USING GESTURE MARKUP LANGUAGE (GML) IN GESTUREWORKS3

In GestureWorks3 there are two methods for applying gesture interactions to touch objects:

1. Using native gml and cml transformations
2. Using gml and traditional gestureEvent listener/handlers.

The root gml file, "my_gestures.gml," contains all gestures available to your application. All gestures must be created here in order to be parsed by the application. Only gestures explicitly selected from this master list are added to touch objects such as touchSprites.
To add gestures to a touch object in your application you must add the gesture to the gestureList associated with that object. There are two ways of doing this:

1. If you are creating touch objects using CML then: using my_application.cml inside touch container in between the "<GestureList>" tags add a gesture by listing the gesture id for example: "<Gesture ref="n-drag" gestureOn="true"/>". This will add the "n-drag" gesture as described in "my_gestures.gml" to your touch object and handle all transformations and events natively using the complete description as defined in the GML document.

2. If you are not using CML to create and manage your touch objects you will have to listen for and handle gestureEvents. To do this: add gestures to a touchSprite or touchMovieClip by adding a gesture to the gestureList property associated with the touchObject, for example: "myTouchSprite.gestureList {"n-drag":true};". Then add a gesture listener for that gesture type. For example "myTouchSprite.addEventListener(GWGestureEvent.DRAG, gestureDragHandler);".

OVERVIEW OF GML DOCUMENT

Each gesture defined in the GML document uses a fully customizable system that can be conceptually broken down into a four step process:
The first step is the definition of the gesture action. This definition is used to match the behavior of the input device to the gesture object.

The second step is the assignment of the analysis module. Currently GML allows you to specify a specific analysis module from the set of "built-in" compiled algorithms in GW3. However the GML specification is also designed to accommodate custom code blocks that can be directly evaluated at run time and directly inserted into the gesture processing pipeline.

The third step is the establishment of post processing filters. For example: in GW3 the values returned from the gesture analysis algorithm can be passed through a simple low pass filter which helps smooth out high frequency noise which can be present in the form of touch point

“jitter”. The “noise filter” can help smooth out these errors and reduce the wobble effect. In addition to this, the values returned from the noise filter can also be fed into a secondary “inertial” filter that can be used to give the effect of inertial mass and friction to gestures adding pseudo-physical behaviour to touch objects associated with the gesture. In this way multiple cumulative filters can be applied to the gesture pipeline in much the same way as multiple filters can be added to display objects in popular image editing applications.

The fourth and final step in defining a gesture using GML is a description of how to map returned values from analysis and processing directly to a defined touch object property or to a gesture event value for a gesture dispatched on the touch object.
With these four steps GML can be used to define surface gestures by performing configured geometric analysis on clusters of points or single touch points. The return values can then be easily processed and assigned to customizable display object properties. This can be done at runtime with re-compiling which effectively separates the gesture interactions from the application code in such a way as to externalize the scripting of touch UI/UX enabling interaction designers to work along side Actionscript developers.

EDITING THE GML DOCUMENT

The individual gesture id of each gesture block is used to directly reference the gesture in the touch object gestureList. Each gesture id used in the GML gesture tag must be unique. Currently only a single "gesture set" is accepted by GW3. All gestures must be in the gesture set "n-manipulate".

There are currently only thirteen specific gesture "types" that are supported in GW3. These types include: drag, rotate, scale, hold, tap, double_tap, triple_tap, orient, pivot, flick, swipe, scroll and tilt. Custom gesture types that will use fully script-able algorithms will available soon.

To edit the number of touch points that will trigger a gesture match; set the cluster "point_number", "point_number_min" and "point_number_max" values. Setting the "point_number" to "2" will only allow 2 points to activate a defined gesture. Otherwise setting "point_number" to "0" and then explicitly setting "point_number_max" and "point_number_min" will determine a range of values that will activate gesture analysis.

The properties returned from the gesture analysis can be defined in the <returns> tag inside the <algoirthm> tag located in the analysis block. These properties must have a unique id value inside the gesture block in which it is defined. These properties can then be referenced in the filtering and mapping blocks.

The processing block allows the activation of multiple cumulative filters to each independent

property. There are currently only two types of filters that can be applied to a given property, the "noise_filter" and the "inertial_filter". The noise filter can be used to smooth values that contain a large percentage of random uncertainty. This can be present as noise like stuttering or jittering when trying to rotate or drag objects. For example the rotate gesture is applied to prevent objects from jittering when trying to slowly drag objects on stage. The higher the filtering percentage the greater the smoothing effect. However high levels of noise filtering applied to a gesture can create a decrease in responsiveness of a gesture. The "inertial_filter" allows for the control of gesture values after touch actions have ceased. Setting "release_inertia" to true and friction to a value less than 1 will give an inertia effect to any gesture property it is applied to. When an object is released the change in the gesture property (delta) is gradually reduced (eased). The closer the value of friction is to 1 the longer it takes for the effect to die down and the gesture delta to reach zero.

The mapping block defines how the values returned from gesture analysis and processing update the touch object to which it is applied. The gesture can be returned in the form of a gestureEvent and monitored using listeners or can be mapped directly to the public properties of the touch object it is attached to using "target_ref". For example target_ref = "dtheta" maps the returned value to the rotation property of the touch object. In addition to the mapping of values, max and min limits can be applied to the values returned by the gesture. This can be done by setting delta_threshold="true" and defining values for delta_max and delta_min.

The configurability of the gesture object as defined in GML has been designed to provide flexibility while developing new gestures and still provide fundamental methods for reducing activity when gesture related calculations are not required. Each gesture can be effectively calibrated to specific touch screens and CPU system requirements and preferred response or "feel".

CURRENT LIMITATIONS

Note:TouchObjects (TouchSprites and TouchMovieClips) can be nested inside Sprites, MovieClips, TouchSprites and ToucMovieClips but transformations made on the parent display object are so far only accounted for in child transformations one layer deep. That is to say that multi-layered nested transformations are currently not possible but will be implemented post launch in GW3.1.