Shake Gestures Library

A Windows Phone Recipe

# Document Purpose

This document introduces a helper library for identifying shake gestures by using the accelerometer built into Windows Phone 7 devices. It explains how to use the library, how the library works internally, and how you can configure the library’s parameters to adapt gesture detection to your needs.

# Library Features

The shake gestures library uses the accelerometer to detect movement in three directions:

* Left-Right (X direction)
* Top-Bottom (Y direction)
* Forward-Backward (Z direction)

If you’re looking for a general purpose shake gesture and don’t care about the direction of the gesture, then you can simply use any combination of one or more of the supported gestures. However, sometimes you need to have better control and to understand if the gesture was in the “right” direction.

# How to Use the Shake Gesture Library

Since the shake gesture library is doing all the heavy lifting for you, using the library requires very little effort from you.

**Step 1**: Add reference to shake gestures library: **ShakeGestures.dll**

**Step 2**: Add a **using** statement to file header

*using ShakeGestures;*

**Step 3**: Register to **ShakeGesture** event

// register shake event

ShakeGesturesHelper.Instance.ShakeGesture += new   
 EventHandler<ShakeGestureEventArgs>(Instance\_ShakeGesture);

// optional, set parameters

ShakeGesturesHelper.Instance.MinimumRequiredMovesForShake = 4;

**Step 4**: implement the **ShakeGesture** event handler from step 3

private void Instance\_ShakeGesture(object sender, ShakeGestureEventArgs e)

{

\_lastUpdateTime.Dispatcher.BeginInvoke(

() =>

{

\_lastUpdateTime.Text = DateTime.Now.ToString();

CurrentShakeType = e.ShakeType;

});

}

The **ShakeGestureEventArgs** holds a ShakeType property that identifies the direction of the shake gesture.

Finally, you need to activate the library, which binds to the phone’s accelerometer and starts listening to incoming sensor input events.

// start shake detection

ShakeGesturesHelper.Instance.Active = true;

***Note****: You can continue working directly with the phone’s sensor. The ShakeGesturesHelper class doesn’t block any sensor events—it just listens to them.*

***Note****: The* ***ShakeGesture*** *event arrives on a thread different from the UI thread, so if you want to update the UI from this event you should dispatch your code to run on the UI thread. This can be done by using the method myControl.*[*Dispatcher.BeginInvoke*](http://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher.begininvoke(VS.95).aspx)*(), where myControl is the control you want to update.*

# How Does the Shake Gesture Library Work?

We need to figure out what a shake gesture is and how to identify a shake when it occurs. Then we need to classify the shake according to the gesture direction.

The accelerometer sensor in a Windows Phone measures the gravity forces applied on the phone and reports an **AccelerometerReading** event about 50 times per second. When you move the phone up and down in 3D space, you’re getting a lot of readings (the sensors are quite noisy), and you don’t really know the phone’s orientation or the distance that the phone moved, since all you have are changes in the relative gravity force on the phone during a given period of time.

With that in mind, we set out to create a simple, high-performance shake gesture library, and came up with the following shake gestures detection process:

1. Noise reduction
2. Segmentation into "Shake" and "Still" signal segments
3. Shake direction classification

We report sensor readings as vectors, since we obtain 3-axis readings (X, Y, and Z, with each having a double value associated with it). We can set the initial point to (0, 0, 0), hence a [vector](http://en.wikipedia.org/wiki/Euclidean_vector).

## Noise Reduction

Before performing gesture detection, the noise must be removed from the accelerometer’s input vectors. This is done by passing the raw input vectors through a low-pass filter. This smooths the signal so that it ignores small changes and only takes into account any “large enough” changes. We’re using an existing noise reduction implementation by Dave Edson. For more details on Dave’s algorithm, check out this [blog post](http://windowsteamblog.com/windows_phone/b/wpdev/archive/2010/09/08/using-the-accelerometer-on-windows-phone-7.aspx).

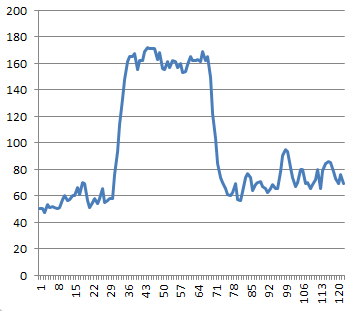


Figure 1: Before noise reduction

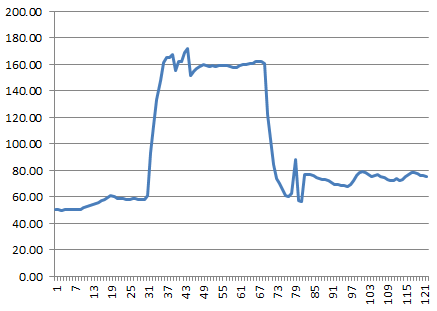


Figure 2: After noise reduction

As you can see, the signal after noise reduction is much cleaner and easier to work with.

## Segmentation into Shake and Still Signal Segments

After cleaning the signal, our next step is to separate shake and still segments. Basically, we want to identify shake segments when the following conditions are satisfied:

1. The magnitude of one or more of the vectors (X, Y, or Z) crosses a certain threshold.
2. The vector stays above this threshold long enough so that we don’t mistake a one-time blip for a shake.

***Note****: We don’t use a system timer to measure time. Instead, we depend on the fact that the Windows Phone sensor generates about 50 events per second, so the time between each event is around 20 msec. Therefore, five events total roughly 100 msec*. You’ll see in the code that we measure event intervals and not real world clock time.

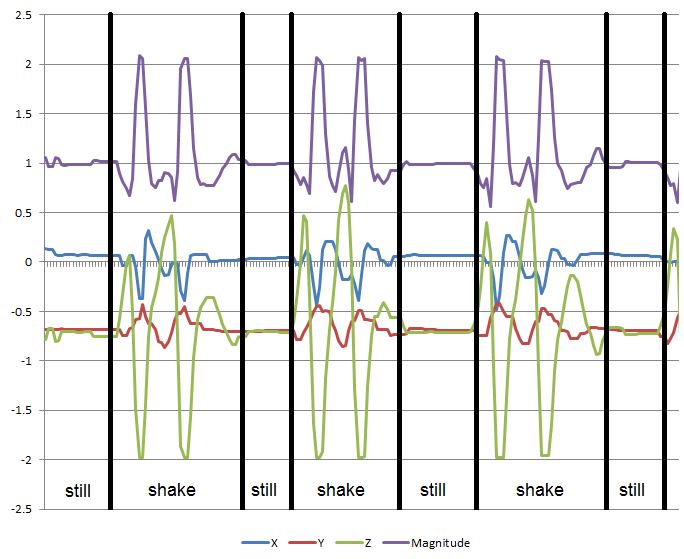


Figure 3: Signal segmentation

Ignore for a second how we calculate vector magnitudes; we’ll address that topic in the next section. For now, look at Figure 3. You can see from the green line pattern that it’s very clear when the phone was in a shake segment and when it was in a still segment. Therefore, our goal is to be able to understand when a series of vectors represents a shake segment. Once we identify a shake segment, we’ll further process it to extract the shake type and raise the shake event.

### Calculating Gravitation Vectors

To start with, we need a reference point that reflects the still state of the phone. The Still segment is used to compute something we call the **last-known-gravitation-vector**. The last-known-gravitation-vector will be used later to eliminate the effect of gravitation when classifying the shake type.

***Note****: We can't assume that the gravitation vector will always have the value (0,-1, and 0) for two reasons:*

1. Gravity isn’t exactly [1G](http://en.wikipedia.org/wiki/Standard_gravity) everywhere on earth. It varies somewhat according to location and altitude.
2. The direction of the gravity vector depends on how you hold your phone. If you rotate it, the direction varies. If you’re moving, your hand direction varies.

To calculate the gravitation vector, we take the most recent vectors in the still signal ([MaximumStillVectorsNeededForAverage](#_MaximumStillVectorsNeededForAverage) parameter) and average the vectors that have very low magnitude ([StillMagnitudeWithoutGravitationThreshold](#_StillMagnitudeWithoutGravitationThr) parameters. In case we don't have enough still vectors for a representative sample ([MinimumStillVectorsNeededForAverage](#_MinimumStillVectorsNeededForAverage) parameter), we abort the calculation, since we won't get a valid gravitation vector.

The gravitation vector is re-calculated for each new still segment that we get, to account for any changes in how you hold your phone. Remember that we’re looking for real time effects; therefore, we need to maintain an accurate state of the current gravitation forces that are affecting the phone.

### Finding Signal Boundaries

Now, let’s assume the user starts shaking his phone. We need to find the shake segment starting and ending points. To do so, we use the following algorithm:

There are only two possible states:

* **Shake state** – Indicates that we’re currently in the middle of a shake signal
* **Still state** – Indicates that we’re currently in the middle of a still signal (that is, not in the middle of a shake signal)

Based on the current state and the input vector type, we decide what the next step is. Each time we get a new input vector from the accelerometer (after noise reduction), we check whether the vector magnitude is equal or larger than a certain shake threshold [ShakeMagnitudeWithoutGravitationThreshold](#_ShakeMagnitudeWithoutGravitationThr) parameter. If it’s larger, then this input vector is addressed as a potential shake vector. Otherwise, it’s considered a still vector.

Per each input vector and the current “shake” state we use the following state-machine to determine the next state..

* **Condition:** The current state of the phone is still, and the new vector has a still magnitude value.
* **Operation:** Add the vector to the still signal array (the array is a cyclic array).
* **Condition:** The current state of the phone is still, and the new input vector has a magnitude that is higher than the minimum shake threshold.
* **Operation:** Set shake state, process still signal, add vector to shake signal array (this doesn’t mean that we identified a shake, we’re just adding it to the array for further processing later).
* **Condition:** The current phone state is shake, and the new input vector has a shake magnitude.
* **Operation:** Add vector to shake signal, try to process shake signal array. Here we’re actually trying to identify a shake, since we already got a few shake vectors in the array. Again, we need to wait for the minimum amount of shake vector before we can start processing a shake single.
* **Condition:** The current phone state is shake, and the new input vector has a still magnitude (it’s below the minimum shake magnitude).
* **Operation:** Add vector to shake signal, unless we’ve already received too many sequential still vectors ([StillCounterThreshold](#_StillCounterThreshold)  parameter), in which case move the still vectors to the still signal and change the phone state to still state.

## Shake Type Classification

It’s time to dive into the actual vector processing. The following describes the algorithm we’re using, but for performance reasons the code performs some of the processing while the shake vectors are being collected, but the idea remains the same.

### Neutralizing Gravitation Effects (Removing Earth Gravity Effects)

All input vectors include the earth’s gravitation forces. And since we don’t have any idea about the position of your phone in 3D space, we don’t know which of the vectors (X, Y, or Z) is affected by the earth’s gravity. Therefore, we need to remove the earth’s gravity effect.

To eliminate the effect of gravity, we first reduce the last known gravitation vector from all the vectors in the shake signal. In this way, when we get acceleration in some axis we can be certain that the user moved the phone along that axis instead of wondering whether this is just the force of gravity trying to fool us. Therefore, for each input vector we’re subtracting the last known gravity vector.

### Finding Shake Main Direction

Now we want to find the main direction of the shake signal, which can be one of the following three directions:

* X (left-right)
* Y (top-bottom)
* Z (forward-backward)

To do this, we first need to find the main direction of each vector in the shake signal. We do this by checking which of the vector products has the biggest absolute value. To get a better feeling of why we’re breaking the vector into its products, let’s review a 2D vector. As you can see in Figure 4, the red vector products are broken into the X and Y values [Red value = squrt(pow(x,2) + pow(y,2))].

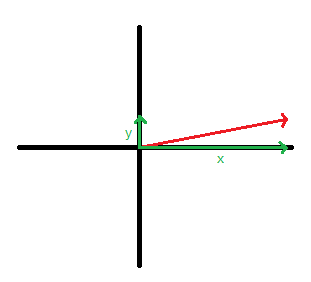


Figure 4: Illustration of classifying a 2D vector

It’s clear that the red vector should be classified as an X axis vector, because its X product is much stronger than its Y product. The same methodology is implemented in the library for 3D vectors.

At this stage, we’ve got a vector that isn’t affected by the earth’s gravity, we’ve broken the vector into its products, and we know the main direction of the vector.

Our next step is to create a histogram of the vectors’ products, and then to select the direction with the most products in it. Remember, we started with a vector whose magnitude is strong enough to be considering a **shake** vector. We removed the earth’s gravity effect, and we broke it into its distinct products, and classified them. For example, Figure 5 shows a histogram of a shake signal that has most of its vectors pointing in X direction, hence the shake signal’s main direction will be X.

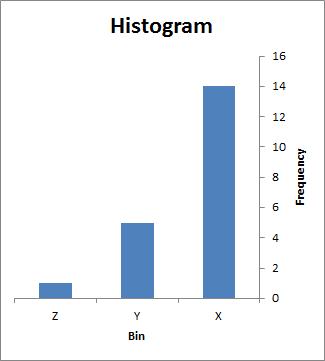


Figure 5: Histogram of vector directions in a shake signal

To prevent weak vectors ([WeakMagnitudeWithoutGravitationThreshold](#_WeakMagnitudeWithoutGravitationThre) parameter) that got into the shake signal from affecting the histogram result, we’ll only consider vectors above a certain threshold when calculating the histogram.

In addition, we’ll consider the histogram result as valid only if the number of vectors in the main direction passes a certain threshold ([MinimumShakeVectorsNeededForShake](#_MinimumShakeVectorsNeededForShake) parameter). This will enable us to eliminate false results due to histograms done on small amounts of data.

### Recognizing the Gesture

Up until now, we’ve identified a movement of the phone in 3D space that’s fast (powerful) enough to be consider a shake. Basically, we know the shake signal’s axis, but that isn’t sufficient. In order to make sure the movement we detected is really a shake gesture, and not just a random (single) movement in one direction, we need to count several intervals of movement and, more importantly, changes in the vector direction. A shake is defined as a movement of the phone on one axis back and forth a few times.

To check whether the shake signal is a shake gesture, we’ll check the sign of the main direction coordinate. For example, if we found that the main direction of our shake signal was X, then we’ll inspect the sign of the X products of all the vectors in the signal. On a real shake gesture, we expect the sign to change a few times, since we move the phone back and forth and thus keep changing the force (acceleration), which translates to a direction change.

For example, in the following figure we see a normalized (without gravitation) shake signal of a real shake gesture. You can clearly see the main direction being Z, and the Z values go back and forth between positive and negative.

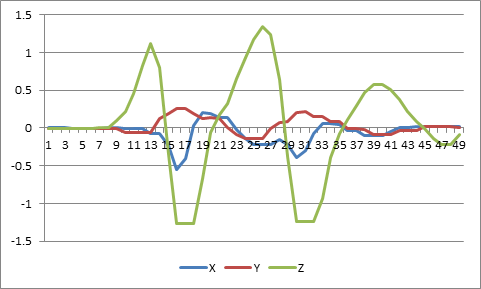


Figure 6: Example of a shake vector

In order to identify the gesture, we’ll count how many times the X values goes from positive to negative and back. If the count is bigger than [MinimumRequiredMovesForShake](#_MinimumRequiredMovesForShake), we’ll finally raise the shake gesture event!

# Configuring Parameters

During the coding and testing of the library, we found that shake is very individualized. It depends on the nature of the application and the actual shake gesture made by the phone’s user. There are alternative algorithms available that are adaptive and learning algorithms. While these might be more accurate, they require a learning phase that we didn’t want to force on the end user and on the developer building the app. Therefore, we give you plenty of different properties to tweak and tune for your own purposes.

These parameters control various aspects of the gesture detection algorithm. By changing these parameters, you can change your application’s sensitivity to shakes and to the duration of a shake. All parameters have default values kept as constants in the library.

The following section describes the available parameters. Each parameter is mentioned in context in the above algorithm.

## ShakeMagnitudeWithoutGravitationThreshold

**Description**: Any vector that has a magnitude (after reducing gravitation force) bigger than this parameter value is considered as a shake vector.

**Default value**: 0.2

## StillCounterThreshold

**Description**: This parameter determines how many consecutive still vectors are required to stop a shake signal.

**Default value**: 20 (about 400 msec)

## StillMagnitudeWithoutGravitationThreshold

**Description**: This parameter determines the maximum allowed magnitude (after reducing gravitation) for a still vector to be considered.

**Default value**: 0.02

## MaximumStillVectorsNeededForAverage

**Description**: The maximum number of still vectors needed to create a still vector average. Instead of averaging the entire still signal, we just look at the top recent still vectors. This is performed as runtime optimization.

**Default value**: 20

## MinimumStillVectorsNeededForAverage

**Description**: The minimum number of still vectors needed to create a still vector average. Without enough vectors, the average won’t be stable and thus will be ignored.

**Default value**: 5

## MinimumShakeVectorsNeededForShake

**Description**: Determines the number of shake vectors needed in order to recognize a shake.

**Default value**: 10

## WeakMagnitudeWithoutGravitationThreshold

**Description**: Shake vectors with a magnitude lower than this parameter won’t be considered for gesture classification.

**Default value**: 0.2

## MinimumRequiredMovesForShake

**Description**: Determines the number of moves required to get a shake signal.

**Default value**: 3

# Summary

In this document we showed how you can use the Shake Gestures Library in your application. We did an in-depth tour of the algorithms used for recognizing shakes. Finally, we skimmed through the various parameters that you can change to better fit gesture detection to your application’s needs.

Shake it up!