Swift Gestures: Seamless Bend Gestures Using Graphics Framework Capabilities

Samudrala Nagaraju(✉)

Web and Services Team, Samsung R&D Institute Bangalore, Bangalore, India
raju.sn@samsung.com

Abstract. With the advent of bendable devices, Lahey et al. [1], explored bend gestures for mobile phone applications. Considering **millions of applications present on app stores** [2], it would be a challenge to modify source code to handle bend gestures. We propose a novel approach to assign bend gestures *using graphics framework capabilities, which does not require application source code changes*. Because of the ease in use of the proposed approach, bend gestures get acceptance from research community and industry.

Keywords: Bendable devices · Rendering engine · Graphics event processing · Bend gestures · Tangible interaction · Usability study · Rendering tree

1 Introduction

Various types of gestures like air gestures, shake gestures and write gestures are developed for hands-free input or partial-hands-free input to ease device interaction [3, 4]. With the advent of bendable devices, bend gestures [1] are being researched for technical feasibility and usability factors. Modern device platforms support downloadable applications for installation on the device. In this paper, we focus on practical issues in assigning bend gestures to applications in a uniform way by introducing logic in graphics framework without changing application code.

1.1 Definition of Terms

Definitions used in this paper are detailed based on [5]. Device platform – A software stack which includes operating system and middleware. Graphics Framework – Widgets and layouts are the primary elements for creating user interfaces using a graphics framework. Graphics Layout - A layout defines the visual structure for an activity or application widget. Widget - A graphical control element for interaction in a graphical user interface (GUI), such as a button or a scroll bar. View - A View is an object that draws something on the screen that the user can interact with. Graphics Event - Events are objects sent to an app to inform it of user actions e.g. multi-touch. Render Tree - This is generated from application UI and is responsible for the layout and subsequent rendering. Application Packaging - containers for application binaries, based on build settings e.g. apk file for each app. User - A smartphone or tablet user.

M. Kurosu (Ed.): Human-Computer Interaction, Part II, HCII 2015, LNCS 9170, pp. 118–129, 2015.
DOI: 10.1007/978-3-319-20916-6_12

1.2 Types of Applications

An application is classified based on the application shipment type along with the device and development ownership. *Pre-embed applications* are present in limited number and are shipped along with the device. *Downloadable applications* are developed by 3rd party companies and are hosted on app stores [2].

1.3 Graphics Frameworks for Application Development

State of the art device platforms provide at least 3 types of graphics capabilities for application development. (1) Native graphics framework (iOS UIKit, Android native widgets and layouts, etc.). (2) Web framework (HTML, CSS, JavaScript) and (3) Advanced graphics framework (OpenGL ES) [5]. Application developers make use of one or more of above mentioned graphics capabilities for application development. In this paper, we work with native graphics framework in Android platform.

1.4 Bendable Device

Bendable device research and development is in full swing from both academic and commercial fronts. Announcements from device manufacturers about the launch of these devices in market created lot of buzz around [6]. Numerous challenges are waiting with hardware, software, electro mechanical, battery and usability domains [7]. In parallel, this calls for taking the existing research needs to be accommodated into commercial setup to get acceptance in main stream development. This is one of the main goals of this paper for bendable device gestures.

2 Background

Background of this paper is focused in explaining the ongoing research in bendable devices and device frameworks essential for the topics discussed in further sections. Along with these topics, this section also focuses on how research teams are prototyping the bendable device concepts.

2.1 Bendable Devices and Gestures

Research is ongoing in multiple areas that are essential in developing the base technologies for bendable devices [8], e.g. user interactions, bend gestures and physical materials. These technologies are essential to make a significant shift in approaching and solving the problem space of bendable device products. Among the whole set of topics available for research, bendable gesture is a topic actively researched by both academic and industry because of the unique problem space not available earlier when flexibility in device physical characteristics is not thought of. In bendable gesture space, researchers are focusing on gesture interactions, gestures classification and

usability factors [9] for detailed study. Role of bend sensors in identifying the bend gesture parameters is also studied [10].

2.2 Bendable Device Prototypes

Researchers substantiated bendable device concepts with various types of prototypes which are classified as follows based on the material used for prototype development. (1) Low fidelity are developed using materials like paper and cloth [11]. (2) Near high fidelity emulations are developed using projected displays and software emulations [12]. (3) High fidelity are developed using device hardware. The scope of prototype emulation for Point (2) encompasses one or all of following. (1) Emulation of the proposed concept implementation. E.g. gestures classification. (2) Emulation of the device hardware. e.g. bendable mobile phone and bendable watch. (3) Emulation of the device sensors. E.g. bend sensors. In this paper, a software implementation of the proposed approach on android graphics framework using mobile phone hardware is used for implementation of "Swift Gestures". Emulation of bend sensors is achieved using a software based implementation on mobile phone hardware.

2.3 Device Platform

Some researchers extend the device capabilities by adding logic in one or more frameworks of device [13] without changing the API. But, some implementations require change in the API structure based on the extent to which control is exposed to 3rd party developers. In this paper, we use the earlier methodology which does not require changes in device platform API on Android.

3 Problem Statement

Focused research is ongoing with bendable device interactions and gesture assignment [10–12]. Even though this research is interesting and productive, bend gesture assignment procedures mentioned in these papers talk about changing application code for handling bend gestures. Second problem is the dependence of bend gestures on device bend physical parameters.

3.1 Source Code Changes

With introduction of bend gestures, application developers face unique problem adapting the legacy downloadable applications to handle these gestures, as it may not be feasible to change the application source code. Figure 1 (a) is Android sample application used to explain the source code changes required to handle bend gestures. Figure 1 (b) shows corresponding source code to Fig. 1(a) to display a dialog 'Dialog Title' on clicking a button titled 'Click Me'. Figure 1(c) shows source code changes required on top of Fig. 1(b) to handle bend gesture for the same action performed on

the button 'Click Me'. A source code change needs compile, package (APK generation in case of Android) and upload stages to upload a new version of the downloadable application to app store, which is not feasible for all apps on app store.

3.2 Device Bend Physical Parameters

Bend gestures depend on the physical properties of a device to the extent it is flexible and provide response to application. The role of physical parameters like stiffness, deformable range and feedback is studied [14]. Device size and aspect ratio also play an important role. In this paper, we focus to assign and use a bend gesture to an application in a uniform way with addition of additional logic to graphics frameworks of device platform. This is achieved by mapping the bend gestures to actions already registered by the application.

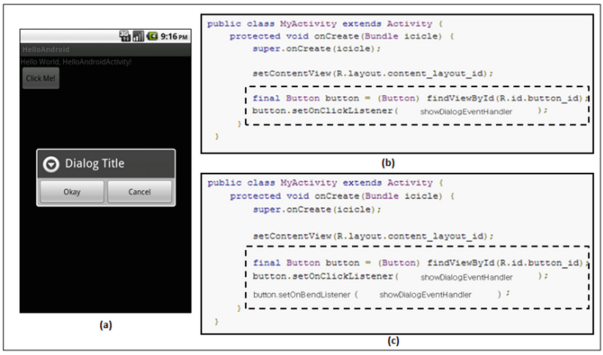

Fig. 1. (a) Sample application with clickable button and dialog (b) Action registration for click event (c) Same action registration extended for bend gesture

4 Proposed Approach

4.1 User Experience (UX) Flow

User experience flow for bend gesture assign and bend gesture invoke procedures is explained in detail in this section. Scope of a bend gesture can either be per application or for the entire device.

4.2 Bend Gesture Assign Procedure

Steps to assign bend gesture are as follows. (1) Select a subset of widgets from application layout at native graphics framework based on the render tree of application. (2) Selected widgets from application layout are highlighted as in Fig. 2 (c). Visual highlighting is done by native graphics framework. (3) User selects a widget from the highlighted list to assign gesture as in Fig. 2 (d). Let's call this as target widget. User does the selection by interacting with highlighted widgets. All interactions at this stage

are taken care by native graphics framework and no control is given to application. (4) Visual indication is given to target widget selected as in Fig. 2 (e) by native graphics framework. (5) User does a physical bend of the device and holds it for a short span of time as in Fig. 2 (e). At this stage, native graphics framework registers the bend gesture and assigns to target widget. This step makes bend gesture assignment independent of device physical parameters as in Sect. 3.2 (6) Visual indication is changed for the target widget post gesture assignment, as in Fig. 2 (f).

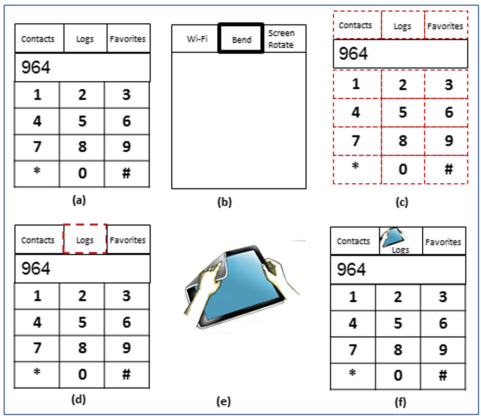

Fig. 2. UX Flow for Gesture Assign (a) Open target application view (b) Drop down notification bar, select 'bend gesture' assignment mode and drop back notification bar (c) Highlight widgets in target application layout (d) Select target widget for gesture assignment (e) Do actual device bend and hold (f) Target widget icon changed based on physical bend parameters

4.3 Bend Gesture Invoke Procedure

As we scoped the bend gesture to the entire mobile, same gesture cannot be duplicated at other applications and this check point is taken care by native graphics framework at bend gesture assign procedure. Post bend gesture assignment, device user needs to invoke the mapped action by physically bending the device as shown in Fig. 3 (d) from any application as shown in Figs. 3 (a) (b) (c).

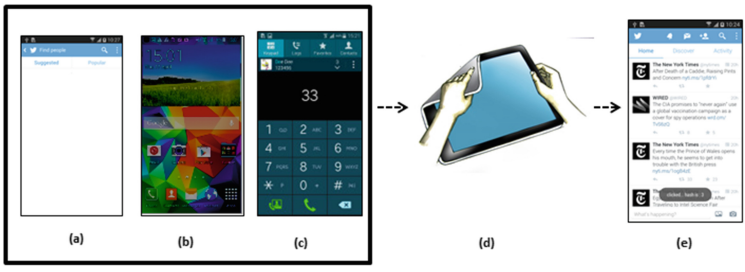

Fig. 3. UX Flow for gesture invoke (a)(b)(c) Invoke from any application (d) Physically bend device and hold (e) Invoke target application and registered action

4.4 Realization Using Android Device Platform

As described in the previous sub-sections of this section, device platform takes care of the following architectural changes for user experience procedures detailed. (1) Identify the subset of widgets from target application layout. (2) Highlight the subset of widgets in target application. (3) Accept the user click of target widget. (4) Highlight the target widget. (5) Capture the physical parameters of device bend. (6) Assignment of device bend gesture to target widget. (7) Changing the visual indication of target widget. (8) Handle gesture invoke trigger.

4.5 Modules Affected in Android Device Platform

Figure 4 highlights the modules affected in android device platform for implementing "Swift gestures". Key responsibility of each module for bend assign and bend invoke procedures is listed in Fig. 5. (1) Quick Setting Panel: This panel is a tiled pane on notification bar to access common settings and can be modified at device platform (2) View System: View occupies a rectangular area on the screen and is responsible for drawing and event handling used to create interactive UI components (buttons, text fields, etc.) (3) Activity Manager: An activity is a focused part of the visible application that the user can interact. Activity takes care of creating a window for application in which it can place a user interface. This is also used to interact with other Activities running in the system. (4) Window Manger: This is a software component that controls the placement and appearance of windows within a windowing system [5].

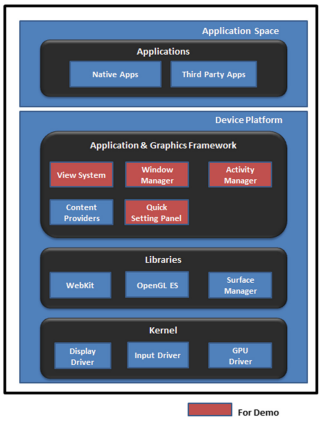

Fig. 4. Modules affected in device platform

4.6 State Transition Flow

Figure 6 shows states introduced at Activity Manger and Window Manger to distinguish among normal application interaction, gesture assign procedure and gesture invoke procedure. Description of each state is as follows (1) NO_BEND: Used for

Module	Responsibility
Quick Panel Setting	• Display notification panel button • Initiate Register Bend Session
View System	• Highlight Clickable widgets • Highlight Selected widget
Activity Manager	• Bend Gesture Callback Handle • State Change Initiation
Window Manager	• State Management • Differentiated Event Handling • Store Bend Gesture Assignment • Invoke mapped action to Bend Gesture

Fig. 5. Module responsibilities for handling bend assign and invoke procedure

normal user interaction with device. (2) HIGHLIGHT_WIDGETS: Used for highlighting the shortlisted widgets and providing visual indication as in Fig. 2(c) (3) ASSIGN_BEND: This state is active from selection of target widget till the physical bend as in Figs. 2 (d) (e) (4) BEND_GESTURE_ASSIGNED: This state is active post bend assignment procedure as in Fig. 2 (f).

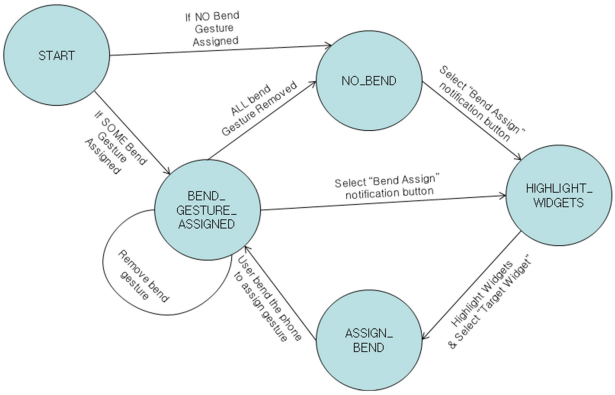

Fig. 6. State transition flow for handling bend assign and invoke procedures

5 Prototype Setup

'Swift gestures' explained in this paper is software driven and hardware dependence is on mobile hardware and the availability of bend sensors with the hardware. We developed the implementation using two mobile phones. First device is demo prototype, which has changes implemented to handle bend gesture assign and invoke procedures on device platform as shown in Fig. 8. Second mobile is bend emulator, on which bend emulation application is developed. Bend emulator is developed on Samsung galaxy S3 on Android ICS platform and is connected to demo prototype over Wi-Fi. Gestures from [1] are reused for bend emulation as shown in Fig. 7 (b).

Dialer is a preloaded application on the mobile phone and ChatON is a downloadable application from app store. We chose these two applications to work with

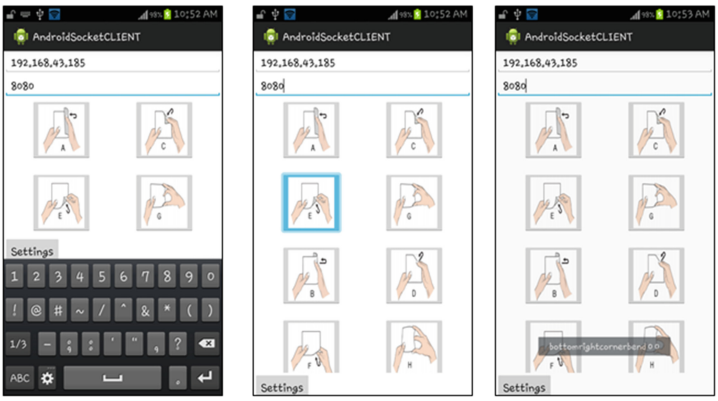

Fig. 7. BEND EMULATOR APPLICATION (a) Input peer ip address and port no on bend emulator application (b) Select bend gesture from the gesture list on bend emulator application (c) Send bend gesture application to peer demo prototype device.

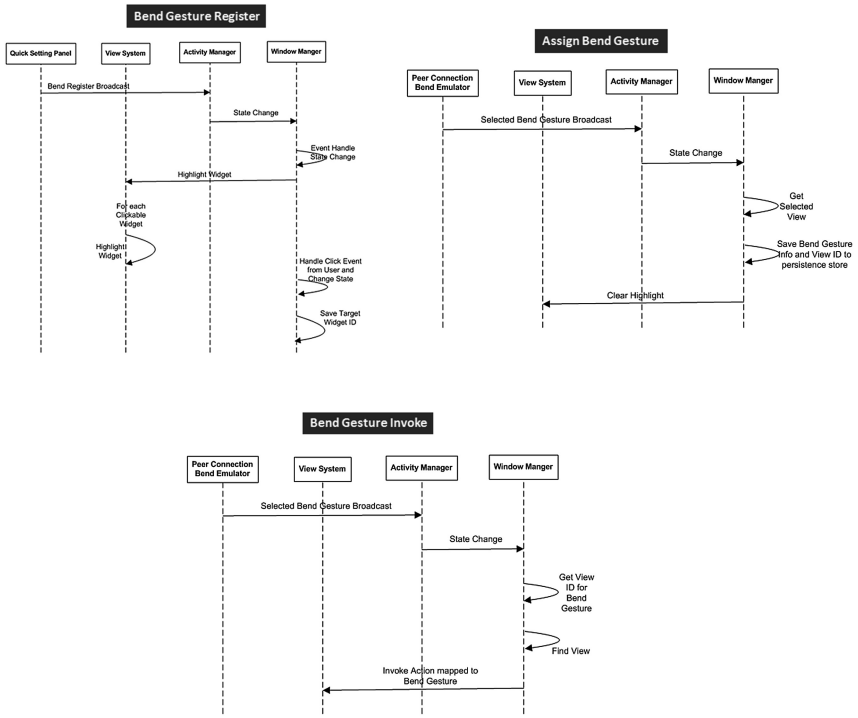

Fig. 8. Call sequence on demo prototype device (a) Bend gesture assign procedure – involves peer connection bend gesture receive, selection of quick settings panel and handling at device platform (b) Bend gesture invoke procedure.

Fig. 9. GESTURE ASSIGN (a) Open Target Screen (b) Drop down notification bar and select bend register (c) Highlight selectable widgets and select target widget (d) Select bend type from bend emulator and assign gesture on demo prototype. GESTURE INVOKE (e) optionally open the desired screen (f) Select bend type from 'Bend gesture emulator' (g) This triggers the action registered on the target widget (Delete chat action for ChatON& Last call action for Dialer).

'Swift gestures' as bend assign and invoke procedures need to work with both preload and downloadable applications. These applications are installed on demo prototype mobile, a Samsung Galaxy S5 running on Android Kitkat platform. ChatON version used is v3.5 downloaded from Google play store [2]. Steps to connect both the mobiles are as follows (1) Launch connection app in both mobile phones (2) Input peer ip address and port no (3) Establish the connection.

6 Bend Emulation and Prototype Usage

The demo prototype mobile is referred as "Mobile A" and bend emulator is referred to as "Mobile B" in Fig. 10. Procedure to work with the setup post connection establishment is as follows. (1) Launch Dialer or ChatON application on demo prototype device (2) Select a bend gesture from the predefined list (3) Send the selected bend gesture parameters – bend position, bend angle, direction of bend, bend extent - to demo prototype mobile. *Bend position* parameter are – top, bottom, left, right – based on the position at which device is bent. *Bend angle* gives the angle at which device is bent between 0 and 90. *Bend direction* is either inward or outward. *Bend extent* is the

distance from edge where bend angle is calculated. Figure 9 gives the screen shots of bend assign and invoke procedures. Based on the definitions given in Sect. 4, target widget for ChatON is "Delete Chats" button and "Initiate Call" button for Dialer application. Figure 8 gives the sequence flow of the implementation.

Fig. 10. Hierarchy Viewer for ChatON application (a) Screen without any highlighting (b) Selected widgets highlight (c) Target widget highlight

7 Performance Overhead

As overall rendering time is an important consideration for graphics framework, we captured rendering time using hierarchical viewer debug tool provided by Android, which provides total screen render time in terms of Measure, Layout and Draw parameters [5]. As shown in Fig. 10, rendering time of a screen for is measured for (a) Original screen when opening target application (b) Highlight widgets in target application and (c) Select target widget for gesture assignment. The average total rendering time is (a) 55.144 ms (b) 57.107 ms (c) 56.181 ms. Overhead for (b) is 1.963 ms (c) is 1.037 ms, which are minimal considering gesture procedures are not used frequently.

8 Usability Study

"Swift gestures" work with legacy applications available in market and so user evaluation is required to gain acceptance. We divided the users to two sets in the age group of 23 and 28 years. First set consist of 7 users aware of developing 3rd party applications. Second set consist of 5 novice users who use smart phones.

8.1 Usability Evaluation Sessions

Usability evaluation sessions are conducted for both the user sets as follows. (1) Moderator introduced "Swift gestures" as a presentation to the participants (2) Bend sensor emulation using mobile application is briefed. (3) Prototype setup using the two

mobiles connected over Wi-Fi is introduced. (4) Users perform the bend gesture assign and invoke procedures with Dialer and ChatON applications. (5) Capture feedback from user using feedback forms. (6) Discussion based on feedback.

8.2 Feedback Questionnaire

Feedback involved answering quantitative rating as follows - between 1 and 5, 1 being highly usable - (1) Ease of using the bend assign task (2) Ease of identifying the widget for which a bend gesture is assigned (3) Ease of using the bend gesture invoke task. These are followed by descriptive questions as follows users can provide comments for these questions. (1) Improvements suggested for bend gesture assign procedure (2) Improvements suggested for bend gesture invoke procedure.

8.3 Evaluation Results

Average rating for quantitative tasks is 2.2, 3.1 and 2.4. Based on the feedback, bend assign gesture task is relatively usable when compared to other two tasks. Comments provided for the descriptive questions are consolidated as follows. (1) Seven participants felt that there needs a limit to the number of target widgets highlighted as sometimes screen looks cluttered. (2) Six participants felt that a preview of the widget action e.g. button click, is helpful. (3) Majority of participants (10 totals) felt a need to list the assigned bend gestures at a common place for latter reference. (4) Users expressed satisfaction as preload and download app interaction is uniform. Based on the comments, recommendations are (1) Widget highlighting can be controlled by graphics framework or user. (2) Assigned bend gestures are listed at a common place like settings.

9 Conclusion

In this paper, bend gestures are assigned to applications without changing source code based on graphics framework capabilities. User experience design and implementation details in device platform are described which give minimal performance overhead and validated based on a usability study. “Swift gestures” takes care of problems caused by device stiffness and deformation parameters as device bend parameters are collected dynamically while assigning.

One limitation of this work is that it doesn’t take into consideration if ‘associated data with gesture’ is relevant at bend gesture invoke time. Swift gestures’ can be extended to scroll, swipe, etc. giving due importance to user experience. We thank G Purushothama Chowdari and C Krishna Bharadwaj for their inputs in implementation and demo.

References

1. Lahey, B., Girouard, A., Burleson, W., Vertegaal, R.: Paper-phone: understanding the use of bend gestures in mobile devices with flexible electronic paper displays. In: CHI 2011 (2011)

2. App Stores: play.google.com, itunes.apple.com. www.samsungapps.com
3. Suarez, J., Murphy, R.R.: Hand gesture recognition with depth images: a review. In: 2012 IEEE RO-MAN (2012)
4. Gesture Play. www.panasonic.com
5. Developer API. https://developer.apple.com, http://developer.android.com
6. www.samsung.com/sec/galaxyround
7. Nagaraju, S.: Novel user interaction styles with flexible/rollable screens. In: CHItaly 2013, Article No. 20 (2013)
8. Samsung Graphene Structure. http://www.sait.samsung.co.kr/saithome/AboutView.do?method=get&newSeq=1091
9. Khalilbeigi, M. Lissermann, R., Mühlhäuser, M., Steimle, J.: Xpaaand: interaction techniques for rollable displays. In: CHI 2011 (2011)
10. Warren, K., Lo, J., Vadgama, V., Girouard, A.: Bending the rules: bend gesture classification for flexible displays. In: CHI 2013 (2013)
11. Wolf, K., Müller-Tomfelde, C., Cheng, K., Wechsung, I.: PinchPad: performance of touch-based gestures while grasping devices. In: Proceedings of TEI 2012, pp. 103–110 (2012)
12. Steimle, J., Jordt, A., Maes, P.: Flexpad: highly flexible bending interactions for projected handheld displays. In: Proceedings of CHI 2013 (2013)
13. Machiry, A., Tahiliani, R., Naik, M.: Dynodroid: an input generation system for Android apps. In: proceedings ESEC/FSE 2013, pp. 224–234 (2013)
14. Johan, K., Graham, W.: Feeling It: The roles of stiffness, deformation range and feedback in the control of deformable UI. In: Proceedings of ICMI 2012 (2012)