

A Theory on Gesture Recognition for the .NET3/WPF Multitouch Framework

Donovan Solms
7 September 2007

<http://blog.whitespaced.co.za/>

Introduction

I've been pretty busy working on the .NET3/WPF Multitouch Framework the past few weeks and I came across some problems. The first was an already existing problem, which is the difficulty (or rather the amount of work) of adding a control to the framework, or writing your own controls. The other problem is that some interaction just isn't practical, i.e. Pressing four fingers on an object to do something isn't always possible due to scale and just not practical. So I decided to start thinking about gestures. When I think about gestures, I think extensibility. Not the gestures itself, but the ability to add any number of gestures, and create your own. Easily. That's when I came up with the idea of a Gesture Engine.

The Gesture Engine

The Gesture Engine will be the first in the .NET3/WPF Multitouch Framework to receive touch events from Touchlib. The reason for that is so that the gesture can be caught before the application takes the touch. I.e. When you press two fingers on an image very close to each other, you might want something else to happen, and not rotate/scale.

At the moment I only have an idea on one-finger gestures and finger proximity gestures, but I'm thinking hard about multiple-finger gestures. This document does not **yet** include how the code will actually match the gestures, but at least a way on how it could be done.

How it will work

To add a gesture to the engine, all you need to do is create a vector graphic for the gesture in SVG (Scalable Vector Graphic). You can do that from (almost) any vector drawing application. The only rule is that the graphic shall contain no colour. Black only drawn in 1pt size brush/line tool/curve tool, etc.

After the graphic is exported to SVG it will be converted into XAML with the ViewerSvg from the Ab2d_svg library that is freely available from <http://www.wpf-graphics.com/>. After the SVG is in XAML markup, the engine will be able to use it. In the long run, a general gesture markup language can be created (as proposed in this thread , <http://nuigroup.com/forums/viewthread/196/>, with an application to create gestures that cut out the process of converting SVG to XAML - in this case)

When the framework starts, it will load all the 'registered' gestures into the engine so that it can easily and quickly access them. New gestures will also be registered using the engine by only specifying the location of the XAML file. It will only be registered once, after which the gesture will stay registered, even if the XAML file is deleted. It will have to be unregistered, or deleted via the RegisteredGestures application that will be included.

How gestures will be recognised

The XAML be converted into geometric shapes. When a gesture is made on the multitouch screen, it will also be converted into a geometric shape. This shape will be matched against the registered gestures until a close-enough (*close-enough gesture*) match is

found (explained in the next part). If a suitable gesture is found, the engine will send a gesture event to the application containing the start and end positions as well as the width and height of the gesture. If no suitable gesture is found, no event will be raised.

Performance can be increased by putting the registered gestures in a tree that is ordered. The root element will be the most used gesture. The tree will adjust itself depending on the gestures use count for the application.

Because the gestures are in vector form, it can be scaled to any size and still be recognised. The engine will first scale the registered gestures to fit the gesture made on screen before matching starts.

The biggest advantage to this approach to gestures, in my opinion, is that a user can add their gestures on the fly. I.e like they way you can customise your shortcuts in an application, the multitouch application can have a gesture setup with a list of commands. Next to each command will be a small thumbnail (from the XAML/SVG) showing the current gesture, then when a user touches 'reconfigure this gesture' he/she can just make a gesture and it will automatically set up and recognisable in the application, even when it is closed and re-opened.

How close is a close-enough gesture?

The following quick sketches show how these gestures will be recognised.



Fig 1.1 - The original gesture in SVG that will be converted to XAML (looks the same)

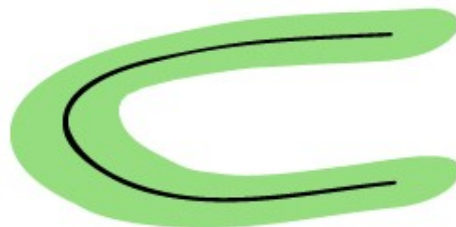


Fig 1.2 – The close-enough gesture

In Fig 1.2 the green area represents the 'close-enough' gesture area. If a gesture made on screen is in the green area, it will be matched to the gesture in Fig 1.1. This will ensure that a gesture is recognised even when it isn't 100% perfect.



Fig 2.1 – The actual gesture made on screen

Even though the gesture made on screen is close, it is not nearly an exact match.

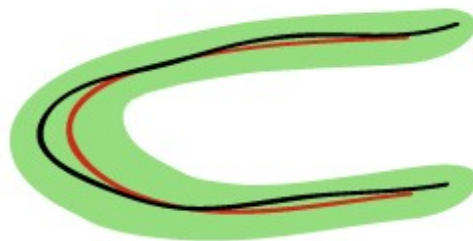


Fig 2.2 – The original gesture (red) and actual on-screen gesture(black)

In Fig 2.2 the red shows the XAML/SVG gesture in the gesture engine. The black shows the gesture made on screen. Green still shows the 'close-enough' gesture area. So when matching starts this gesture will be a match to the registered gesture because all the points fall inside the 'close-enough' area.

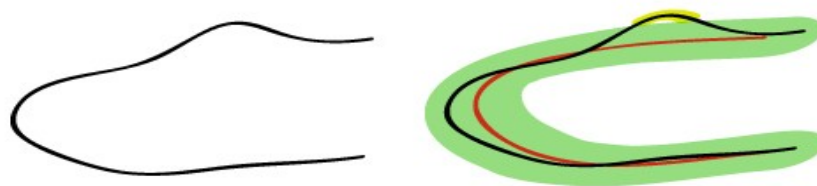


Fig 3 – On-screen gesture (left) and the original gesture(red) (with 'close-enough' green area) combined with the on-screen gesture (black)

Fig 3 shows a gesture that will not be recognised as a point falls outside the 'close-enough' area. Matching will only continue until a point outside the 'close-enough' area is found. If such a point is found (yellow background), the engine will start checking against the next gesture in the tree (or list)

Finger proximity gestures

Proximity gestures can be seen as gestures that are only presses, not movements. There will be three levels. Zero-proximity, normal-proximity and close-proximity.

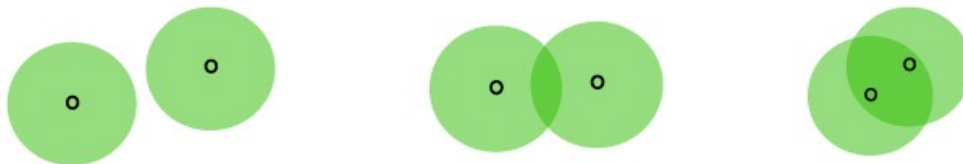


Fig 4 – Zero-proximity (left), normal-proximity (center), close-proximity (right)

As you can see in Fig 4, Zero-proximity is when two (or more) fingers are not within a specified radius of each other, while normal-proximity is when touches' radii are overlapping. Close-proximity is when a finger is in the radius of another finger.

These gestures could be described in some XML format to also allow adding gestures as you like.

This could be useful in the following situations:

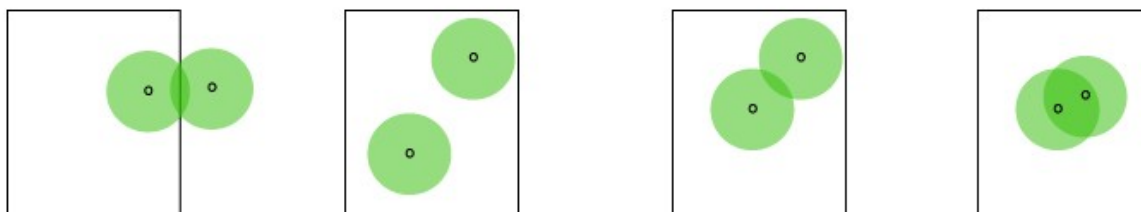


Fig 5 – Some proximity gestures in use. The block box can be any on-screen object. (from left to right G1,G2,G3,G4)

The gesture G1 might be used when you need a menu to pop up for a specific object. G2 will most likely be used for rotate/scaling on objects. G3 could be used when you need something else to happen, apart from rotate/scale, when two fingers are on an object. G4 can be useful when the object is pretty small.

These proximity gesture will probably be most useful on areas where normal movement-based gestures won't really work. I.e you want a specific menu to pop up for an object, but you don't want to make a gesture over the object, then you can maybe use G1 for that menu, G3 for a different menu and G4 for another menu. These gestures can, of course, make us of more than two fingers.