookEx().

The code shows that SetWindowsHookEx() is actually called twice, and two handlers are installed: one attached to WH_GETMESSAGE and one to WH_CALLWNDPROC. This is because the WN_TOUCH messages arrive at the WNDPROC differently in Windows 7 than in Windows 8. On Windows 7 the messages are posted to the queue. In Windows 8, they are directly injected into the WNDPROC. It is important to remember to release the Windows hooks once we are finished since they are a limited resource in the OS.

Applications are not enabled for touch by default. For the Unity application to receive touch messages, the plugin needs to call RegisterTouchWindow() to enable the messages. This call will also require the windows handle obtained earlier.

Once we have finished, our callbacks will be called whenever messages are sent to the Unity window. Then it is a simple matter of scanning for touch messages and returning the rest to Unity.

4. Handling the Touch Data

Two exported functions are provided to access the touch data from the plugin. You can change these if a different interface is more appropriate. Let’s clarify the use of data.

Int GetTouchPointCount() returns an integer that represents the number of contact points currently active on the screen.

Void GetTouchPoint(int ID, tTouchData Data) returns the active touch point with index ID from the plugin.

As previously mentioned, the data provided by the call to GetTouchPoint() requires finesse in order to maintain temporal consistency when handling the touch points in Unity. The plugin maintains an internal list of active points and makes every effort to respect the IDs assigned by Windows. The Windows IDs are not the same as the ID provided to the GetTouchPoint()

exported function. Interrogate the tTouchData structure to find the actual ID of the touch point so you maintain temporal integrity between frames.

5. Conclusion

Although the Unity engine does not process touch messages itself on Windows 7 and Windows 8 desktop, it is possible to register the Unity window for touch from a plugin. Once the window is registered, a little trickery allows us to access touch messages in a script in the Unity application. Many developers favor Unity because it is an intuitive and flexible tool that allows them to develop rich games quickly. Access to touch in Unity will greatly enhance your ability to create dynamic games and media for the new range of touch-enabled Windows devices.

Resources

- MSDN Home: http://msdn.microsoft.com/en-US/
- Plug-in source and sample project: http://software.intel.com/en-us/blogs/2013/05/01/the-unity-multi-touch-source-finally