HackMe 2.0 - XperienZ

HKME-26 – Head Movement Detection Using HTML5 Device Orientation API

Version 1.0

Published on 01/10/2017

Document Revision History

Date	Version	Description	Author
01/09/2017	1.0	First Draft	Saurabh Nutan Sohoni

References

No.	Artefact Name
1.	https://mobiforge.com/design-development/html5-mobile-web-device-orientation-events
2.	https://github.com/ajfisher/deviceapi-normaliser
3.	https://dev.opera.com/articles/w3c-device-orientation-usage

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to guide developers in usage of HTML5's deviceOrientation API. This API makes use of readings from compasses, accelerometers and gyroscope sensor in a mobile device and outputs the current orientation of the device in terms of three angles against 3 axes of rotation.

1.2 Scope

Scope of this project is limited to development of a simple web application which makes use of only values 'alpha' angle as received from deviceOrientation API. In a typical VR application, detecting head movements is essential for identifying user interaction with the rendered scene/game/movie. When user is holds mobile device inside a VR headset, any head movement results in significant change of alpha angle values. This will be dealt in detail in the next section. Using alpha values, one can create an easy but intelligent head tracking web application.
In this application, alpha angle values received are mapped onto a linear slider with a bob inside to demonstrate current head position.

1.3 Hardware requirements

The only hardware required for this application is a mobile device. Majority of latest smartphones available these days have at least a compass, accelerometer or a gyroscope sensor or all of them in-built. These sensors are used by deviceOrientation API for detecting correct orientation and motion of the device.

1.4 Software requirements

This project uses HTML5 deviceOrientation API for interaction between orientation sensors present in a mobile device. Browser support of this API is as follows:

a. Chrome for desktop (6 and above)
b. Firefox for desktop (complete support in 6 and above)
c. Chrome for android (3 and above)
d. Firefox Mobile (complete support in 6 and above)

2. OVERVIEW OF CONCEPT AND METHODOLOGY USED

This section gives an overview of information available from deviceOrientation API. It also lists various angles and their importance w.r.t. orientation of device.

2.1 What is deviceOrientation API?

HTML5 Device Orientation API provides information about the orientation and movement of a device. Information comes from the positional sensors such as compasses, gyroscopes and accelerometers. Via this API, a web app can access and make use of information about how a device is physically oriented in space.

The HTML5 Device Orientation API specifies three events, which are outlined below:

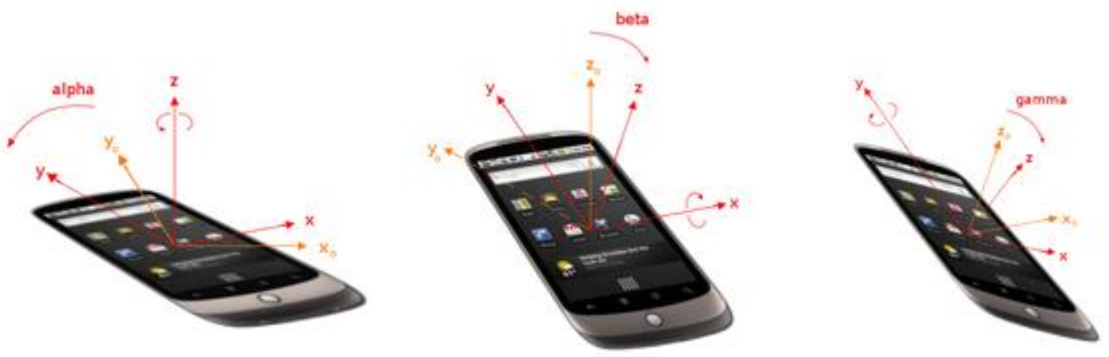
a. deviceorientation – Fired when significant orientation occurs
b. compassneedscalibration – Fired when compass needs calibration and calibration will provide more accurate data
c. devicemotion – Fired regularly with information regarding the motion of device

2.2 Angles associated with device orientation

The API returns raw values of different angles viz. alpha, beta and gamma defined below and illustrated in the figure:

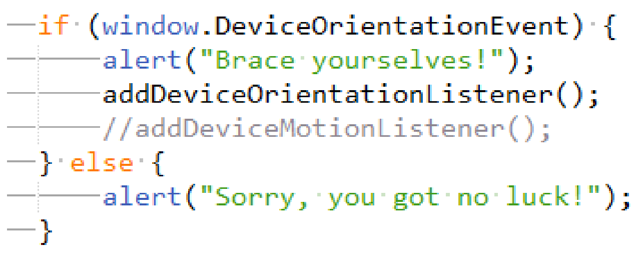
a. Alpha - The direction, the device is facing with rotation allowed around Z axis.
b. Beta – The direction, the device is facing when it is tilted from front to back
c. Gamma – The direction, the device is facing when the device is tilted left to right

Graphic shown below will list all the angles described above:



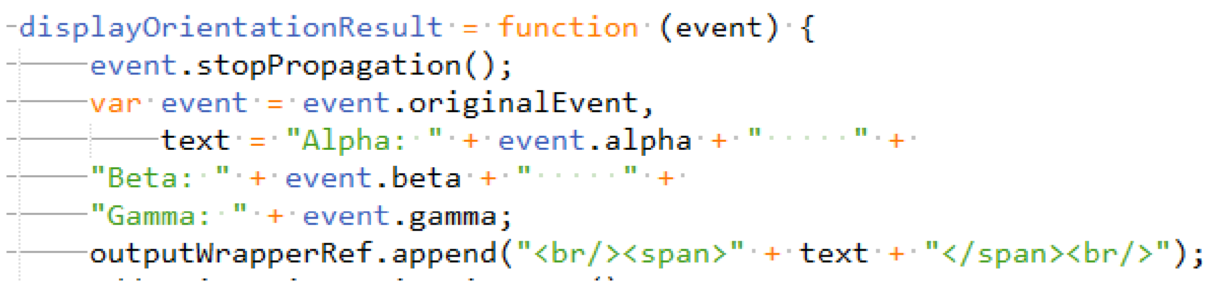
2.3 Accessing orientation data

Browser support and availability of the API must be checked before using it in any application. This can be done by using the code snippet given below:
It checks for availability of the DeviceOrientationEvent on global window object. Events described above will only be triggered if the current browser is supported.



```
if (window.DeviceOrientationEvent) {
    alert("Brace yourselves!");
    addDeviceOrientationListener();
    //addDeviceMotionListener();
} else {
    alert("Sorry, you got no luck!");
}
```

To access the orientation data, we need to attach an event listener to the deviceorientation event. And in the handler function we access the data with event.alpha, event.beta, and event.gamma as shown in the code snippet below:



```
displayOrientationResult = function (event) {
    event.stopPropagation();
    var event = event.originalEvent,
        text = "Alpha: " + event.alpha + "     " +
    "Beta: " + event.beta + "     " +
    "Gamma: " + event.gamma;
    outputWrapperRef.append("<br/><span>" + text + "</span><br/>");
```

3. MOVEMENT TRACKING IN ACTION

We are all set to launch our application now. Sample code base is included in the root folder where this guide is found. In this application, only alpha angle values are used.

Follow following steps to run the code:

a. Go to root folder and execute command: npm install
b. Start the server by command: npm start
c. Application will be available on localhost on port no 8000

Few guidelines on code developed are given below:

3.1 Degree to Radian conversion

The range of values of angle alpha as provided by deviceOrientation API are from 0 degree to 360 degree. To ease out mapping of data on a linear scale and involved calculations, these angles will be converted into radians by using the formula given below:

$$\text{Radians} = \left(\frac{\pi}{180^0}\right) \times \text{degrees}$$

3.2 Catching the offset

Since device orientation obtained is based data from compass, there is a fixed direction, north, where the value of angle alpha is 0 or 360 degree. But the angle to which user is facing while using VR headset cannot be predicted. Hence, an offset needs to be calculated based on the current angle of orientation of user. Code snippet below loops for 50 times (10 times in second) to get accurate value of the offset:

```
        offset = alpha;
        offset = (offset*pi)/180;
        range = [offset - viewingRange, offset + viewingRange]
    }
    offsetCounter += 1;
```

3.3 Range of head movement

While viewing a VR application, the end user mostly moves his/her head in a horizontal span of 90 degree or pi/2 radians. Hence, in this application, the range or span of values after calibration fall in an interval of pi/2 radians. Code snippet given below does this task:

```
if (alpha > range[1]) {
    alpha = range[1];
} else if (alpha < range[0]) {
    alpha = range[0];
}
normalizedAlpha = ((alpha-range[0])*boxWidth)/(pi);
```

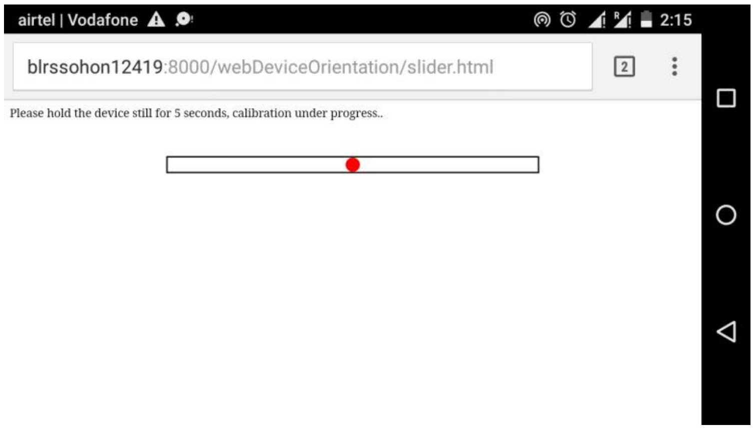
3.4 Convert orientation to linear axis

This is the last step in which the normalized alpha calculated above is converted to map to a linear box. The blob in the center of the box is moved by making use of CSS property 'margin'. In this example, the total width of box is 520 pixel with the width of blob being 20 pixel. This gives the total width for motion of blob to be 500 pixel. The code snippet given below makes the necessary adjustment to CSS margin property of slider blob:

```
rollerRef.css({
    marginLeft: normalizedAlpha > (boxWidth/2) ?
    ((boxWidth/2) - (normalizedAlpha - (boxWidth/2))) :
    (((boxWidth/2) - normalizedAlpha) + (boxWidth/2))
});
```

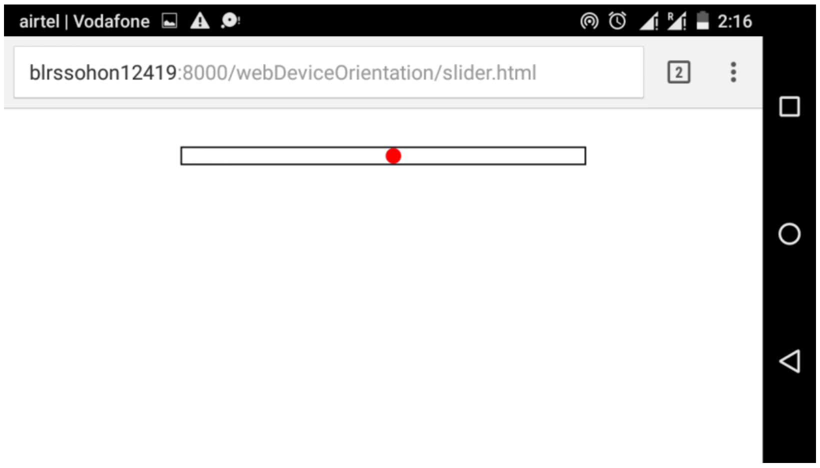
Few snapshots taken during the test run of application are given below:

a. Calibration in process:

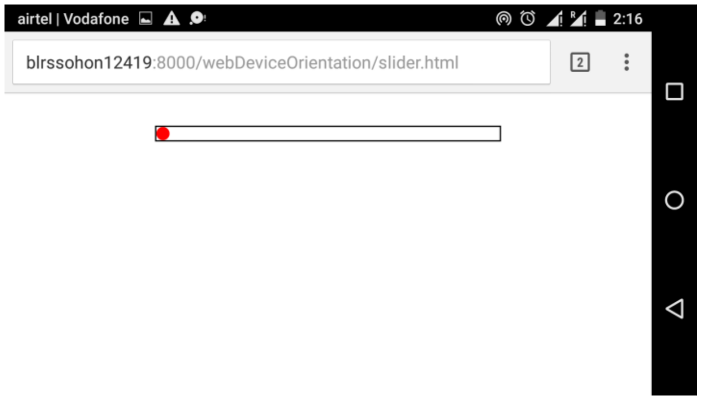


Please hold the device still for 5 seconds, calibration under progress..

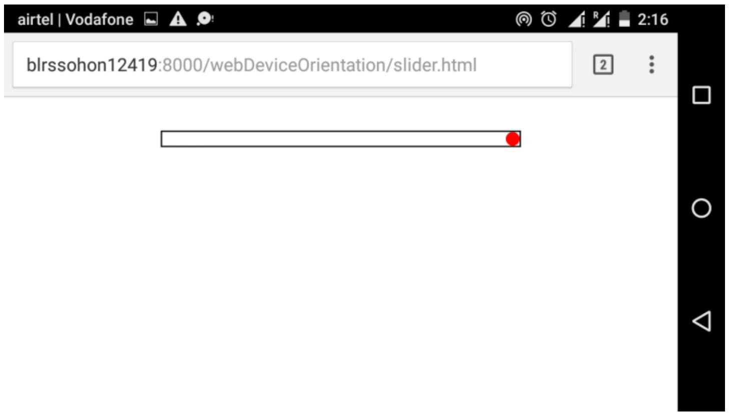
b. Initial position of head after calibration:



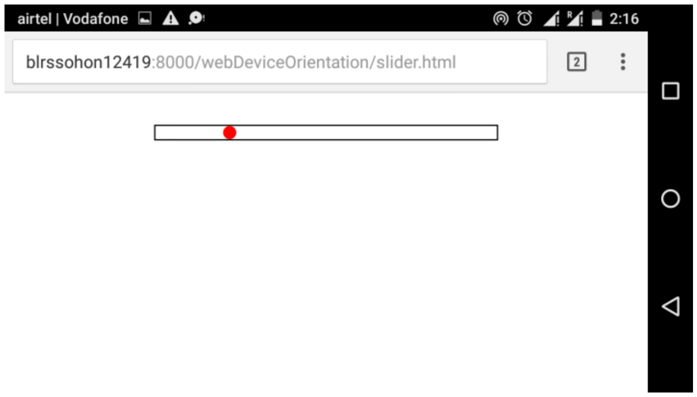
c. Head position at the left most corner and/or beyond it:


airtel | Vodafone 2:16
blrssohon12419:8000/webDeviceOrientation/slider.html

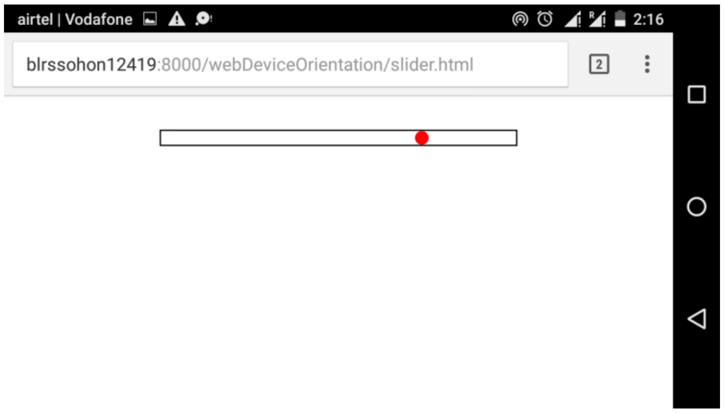
d. Head position at the right most corner and/or beyond it:


airtel | Vodafone 2:16
blrssohon12419:8000/webDeviceOrientation/slider.html

e. Head position between mid and left end:


airtel | Vodafone 2:16
blrssohon12419:8000/webDeviceOrientation/slider.html

f. Head position between mid and right end:


airtel | Vodafone
2:16
blrssohon12419:8000/webDeviceOrientation/slider.html
2