Document serial number: 000004892117

Touch Measure Tutorial

000004892117

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

© 2014 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Document serial number: 000004892117

Table of Contents

000004892117

Document serial number: 000004892117

About This Document

Purpose

This document explains techniques used in the "Touch Measure Tutorial" sample code (see Other Resources).

Audience

This document is intended for engineers who are interested in implementing fast-response touch controls or testing touch capabilities with the goal of improving or prototyping their user interfaces. A basic knowledge of real-time rendering is assumed.

Typographic Conventions

The typographic conventions used in this guide are explained in this section.

Text

- Names of keyboard functions or keys are formatted in a bold serif font. For example, **Ctrl**, **Delete**, **F9**.
- File names, source code, and command-line text are formatted in a fixed-width font. For example:

```
m_targetBufferData[idx]);     // pointer to the surface data
```

Other Resources

Updates/Errata

Any updates or amendments to this guide can be found in the release notes that accompany the SDK release packages.

Sample Code

The sample code is located in the PlayStation®Vita SDK package at:

```
%SCE_PSP2_SDK_DIR%/target/samples/sample_code/input_output_devices/
tutorial_touch_measure
```

000004892117

Document serial number: 000004892117

1 Introduction

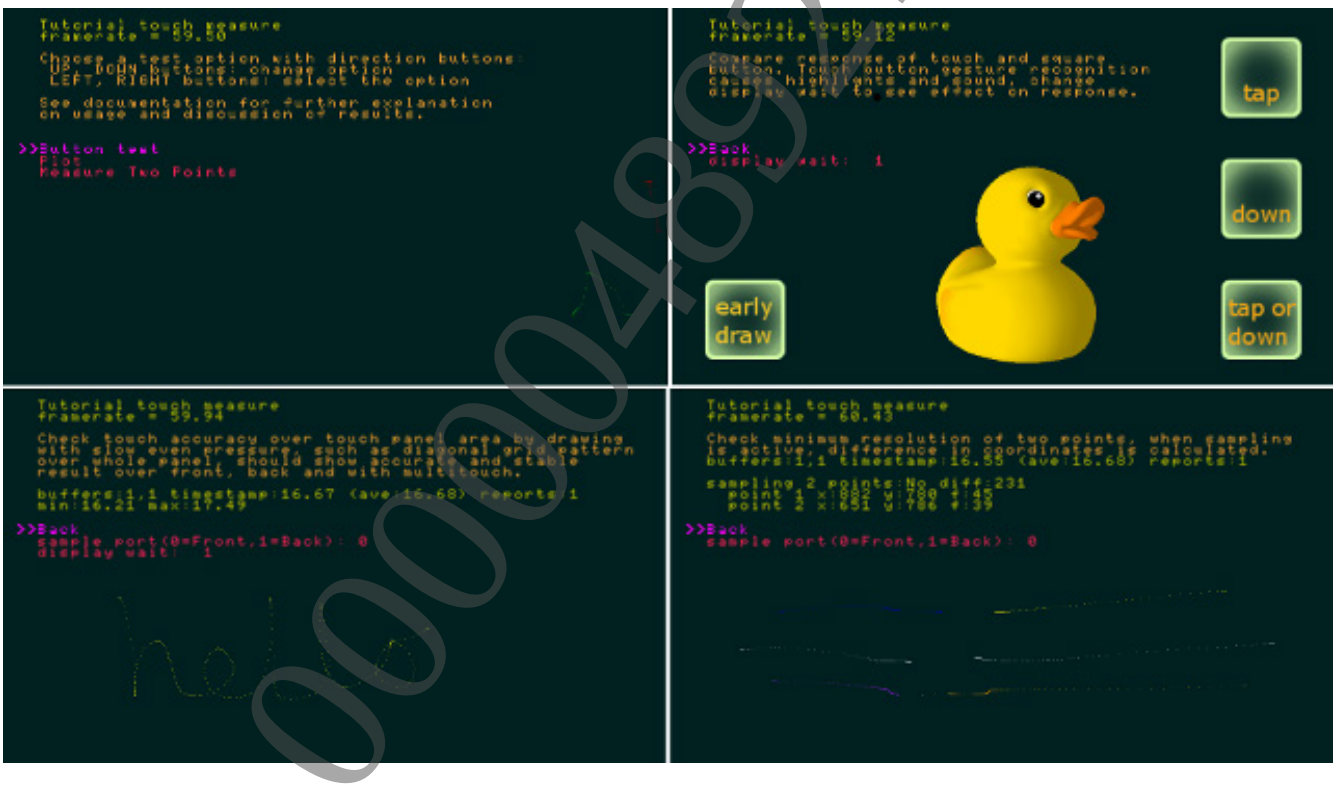
The “Touch Measure Tutorial” sample code demonstrates both the usability of touch buttons when a fast reaction is required and touch-data plotting for measurement of additional capabilities of touch panels. This document provides implementation details and examples, as well as an explanation of various measurements and effects that can be observed when running the sample.

The sample offers three test options:

- The button test, in which a 3D model of a duck and four on-screen buttons are displayed. Each button reacts to touch with a different method of gesture recognition or highlight rendering. Also, both the touch buttons and the square (□) button trigger the application to play a sound and change the 3D model rotation. Therefore, user interaction and the effect of latency on different types of buttons can be compared. Additionally, the frame rate can be adjusted, and changes in the response of different buttons due to the application running at lower frame rates can be observed.
- The plot test, which allows additional measurements of touch-screen properties to be observed. Touch-data plotting can be used to test the accuracy of touch detection across touch-panel areas.
- The measure-two-points test, which allows the resolution of two touch points to be measured.

Figure 1 shows screenshots of the tutorial’s main menu and the three test options mentioned above.

Figure 1 Screenshots of Main Menu (upper left) and Test Options



Document serial number: 000004892117

000004892117

2 Overview

The sample demonstrates the following topics related to using the touch-screen user interface:

- Implementation of touchable buttons, showing fast response and handling frame drop
- Examples of expected latency in touch detection and display
- Display of touch-panel accuracy over the whole area of the touch panel
- Measurement of the resolution of two distinct touch points

Menu Settings and Text Display

You can navigate to and select the following main-menu options via the directional buttons:

- Button test
- Plot
- Measure Two Points

Throughout the sample, the triangle button hides the debug text, the circle button takes a screenshot, and the X button clears the plotted samples.

"Button Test" Option

This option shows the touch buttons and compares the response of the touch and square buttons. Touch button gesture recognition causes highlight rendering (a sound is also played and the 3D model rotation changes).

- **display wait** can be set to a value from 1 to 11; it represents the number of frames to wait between each display buffer swap. Therefore this setting controls the overall frame rate and determines the application update and render loop and how often touch data is read.

"Plot" Option

This option allows touch accuracy to be viewed over both touch-panel areas as touch-data coordinates are plotted as pixels on screen. The printed measurements provide further touch data collected by the sample.

- **buffers : x,y** shows the number of data buffers returned by `sceTouchRead()`, where x is the number of buffers from the front panel and y is the number of buffers from the back panel. Each data buffer contains all touch-data reports at a given timestamp.
- **sample port** menu option can be set at 0 or 1, where 0 represents the front touch port and 1 represents the back touch port. It determines which touch-panel data is used for the calculation or reporting of measurements.
- **timestamp** is the difference comparing timestamps of the most recent touch-data buffer at each update loop.
- **ave** is the average timestamp delta over the last continuous touch.
- **reports** the number of touch reports in the current data buffer; this is the number of touch points detected by the touch library.
- **min** and **max** are the minimum and maximum timestamp delta over the last continuous touch.
- **display wait** (see the "Button Test" Option section for an explanation). Changing the frame rate in this case also influences the number of data buffers returned and timestamps.

Document serial number: 000004892117

000004892117

"Measure Two Points" Option

This option allows the resolution of two points to be measured. When sampling is active, the difference in touch coordinates is calculated.

- **sample port** menu option can be set to a value of 0 or 1, where 0 represents the front touch port and 1 represents the back touch port. It determines which touch-panel data is used for the calculation or reporting of measurements.
- **sampling 2 points: (Yes/No)** indicates whether two touch points are currently detected and being sampled.
- **diff** shows the distance calculated between two sampled points, in touch coordinate units.
- **point 1 x: y: f:** shows the x and y coordinates and the force reading of the touch points being sampled. The coordinates are used to calculate the distance shown in **diff**.

000004892117

Document serial number: 000004892117

3 Implementation

Touch-Data Plot

The sample code plots data points to display a trace so that various tests can be performed. Each plot point corresponds to every touch report contained in the most recent touch-data buffer returned from `sceTouchRead()`, so only the first set of reports are used if multiple buffers are read. This is equivalent to an application that only requires the latest touch position for processing and for displaying visual feedback. Therefore the "touch data point" option allows you to see the effect of low frame rate on the sample points and on the number of buffers of history.

Each touch report is plotted as one pixel. The pixel position is calculated depending on the screen display's width and height and the touch-sensor coordinate range returned by the touch library, and its color is dependent on the touch-report ID. The data point is drawn by directly adding the color value to the frame buffer immediately before swapping display buffers.

At each frame update, the data from the most recent touch report is also stored and used for calculation of measurements. So each plotted pixel corresponds to one sample point used in the calculation of displayed timestamp delta. The X button resets the plot display as well as the internal sample buffers and measurements (when the internal sample buffer is full the plot data and display is automatically reset).

Touch Operation Buttons

The buttons on the touch screen (touch operation button) and the square button will trigger the same event, which causes the application to play a sound and to change the direction of model rotation. Therefore any differences in the feel of using each type of button or perceived lag should be due to either the differences in the methods used to implement each button or the physical differences between real and touch buttons.

In this tutorial, there are four different types of touch operation buttons that demonstrate the use of gesture detection and highlight rendering, which provides visual feedback (highlight/non-highlight) so that the user knows that the touch operation button was pressed. The time delay between a user touching the button and the highlighting can vary a great deal; this latency can be due to several factors including the processing of raw touch data and the display rate.

The following list provides a general overview of each type of touch button implemented in the sample.

- **tap** – uses tap gesture recognition and draws a highlight (displays a star mark in the top left corner of the button) if a tap operation is detected. This is implemented using the libSystemGesture tap gesture recognizer.
- **down** – uses gesture recognition and draws a highlight (displays a star mark on the left edge of the button) when a user touched the panel. This is implemented using the libSystemGesture primitive touch recognizer.
- **tap or down** – if tap is detected or a user touched the panel, draws the respective highlight. For example, if tap is detected, the "down" highlight should display first, closely followed by the "tap" highlight.
- **early draw** – if tap is detected or a user touched the panel, draws the respective highlight directly into the display buffer. The highlights in this case are plain squares: a large square for "touch/down", and a small square in the top right corner for "tap".

In the source code, the button class manages the behavior and state of button operation instances, whereas the Controller class updates and processes all touch-related data. In cases where frames might be dropped, more than one data set could be returned when the touch data is read. In such cases, each returned buffer is processed in order inside the Controller class for gesture events, and all events are recorded to avoid missing a gesture.

Document serial number: 000004892117

000004892117

Normally, the number of data buffers returned by the libtouch API is equal to the number of frames that were missed since the last call to the API. Because the Controller class stores a maximum of 12 buffers, it has the capacity to store raw touch data to support frame rates of approximately 5 fps. However, at such artificially slow update rates it is quite easy to have a succession of fast taps or other gestures within one frame and dealing fully with this case is beyond the scope of this sample. So, for example, at extremely slow frame rates two touches could result in the highlight flashing only once. However, if the frame rate is approximately 20 fps, the button class should not miss any touches.

Render Method

The default setting for a touch operation button is to render everything using the regular libgxm pipeline and a shader for drawing textured triangles. This will generally provide the best quality graphics and flexibility in code and assets.

The “early draw” button is set to draw the button highlight without using the normal shader pipeline. Instead it uses the same method as plot rendering, drawing by adding color value directly into the frame buffer immediately before swapping display buffers. This method of rendering is very restrictive and occurs inside the display callback so it is not generally recommended. The highlight implemented is a simple square shape, and although plotted pixel values could be changed it would be difficult to achieve the same quality of rendering as textured and blended polygons rendered through libgxm. However, it is included as a simple test because it represents a case of the smallest possible drawing delay.

000004892117

Document serial number: 000004892117

4 Analysis and Discussion

Demonstration of Touch-Panel Accuracy

The general accuracy can be demonstrated across the whole area of the touch panel if a user draws a grid of straight diagonal lines, at even pressure. An example of the resulting plot is shown in the screenshot shown in Figure 2.

The spatial accuracy of data over the entire screen is very good, although there is a small discrepancy near the screen edge; this is likely due to the average touch range point being reported – if half of your finger is off the edge of the panel, the average touch coordinate does not correspond to the whole area pressed by your finger. For similar reasons, the lines in the screenshot are not perfectly straight because it is difficult to keep the pressure exactly the same.

Figure 2 Screenshot of Grid Drawn Slowly over Whole Screen with Finger and Ruler

Demonstration of Touch-Panel Resolution

The resolution can be estimated for two simultaneous touch points on screen by making two approaching traces, getting as close as is physically possible and checking the closest point reading. When two touch points are detected, the text display for **sampling 2 points** is Yes; it switches to No when the two points become too close to resolve. In the latter case, the display continues to show the coordinates (x:,y:,f:) and the difference (diff) for the last points sampled.

Touch-screen resolution is approximately 1 to 1.5 cm, as specified in SDK Touch Service documentation. In practice, the observed resolution can be more variable and it can also be dependent on the user. For example, if a user makes a large contact area with her fingers, she may not be able to physically get her touch points close enough to observe the limitation due to the touch screen.

Table 1 and Figure 3 show examples of results for this test. The resolutions observed in normal use were within the expected range, regardless of the force reported or whether the front or back panel was tested. The resolution has implications for user-interface layout, because there is a limit on how close each touch

Document serial number: 000004892117

000004892117

control can be. Also, if you were to hold the console by placing your thumb immediately to the right of the "tap or down" button, it is not always possible to press the "tap or down" button anymore.

Mapping from touch coordinates to pixel sizes for art assets and layout will depend on the display width and height in your application. In this sample the screen size is 960x544, so, for example, the conversion for the x coordinate is:

1 pixel = panel width / display width = 1920/960 = 2 touch units

(For the y direction, the result is the same: 1088/544 = 2 touch units.)

And the relation between touch units and size on screen is:

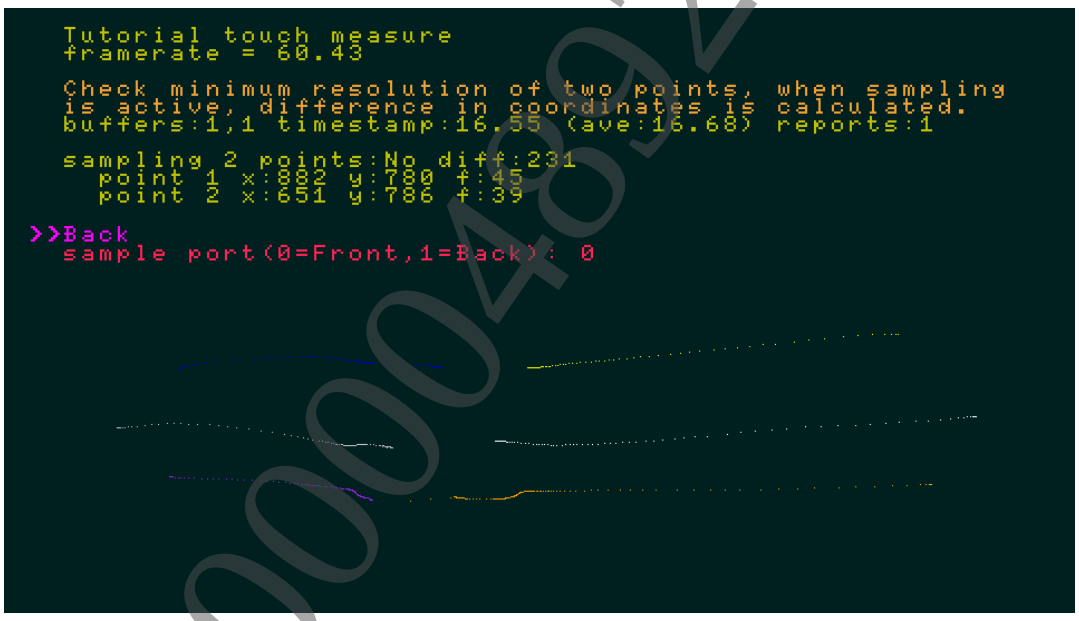
1 touch unit ≈ screen width / panel coordinate range = 11 cm / 1919 = 0.0057 cm

Table 1 Difference Between Two Points Readings*

Point 1: x,y,f	Point 2: x,y,f	diff	Size on screen (cm)
600, 696,122	883,788,86	177	1.0
683,627,100	906,612,78	223	1.3
1296,491,62	1041,499,61	255	1.5

* Note: Numbers and the conversion to screen size are approximate because coordinate ranges may change between different hardware versions.

Figure 3 Closest Distance Between Two Simultaneous Touch Reports



Latency of Touch Panel and Feedback to User

The perceived latency of the touch screen depends on several factors:

- Latency of the device
- Possible latency in `sceTouchRead()`/`sceTouchPeek()`
- Gesture recognition/other processing
- Delay in feedback to user (this sample mainly focuses on visual feedback due to rendering)

Details are discussed below.

Document serial number: 000004892117

Device and Read Latency

The update frequency from the libtouch API is similar to game pad controls and does not seem to change significantly with more than one simultaneous touch point or depending on whether the front or back panel is used. The Controller class gets touch data at each frame update using `sceTouchRead()`. Using `sceTouchRead()` instead of `sceTouchPeek()` is recommended if most up to date data is required; see the SDK documentation for details.

The difference in timestamps of successive touch data is approximately 16.67 ms, and the average rate is fairly stable. This is shown when the sample runs at 60 fps (`displaywait`=1); every call to API returns only one data set per panel (num buffer:1,1), and the average timestamp=16.xx.

Table 2 Timestamp Delta Readings

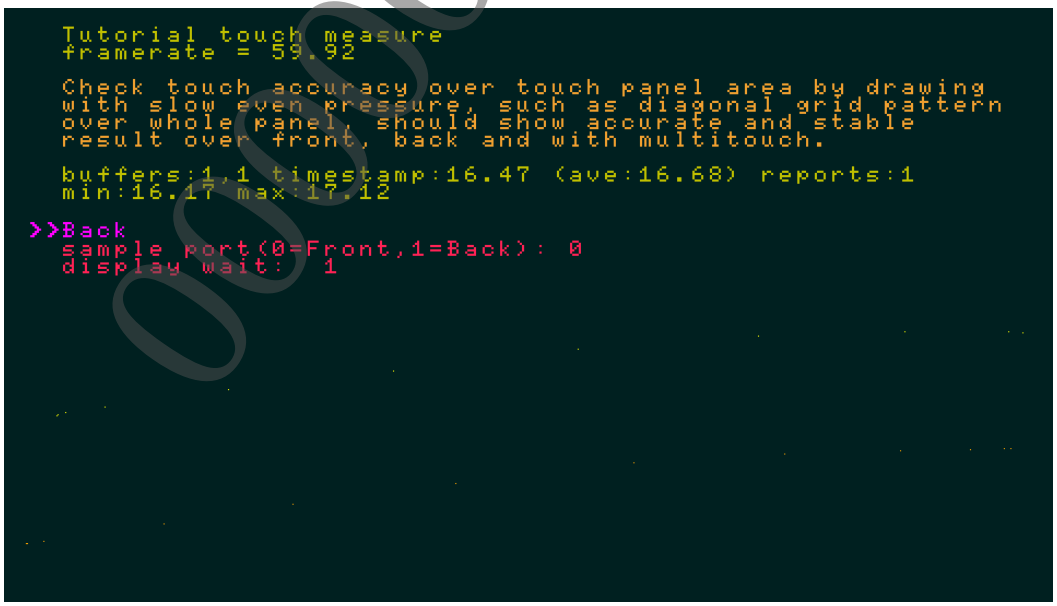
Current Delta (ms)	Average (ms)	Minimum (ms)	Maximum (ms)
16.60	16.67	16.39	17.13
16.68	16.68	16.27	17.17
16.56	16.65	16.41	17.12

It is easy to see the effect of this latency when a fast dragging trace is plotted, as in Figure 4. The spacing between each touch point can be large because a user can move his finger relatively far within 1 frame timing. For example, assuming drawing and sampling input at 60 fps, a user moving his finger at 1meters/sec means that in .016 seconds, he moves it 16 mm; so that will be the distance between sample points on the touch panel. During this movement, 16 mm will also be the distance between the plotted point and the actual position of the user's finger, which is quite noticeable. Figure 4 shows a similar example of two plotted traces. I moved my finger very quickly from one side of the screen to the other; each trace has pixels that are quite widely spaced.

Therefore, a control mechanism in which a drag-type gesture causes graphics to reposition or follow the exact touch point made by the user has some limitations. It can be implemented well if the movement is going to be relatively slow. In other cases, a bit of lag might be acceptable; sometimes it can be beneficial because it allows users to see what they are dragging.

Another SDK sample – Touch Trace – shows this well; moving fast on the front panel causes the square to lag a little.

Figure 4 Screenshot of Plot When Moving Finger Quickly



Document serial number: 000004892117

libSystemGesture Library

If touch data is processed to detect gestures using the libSystemGesture library, then, depending on the gestures operation trying to be recognized, the timing which the status of the gesture changed is different.. See the SDK documentation for detailed definitions of each gesture type.

The touch operation button implementation in this sample supports the following two gesture operations:

- **tapped** – touch gesture event retrieved from the libSystemGesture tap gesture recognizer.
- **down** – the begin state of primitive touch gesture event.

These are sufficient to demonstrate the difference in latency due to the gesture definitions on button press reactions. The tap operation is recognized if the difference in position and time between the initial touch until the finger is lifted is within a certain range. The time difference parameter used in the library is shared with the system software GUI. In contrast, the primitive touch detection can be set up to detect the event with no time delay. So if a faster response is needed, it is preferable to act on the button "down" state, or more generally use the touch report data directly when this is appropriate.

However, in many cases the short time delay in the tap operation is not a problem, and it may be preferable to keep consistent operation with other software that uses the tap gesture recognizer of the libSystemGesture library.

Audio Feedback

Simple audio feedback was added using the sound utility sample code. Current testing suggests that the latency between button activation and the sound being played is minimal (seeming to occur almost simultaneously), and as expected it did not make much difference when frame rates were changed. Therefore audio cues could be a useful way of giving user feedback for touch controls, or indeed for any input device in a situation when high frame rate cannot be assumed.

Visual Feedback and Rendering Method

Generally in libgxm rendering there is a 2-frame delay between (a) updating data input from the user and updating the graphics and (b) the user seeing the frame displayed on screen. This sample uses a triple buffered display system (see the libgxm documentation for pipeline information).

If the fps is 60, the user perceives visual feedback to be instantaneous (~32 ms); however, as the frame rate for update and rendering becomes slower, the delay increases between user input and the highlighting of the touched button.

Table 3 Estimated Lag When Update Rate Drops

Display Wait, fps	Timestamp Delta (ms)	Render Delay (ms)	Theoretical Min (ms)	Max (ms)
1, 60	~16.6	~32	32	48.6
3, 20	~50	~96	96	146

When the display wait is 3, meaning that the sample updates at 20 fps, the 3D model animation and reaction appear to be quite slow with any type of button. As expected, the tap button has the longest delay, with the highlight appearing long after the touch panel was first touched. In this case there is a small improvement in lag when drawing directly into the frame buffer; however, the gain is not very noticeable – in fact some users may not notice the difference at all.

So generally, if a fast reaction is required, it would be much better to keep the higher quality libgxm rendering of the "tap or down" button and to improve the overall frame rate of your game.

Summary

Touch screen and pad devices have similar response times; however, the processing and possibly the display of graphics may introduce further delay.

Document serial number: 000004892117

A small amount of latency is not necessarily a problem in some user interfaces, but when a quick (comparable to pad) response is required from a touch interface, it can be achieved under certain conditions: for example, by using the default setting on a button as implemented in the sample code.

The accuracy and resolution of the touch panels are good and allow very detailed touch input from a user with average finger size. As discussed in this chapter, there are some limitations related to touch input methods.

There are many design considerations when comparing touch input with traditional pad input, due to the physical differences of the input methods and the way users interact with them. For useful information about various input methods as well as user-interface design advice, see Other Resources in the About This Document section.

000004892117

Document serial number: 000004892117