US 20110202834A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2011/0202834 A1**

Mandryk et al. (43) **Pub. Date: Aug. 18, 2011**

(54) VISUAL MOTION FEEDBACK FOR USER INTERFACE

(75) Inventors: **Luciano Baretta Mandryk**, Seattle, WA (US); **Jeffrey Cheng-Yao Fong**, Seattle, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(21) Appl. No.: **12/773,803**

(22) Filed: **May 4, 2010**

Related U.S. Application Data

(60) Provisional application No. 61/304,004, filed on Feb. 12, 2010.

Publication Classification

(51) **Int. Cl.**

G06F 3/01	(2006.01)
G06F 3/033	(2006.01)
G06F 3/048	(2006.01)

(52) **U.S. Cl.** **715/701**; 715/863; 715/800

(57) **ABSTRACT**

Aspects of a user interface that provides visual feedback in response to user input. For example, boundary effects are presented to provide visual cues to a user to indicate that a boundary in a movable user interface element (e.g., the end of a scrollable list) has been reached. As another example, parallax effects are presented in which multiple parallel or substantially parallel layers in a multi-layer user interface move at different rates, in response to user input. As another example, simulated inertia motion of UI elements is used to provide a more natural feel for touch input. Various combinations of features are described. For example, simulated inertia motion can be used in combination with parallax effects, boundary effects, or other types of visual feedback.

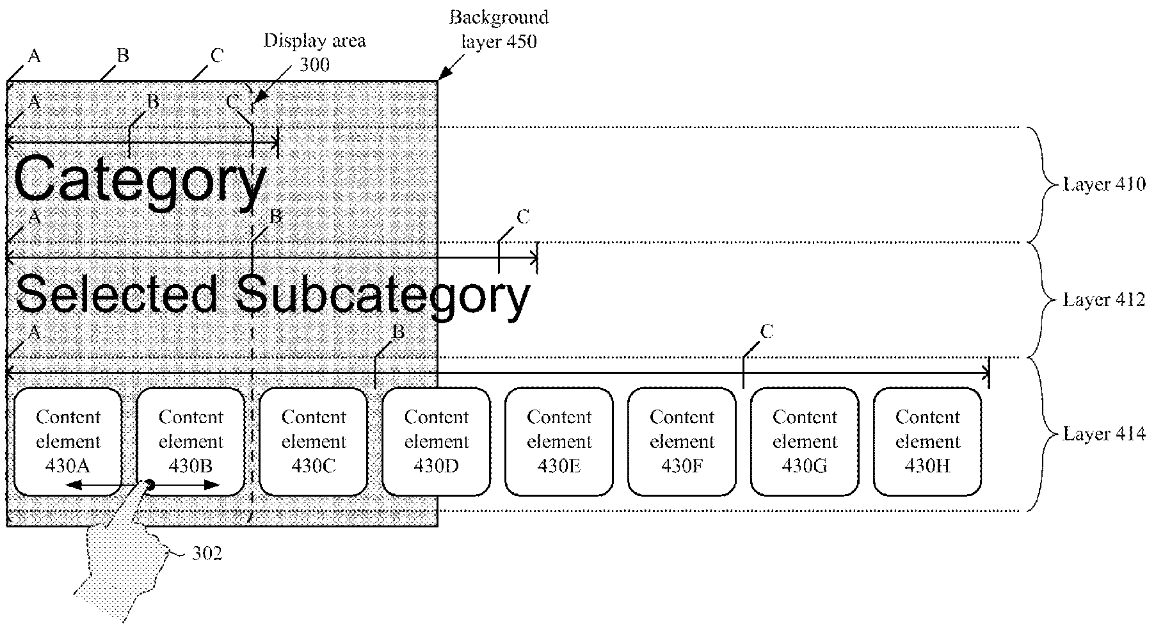


Figure 1A

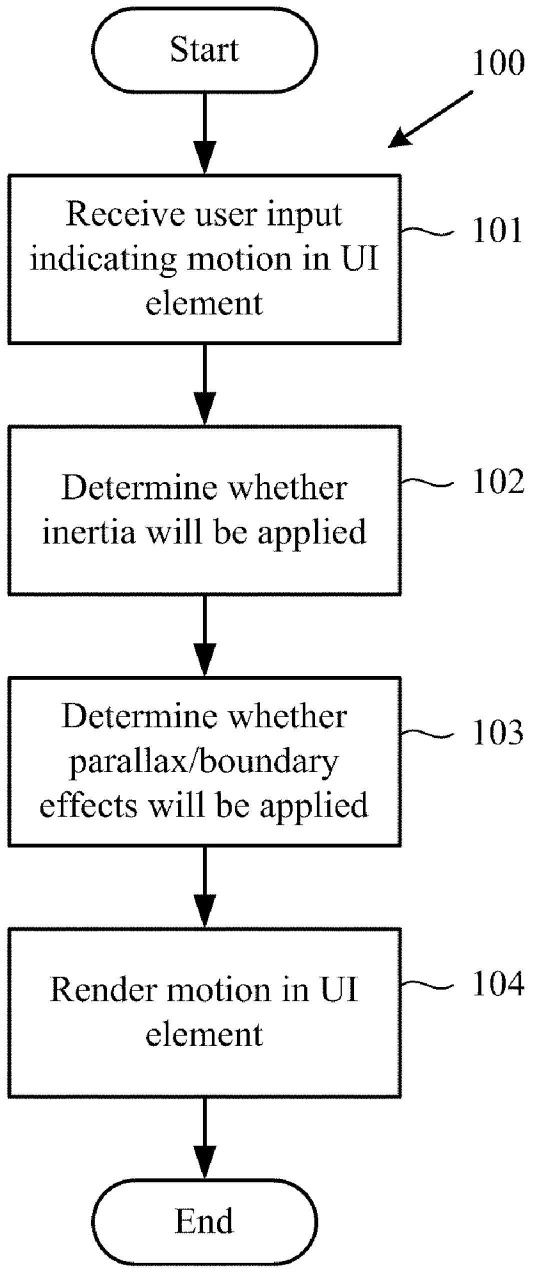

Start
100
Receive user input indicating motion in UI element
101
Determine whether inertia will be applied
102
Determine whether parallax/boundary effects will be applied
103
Render motion in UI element
104
End

Figure 1B

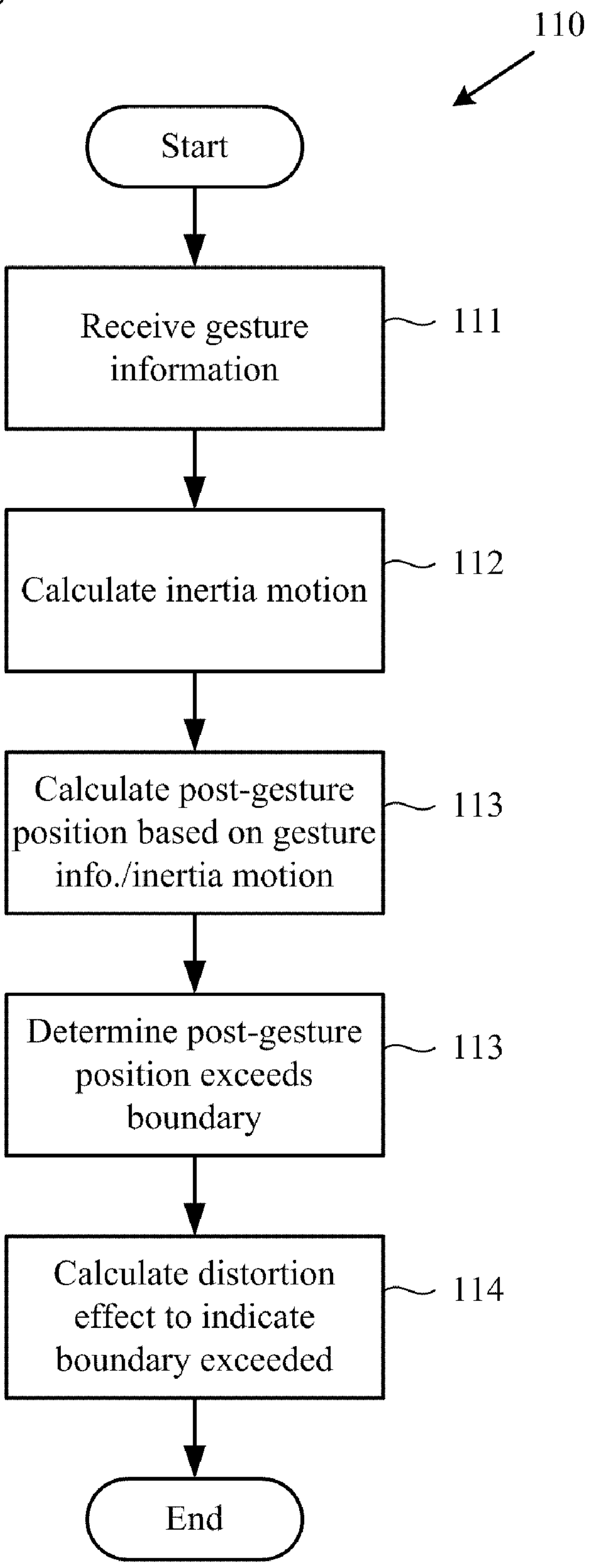

110
Start
Receive gesture information
111
Calculate inertia motion
112
Calculate post-gesture position based on gesture info./inertia motion
113
Determine post-gesture position exceeds boundary
113
Calculate distortion effect to indicate boundary exceeded
114
End

Figure 1C

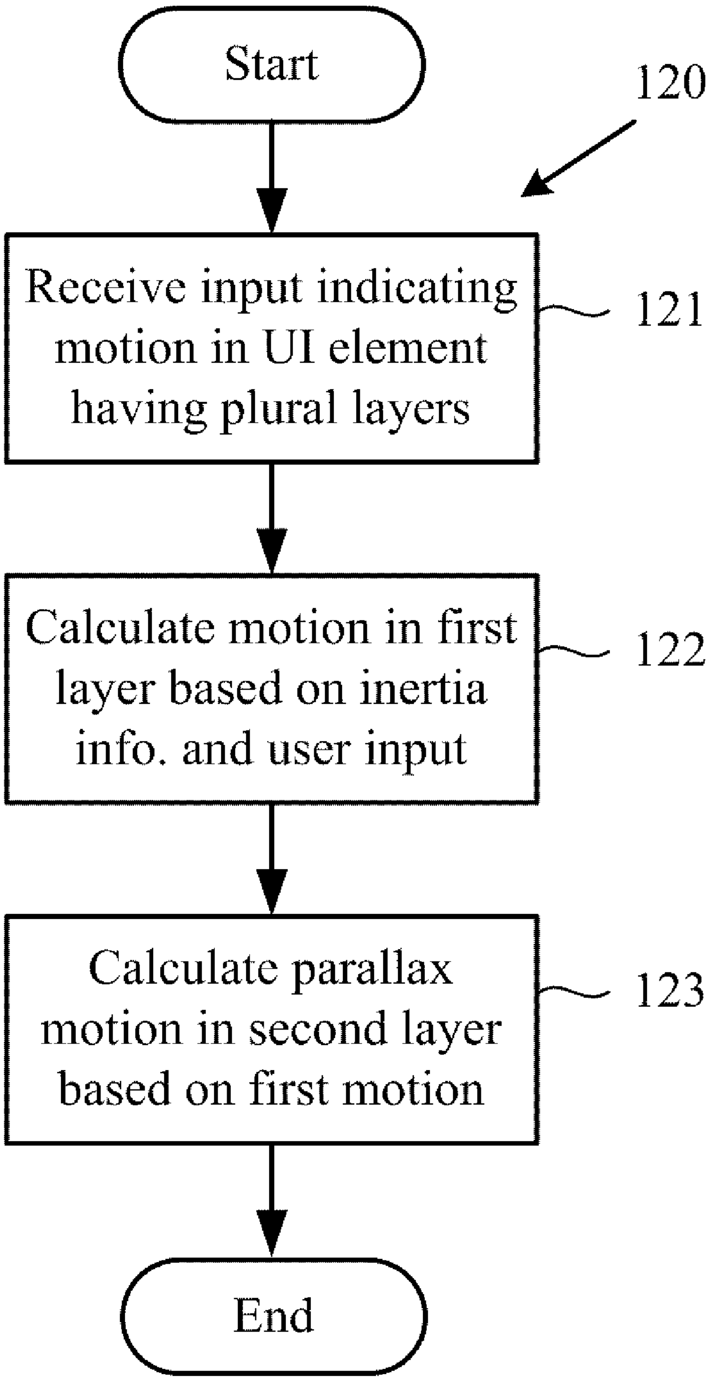

Start
120
Receive input indicating motion in UI element having plural layers
121
Calculate motion in first layer based on inertia info. and user input
122
Calculate parallax motion in second layer based on first motion
123
End

Figure 2

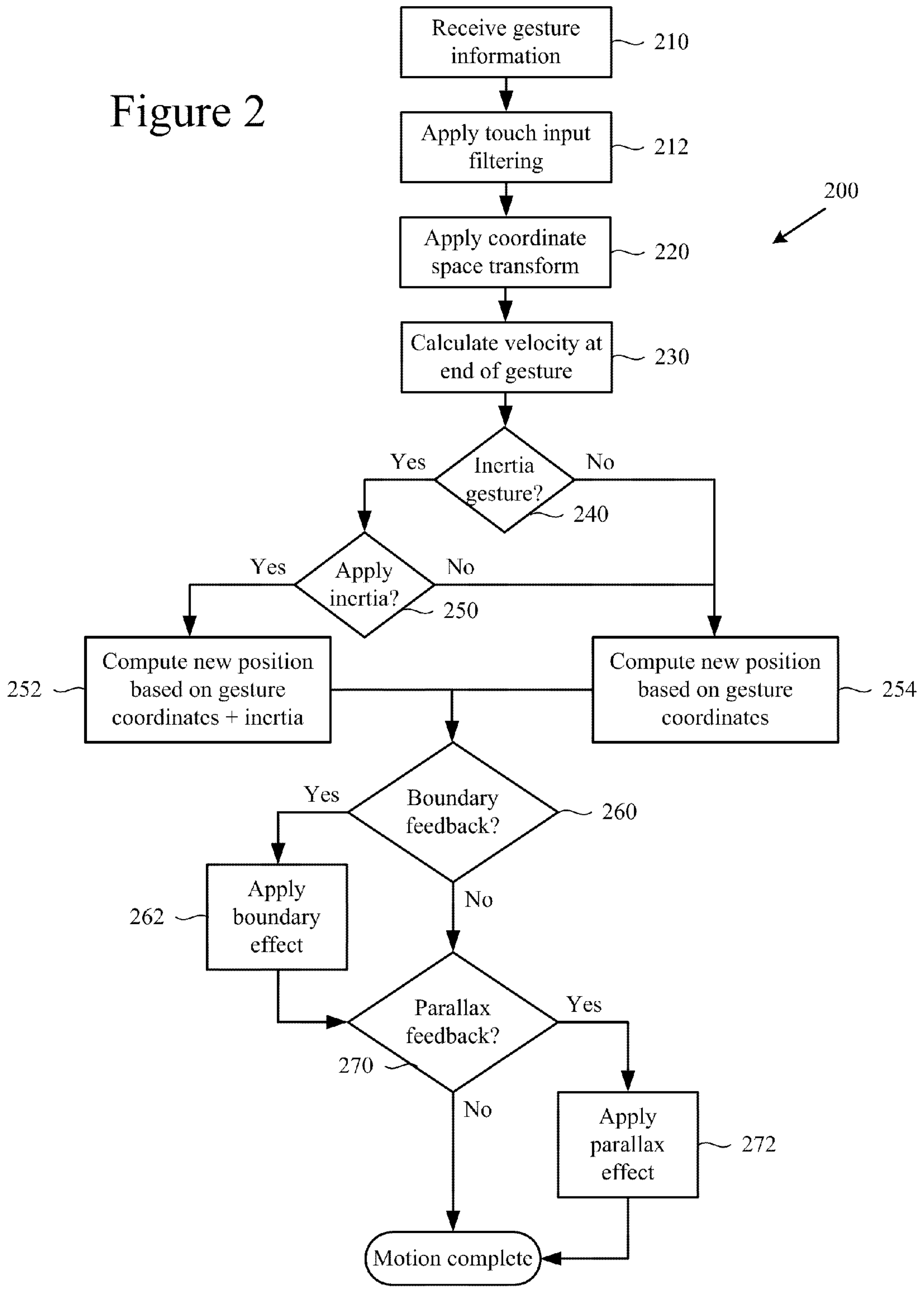

Receive gesture information
210
Apply touch input filtering
212
200
Apply coordinate space transform
220
Calculate velocity at end of gesture
230
Yes
Inertia gesture?
No
240
Yes
Apply inertia?
No
250
252
Compute new position based on gesture coordinates + inertia
Compute new position based on gesture coordinates
254
Yes
Boundary feedback?
260
262
Apply boundary effect
No
Parallax feedback?
Yes
270
No
Apply parallax effect
272
Motion complete

Figure 3

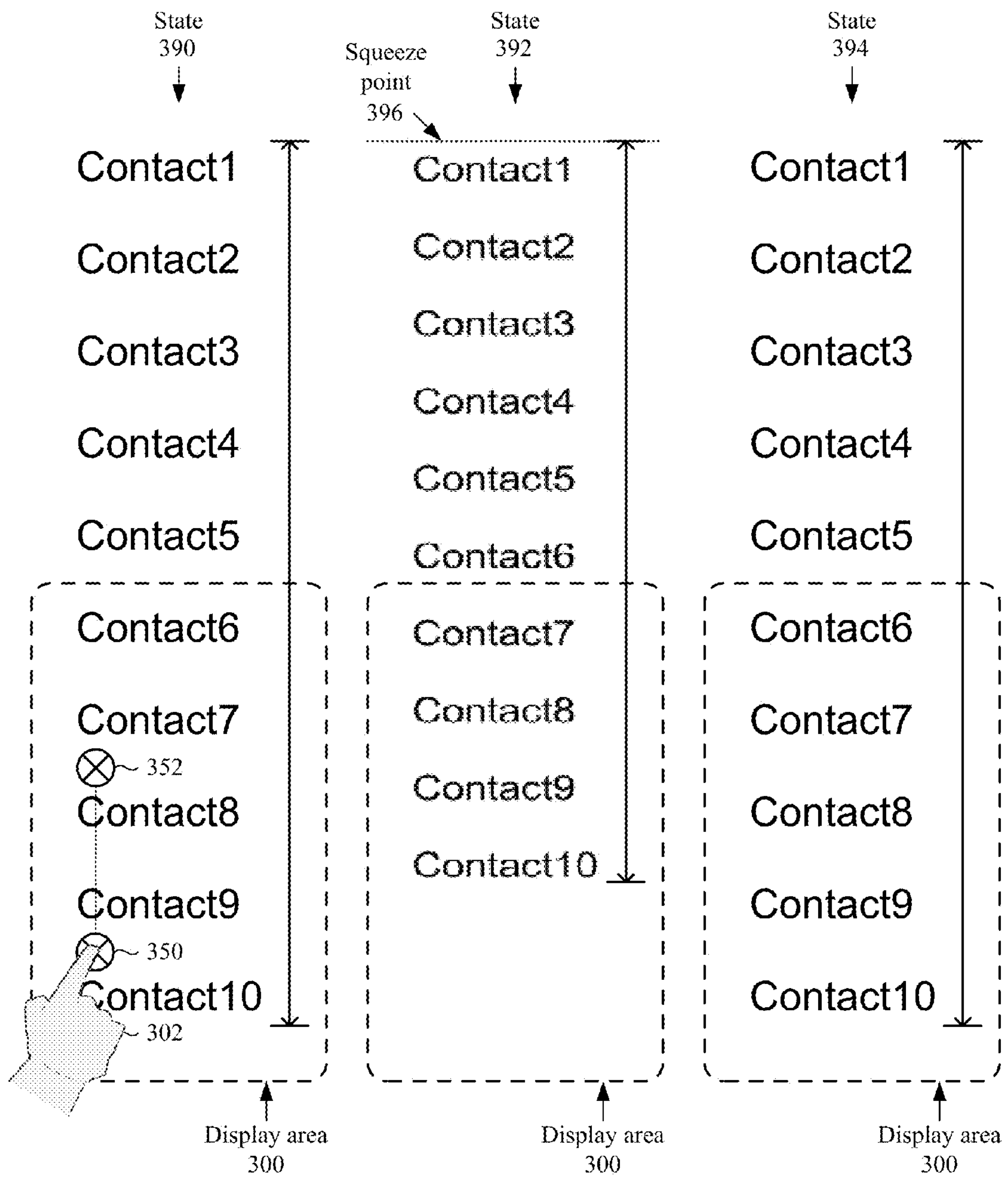
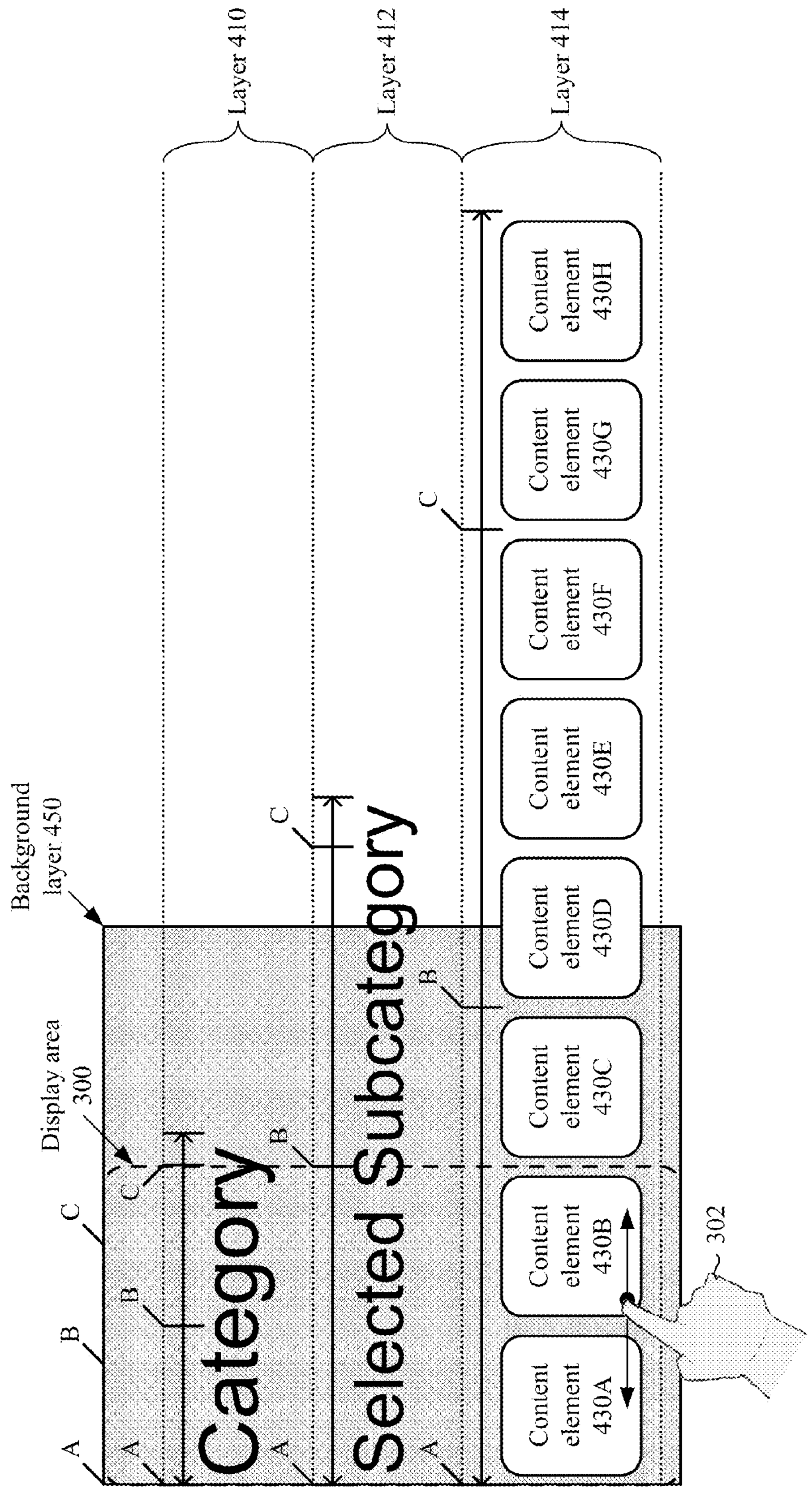

State 390
State 392
State 394
Squeeze point 396
Contact1
Contact2
Contact3
Contact4
Contact5
Contact6
Contact7
352
Contact8
Contact9
350
Contact10
302
Display area 300
Contact1
Contact2
Contact3
Contact4
Contact5
Contact6
Contact7
Contact8
Contact9
Contact10
Display area 300
Contact1
Contact2
Contact3
Contact4
Contact5
Contact6
Contact7
Contact8
Contact9
Contact10
Display area 300

Figure 4A


Layer 410
Layer 412
Layer 414
Background layer 450
Display area 300
Category
Selected Subcategory
Content element 430A
Content element 430B
Content element 430C
Content element 430D
Content element 430E
Content element 430F
Content element 430G
Content element 430H
302
A
B
C

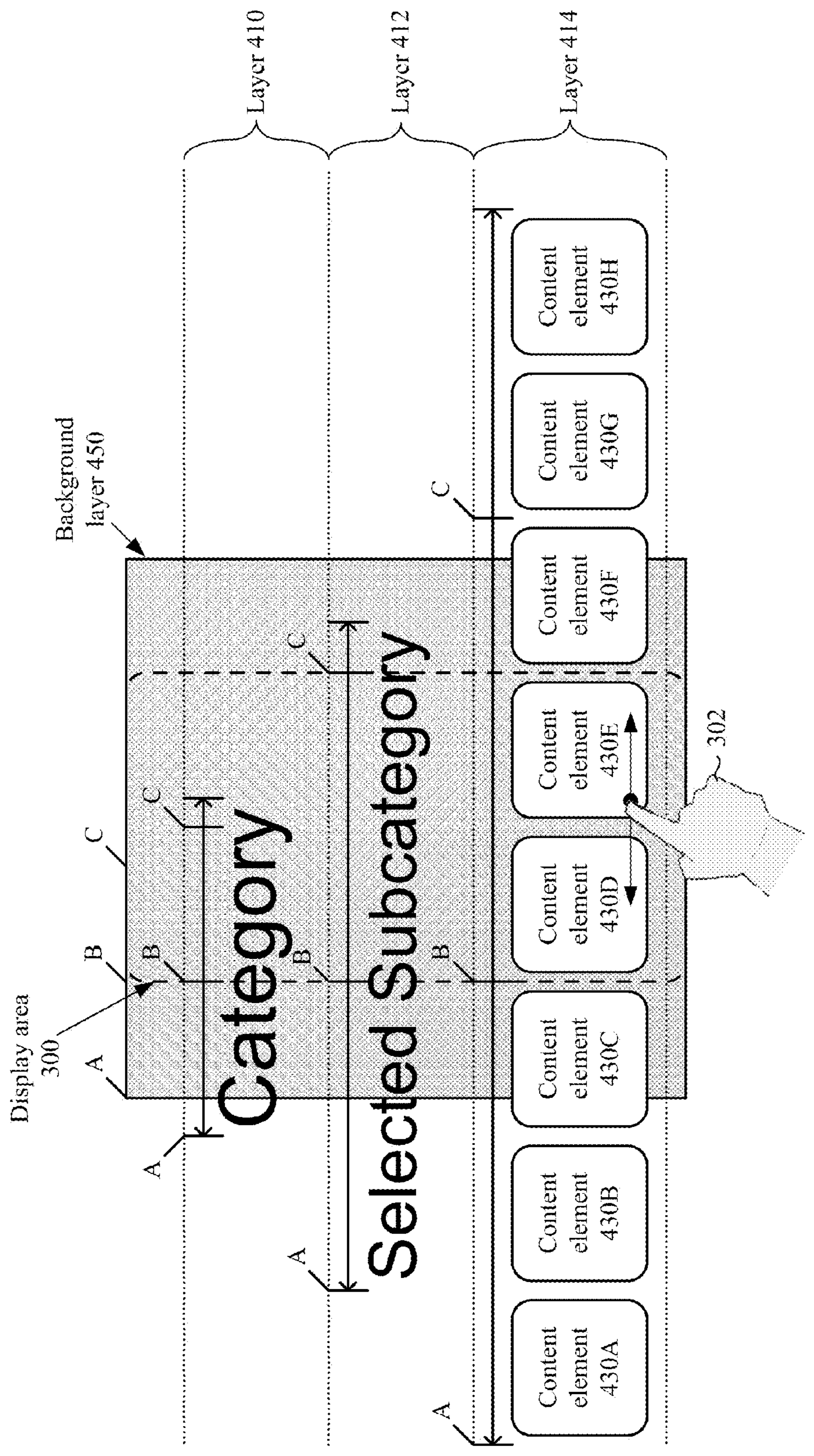

Layer 410
Layer 412
Layer 414
Background layer 450
Display area 300
A
B
C
Category
Selected Subcategory
Content element 430A
Content element 430B
Content element 430C
Content element 430D
Content element 430E
Content element 430F
Content element 430G
Content element 430H
302

Figure 4B

Figure 4C

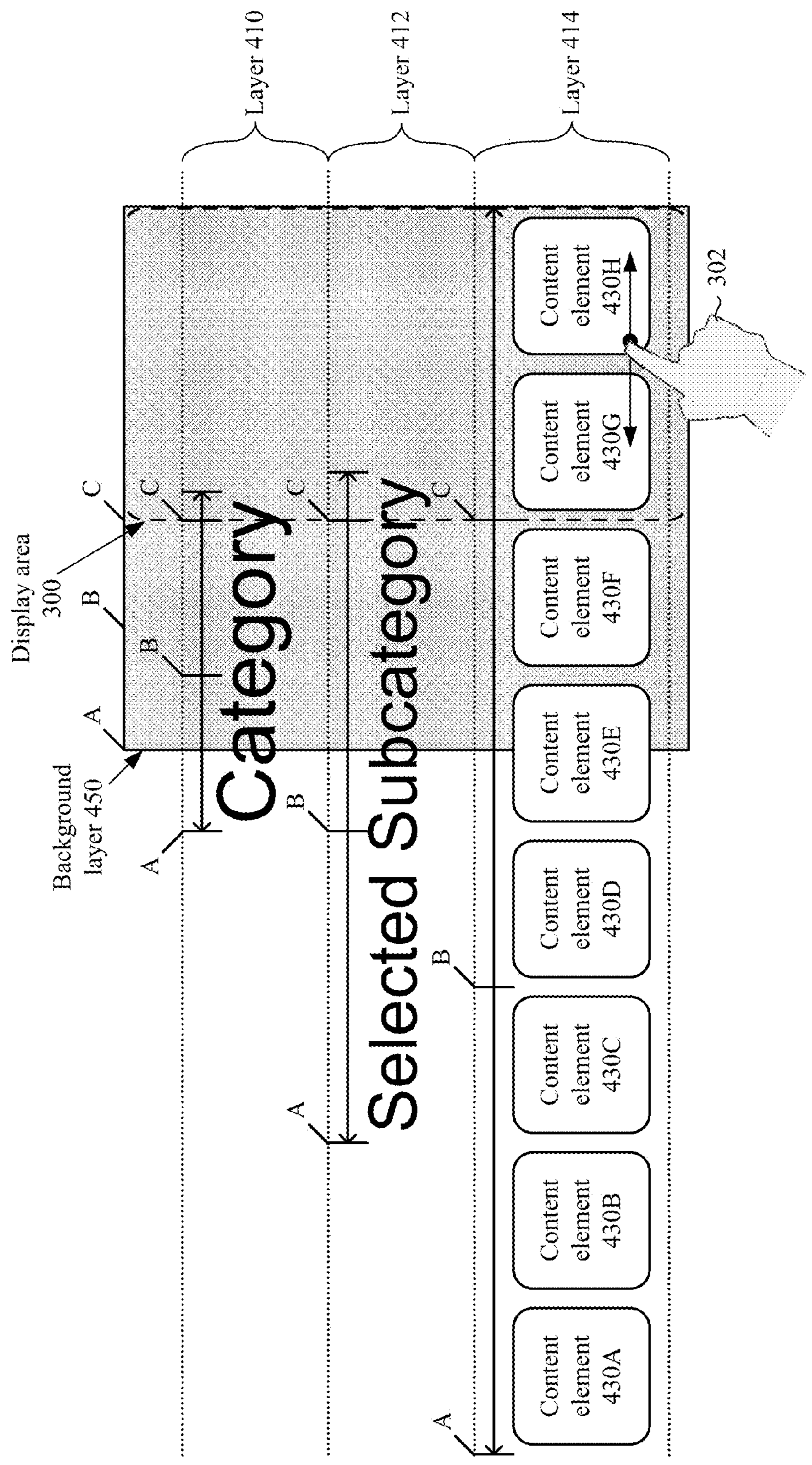

Layer 410
Layer 412
Layer 414
Display area 300
Background layer 450
A
B
C
Category
Selected Subcategory
Content element 430A
Content element 430B
Content element 430C
Content element 430D
Content element 430E
Content element 430F
Content element 430G
Content element 430H
302

Figure 5

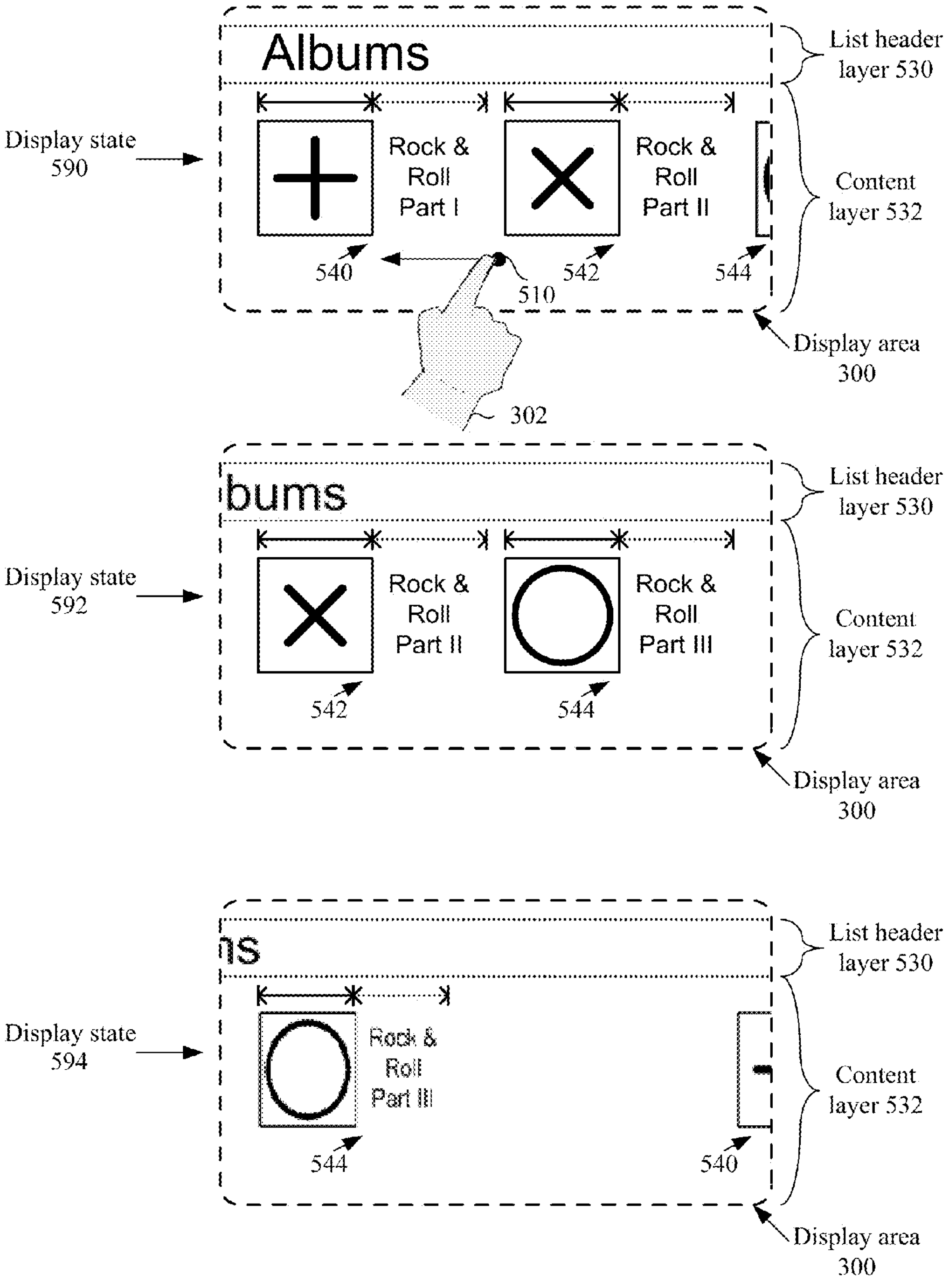

Albums
List header layer 530
Display state 590
Rock & Roll Part I
Rock & Roll Part II
Content layer 532
540
542
544
510
Display area 300
302
bums
List header layer 530
Display state 592
Rock & Roll Part II
Rock & Roll Part III
Content layer 532
542
544
Display area 300
ns
List header layer 530
Display state 594
Rock & Roll Part III
Content layer 532
544
540
Display area 300

Figure 6A

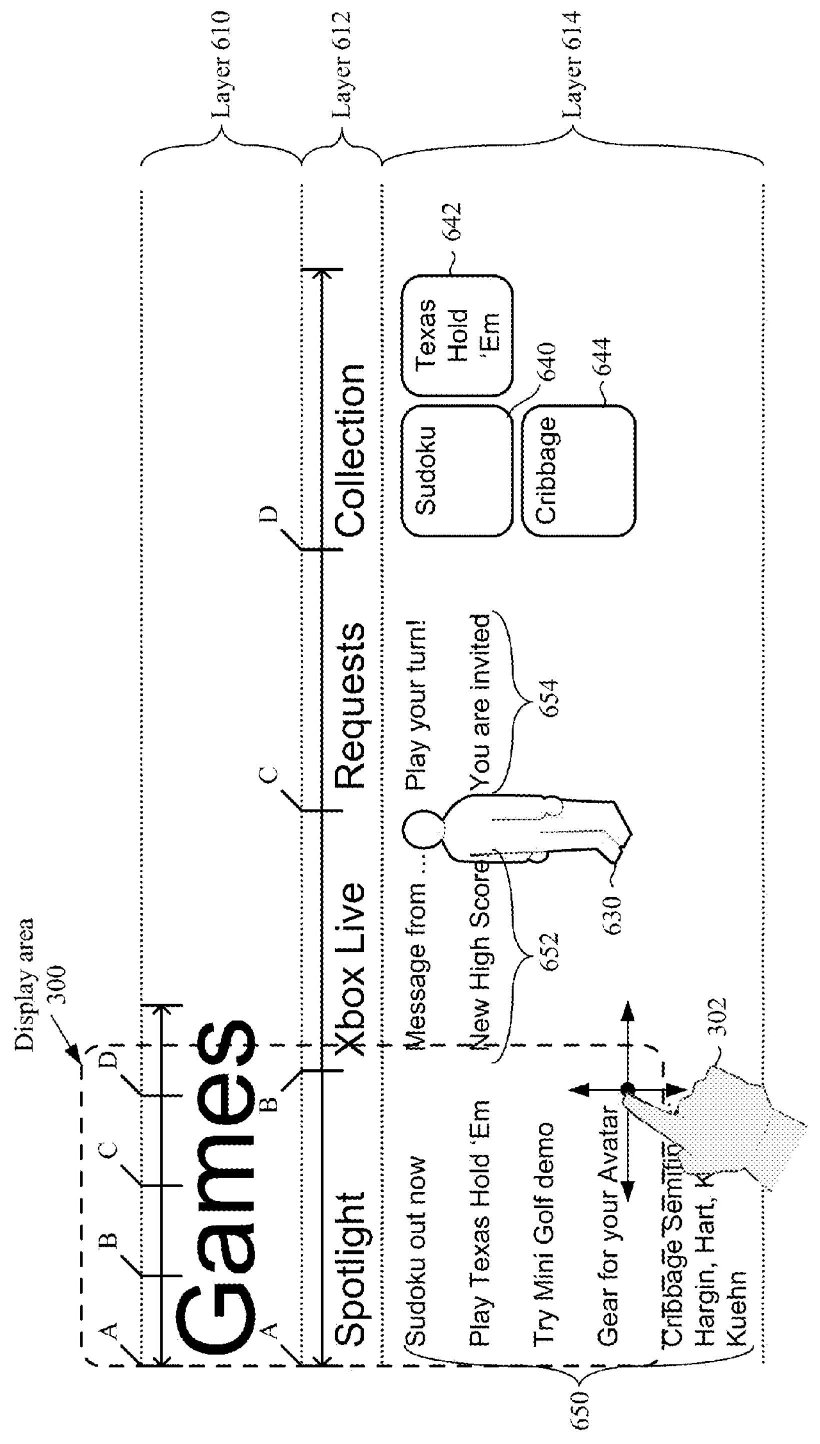

Layer 610
Layer 612
Layer 614
Display area 300
A
B
C
D
Games
Spotlight
Xbox Live
Requests
Collection
Sudoku out now
Play Texas Hold 'Em
Try Mini Golf demo
Gear for your Avatar
Cribbage Semifin
Hargin, Hart, K
Kuehn
Message from ...
New High Score!
Play your turn!
You are invited
Sudoku
Texas Hold 'Em
Cribbage
640
642
644
650
652
654
630
302

Figure 6B

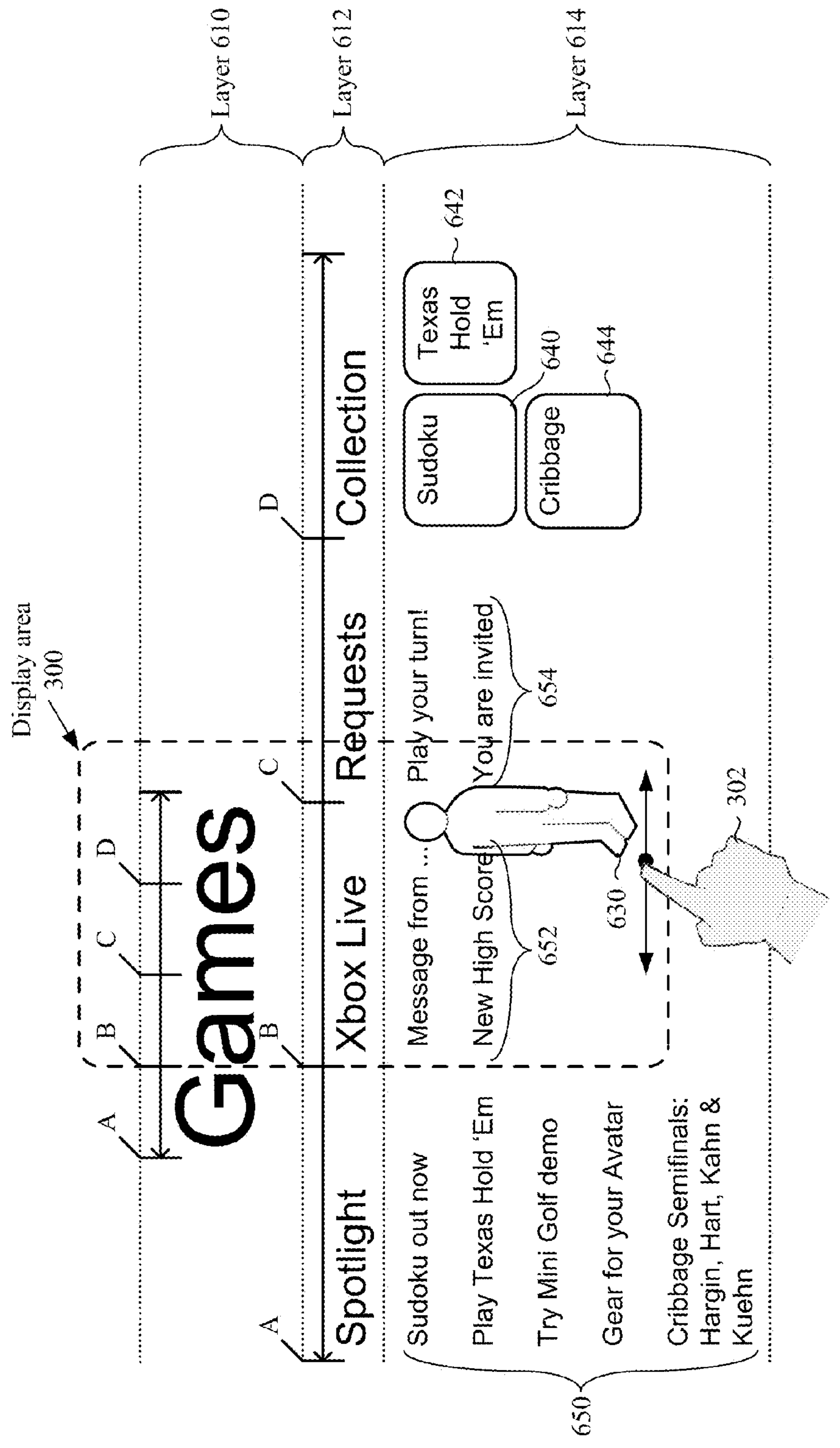

Layer 610
Layer 612
Layer 614
Display area 300
A
B
C
D
Games
A
B
C
D
Spotlight
Xbox Live
Requests
Collection
Sudoku out now
Play Texas Hold 'Em
Try Mini Golf demo
Gear for your Avatar
Cribbage Semifinals: Hargin, Hart, Kahn & Kuehn
Message from ...
New High Score!
Play your turn!
You are invited
Sudoku
Texas Hold 'Em
Cribbage
642
640
644
650
652
654
630
302

Figure 6C

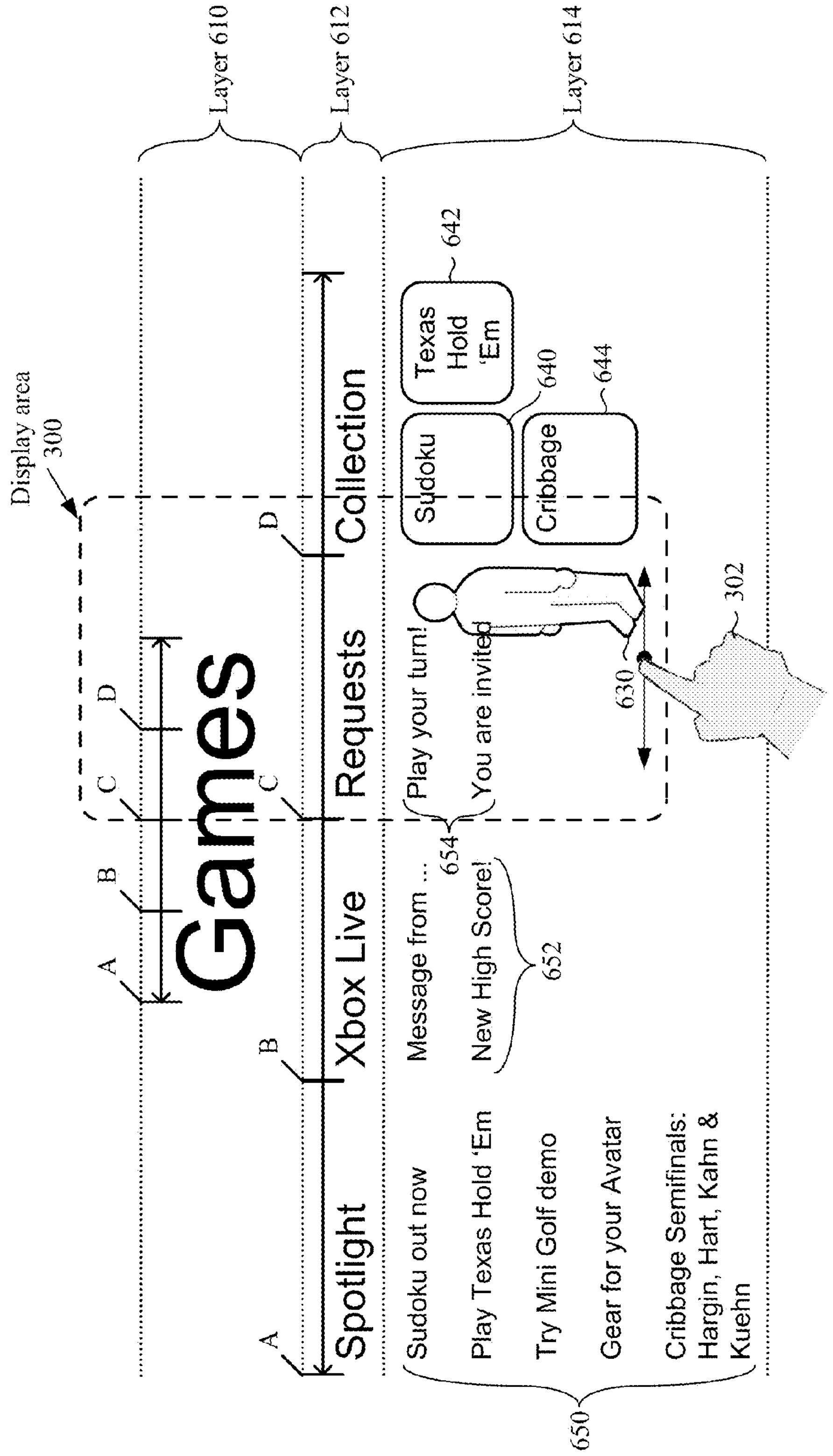
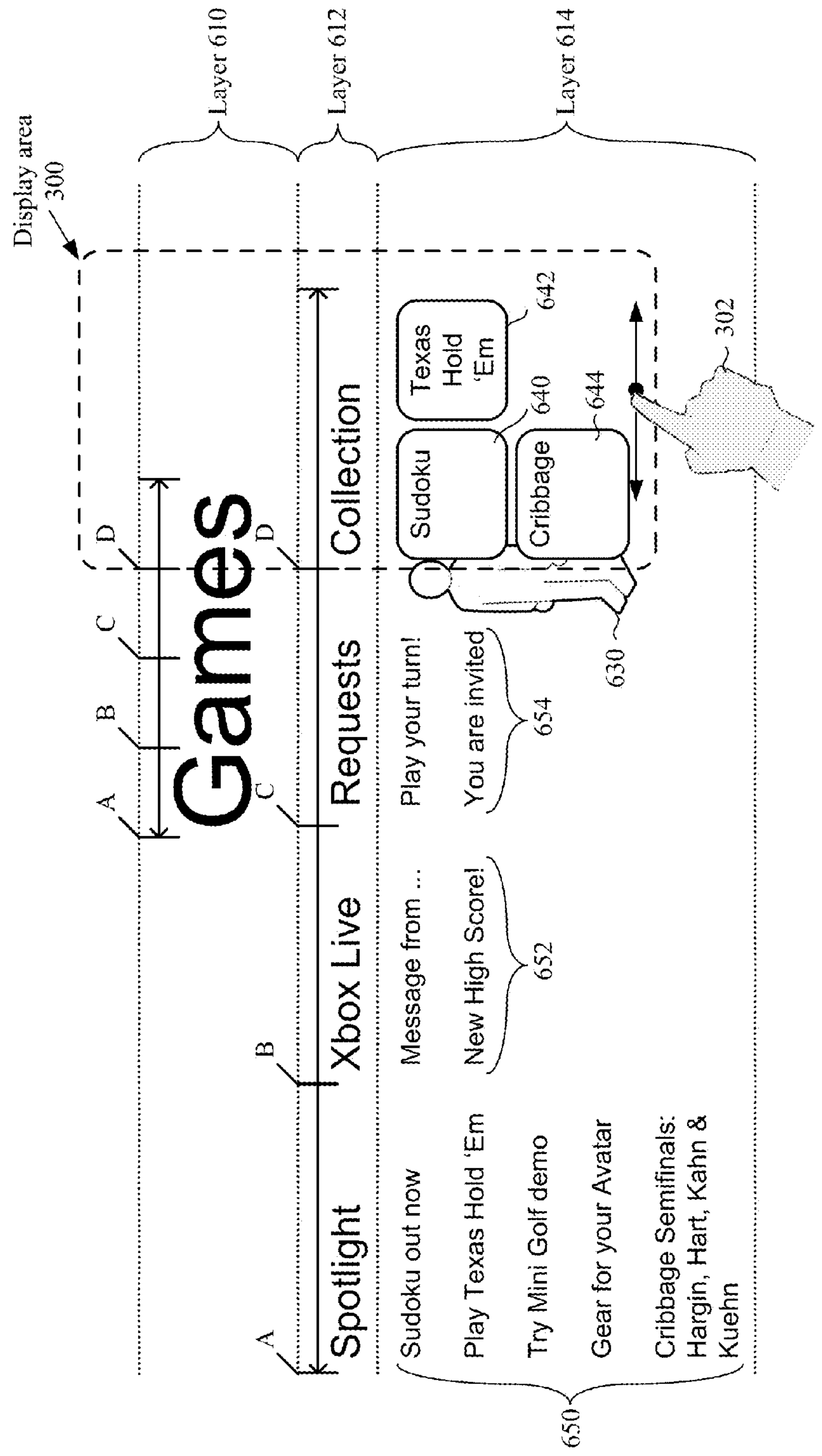

Layer 610
Layer 612
Layer 614
Display area 300
Games
A
B
C
D
Spotlight
Xbox Live
Requests
Collection
Sudoku out now
Play Texas Hold 'Em
Try Mini Golf demo
Gear for your Avatar
Cribbage Semifinals: Hargin, Hart, Kahn & Kuehn
Message from ...
New High Score!
Play your turn!
You are invited
Sudoku
Texas Hold 'Em
Cribbage
650
652
654
630
640
642
644
302

Figure 6D


Display area 300
Layer 610
Layer 612
Layer 614
Games
A
B
C
D
Spotlight
Xbox Live
Requests
Collection
Sudoku out now
Play Texas Hold 'Em
Try Mini Golf demo
Gear for your Avatar
Cribbage Semifinals: Hargin, Hart, Kahn & Kuehn
650
Message from ...
New High Score!
652
Play your turn!
You are invited
654
Sudoku
640
Texas Hold 'Em
642
Cribbage
644
630
302

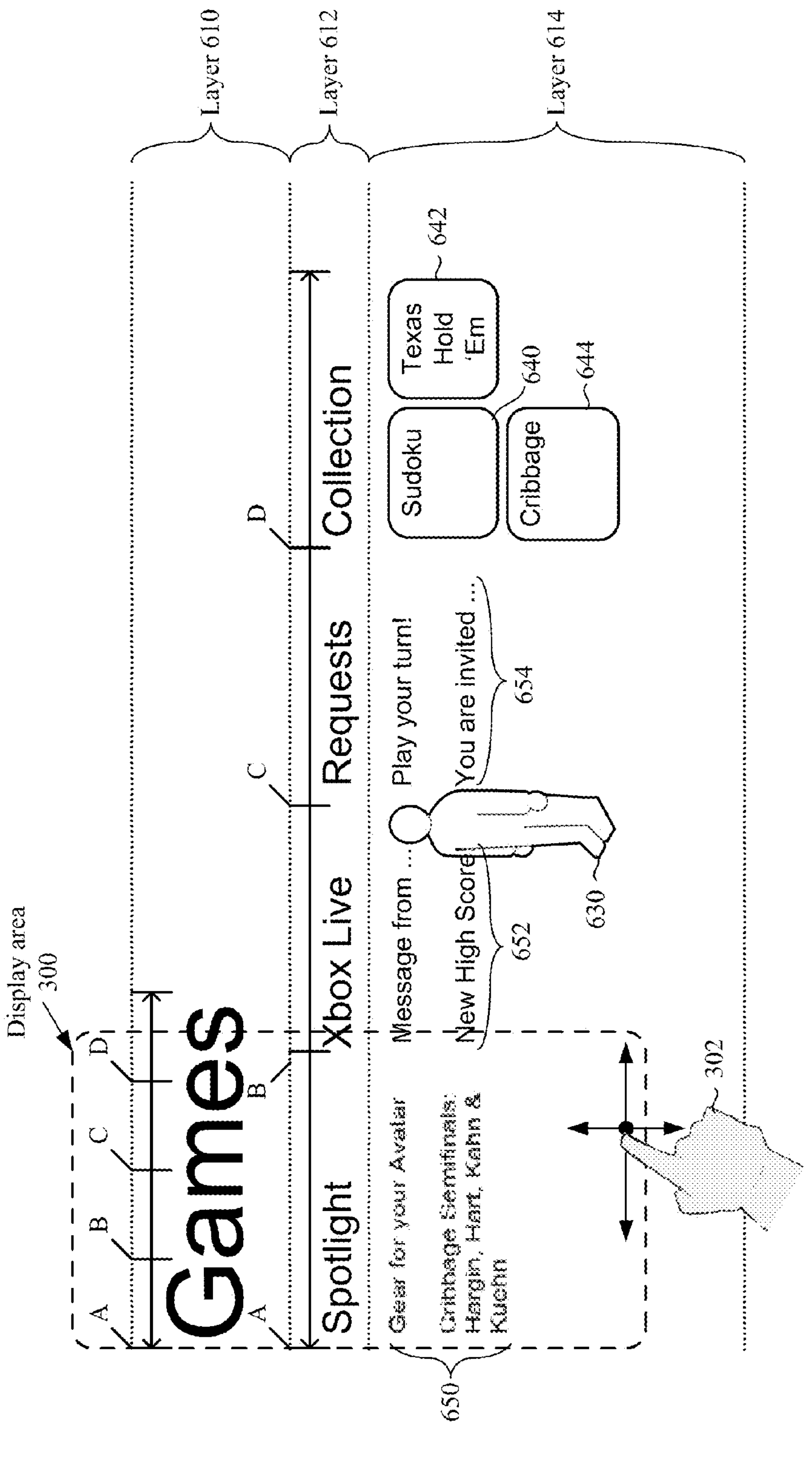
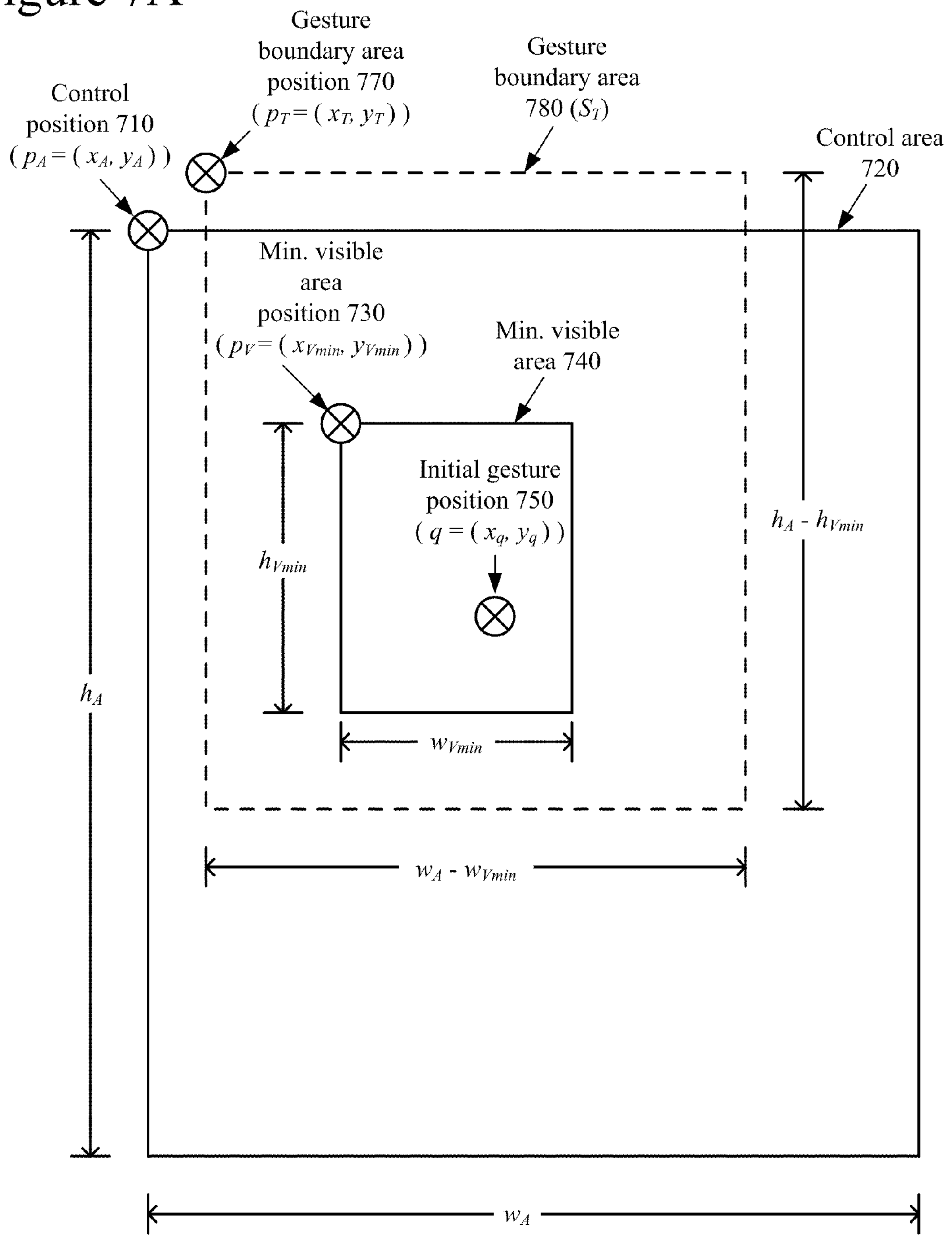

Layer 610
Layer 612
Layer 614
Display area 300
A
B
C
D
Games
A
B
C
D
Spotlight
Xbox Live
Requests
Collection
Gear for your Avatar
Cribbage Semifinals: Hargin, Hart, Kahn & Kuchn
Message from ...
New High Score!
Play your turn!
You are invited ...
Sudoku
Texas Hold 'Em
Cribbage
642
640
644
654
652
650
630
302

Figure 6E

Figure 7A


Gesture boundary area position 770 ($p_T = (x_T, y_T)$)
Gesture boundary area 780 (S_T)
Control position 710 ($p_A = (x_A, y_A)$)
Control area 720
Min. visible area position 730 ($p_V = (x_{Vmin}, y_{Vmin})$)
Min. visible area 740
Initial gesture position 750 ($q = (x_q, y_q)$)
h_{Vmin}
$h_A - h_{Vmin}$
h_A
w_{Vmin}
$w_A - w_{Vmin}$
w_A

$$x_T = x_q + (x_{Vmin} + w_{Vmin}) - (x_A + w_A) = x_q$$
$$y_T = y_q + (y_{Vmin} + h_{Vmin}) - (y_A + h_A) = y_q$$
$$S_T = (h_A - h_{Vmin}, w_A - w_{Vmin})$$

Figure 7B

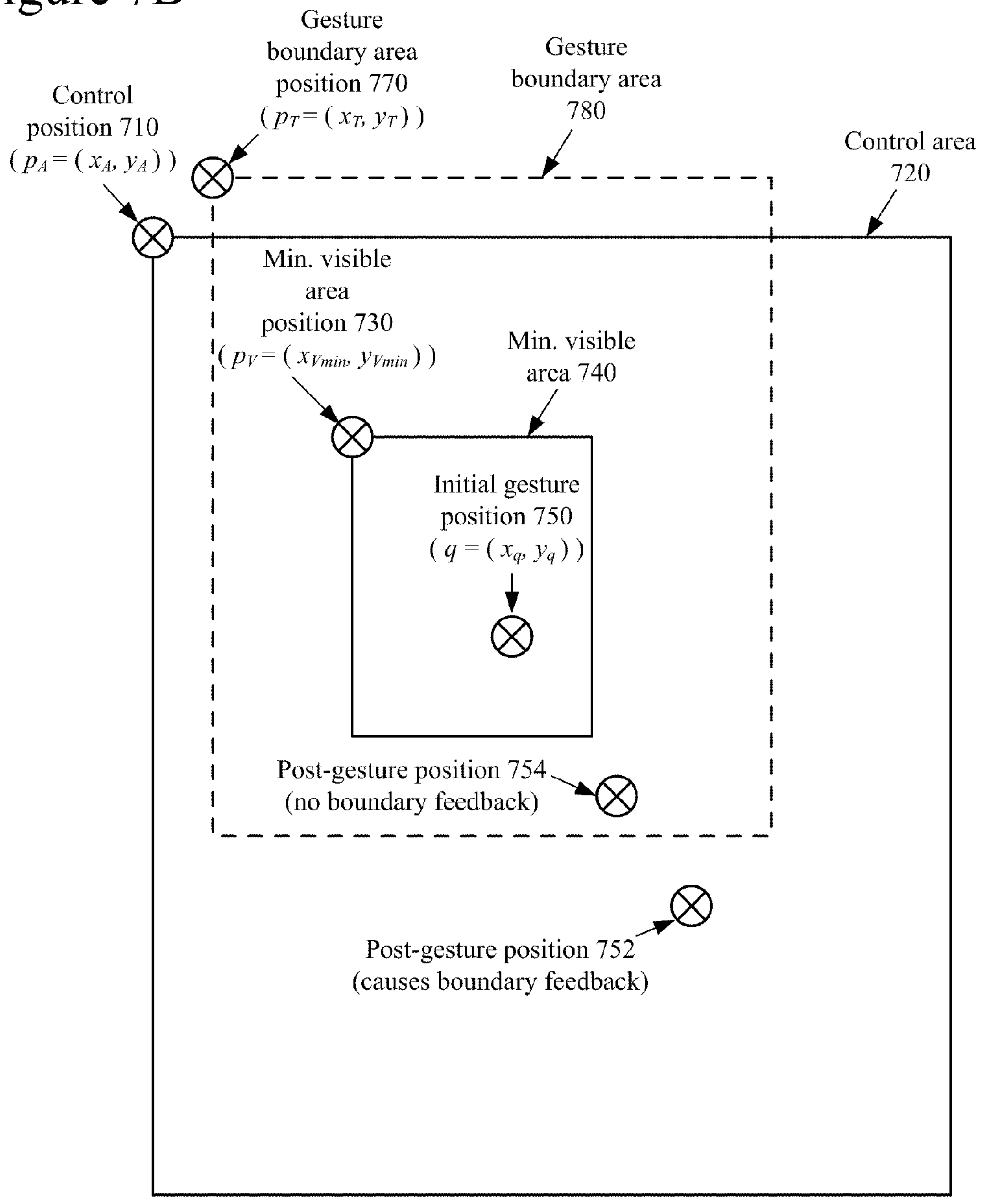
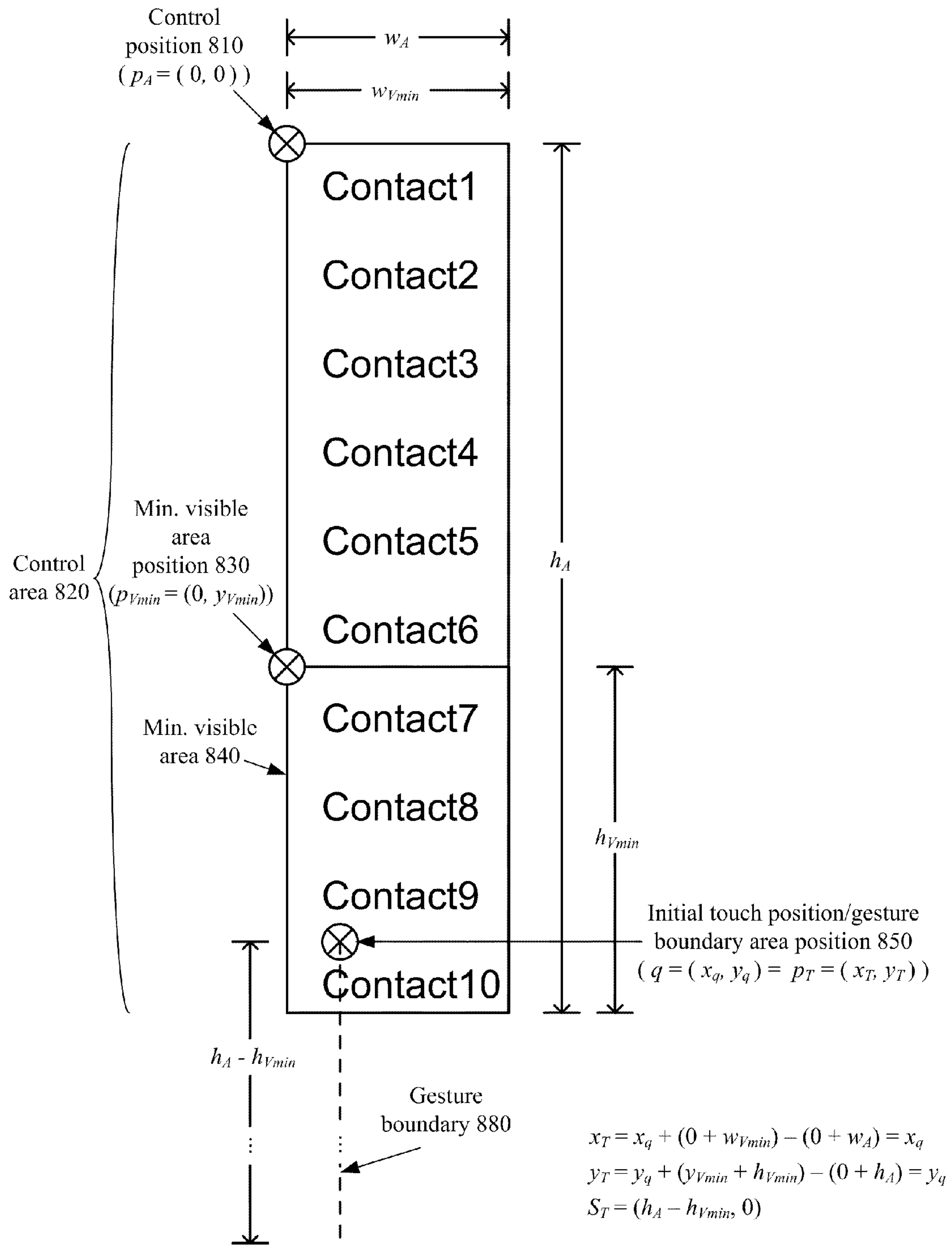

Gesture boundary area position 770 ($p_T = (x_T, y_T)$)
Gesture boundary area 780
Control position 710 ($p_A = (x_A, y_A)$)
Control area 720
Min. visible area position 730 ($p_V = (x_{Vmin}, y_{Vmin})$)
Min. visible area 740
Initial gesture position 750 ($q = (x_q, y_q)$)
Post-gesture position 754 (no boundary feedback)
Post-gesture position 752 (causes boundary feedback)

Figure 8A


Control position 810 (p_A = (0, 0))
w_A
w_{Vmin}
Contact1
Contact2
Contact3
Contact4
Contact5
Contact6
Contact7
Contact8
Contact9
Contact10
Control area 820
Min. visible area position 830 (p_{Vmin} = (0, y_{Vmin}))
Min. visible area 840
h_A
h_{Vmin}
Initial touch position/gesture boundary area position 850 ($q = (x_q, y_q) = p_T = (x_T, y_T)$)
h_A - h_{Vmin}
Gesture boundary 880

$x_T = x_q + (0 + w_{Vmin}) - (0 + w_A) = x_q$

$y_T = y_q + (y_{Vmin} + h_{Vmin}) - (0 + h_A) = y_q$

$S_T = (h_A - h_{Vmin}, 0)$

Figure 8B

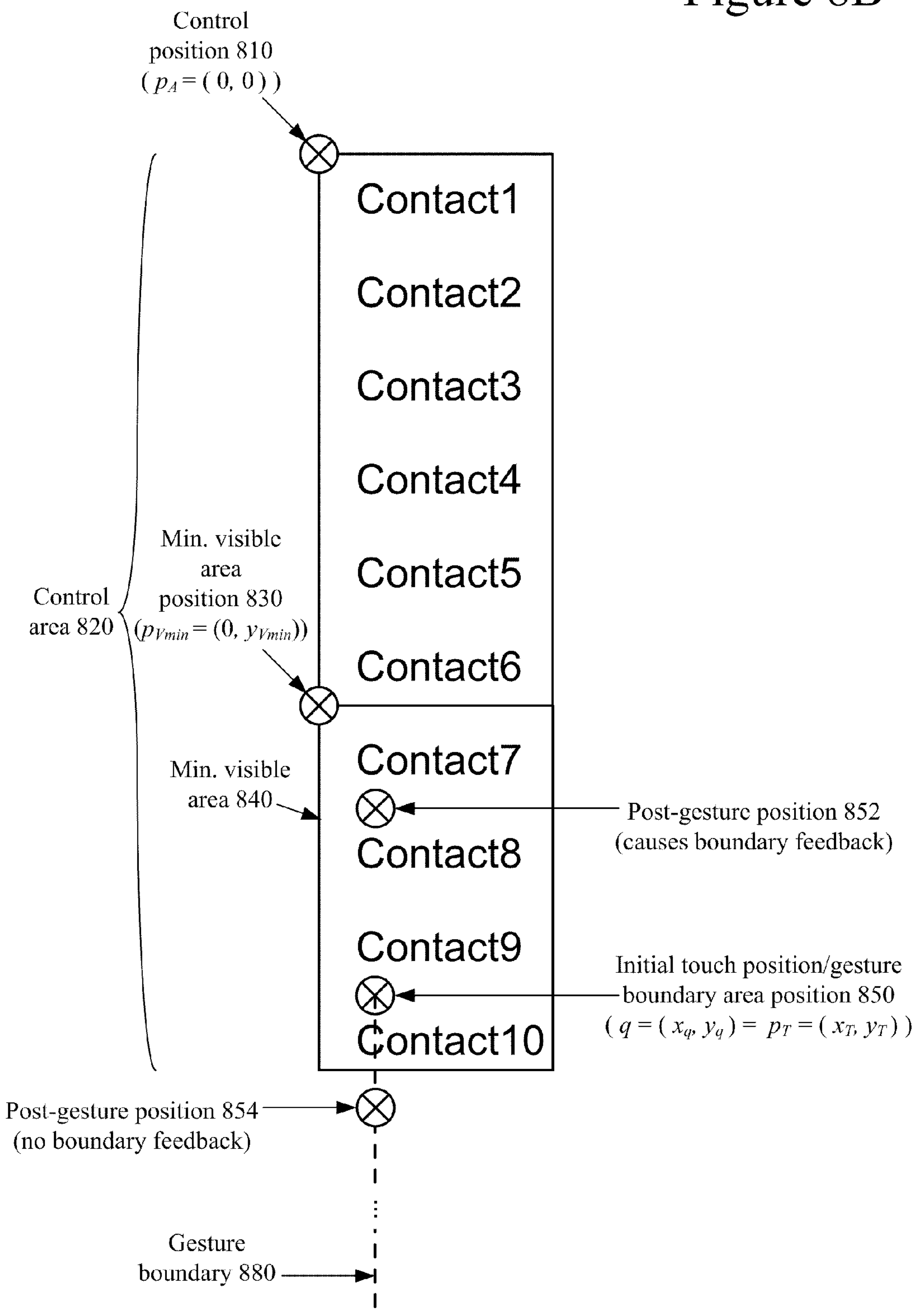

Control position 810 (p_A = (0, 0))
Contact1
Contact2
Contact3
Contact4
Contact5
Contact6
Contact7
Contact8
Contact9
Contact10
Control area 820
Min. visible area position 830 (p_{Vmin} = (0, y_{Vmin}))
Min. visible area 840
Post-gesture position 852 (causes boundary feedback)
Initial touch position/gesture boundary area position 850 ($q = (x_q, y_q) = p_T = (x_T, y_T)$)
Post-gesture position 854 (no boundary feedback)
Gesture boundary 880

Figure 9

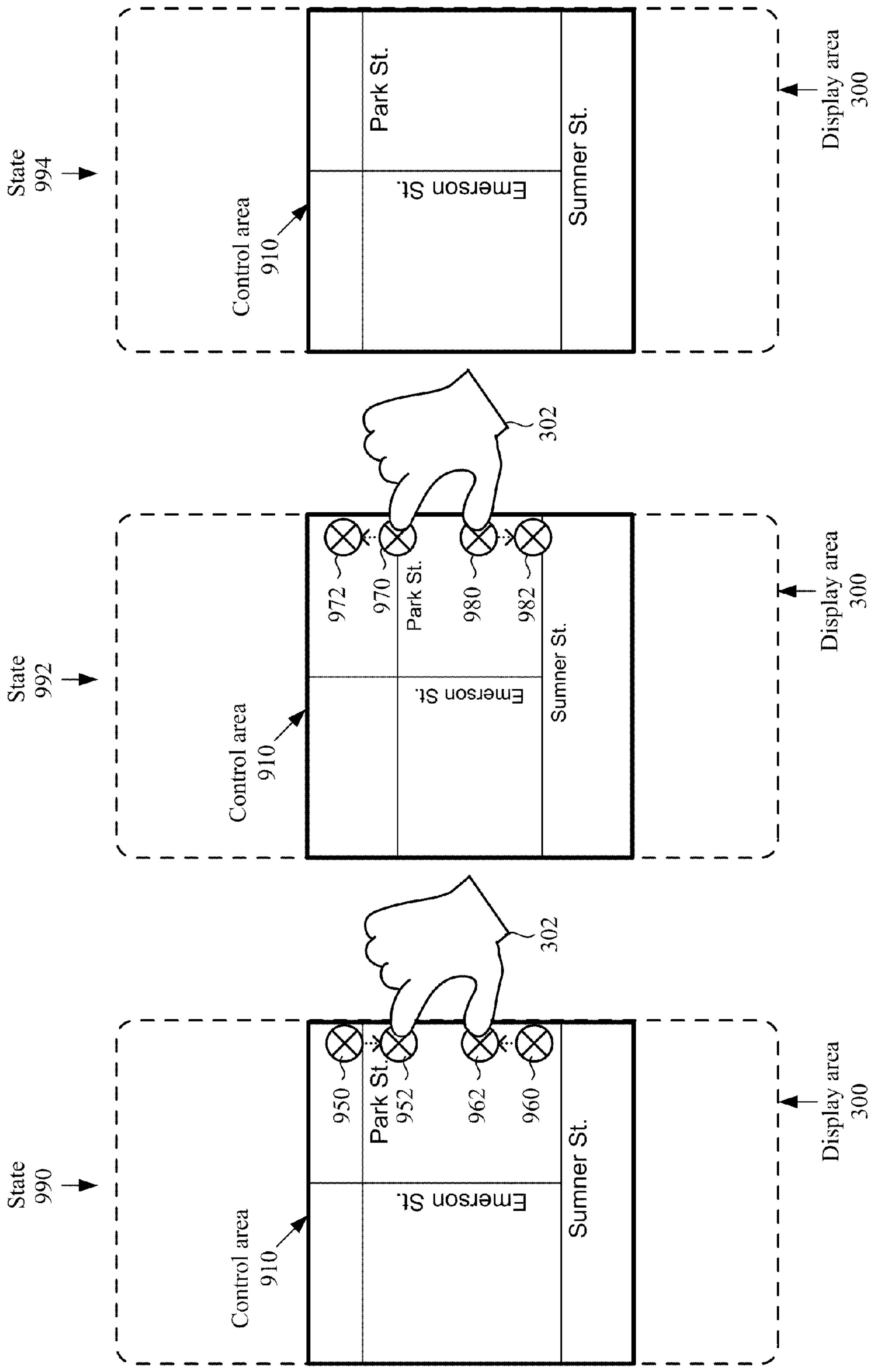

State 990
Control area 910
Park St.
Emerson St.
Sumner St.
950
952
962
960
302
Display area 300
State 992
Control area 910
972
970
Park St.
980
982
Emerson St.
Sumner St.
302
Display area 300
State 994
Control area 910
Park St.
Emerson St.
Sumner St.
Display area 300

Figure 10

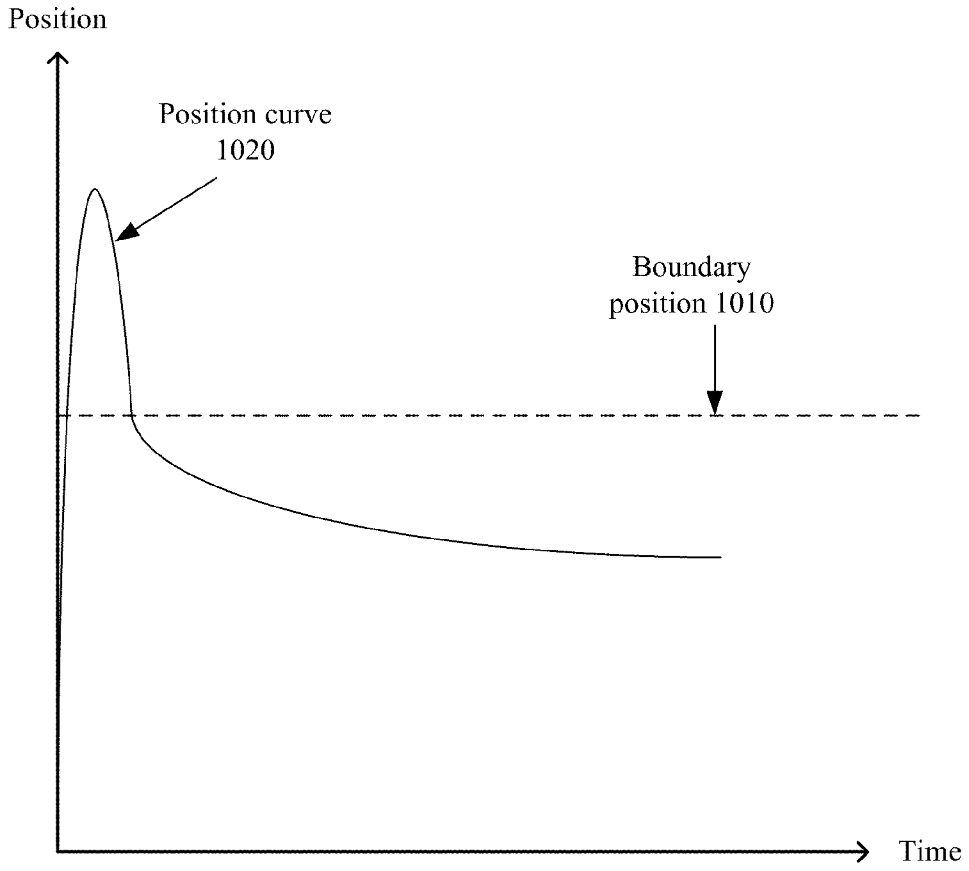


Figure 11

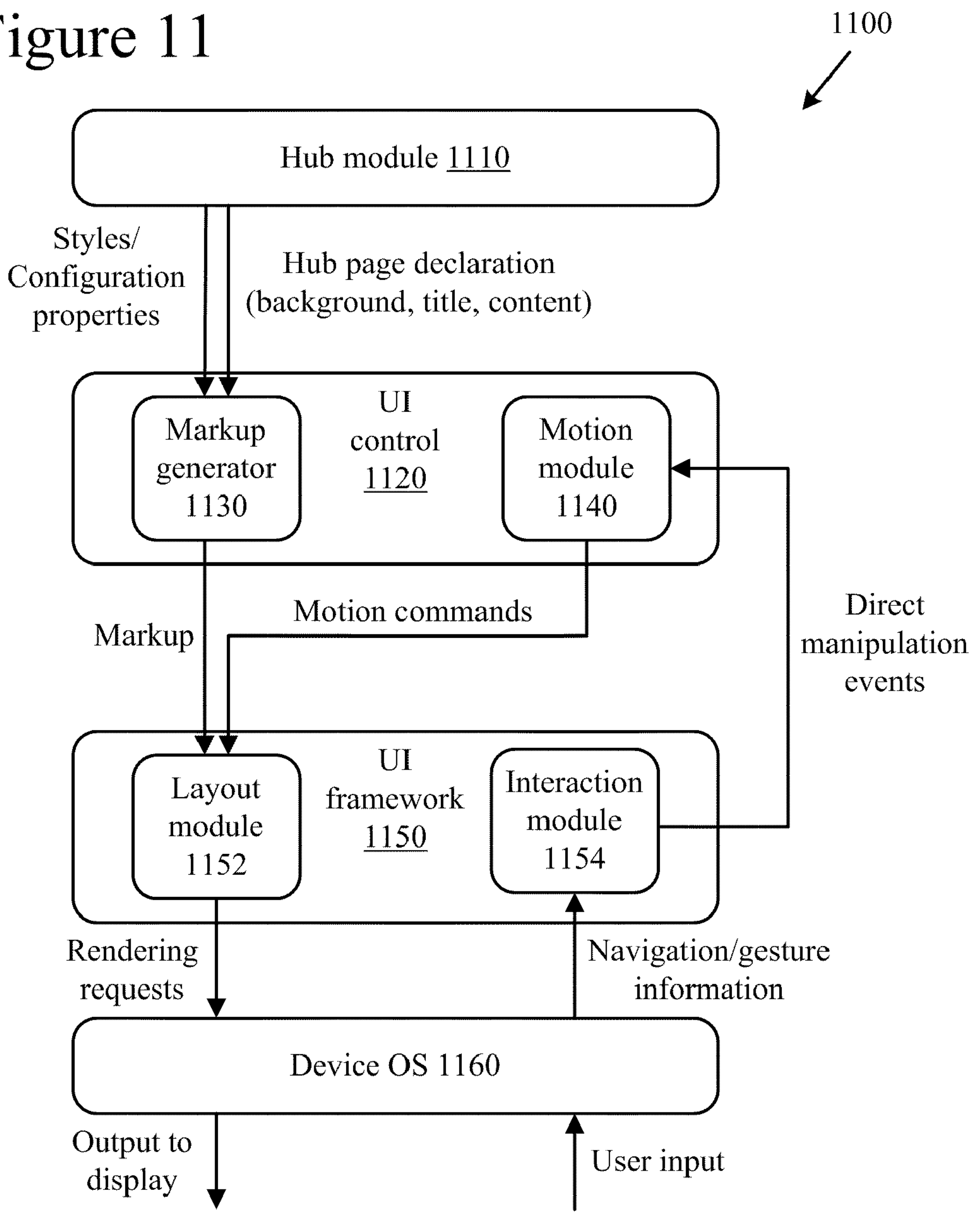

1100
Hub module 1110
Styles/ Configuration properties
Hub page declaration (background, title, content)
Markup generator 1130
UI control 1120
Motion module 1140
Markup
Motion commands
Direct manipulation events
Layout module 1152
UI framework 1150
Interaction module 1154
Rendering requests
Navigation/gesture information
Device OS 1160
Output to display
User input

Figure 12

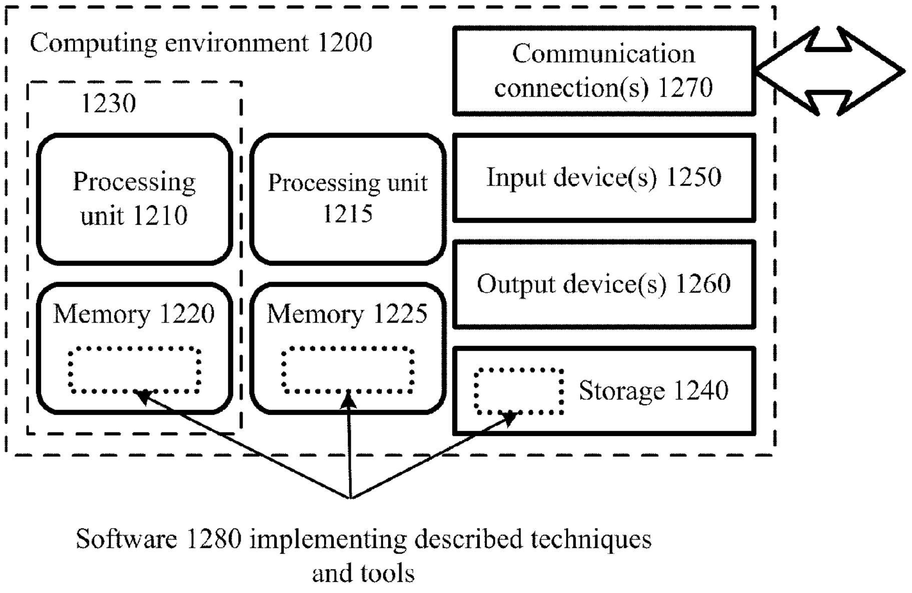

Computing environment 1200
1230
Processing unit 1210
Processing unit 1215
Memory 1220
Memory 1225
Communication connection(s) 1270
Input device(s) 1250
Output device(s) 1260
Storage 1240
Software 1280 implementing described techniques and tools

Figure 13

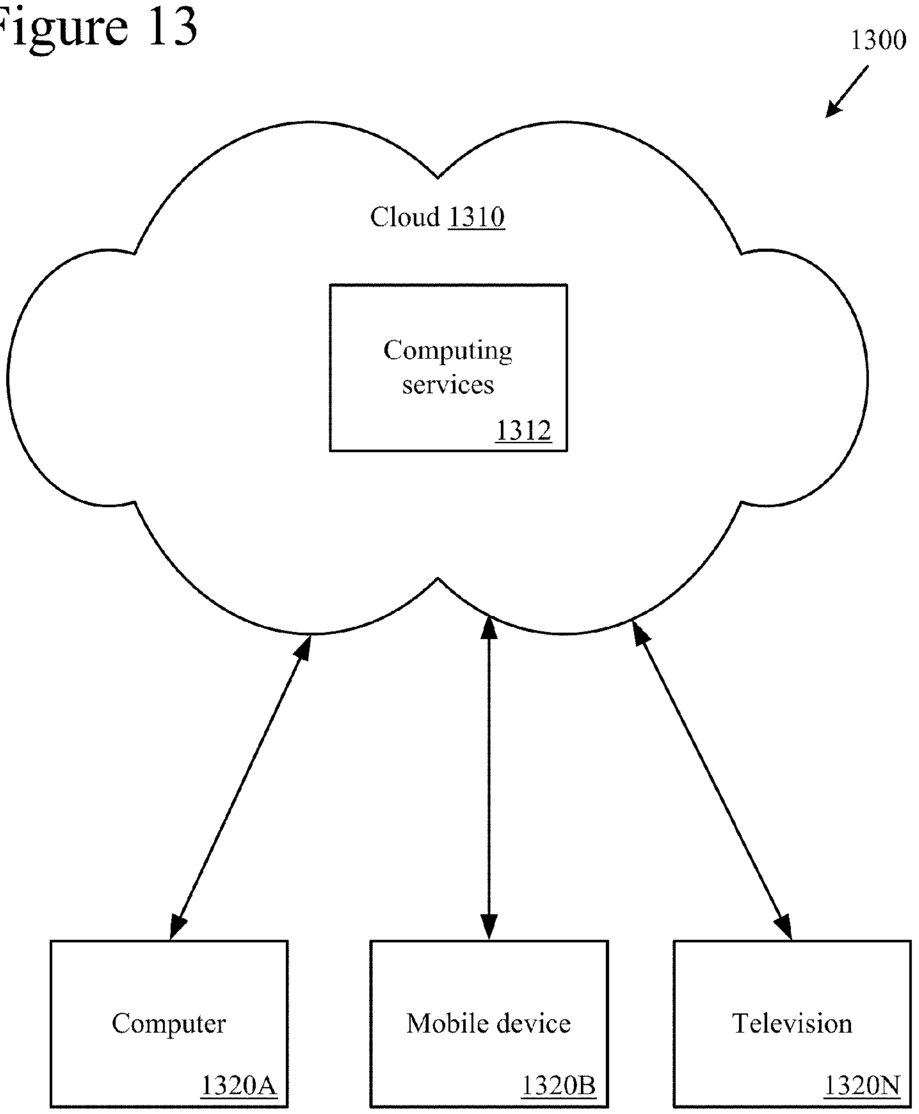
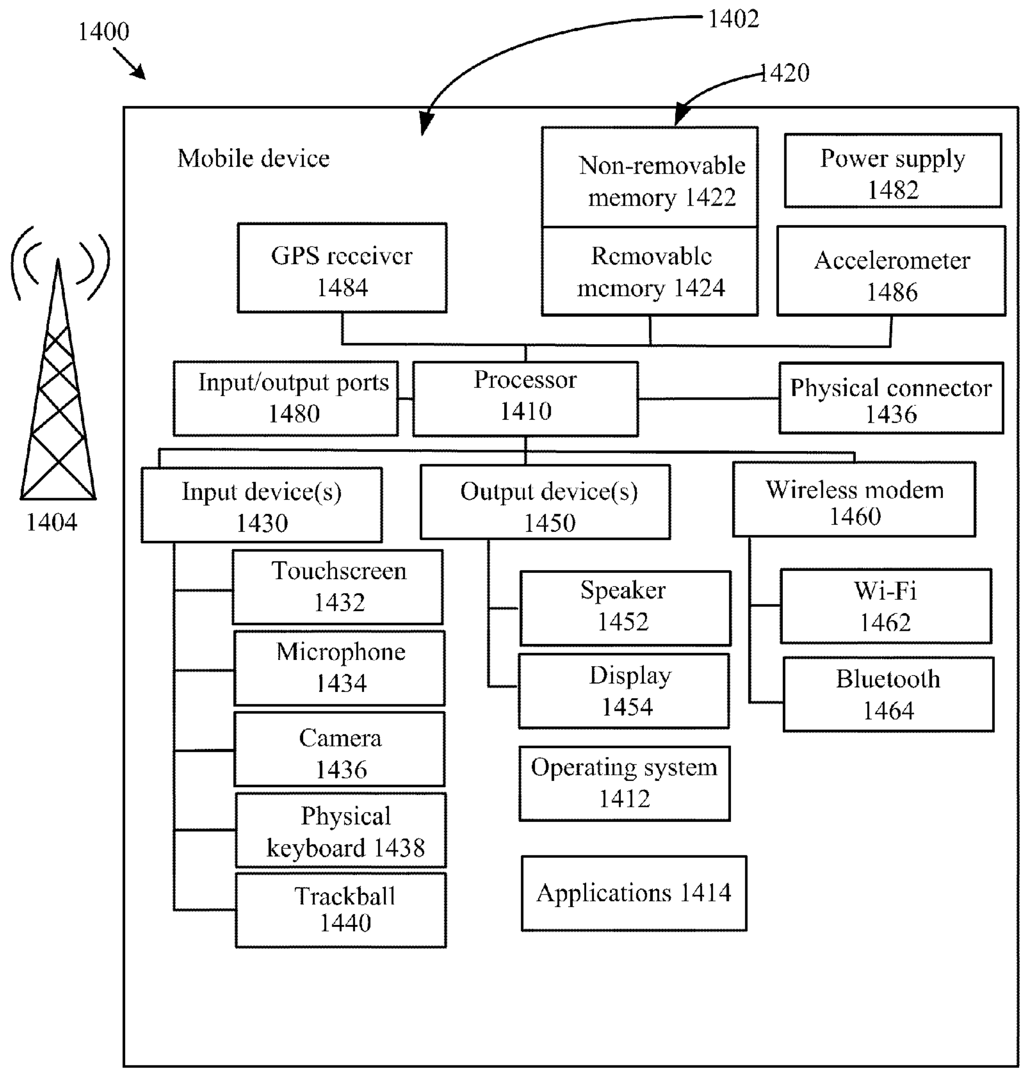

1300
Cloud 1310
Computing services
1312
Computer
1320A
Mobile device
1320B
Television
1320N

Figure 14


1400
1402
1420
1404
Mobile device
Non-removable memory 1422
Removable memory 1424
Power supply 1482
Accelerometer 1486
GPS receiver 1484
Input/output ports 1480
Processor 1410
Physical connector 1436
Input device(s) 1430
Output device(s) 1450
Wireless modem 1460
Touchscreen 1432
Microphone 1434
Camera 1436
Physical keyboard 1438
Trackball 1440
Speaker 1452
Display 1454
Operating system 1412
Applications 1414
Wi-Fi 1462
Bluetooth 1464

VISUAL MOTION FEEDBACK FOR USER INTERFACE

CROSS REFERENCE TO RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Patent Application No. 61/304,004, filed on Feb. 12, 2010, entitled "MULTI-LAYER USER INTERFACE WITH FLEXIBLE MOVEMENT," which is incorporated herein by reference.

BACKGROUND

[0002] The design of an effective user interface poses many challenges. One challenge is how to provide a user with an optimal amount of visual information or functionality, given the space limitations of a display and the needs of a particular user. This challenge can be especially acute for devices with small displays, such as smartphones or other mobile computing devices. This is because there is often more information available to a user performing a particular activity (e.g., browsing for audio or video files in a library of files) than can fit on the display. A user can easily become lost unless careful attention is paid to how information is presented on the limited amount available display space. Visual cues are useful for indicating, for example, a user's location when browsing a list or other collection of data, since it is often not possible to show an entire collection (e.g., a list of contacts stored in a smartphone) on a small display.

[0003] Another challenge is how to provide a high level of functionality while maintaining a satisfying and consistent user experience. As devices have become more complex, and as consumers have become more demanding, it has become increasingly difficult to design user interfaces that are convenient and pleasing for users, without sacrificing reliability, flexibility, functionality or performance.

[0004] Whatever the benefits of previous techniques, they do not have the advantages of the techniques and tools presented below.

SUMMARY

[0005] Techniques and tools are described that relate to different aspects of a user interface that provides visual feedback in response to user input. For example, boundary effects are presented to provide visual cues to a user to indicate that a boundary in a movable user interface element (e.g., the end of a scrollable list) has been reached. As another example, parallax effects are presented in which multiple parallel or substantially parallel layers in a multi-layer user interface move at different rates, in response to user input. As another example, simulated inertia motion of UI elements is used to provide a more natural feel for touch input. Various combinations of features are described. For example, simulated inertia motion can be used in combination with parallax effects, boundary effects, or other types of visual feedback.

[0006] In one aspect, a user interface (UI) system receives gesture information corresponding to a gesture on a touch input device. The UI system calculates simulated inertia motion for a movable user interface element based at least in part on the gesture information, and potentially on other inertia information such as a friction coefficient or a parking speed coefficient. Based at least in part on the gesture information and on the simulated inertia motion, the UI system calculates a post-gesture position of the movable user interface element. The UI system determines that the post-gesture position exceeds a gesture boundary of the movable user interface element, and calculates a distortion effect (e.g., a squeeze, compression or squish effect) in the movable user interface element to indicate that the gesture boundary has been exceeded. Calculating the distortion effect can include, for example, determining an extent by which the gesture boundary has been exceeded, determining a compressible area of the movable user interface element, determining a scale factor for the distortion effect based at least in part on the compressible area and the extent by which the gesture boundary has been exceeded, and scaling the compressible area according to the scale factor. The distortion effect can be calculated based on a distortion point (which, for compression, can be referred to as a compression point or squeeze point), which can indicate the part of the UI element to be distorted.

[0007] In another aspect, user input (e.g., a gesture on a touch screen) indicates movement in a graphical user interface element having plural movable layers. Based at least in part on inertia information and the user input, a UI system calculates a first motion having a first movement rate in a first layer of the plural movable layers, and calculates a parallax motion in a second layer of the plural movable layers. The parallax motion is based at least in part on the first motion (and potentially simulated inertia motion), and the parallax motion comprises a movement of the second layer at a second movement rate that differs from the first movement rate. The parallax motion can be calculated based on, for example, a parallax constant for the second layer, or an amount of displayable data in the second layer.

[0008] In another aspect, a UI system receives gesture information corresponding to a gesture on a touch input device, the gesture information indicating a movement of a user interface element having a movement boundary. Based at least in part on the gesture information, the UI system computes a new position of the user interface element. Based at least in part on the new position, the UI system determines that the movement boundary has been exceeded. The UI system determines an extent by which the movement boundary has been exceeded, determines a compressible area of the user interface element, determines a scale factor for a distortion effect based at least in part on the compressible area and the extent by which the movement boundary has been exceeded, and presents a distortion effect in the user interface element. The distortion effect comprises a visual compression of content in the compressible area (e.g., text, images, graphics, video or other displayable content) according to the scale factor. Depending, for example, on the size of the compressible area and the size of the display area, some parts of the compressible area may not be visible on a display, so the distortion can be virtual (e.g., in areas that are not visible on a display) or the distortion can be actually displayed, or some parts of the distorted content can be displayed while other parts of the distorted content are not displayed. The visual compression is in a dimension that corresponds to the movement of the user interface element. For example, a vertical movement in a UI element that exceeds a movement boundary can cause content in the UI element to be vertically compressed or squeezed.

[0009] The foregoing and other objects, features, and advantages of the invention will become more apparent from the following detailed description, which proceeds with reference to the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] FIGS. **1**A-**1**C and **2** are flow charts showing example techniques for presenting motion feedback in user interface elements, according to one or more described embodiments.

[0011] FIG. **3** is a diagram showing a boundary effect, according to one or snore described embodiments.

[0012] FIGS. **4**A-**4**C are diagrams showing parallax effects, according to one or more described embodiments.

[0013] FIGS. **5** and **6**A-**6**E are diagrams showing parallax effects and boundary effects in a user interface having parallel layers, according to one or more described embodiments.

[0014] FIGS. **7**A, **7**B, **8**A and **8**B are diagrams showing gesture boundary areas which can be used to determine whether to present boundary effects, according to one or more described embodiments.

[0015] FIG. **9** is a diagram showing example pinch and stretch gestures, according to one or more described embodiments.

[0016] FIG. **10** is a graph showing changes in position over time of a UI element that exhibits a boundary feedback effect, according to one or more described embodiments.

[0017] FIG. **11** is a system diagram showing a UI system in which described embodiments can be implemented.

[0018] FIG. **12** illustrates a generalized example of a suitable computing environment in which several of the described embodiments may be implemented.

[0019] FIG. **13** illustrates a generalized example of a suitable implementation environment in which one or more described embodiments may be implemented.

[0020] FIG. **14** illustrates a generalized example of a mobile computing device in which one or more described embodiments may be implemented.

DETAILED DESCRIPTION

[0021] Techniques and tools are described that relate to different aspects of a user interface that provides visual feedback in response to user input. For example, boundary effects are presented to provide visual cues to a user to indicate that a boundary in a movable user interface element (e.g., the end of a scrollable list) has been reached. As another example, parallax effects are presented in which multiple parallel or substantially parallel layers in a multi-layer user interface move at different rates, in response to user input. As another example, simulated inertia motion of UI elements is used to provide a more natural feel for touch input. Various combinations of features are described. In one implementation, a UI system that accepts touch input includes detailed motion rules (e.g., rules for interpreting different kinds of touch input, rules for presenting inertia motion in UI elements in response to touch input, rules for determining boundaries in UI elements, etc.). The motion rules can be combined with various combinations of optional motion features such as parallax effects, boundary effects, and other visual feedback. The visual feedback that is presented according to motion rules and optional motion features in a UI element can depend on many factors, such as the type of the UI element and the content of the UI element.

[0022] Various alternatives to the implementations described herein are possible. For example, techniques described with reference to flowchart diagrams can be altered by changing the ordering of stages shown in the flowcharts, by repeating or omitting certain stages, etc. As another example, systems described with reference to system diagrams can be altered by changing the ordering of processing stages shown in the diagrams, by repeating or omitting certain stages, etc. As another example, user interfaces described with reference to diagrams can be altered by changing the content or arrangement of user interface features shown in the diagrams, by omitting certain features, etc. As another example, although some implementations are described with reference to specific devices and user input mechanisms (e.g., mobile devices with a touchscreen interface), described techniques and tools can be used with other devices and/or user input mechanisms.

[0023] The various techniques and tools can be used in combination or independently. Different embodiments implement one or more of the described techniques and tools.

I. Overview of Motion Feedback Features for User Interfaces

[0024] As devices have become more complex, and as consumers have become snore demanding, it has become increasingly difficult to design user interfaces that are convenient and pleasing for users, without sacrificing reliability, flexibility, functionality or performance. The feel of a user interface (UI) is becoming increasingly important to distinguish the underlying product from its competitors. An important contributor to the feel of a UI is how it reacts when a user interacts with it. This is especially true for touch-based interfaces.

[0025] Accordingly, techniques and tools are described for providing feedback (e.g., visual cues such as parallax effects, boundary effects, etc.) to users in response to user input (e.g., touch input). In some embodiments, movements in elements (also referred to as "controls") are based at least in part on user input (e.g., gestures on a touchscreen) and an inertia model. For example, a movement in a UI element can be extended beyond the actual size of a gesture on a touchscreen by applying inertia to the movement. Applying inertia to a movement in a UI element typically involves performing one more calculations using gesture information (e.g., a gesture start position, a gesture end position, gesture velocity and/or other information) and one or more inertia motion values (e.g., friction coefficients) to determine a post-gesture state (e.g., a new position) for the UI element. Simulated inertia motion can be used in combination with other effects (e.g., parallax effects, boundary effects, etc.) to provide feedback to a user. In any of the examples herein, movements in UI elements can be rendered for display (e.g., depicting calculated distortion, parallax, or other effects, if any).

[0026] Movement in UI elements typically depends to some extent on user interaction. For example, a user that wishes to navigate from one part of a UI element to another (e.g., from the beginning of a scrollable list to the end of the list) provides user input to indicate a desired movement. The user input can then cause movement in the UI element and potentially other elements in the user interface. In some embodiments, a user causes movement in a display area of a device by interacting with a touchscreen. The interaction can include, for example, contacting the touchscreen with a fingertip, stylus or other object and moving it (e.g., with a flicking or sweeping motion) across the surface of the touchscreen to cause movement in a desired direction. Alternatively, a user can interact with a user interface in some other way, such as by pressing buttons (e.g., directional buttons) on a keypad or keyboard, moving a trackball, pointing and clicking with a mouse, making a voice command, etc.

[0027] The actual amount and direction of the user's motion that is necessary to produce particular movements can vary depending on implementation or user preferences. For example, a user interface system can include a default setting that is used to calculate the amount of motion (e.g., in terms of pixels) as a function of the size or rate of a user movement. As another example, a user can adjust a touchscreen sensitivity control, such that the same motion of a fingertip or stylus on a touchscreen will produce smaller or larger movements, depending on the setting of the control. Gestures can be made in various directions to cause movement in UI elements. For example, upward and downward gestures can cause upward or downward movements, respectively, while rightward and leftward movements can cause rightward and leftward movements, respectively. Upward/downward motion can even be combined with left/right motion for diagonal movements. Other kinds of motion, such as non-linear motion (e.g., curves) or bi-directional motion (e.g., pinch or stretch motions made with multiple contact points on a touchscreen) also can be used to cause movement in UI elements.

Example 1

Inertia, Boundary Effects and Parallax Effects: Overview

[0028] FIG. **1**A is a flow chart showing a general technique **100** for providing motion feedback in a UI. At **101**, a device receives user input indicating motion in a UI element. For example, a UI system on a mobile device receives gesture information corresponding to a gesture on a touchscreen on the mobile device. At **102**, the device determines whether inertia will be applied to the motion indicated by the user input. For example, a UI system determines based on gesture information (e.g., gesture start position, gesture end position, gesture direction, gesture velocity) whether to apply inertia to the motion in the UI element. At **103**, the device determines whether visual effects (e.g., boundary effects, parallax effects, etc.) will be applied to the motion indicated by the user input. For example, the device determines whether to apply a distortion effect (e.g., a compression or squeeze effect) to indicate that a boundary in the UI element (e.g., a boundary at the end of a scrollable list) has been reached. As another example, the device determines whether to apply a parallax effect (e.g., by moving parallel layers in a multi-layer UI element at different rates). The applied effects also can be based on inertia, where inertia is applied to the motion indicated by the user input. For example, if a UI system applies inertia to a movement and calculates, based on the inertia, a new position for a UI element that is outside a boundary for the UI element, the UI system can apply a boundary effect to provide a visual indicator that the boundary has been reached. At **104**, the motion in the UI element is rendered for display.

[0029] FIG. **19** is a flow chart showing a technique **110** for providing boundary effects in combination with inertia motion. At **111**, a UI system receives gesture information corresponding to a gesture. For example, the UI system receives gesture coordinates and velocity information for the gesture. At **112**, the UI system calculates inertia motion based on the gesture information. For example, the UI system determines that inertia motion is applied based on the velocity information, and calculates a duration of inertia motion for the gesture. At **113**, the UI system calculates a post-gesture position based on the gesture information and the inertia motion. For example, the UI system calculates the post-gesture position based on the gesture coordinates and the duration of the inertia motion. At **114**, the UI system determines that a boundary for the UI element has been exceeded. For example, the UI system compares one or more coordinates (e.g., vertical or horizontal coordinates) of the post-gesture position and determines an extent by which the post-gesture position exceeds the boundary. At **115**, the UI system calculates a distortion effect to indicate that the boundary has been exceeded. For example, the UI system calculates a squeeze or compression effect in the content of the UI element based on the extent to which the post-gesture position exceeds the boundary.

[0030] FIG. **1**C is a flow chart showing a technique **120** for providing parallax effects in combination with inertia motion. At **121**, a UI system receives user input indicating motion in a UI element having plural layers. For example, the UI system receives gesture coordinates and velocity information for a gesture on a touch screen, where the gesture is directed to a content layer in multi-layer UI. At **122**, the UI system calculates motion in a first layer based on inertia information and the user input. For example, the UI system determines that inertia motion should be applied to movement in the content layer based on the velocity information, and calculates a duration of inertia motion for the movement. At **123**, the UI system calculates a parallax motion in a second layer based on the first motion in the first layer. For example, the UI system calculates the parallax motion in a layer above the content layer based on the motion in the content layer, with the parallax motion having a different movement rate than the motion in the content layer. The parallax motion also can include inertia motion, or inertia motion can be omitted in the parallax motion.

[0031] In any of the above techniques, any combination of the inertia, boundary, parallax, distortion, and other effects described herein can be applied. Depending on implementation and the type of processing desired, processing stages shown in example techniques **100**, **110**, **120** can be rearranged, added, omitted, split into multiple stages, combined with other stages, and/or replaced with like stages.

Example 2

Inertia, Boundary Effects and Parallax Effects: Detailed Technique

[0032] FIG. **2** is a flow chart showing a detailed example technique **200** for providing visual feedback in a UI in response to a user gesture.

[0033] At **210**, a UI system on a device receives touch input information in a touch input stream. For example, the touch input stream comprises data corresponding to a gesture on a touchscreen of a mobile device. Data received from the touch input stream can include, for example, gesture information such as a gesture start position, a gesture end position, and timestamps for the gesture. The touch input stream is typically received from a device operating system, which converts raw data received from a touch input device (e.g., a touchscreen) into gesture information. Alternatively, data received from the touch input stream can include other information, or gesture information can be received from some other source.

[0034] At **212**, filtering is applied to the touch input stream. In the filtering stage, one or more algorithms are applied to the touch input stream coming from the OS to filter out or correct anomalous data. For example, the filtering stage can correct misaligned touch data caused by jittering (e.g., values that are not aligned with previous inputs) or filter out spurious touch contact points (e.g., incorrect interpretation of a single touch point as multiple touch points that are close together), etc. As another example, if only single-touch-point gestures are allowed, the filtering stage can convert any multi-touch input into a single-touch input. Alternatively, touch input filtering can be performed during generating of the touch input stream (e.g., at the device OS). As another alternative, touch input filtering can be performed during a coordinate space transform stage (e.g., coordinate space transform **220**). As another alternative, touch input filtering can be omitted.

[0035] At **220**, the UI system applies a coordinate space transform to data in the touch input stream corresponding to the gesture. For example, a coordinate space transform is applied to the data from the touch input stream in order to account for possible rotations of the device, scale changes, influence from other animations, etc., in order to properly interpret the original input stream. For example, if a UI element is rotated 90 degrees such that vertical movement in the UI element becomes horizontal movement (or vice versa), a vertical gesture can be transformed to a horizontal gesture (or vice versa) to account for the rotation of the device. If no adjustments are necessary, the coordinate space transform can leave gesture information unchanged. Alternatively the coordinate space transform state can be omitted.

[0036] At **230**, the UI system calculates the velocity at the end of the gesture. For example, the velocity is calculated by determining a first position near the end of the gesture and an end position of the gesture, and dividing by the time elapsed during the movement from the first position near the end of the gesture to the end position. In one implementation, the first position is determined by finding the gesture position at approximately 100 ms prior to the end of the gesture. Measuring velocity near the end of the gesture can help to provide a more accurate motion resulting from the gesture than measuring velocity over the entire course of the gesture. For example, if a gesture starts slowly and ends with a higher velocity, measuring the velocity at the end of the gesture can help to more accurately reflect the user's intended gesture (e.g., a strong flick). Alternatively, the velocity is calculated by determining the distance (e.g., in pixel units) between the start position for the gesture and the end position of the gesture, and dividing by the time elapsed during the movement from the start position to the end position. The time elapsed can be calculated, for example, by computing the difference between a timestamp associated with the start position and a timestamp associated with the end position.

[0037] At **240**, the UI system determines whether the gesture is an inertia gesture. As used herein, an inertia gesture refers to a gesture, such as a flick gesture, capable of causing movement in one or more user interface elements to which inertia can be applied. The UI system can distinguish between a non-inertia gesture and an inertia gesture by determining how quickly the user's finger, stylus, etc., was moving when it broke contact with the touchscreen, and whether the velocity exceeds a threshold. If the gesture ends with a velocity above the threshold, the gesture can be interpreted as an inertia gesture. For example, a gesture that starts with panning motion at a velocity below the threshold and ends with a velocity above the threshold can be interpreted as ending with a flick that causes movement to which inertia can be applied. If the gesture ends with a velocity below the threshold, the gesture can be interpreted as a non-inertia gesture. Exemplary techniques and tools used in some implementations for gesture interpretation are described in detail below.

[0038] If the gesture is an inertia gesture (e.g., a flick gesture), at **250** the UI system determines whether inertia will be applied to the motion indicated by the gesture. For example, the UI system determines based on gesture information (e.g., end-of-gesture velocity) and/or other information (e.g., user preferences) whether to apply inertia to the motion in the UI element. Despite being considered an inertia gesture, a gesture such as a flick may still not have inertia applied to its resulting movements, such as when a flick gesture is received for a UI element that does not support inertia movements, or for a UI element for which inertia movement has been deactivated (e.g., according to user preference).

[0039] If inertia is not to be applied (e.g., when the gesture is not an inertia gesture), at **254** the UI system computes a new position for the UI element based on gesture information (e.g., end-of-gesture position coordinates). If inertia is to be applied, at **252** the UI system computes a new position based on the gesture information (e.g., end-of-gesture position coordinates) and simulated inertia. For example, the simulated inertia involves treating a UI element, or part of a UI element, as a physical object of non-zero mass that moves according to an approximation of Newtonian physics. The approximation can include, for example, a friction coefficient and/or other parameters that control how the movement is calculated and/or rendered.

[0040] When the new position of the UI element has been computed (with or without simulated inertia), the UI system determines at **260** whether boundary feedback will be presented. Determining whether boundary feedback will be presented involves determining whether the new position is within boundaries (if any) of the UI element. For example, in a scrollable list, the UI system can determine whether the new position is calculated to be outside the boundaries of the scrollable list (e.g., below the end of a vertically scrollable list). Some UI elements may not have boundaries that can be exceeded by any permitted motion. For example, a UI element may take the form of a wrappable list, which may have a default entry position but no beginning or end. If the wrappable list is axis-locked (e.g., if movement is only permitted along a vertical axis for a vertically scrolling list), the list may have no boundaries that can be exceeded by any permitted motion. For UI elements without any boundaries, or without boundaries that can be exceeded by permitted motion, the determination of whether the new position is within boundaries can be skipped. Axis locking is described in more detail below.

[0041] If boundary feedback is to be presented, at **262** the UI system applies a boundary effect to the UI element. For example, the UI system can apply a visual distortion effect such as a "squish" or compression of text, images or other visual information in the UI element, to provide a visual cue that a boundary of the UI element has been reached. Boundary effects are described in more detail below.

[0042] The UI system determines at **270** whether parallax feedback will be presented. Determining whether parallax feedback will be presented involves determining whether the UI element has multiple parallel layers or substantially parallel layers that can be moved at different rates based on the same gesture. If parallax feedback is to be presented, at **272** the UI system applies a parallax effect to the UI element. In general, a parallax effect involves movement of multiple parallel layers, or substantially parallel layers, at different rates. Example parallax effects are described in more detail below.

[0043] The processing stages in example technique **200** indicate example flows of information in a UI system. Depending on implementation and the type of processing desired, processing stages can be rearranged, added, omitted, split into multiple stages, combined with other stages, and/or replaced with like stages.

[0044] For example, although example technique **200** shows stages of receiving data from a touch input stream, applying touch input filtering, applying a coordinate space transform, calculating a velocity at the end of a gesture, and determining whether the gesture is an inertia gesture, such processing stages are only exemplary. Gesture information (e.g., gesture velocity, position, whether the gesture is a candidate for simulated inertia, etc.) can be obtained in other ways. As an example, a module that determines whether to apply inertia motion and determines whether to apply boundary feedback or parallax effects can obtain gesture data from another source, such as another module that accepts touch input and makes calculations to obtain gesture information (e.g., gesture velocity, end-of-gesture position).

[0045] As another example, although example technique **200** shows a determination of whether to present boundary feedback occurring before a determination of whether to present parallax feedback, such an arrangement is only exemplary. A determination of whether to present boundary feedback and/or parallax feedback can be performed in other ways. As examples, once a new position has been calculated, determinations of whether to present boundary feedback and/or parallax feedback can occur in parallel, or the determination of whether to present a parallax effect can occur before the determination of whether to present a boundary effect. Such arrangements can be useful, for example, where a gesture may cause movements in multiple parallel layers of a UI element prior to reaching a boundary of the element. A UI system also can determine (e.g., based on characteristics of a current UI element) whether boundary effects and/or parallax effects are not available (e.g., for UI elements that do not have multiple layers or boundaries), and skip processing stages that are not relevant.

II. Boundary Effects

[0046] Boundary feedback can be used to provide visual cues to a user to indicate that a boundary (e.g., a boundary at the end, beginning, or other location) in a UI element (e.g., a data collection such as a list) has been reached. In described implementations, a UI system presents a boundary effect in a UI element (or a portion of a UI element) by causing the UI element to be displayed in a visually distorted state, such as a squeezed or compressed state (i.e., a state in which text, images or other content is shown to be smaller than normal in one or more dimensions), to indicate that a boundary of the UI element has been reached.

[0047] Described techniques and tools for presenting boundary feedback can be applied to any UI element with one or more boundaries that can be manipulated by moving the element. For example, described techniques and tools can be used in an email viewer, such that text in a scrollable email message is distorted (e.g., squeezed or compressed) to indicate that the end of the email message has been reached.

[0048] Boundary effects (e.g., distortion effects) can be presented in different ways. For example, a boundary effect can be held in place for different lengths of time depending on user input and/or design choice. A boundary effect can end, for example, by returning the UI element to a normal (e.g., undistorted) state when a user lifts a finger, stylus or other object to end an interaction with a touchscreen after reaching a boundary, or when an inertia motion has completed. As another example, distortion effects other than a squish, squeeze or compression can be used. One alternative distortion effect is a visual stretch. A stretch effect can be used, for example, in combination with a snap-back animation to indicate that boundary has been reached.

[0049] Boundary effects can be presented even when it is possible to continue a movement beyond a boundary. For example, if a user scrolls to the end of a vertically-oriented list, causing a distortion of text or images at the end of the list, further motion can cause the list to wrap past the boundary and back to the beginning of the list. The UI also can show an element (or part of an element) at the top of the list to indicate that further movement can allow the user to wrap back to the beginning of the list.

Example 3

Boundary Effect: Distortion

[0050] FIG. **3** is a diagram showing a graphical user interface (GUI) presented by a UI system that uses a distortion effect to indicate that a boundary of UI element has been reached. According to the example shown in FIG. **3**, a user **302** (represented by the hand icon) interacts with a list comprising list elements ("Contact1," "Contact2," etc.). In this example, distortion effects depend at least in part on the location of a squeeze point **396**. Some list elements with distortion effects are shown as being outside display area **300**.

[0051] FIG. **3** shows example states **390**-**394**. In state **390**, user **302** interacts with a touchscreen by making an upward motion, indicated by an initial gesture position **350** and an end-of-gesture touch position **352**. The interaction can include, for example, contacting the touchscreen with a fingertip, stylus or other object and moving it (e.g., with a flicking or sweeping motion) along the surface of the touchscreen. Although FIG. **3** shows user **302** interacting with the touchscreen at particular locations in the display area **300**, the UI system allows interaction with other parts of the touchscreen to cause movement in the list. Furthermore, although the example shown in FIG. **3** shows user **302** making an upward motion to scroll towards the end of the list, user **302** also can make other motions (e.g., downward motions to scroll towards the beginning of the list). The UI system can interpret different kinds of upward or downward user movements, even diagonal movements extending to the right or left of the vertical plane, as a valid upward or downward motion.

[0052] From state **390**, the upward motion causes a distortion effect shown in state **392**. In this example, the upward motion is finger-tracking motion caused by a drag gesture, but distortion effects also can be caused by other motion resulting from other kinds of gestures, such as inertia motion caused by a flick gesture. The distortion effect indicates that a boundary in the list has been reached. In the example shown in state FIG. **3**, the entire list is treated as a single surface, as indicated by the single dimension line to the right of the list in states **390**, **392** and **394**, respectively. In state **392**, the list has been squeezed or compressed in a vertical dimension, as shown by the reduced length of the dimension to the right of the list. The text of each list element has been squeezed or compressed in a vertical dimension. The elements are distorted proportionally. The effect in state **392** is as if all the list elements are being compressed against a barrier at the squeeze point **396**.

[0053] In the example shown in state **392**, the squeeze point **396** is indicated at the top of a list, outside the display area **300**. Other squeeze points are also possible. For example, the squeeze point could be at the center of a list (e.g., at item **50** in a **100** item list) or at the top of a visible portion of a list. In this example, the list can be considered as having two parts—one part above the squeeze point, and one part below the squeeze point—where only one part of the list is squeezed. The squeeze point can change dynamically, depending on the state of the list and/or display. For example, a squeeze point can move up or down (e.g., in response to where the center of the list is) as elements are added to or removed from the list, or a squeeze point can update automatically (e.g., when the end of the list has been reached) to be at the top of a visible portion of the list. As another example, a squeeze point can be placed outside of a list. This can be useful to provide more consistent visual feedback, such as when a UI element does not fill the visible area.

[0054] In state **394**, the list has returned to the undistorted state shown in state **390**. For example, the list can return to the undistorted state after the gesture shown in state **390** is ended (e.g., when the user breaks contact with the touchscreen).

[0055] The upward motion shown in FIG. **3** is only an example of a possible user interaction. The same motion and/or other user interactions (e.g., motions having different sizes, directions, or velocities) can cause different effects, different display states, different transitions between display states, etc. For example, a motion that causes a distortion effect in a UI element (e.g., at the end of a vertically scrollable list) also can cause another portion of the UI element (e.g., a list item at the beginning of a vertically scrollable list) to be displayed to indicate availability of a wrapping feature in the list. Further movement can then cause wrapping in the UI element (e.g., from the end back to the beginning of a vertically scrollable list).

[0056] States **390-394** are only examples of possible states. In practice, a UI element can exist in any number of states (e.g., in intermediate states between example states **390-394**, etc.) in addition to, or as alternatives to, the example states **390-394**. For example, it is preferable to show a gradual transition between an undistorted state (e.g., state **390**) and a distorted state (e.g., state **392**), or from a distorted state to an undistorted state, to provide a more natural feel and avoid the appearance of abrupt changes in the display. Intermediate states, such as states that may occur between state **390** and state **392**, or between state **392** and state **394** can show gradually increasing or decreasing degrees of distortion, as appropriate.

III. Parallax Effects

[0057] In described embodiments, a UI system can present parallel, or substantially parallel, movable layers. The UI system can present a parallax effect, in which layers move at different speeds relative to one another. The effect is referred to as a parallax effect because, in a typical example, a layer that is of interest to a user moves at a faster rate than other layers, as though the layer of interest were closer to the user than the other, slower-moving layers. However, the term "parallax effect" as used herein refers more generally to effects in which layers move at different rates relative to one another.

[0058] The rate of movement in each layer can depend on several factors, including the amount of data to be presented visually (e.g., text or graphics) in the layers, or the arrangement of the layers relative to one another. The amount of data to be presented visually in a layer can measured by, for example, determining the length as measured in a horizontal direction of the data as rendered on a display or as laid out for possible rendering on the display. Length can be measured in pixels or by some other suitable measure (e.g., the number of characters in a string of text). A layer with a larger amount of data and moving at a faster rate can advance by a number of pixels that is greater than a layer with a smaller amount of data moving at a slower rate. Layer movement rates can be determined in different ways. For example, movement rates in slower layers can be derived from movement rates in faster layers, or vice versa. Or, layer movement rates can be determined independently of one another. Layers that exhibit parallax effects can be overlapping layers or non-overlapping layers.

[0059] When user interaction causes movement in layers, the movement of the layers is a typically a function of the length of the layers and the size and direction of the motion made by the user. For example, a leftward flicking motion on a touchscreen produces a leftward movement of the layers relative to the display area. Depending on implementation and/or user preferences, user input can be interpreted in different ways to produce different kinds of movement in the layers. For example, a UI system can interpret any movement to the left or right, even diagonal movements extending well above or below the horizontal plane, as a valid leftward or rightward motion of a layer, or the system can require more precise movements. As another example, a UI system can require that a user interact with a part of a touchscreen corresponding to the display area occupied by a layer before moving that layer, or the system can allow interaction with other parts of the touchscreen to cause movement in a layer. As another example, a user can use an upward or downward motion to scroll up or down in a part of the content layer that does not appear on the display all at once.

[0060] In some embodiments, lock points indicate corresponding positions in layers with which a display area of a device will be aligned. For example, when a user navigates to a position on a content layer such that the left edge of the display area is at a left-edge lock point "A," the left edge of display area will also be aligned at a corresponding left-edge lock point "A" in each of the other layers. Lock points also can indicate alignment of a right edge of a display area (right-edge lock points), or other types of alignment (e.g., center lock points). Typically, corresponding lock points in each layer are positioned to account for the fact that layers will move at different speeds. For example, if the distance between a first lock point and a second lock point in a content layer is twice as great as the distance between corresponding first and second lock points in a background layer, the background layer moves at half the rate of the content layer when transitioning between the two lock points.

[0061] In addition to indicating corresponding positions in layers, lock points can exhibit other behavior. For example, lock points can indicate positions in a content layer to which the layer will move when the part of the layer corresponding to the lock point comes into view on the display. This can be useful, for example, when an image, list or other content element comes partially into view near an edge of the display area—the content layer can automatically bring the content element completely into view by moving the layer such that an edge of the display area aligns with an appropriate lock point. A lock animation can be performed at the end of a gesture, such as a flick or pan gesture, to align the layers with a particular lock point. As an example, a lock animation can be performed at the end of a gesture that causes movement of a content layer to a position between two elements in a content layer (e.g., where portions of two images in a content layer are visible in a display area). A UI system can select an element (e.g., by checking which element occupies more space in the display area) and transition to focus on that element using the lock animations. This can improve the overall look of the layers and can be effective in bringing information or functional elements into view in a display area. A lock animation also can be used together with simulated inertia motion. For example, a lock animation can be presented after inertia motion stops, or a lock animation can be blended with inertia motion (such as by extending inertia motion to a lock point, or ending inertia motion early by gradually coming to a stop at a lock point) to present a smooth transition to a lock point.

[0062] The amounts and rates of movements presented in parallax effects can be calculated and presented in different ways. In a detailed example described below, equations are described for calculating parallax effect movements in which a parallax constant is used to determine anew position for a layer after a gesture. As another example, motion in layers and/or other elements, such as lists, can be calculated based on motion ratios. For example, a UI system can calculate motion ratios for a background layer and a title layer by dividing the width of the background layer and the width of the title layer, respectively, by a maximum width of the content layer. Taking into account the widths of the background layer and the title layer, a system can map locations of lock points in the background layer and the title layer, respectively, based on the locations of corresponding lock points in the content layer.

[0063] Movement of various layers can differ depending on context. For example, a user can navigate left from the beginning of a content layer to reach the end of a content layer, and can navigate right from the end of the content layer to reach the beginning of a content layer. This wrapping feature provides more flexibility when navigating through the content layer. Wrapping can be handled by the UI system in different ways. For example, wrapping can be handled by producing an animation that shows a rapid transition from the end of layers such as title layers or background layers back to the beginning of such layers, or vice-versa. Such animations can be combined with ordinary panning movements in the content layer, or with other animations in the content layer, such as locking animations. However, wrapping functionality is not required.

Example 4

Parallax Effects: Multiple Layers with Background Layer

[0064] FIGS. **4**A-4C are diagrams showing a GUI presented by a UI system with three layers **410**, **412**, **414** and a background layer **450**. In this example, a user **302** (represented by the hand icon) interacts with content layer **414** by interacting with a touchscreen having a display area **300**.

[0065] Background layer **450** floats behind the other layers. Data to be presented visually in background layer **450** can include, for example, an image that extends beyond the boundaries of display area **300**. The content layer **414** includes content elements (e.g., images) **430**A-H. Layers **410**, **412** include text information ("Category" and "Selected Subcategory," respectively). The length of content layer **414** is indicated to be approximately twice the length of layer **412**, which is in turn indicated to be approximately twice the length of layer **410**. The length of background layer **450** is indicated to be slightly less than the length of layer **412**.

[0066] In FIGS. **4**A-4C, the direction of motion that can be caused in the layers **410**, **412**, **414**, **450** by user **302** is indicated by a left-pointing arrow and a right-pointing arrow. These arrows indicate possible movements (left or right horizontal movements) of layers **410**, **412**, **414**, **450** in response to user movements. In this example, the system interprets user movements to the left or right, even diagonal movements extending above or below the horizontal plane, as a valid leftward or rightward motion of a layer. Although FIGS. **4**A-4C show user **302** interacting with a portion of display area **300** that corresponds to content layer **414**, the system also allows interaction with other parts of the touchscreen (e.g., parts that correspond to portions of display area **300** occupied by other layers) to cause movement in layers **410**, **412**, **414**, **450**.

[0067] When user input indicates a motion to the right or left, the system produces a rightward or leftward movement of the layers **410**, **412**, **414**, **450** relative to display area **300**. The amount of movement of layers **410**, **412**, **414**, **450** is a function of the data in the layers and the size or rate of the motion made by the user.

[0068] In FIGS. **4**A-4C, example left-edge lock points "A," "B" and "C" are indicated for layers **410**, **412**. **414**. **450**. The left-edge lock points indicate the corresponding position of the left edge of the display area **300** on each layer. For example, when a user navigates to a position on content layer **414** such that the left edge of display area **300** is at lock point "A," the left edge of display area **300** will also be aligned at lock point "A" of the other layers **410**, **412**, **450**, as shown in FIG. **4**A. In FIG. **4**B, the left edge of display area **300** is at lock point "B" in each of the layers **410**, **412**, **414**, **450**. In FIG. **4**C, the left edge of the display area **300** is at lock point "C" in each of the layers **410**, **412**, **414**, **450**.

[0069] The lock points shown in FIGS. **4**A-4C are not generally representative of a complete set of lock points, and are limited to lock points "A," "B" and "C" only for brevity. For example, left-edge lock points can be set for each of the content elements **430**A-430H. Alternatively, fewer lock points can be used, or lock points can be omitted. As another alternative, lock points can indicate other kinds of alignment. For example, right-edge lock points can indicate alignment with the right edge of display area **300**, or center lock points can indicate alignment with the center of display area **300**.

[0070] In this example, layers **410**, **412**, **414**, **450** move according to the following rules, except during wrapping animations:

[0071] 1. Content layer **414** will move at approximately twice the rate of layer **412**, which is approximately half the length of layer **414**.

[0072] 2. Layer **412** will move at approximately twice the rate of layer **410**, which is approximately half the length of layer **412**.

[0073] 3. Content layer **414** will move at approximately four times the rate of layer **410**, which is approximately ¼ the length of layer **414**.

[0074] 4. Background layer **450** will move slower than layer **410**. Although background layer **450** is longer than layer **410**, the distance to be moved between neighboring lock points (e.g., lock points "A" and "B") in layer **410** is greater than the distance between the corresponding lock points in background layer **450**.

[0075] Movement of layers **410**, **412**, **414**, **450** may differ from the rules described above in some circumstances. In this example, wrapping is permitted. User **302** can navigate left from the beginning of content layer **414** (the position shown in FIG. **4**A), and can navigate right from the end of content layer **414** (the position shown in FIG. **4**C). During a wrapping animation, some layers may move faster or slower than during other kinds of movements. In this example, the image in background layer **450** and the text in layers **410** and **412** moves faster when user input causes wrapping back to the beginning of content layer **414**. In FIG. **4**C, display area **300** shows portions of one and two letters, respectively, in layers **410** and **412**, at the end of the respective text strings. Display area **300** also shows the rightmost portion of the image in background layer **450**. A wrapping animation to return to the state shown in FIG. **4**A can include bringing the leftmost portion of the image in background layer **450** and the beginning of the text in layers **410**, **412** into view from the right. This results in a more rapid movement in layers **410**, **412** and **450** than in other contexts, such as the transition from the state shown FIG. **4**A to the state shown in FIG. **4**B.

Example 5

Inertia Motion with Parallax Effects and Boundary Effects

[0076] Parallax effects can be used in combination with boundary effects and inertia motion. For example, boundary effects can be used to indicate when a user has reached a boundary of a layer, or a boundary of an element within a layer. As another example, inertia motion can be used to extend motion of UI elements caused by some gestures (e.g., flick gestures). If inertia motion causes movement of a UI element (e.g., a layer) to extend beyond a boundary, a UI system can present a boundary effect.

[0077] FIG. **5** is a diagram showing two layers **530**, **532**. Display area **300** is indicated by a dashed line and has dimensions typical of displays on smartphones or similar mobile computing devices. The content layer **532** includes content elements **540-544**. In this example, each content element **540-544** comprises an image representing a music album, and text indicating the title of the respective album. The list header layer **530** includes a text string ("Albums"). According to the example shown in FIG. **5**, a user **302** (represented by the hand icon) interacts with content layer **532** by interacting with a touchscreen having the display area **300**. The interaction can include, for example, contacting the touchscreen with a fingertip, stylus or other object and moving it (e.g., with a flicking or sweeping motion) across the surface of the touchscreen.

[0078] FIG. **5** shows example display states **590-594**. In display state **590**, user **302** interacts with a touchscreen by making a flick gesture **510**, which is indicated by a leftward-pointing arrow. The flick gesture **510** causes an inertia motion in content layer **532**, which continues to move after the gesture **510** has ended. Although FIG. **5** shows user **302** interacting with the touchscreen at a particular location in the display area **300**, the UI system allows interaction with other parts of the touchscreen to cause movement. Furthermore, although the example shown in FIG. **5** shows user **302** making a leftward flick gesture, user **302** also can make other motions (e.g., rightward motions to scroll towards the beginning of the list). The UI system can interpret different kinds of leftward or rightward user movements, even diagonal movements extending below or above the horizontal plane, as a valid leftward or rightward motion.

[0079] In response to the flick gesture **510**, the UI system produces leftward movement of the layers **530**, **532** relative to the display area **300**. For example, from display state **590**, the flick gesture **510** causes a leftward movement in the layers and leads to display state **592**, in which element **540** is no longer visible, and elements **542** and **544** have moved to the left. The text string ("Albums") in the list header layer **530** also has moved to the left, but at a slower rate (in terms of pixels) than the content layer **532**. The movement of the layers **530**, **532** is a function of the data in the layers and the velocity of the flick gesture **510**.

[0080] From display state **592**, the inertia motion causes continued leftward movement of the layers **530**, **532** without further input from the user **302**, and leads to display state **594** in which element **542** is no longer visible. The inertia motion causes the content layer to extend beyond a boundary (not shown) to the right of the element **544** in the content layer **532**, which results in a distortion effect in which an image and text in element **544** is squeezed or compressed in a horizontal dimension. The compression is indicated by the reduced length of the dimension lines above the image and text ("Rock & Roll Part in") of element **544**, respectively. The text string ("Albums") in the list header layer **530** also has moved to the left, but at a slower rate (in terms of pixels) than the content layer **532**. The text in list header layer **530** is uncompressed. The distortion effect gives user **302** an indication that the end of the content layer **532** has been reached.

[0081] Although a motion that is calculated to extend beyond a boundary may result in a distortion effect, the boundary need not prevent further movement in the direction of the motion. For example, if wrapping functionality is available, further movement beyond the boundary can cause the content layer **530** to wrap back to the beginning (e.g., back to display state **590**). In state **594**, element **540** at the beginning of the collection is partially visible, indicating that wrapping is available.

[0082] The display can return from display state **594** to display state **592**, transitioning from a display state with a distortion effect to an undistorted display state. This can occur, for example, without any additional input by the user. The length of time that it takes to transition between states can vary depending on implementation.

[0083] Flick gesture **510** is only an example of a possible user interaction. The same gesture **510** and/or other user interactions (e.g., motions having different sizes, directions, or velocities) can cause different effects, different display states, different transitions between display states, etc. Some display states (e.g., display state **594**) may occur only if a gesture results in a post-gesture position that is calculated to go beyond a boundary for the layer.

[0084] Display states **590-594** are only examples of possible display states. In practice, a display can exist in any number of states (e.g., in intermediate states between example states **590-594**, in states with different visible UI elements, etc.) in addition to, or as alternatives to, the example display states **590-594**. For example, it is preferable to show a gradual transition between an undistorted state (e.g., state **592**) and a distorted state (e.g., state **494**), or from a distorted state to an undistorted state, to provide a more natural feel and avoid the appearance of abrupt changes in the display. Intermediate states, such as states that may occur between state **592** and state **594**, can show gradually increasing or decreasing degrees of distortion, as appropriate. As another example, a UI system can provide a boundary effect by compressing the elements **542** and **544** shown in display state **592** without moving the elements **542** and **544** to the left in the display area **300**.

Example 6

Changes in Display Orientation

[0085] Described techniques and tools can be used on display screens in different orientations, such as landscape orientation. Changes in display orientation can occur, for example, where a UI has been configured (e.g., by user preference) to be oriented in landscape fashion, or where a user has physically rotated a device. One or more sensors (e.g., an accelerometer) in the device can be used to detect when a device has been rotated, and adjust the display orientation accordingly.

[0086] In the example shown in FIG. **5**, the display area **300** is oriented in landscape fashion. Content (e.g., data collection elements **540-544** in content layer **532**) and/or other user interface features in the display area **300** can be dynamically adjusted to take into account effects of a reorientation (e.g., a new effective width of the display area **300**, interpreting directions of user interactions differently, etc.). For example, distortion effects can be adjusted, such as by compressing data collection elements in a horizontal dimension instead of a vertical dimension, to account for display reorientation.

[0087] However, such adjustments are not required. For example, if a display area has equal height and width, reorientation of the display area to a landscape orientation will not change the effective width of the display area.

Example 7

Vertical Boundary Effect with Horizontal Parallax Effect

[0088] FIGS. **6**A-**6**E are diagrams showing a content layer **614** that moves in tandem with layer **612** above it. In this example, a user **302** (represented by the hand icon) navigates through content layer **614** by interacting with a touchscreen having the display area **300**. The interaction can include, for example, contacting the touchscreen with a fingertip, stylus or other object and moving it (e.g., with a flicking or sweeping motion) across or along the surface of the touchscreen. The content layer **614** includes game icons **640**, **642**, **644**, lists **650**, **652**, **654**, and avatar **630** (which is described in more detail below in Example 8). The other layers **610**, **612** include text information ("Games" in layer **610**; "Spotlight," "Xbox Live, "Requests" and "Collection" in layer **612**).

[0089] The direction of motion that can be caused by user **302** is indicated by a left-pointing arrow and a right-pointing arrow in FIGS. **6**A-**6**E, along with additional up- and down-pointing arrows in FIGS. **6**A and **6**E. The right-pointing and left-pointing arrows indicate possible movements (left or right horizontal movements) of the layers **610**, **612**, **614** in response to user movements. In addition to movements of entire layers, a user also can cause movements in elements or parts of layers, depending on the data in the layer and how the layer is arranged. For example, a user can cause movements (e.g., vertical movements) in layer elements (e.g., lists in a content layer) that are orthogonal to movements (e.g., horizontal movements) that can be caused in a layer as a whole. Such can include scrolling vertically in a list embedded in a content layer that moves horizontally. Alternatively, a system that presents layers that move vertically can allow horizontal movements in layer elements.

[0090] The up-pointing and down-pointing arrows indicate possible movements of the list **650** in response to user movements. The amount of movement of list **650** can be a function of the size or rate of the motion made by user **302**, and the data in list **650**. Thus, scrolling of the list **650** can be element-by-element, page-by-page of elements, or something in between that depends on size or rate of the motion. In this example, list **650** includes only one element that is not visible in the display area **300**, as shown in FIG. **6**A, so a range of small or large downward movements may be enough to scroll to the end of list **650**. In the example shown in FIG. **6**E, an upward user movement has caused a boundary effect in list **650**, in which the text of elements in the list are squeezed or compressed in a vertical dimension. This effect gives user **302** an indication that the end of the list has been reached.

[0091] In this example, the amount of movement in layers **610**, **612**, **614** is a function of the data in the layers and the size or rate of the motion made by the user. Horizontal movement in layers **610**, **612**, **614** proceeds according to the following rules, except during wrapping animations:

[0092] 1. The horizontal movement of content layer **614** is locked to layer **612**.

[0093] 2. Layers **612** and **614** will each move at approximately three times the rate of layer **610**, which is approximately ⅓ the length of layers **612** and **614**.

[0094] Movement in the layers **610**, **612**, **614** may differ from the rules described above in some circumstances. In the example shown in FIGS. **6**A-**6**E, wrapping is permitted. The arrows indicate that a user can navigate left from the beginning of the content layer **614** (the position shown in FIG. **6**A and FIG. **6**E), and can navigate right from the end of the content layer **614** (the position shown in FIG. **6**D). During a wrapping animation, some layers may move faster or slower than during other kinds of movements. For example, the text in layer **610** can move faster when wrapping back to the beginning of content layer **614**. In FIG. **6**D, display area **300** shows portions of two letters in layer **610**, at the end of the "Games" text string. A wrapping animation to return to the state shown in FIG. **6**A can include bringing the data in layers **610**, **612**, **614** (including the text of layer **610**) into view from the right, resulting in a more rapid movement in layer **610** than in other contexts, such as a transition from the state shown FIG. **6**A to the state shown in FIG. **6**B.

[0095] In FIGS. **6**A-**6**E, example lock points "A," "B," "C" and "D" are indicated for layers **610** and **612**. In terms of horizontal motion, content layer **614** is locked to layer **612**; the lock points indicated for layer **612** also apply to layer **614**. The lock points for each layer indicate the corresponding position of the left edge of the display area **300** on each layer. For example, when a user navigates to a position on content layer **614** such that the left edge of the display area **300** is at lock point "A," the left edge of display area **300** also is aligned at lock point "A" of the other layers **610**, **612**, as shown in FIGS. **6**A and **6**E. In FIG. **6**B, the left edge of the display area **300** is at lock point "B" in each of the layers **610**, **612**, **614**. In FIG. **6**C, the left edge of the display area **300** is at lock point "C" in each of the layers **610**, **612**, **614**. In FIG. **6**D, the left edge of the display area **300** is at lock point "D" in each of the layers **610**, **612**, **614**.

[0096] The lock points shown in FIGS. **6**A-**6**E are not generally representative of a complete set of lock points, and are limited to lock points "A," "B," "C" and "D" only for brevity. For example, right-edge lock points can be added to obtain alignment with the right edge of display area **300**, or center lock points can be added to obtain alignment with the center of display area **300**. Alternatively, fewer lock points can be used, more lock points can be used, or lock points can be omitted.

[0097] User **302** can move left or right in content layer **614** after making an up or down movement in list **650**. The current position of list **650** can be saved, or the system can revert to a default position (e.g., the top-of-list position indicated in FIG. **6**A) when navigating left or right in content layer **614** from list **650**. Although the arrows in FIGS. **6**A-**6**E (and other figures) that indicate possible movements are shown for purposes of explanation, the display area **300** can itself display graphical indicators (such as arrows or chevrons) of possible movements for the layers and/or list.

[0098] The system can interpret user movements to the left or right, even diagonal movements extending above or below the horizontal plane, as a valid leftward or rightward motion. Similarly, the system can interpret upward or downward movements, even diagonal movement extending to the left or right of the vertical plane, as a valid upward or downward motion. Although FIGS. **6**A-**6**E show the user **302** interacting with a portion of the display area **300** that corresponds to the content layer **614**, the system also allows interaction with other parts of the touchscreen (e.g., those that correspond to display area occupied by other layers) to cause movement in the layers **610**, **612**, **614**, list **650**, or other UI elements.

Example 8

Avatar

[0099] In FIGS. **6**A-**6**E, avatar **630** can provide a visual cue to indicate a relationship between or draw attention to parts of the content layer **614**.

[0100] In FIG. **6**B, avatar **630** is positioned between list **652** and list **654**. In FIG. **6**C, avatar **630** floats behind the text of list **654**, but remains completely within display area **300**. In FIG. **6**D, avatar **630** is only partially within display area **300**; the part that is within the display area floats behind game icons **640**, **642**, **644**. The positioning of avatar **630** at the left edge of display area **300** can indicate to the user **302** that information associated with avatar **630** is available if the user **302** navigates in the direction of avatar **630**. Avatar **630** can move at varying speeds. For example, avatar **630** moves faster in the transition between FIGS. **6**B and **6**C than it does in the transition between FIGS. **6**C and **6**D.

[0101] Alternatively, avatar **630** can move in different ways, or exhibit other functionality. For example, a UI system can present a distortion effect in avatar **630** to indicate a user's location in a data collection with which the avatar is associated. Avatar **630** also can be locked to particular position (e.g., a lock point) in content layer **614** or in some other layer, such that avatar **630** moves at the same horizontal rate as the layer to which it is locked. As another alternative, avatar **630** can be associated with a list that can be scrolled up or down, such as list **650**, and move up or down as the associated list is scrolled up or down.

IV. Detailed Implementation

[0102] In this section, a detailed implementation is described comprising aspects of motion feedback including boundary effects and parallax effects, with reference to the following detailed example.

Example 9

Detailed Example

[0103] In this detailed example, a set of equations, coefficients and rules are described that can allow a UI system (e.g., a UI system provided as part of a mobile device operating system) to interpret user input such as touch gestures (including multi-touch gestures with more than one touch contact point) and generate motion feedback in response to user input. Features described in this detailed example include inertia movement, panning and zooming operations, boundary effects, parallax effects, and combinations thereof. Described features can help to provide natural-looking, smooth motion in response to user input (e.g., touch gestures).

[0104] In this detailed example, processing tasks can be handled by different software modules. For example, a module called "ITouchSession" provides coefficients, gesture positions, and gesture velocity information, and a dynamic motion module in a mobile device operating system uses information provided by ITouchSession to generate motion feedback (e.g., parallax effects, boundary effects, etc.). Preferably, gesture information provided to the dynamic motion module is accurate (e.g., with minimal jitter in position information), detailed (e.g., with time stamps on touch input), and low-latency (e.g., under 30 ms). The information (e.g., motion feedback information) generated by the dynamic motion module can be used by other modules, as well. For example, web browsers or other applications that run on the mobile device operating system can use information generated by the dynamic motion module.

[0105] In this detailed example, the dynamic motion resulting from user interaction is defined by a set of motion rules. The motion rules define how different visual elements react on screen in response to different gestures. For example, some rules apply to finger-tracking gestures such as panning or dragging gestures, some rules apply to flick or toss gestures, and some rules apply to pinch or stretch gestures. Additionally, some rules, such as inertia rules, may apply to more than one type of gesture. The specific motion rules that apply to different UI elements (or "controls") are determined by factors such as the control type and control content; not all motion rules will apply to all UI elements. For example, rules for pinch and stretch gestures do not apply to UI elements where pinch and stretch gestures are not recognized. The motion resulting from the application of motion rules to the input stream generated by the user can be further refined by an optional set of modifiers, which are collectively called "optional motion features."

[0106] In this detailed example, touch input interactions that result in dynamic motion comply with the motion rules. Additionally, different UI elements (or "controls") can apply zero or more optional motion features, which can be determined by factors such as the desired motion, control type and control content. For example, a list control may opt to enhance motion feedback with boundary effects, while a panorama control may apply a parallax feature to some of its layers.

[0107] In addition, when a user interacts with a UI element, it can be helpful to provide some immediate (or substantially immediate) visual feedback to the user (e.g., a change in movement in the UI element, or some other effect such as a tilt or highlight). Immediate or substantially immediate feedback helps the user to know that the user interface is responsive to the user's actions.

[0108] In this detailed example, the following motion rules apply in UI elements where the rules (e.g., rules relating to finger-tracking gestures, inertia, boundaries, pinch/stretch gestures) are relevant to the types of motion that are permitted in the respective UI elements. The motion rules can be modified for some UI elements, such as where optional motion features apply to a UI element.

Motion Rule: Finger Tracking

[0109] For finger tracking movements (e.g., movements caused by dragging or panning gestures), the content at the initial gesture point moves in direct correspondence to the gesture. For example, content under the user's finger at an initial touch point moves with the user's finger during the gesture. The current position of a visual element is given by the following equation:

$$p=p_0+(q-q_0) \quad \text{(Eq. 1)}$$

where p is the (x, y) vector that represents the current position of the visual element, p_0 is the (x_0, y_0) vector that represents the visual element position at the beginning of the gesture, q is the (x, y) vector that represents the current touch contact position, and q_0 is the (x_0, y_0) vector that represents the touch contact position at the beginning of the gesture.

Motion Rule: Inertia

[0110] In a UI element that allows inertia movement (e.g., a scrolling list), when the user finishes a gesture (e.g., by lifting a finger or other object to end the interaction with the touchscreen), a velocity and direction for that movement is identified, and the motion initially continues in the same direction and speed as the gesture, as if the visual element was a real, physical object with a non-zero mass. If the motion is not stopped for some other, permissible reason (e.g., where the UI element reaches a boundary or is stopped by another user gesture), the motion gradually decelerates over time, eventually coming to a stop. The deceleration proceeds according to a combination of equations and coefficients, which can vary depending on implementation. Default system-wide coefficient values can be made available. Default system-wide coefficients can help to maintain a consistent feeling across all controls. Alternatively, different equations or coefficients can be used, such as where a particular control has its own friction coefficient for modeling different kinds of motion.

[0111] The velocity (e.g., in pixels/second) at the end of the gesture is computed by the following equation:

$$v=(q-q_0)/(t-t_0), \quad \text{(Eq. 2)}$$

where v is the (v_x, v_y) velocity vector that represents the inertia velocity at the end of the gesture, q is the (x, y) vector that represents the touch contact position at the end of the gesture, q_0 is the (x_0, y_0) vector that represents the touch contact position at the time t_0, t is the timestamp of the last touch input of the gesture, and t_0 is the timestamp of the least recent touch input that happened within some fixed period of time from the last touch input. Alternatively, the velocity can be calculated in another way. For example, a weighted sum of velocities at different time instances can be calculated, with greater weighting for velocities at the end of the gesture. In this detailed example, calculating the velocity is the responsibility of ITouchSession. However, velocity calculations can be handled by other modules.

[0112] The duration of the inertia motion can be computed according to the following equation:

$$t_{max} = \frac{\ln \frac{\gamma}{|v_0|}}{\ln \mu} \quad \text{(Eq. 3)}$$

where t_{max} is the duration of the inertia motion, $|v_0|$ is the magnitude of the initial velocity vector ($|v_0|>\gamma$), μ is a friction coefficient (e.g., MotionParameter_Friction, $0<\mu<1$), and y is a parking speed coefficient (e.g., MotionParameter_ParkingSpeed, $0<\gamma<|v_0|$) that is used to indicate a threshold velocity, below which inertia motion will stop. In one implementation, the friction coefficient is 0.4, and the parking speed coefficient is 60.0. The duration is computed at the start of the inertia motion, and need not be computed again.

[0113] The following equation will compute the current velocity vector v at any given time t:

$$v=V_0 \cdot \mu^t \quad \text{(Eq. 4).}$$

[0114] The new position p' for the visual element can be computed based on its last known position p and the time elapsed since the last position update (Δt), as shown in the following equation:

$$p'=p+v \cdot \Delta t \quad \text{(Eq. 5).}$$

The motion stops once the velocity reaches a value smaller than γ.

[0115] The actual calculation of values relating to inertia motion (e.g., velocity, etc.) can differ depending on implementation.

Motion Rule: Interacting with an Element in Inertia Motion

[0116] If a new gesture begins while a UI element is in inertia motion, the inertia motion is immediately interrupted. Depending on the new gesture, the motion in the UI element may be stopped, or a new motion may start. If the new gesture causes a new motion in the UI element, the new gesture controls the UI element's motion. The previous gesture and any consequent inertia do not affect the motion generated by the new gesture. Handling of new gestures during inertia motion can be different depending on implementation. For example, new gestures can be ignored during inertia motion or can have different effects on inertia motion.

Motion Rule: Gesture Boundaries

[0117] The motion of some UI elements is limited by gesture boundaries. The dimensions of gesture boundaries and the effects of exceeding gesture boundaries can differ depending on several factors, such as the content of a UI element and/or a minimum visible area of the UI element. For example, lists which don't wrap around indefinitely may only be able to scroll a certain distance based on the number of items in the list and a minimum amount of visible items (e.g., an amount of items that occupies most or all of a display area).

[0118] In this detailed example, for an element A having a width w_A, height h_A, total area S_A and position p_A (x_A, y_A), with a minimum visible area S_{Vmin} (width w_{Vmin}, height h_{Vmin}) currently at position p_{Vmin} (x_{Vmin}, y_{Vmin}), a gesture that begins at an initial position q (x_q, y_q) has a rectangular gesture boundary area S_T (width w_T, height h_T) at position $p_T=(x_T, y_T)$. The minimum visible area indicates a minimum visible amount of the control (e.g., a minimum number of list items in a scrollable list), but does not require any particular part of the control to be visible. Therefore, the content of the minimum visible area for a particular control can vary depending on, for example, the control's current state (e.g., whether the end or beginning of a scrollable list is currently visible).

[0119] Conceptually, the position p_T of the gesture boundary area can be defined according to the following equation:

$$p_T=q+(p_{Vmin}+S_{Vmin})-(p_A+S_A) \quad \text{(Eq. 6).}$$

[0120] The x and y coordinates of the position p_T of the gesture boundary area are defined according to the following equations:

$$x_T=x_q+(x_{Vmin}+w_{Vmin})-(x_A+w_A) \quad \text{(Eq. 7)}$$

$$y_T=y_q+(y_{Vmin}+h_{Vmin})-(y_A+h_A) \quad \text{(Eq. 8).}$$

[0121] The dimensions of the gesture boundary area are defined according to the following equation:

$$S_T=S_A-S_{Vmin}=(h_A-h_{Vmin}, w_A-w_{Vmin}) \quad \text{(Eq. 9).}$$

[0122] If the new position of the UI element, resulting from user interaction or simulated inertia or some combination, falls outside the area defined by p_T+S_T, a boundary has been exceeded. An appropriate boundary feedback modifier can be applied or the new position can be clamped (i.e., kept within the allowed boundaries).

[0123] FIG. 7A shows an example boundary diagram for a control having a position **710** and area **720**. The control has a minimum visible area **740** (at position **730**). For example, the position **730** of the minimum visible area can be located at the top left of a display area. Based on an initial gesture position **750**, a gesture boundary at position **770** and having area **780** is calculated.

[0124] In FIG. 7B, example post-gesture positions **752**, **754** are shown. Post-gesture position **752** is outside the gesture boundary area **780**, and causes boundary feedback. Post-gesture position **754** is inside the gesture boundary **780**, and does not cause boundary feedback.

[0125] FIG. **8**A shows an example boundary diagram for a control corresponding to the scrollable list shown in FIG. **3**. In FIG. **8**A, the control at position **810** has a control area **820** (width W_A, height h_A). In this detailed example, the coordinates of the control position are considered to be (0, 0). The control has a minimum visible area **840** (at position **830**). For example, the position **830** of the minimum visible area **840** can be at the top left of a display area. Based on an initial gesture position **850**, a gesture boundary at position **850** (the same position as the initial gesture position) is calculated. In this detailed example, the gesture boundary **880** has a height of h_A-h_{Vmin}, and a width of 0. (Due to space limitations, the boundary **880** as shown in FIG. **8**A is not to scale.) Therefore, the gesture boundary **880** is actually a vertical line. Although a control having a gesture boundary area with no width could cause a boundary feedback effect with any horizontal movement, boundary feedback can be enabled or disabled on an axis basis (e.g., permitting boundary feedback for vertical movements but not for horizontal movements). Such a control also can be a candidate for axis locking, to allow only vertical movements and remove any need for boundary feedback for horizontal movements. Axis locking is explained in more detail below.

[0126] In FIG. **8**B, example post-gesture positions **852**, **854** are shown. Post-gesture position **852** is outside the gesture boundary area **880**, and causes boundary feedback. For example, referring again to FIG. **3**, a UI system can present a squeeze or compression effect to indicate that the post-gesture position is outside the gesture boundary area, as shown in state **392**. Post-gesture position **854** is inside the gesture boundary area **880**, and does not cause boundary feedback.

[0127] The actual calculation of values relating to motion boundaries and the effects of motion boundaries can differ depending on implementation. For example, if a wrapping feature is available in a UI element, a boundary can indicate a position at which a boundary effect will be presented (e.g., to indicate that the end of a list has been reached) without preventing further movement beyond the boundary (e.g., wrapping movement from the end of the list back to the beginning of the list).

Motion Rule: Pinch/Stretch

[0128] Pinch gestures and stretch gestures are gestures that can change the scale (zoom) of the subject area of a control (e.g., a map or image with zoom capability). Pinch gestures and stretch gestures are considered to be multi-touch gestures because they typically have multiple points of interaction. In a typical pinch or stretch gesture scenario, a user places two fingers some distance apart from each other on a touchscreen, and either increases (for a stretch gesture) or decreases (for a pinch gesture) the distance between them.

[0129] FIG. **9** is a diagram showing example pinch and stretch gestures. On a device having a display area **300**, a user **302** (represented by a hand icon) interacts with a control (e.g., a map with zoom features) having a control area **910**. From display state **990**, the user **302** performs a pinch gesture beginning at touch points **950**, **960** and ending at touch points **952**, **962**. This results in a zoomed-out version of the map (relative to display state **990**) in control area **910** in state **992**. From display state **992**, the user **302** performs a stretch gesture beginning at touch points **970**, **980** and ending at touch points **972**, **982**. This results in a zoomed-in version of the map (relative to display state **992**) in control area **910** in state **994**. Alternatively, a pinch or stretch gesture can begin or end at other touch points (e.g., with a greater or lesser distance between beginning and ending touch points) or can use a different orientation of touch points (e.g., horizontal or diagonal).

[0130] The scale adjustment caused by a pinch or stretch gesture can be represented as follows. Let $qA_0=(x_{A0}, y_{A0})$ and $qB_0=(x_{B0}, y_{B0})$ be the positions for initial touch points A and B. The distance d_0 between the two points represents a 100% scale factor, and can be calculated according to the following equation:

$$d_0=|qA_0-qB_0| \quad \text{(Eq. 10).}$$

[0131] The distance d_0 includes a horizontal component x_{d0} and a vertical component y_{d0}.

[0132] Let qA and qB be updated positions for touch points A and B, and let $d=(x_d, y_d)$ be the distance between them, calculated in a similar manner. The distance d also includes a horizontal component x_d and a vertical component y_d. The scale factor s_{zoom} to apply to the UI element can be calculated according to the following equation:

$$s_{zoom} = \frac{d}{d_0} = \left(\frac{x_d}{x_{d0}}, \frac{y_d}{y_{d0}}\right). \quad \text{(Eq. 11)}$$

Note that in Equation 11, the scale s is not isometric, i.e., the X and Y axes will be scaled differently. For isometric scaling, the following equation can be used instead:

$$s_{zoom} = \frac{\sqrt{x_d^2 + y_d^2}}{\sqrt{x_{d0}^2 + y_{d0}^2}}. \quad \text{(Eq. 12)}$$

In this case, s_{zoom} is a scalar, so the same factor is applied to both X and Y components.

[0133] Alternatively, a scale factor can be calculated in different ways. For example, inertia can be applied to a pinch or stretch gesture (such as when the gesture ends with a velocity above a threshold), and the scale factor can be based at least in part on the inertia of the gesture (e.g., increasing the scale of the zoom when a stretch gesture ends with a velocity above a threshold).

[0134] To make zooming feel natural, the scale factor can be applied to a zooming point (e.g., a center point between touch points qA and qB). The zooming point $c_z=(x_{cz}, y_{cz})$ can be calculated by averaging the two touch contact positions, as shown in the following equation:

$$c_z = \frac{qA + qB}{2}. \quad \text{(Eq. 13)}$$

Alternatively, a zooming point can be calculated in a different way, or a calculation of a zooming point can be omitted.

[0135] A pinch/stretch gesture can also produce position changes (panning) in addition to scale changes. Panning position changes can occur simultaneously with scale changes. The zooming point calculation in Equation 13 is used when simultaneous panning is not allowed. If simultaneous panning is allowed, the zooming point is calculated using the initial touch contact positions qA_0 and qB_0 rather than the updated touch contact positions qA and qB. If c_{z0} is the initial zooming point and c_z is the updated zooming point, the distance $d_{pan}=(x_{dpan}, y_{dpan})$ between the two zooming points represents a panning offset to be applied to the UI element, as shown in the following equation:

$$d_{pan}=c_z-c_{z0}=(x_{cz}-x_{cz0}, y_{cz}-y_{cz0}) \quad \text{(Eq. 14)}.$$

Alternatively, a panning offset can be calculated in a different way, or a panning offset can be omitted.

Optional Motion Features: Overview

[0136] Optional motion features can be used (e.g., when requested by a control) to refine or add visual feedback to motion generated by gestures. Optional motion features can depend on control type and content. For example, some controls (e.g., a scrolling list) may use an optional axis locking feature that is appropriate for the orientation of the control (e.g., allowing only vertical movements in a vertically scrolling list). Optional motion features can be used in combination with each other and with various motion rules. For example, a vertically scrolling list can use an axis locking feature and a boundary effect feature, while following rules for inertia motion and finger tracking motion. Different UI elements can use different combinations of rules and optional motion features, even when the different UI elements are visible at the same time. For example, a movable layer can use parallax effects but omit boundary effects, while a vertically scrolling list in the movable layer can use boundary effects but omit parallax effects. UI elements of the same basic type can use different sets of optional motion features. For example, a first pair of movable layers can use parallax effects and move at different rates relative to one another, while a third layer parallel to the first pair remains stationary.

[0137] When present, optional motion features act like filters, modifying the values generated according to other motion rules, such as the motion rules described above.

Optional Motion Features: Axis Locking

[0138] For some controls, it may make sense to permit movement only along a particular axis. For example, it can be useful to restrict movement of a movable, horizontal UI layer (sometimes referred to as a panorama control) to movements along the X axis, or to restrict movement of a vertically scrolling list to movements along the Y axis. In such cases, axis locking can be used as an optional motion feature.

[0139] In this detailed example, axis locking is applied to a UI element by using the relevant equations in the motion rules described above, but only applying an X or Y component (as appropriate) to the motion of the axis-locked UI element. Changes to the other component are ignored and not applied to the UI element's motion.

[0140] Alternatively, axis locking can be performed in another way. For example, in a UI element such as a wheel element that moves about an axis such as a Z axis, axis locking can be used to permit only rotational motion about the axis. As another alternative, axis locking can be omitted.

Optional Motion Features: Parallax Effects

[0141] Parallax effects can be applied to controls that present multiple layers of content. In a parallax effect, multiple layers are animated differently (e.g., moving at different speeds), but the movements of the layers are based on the same input stream generated by the user.

[0142] In a parallax effect, layers that are animated in response to a gesture move at different speeds relative to one another. The layer that the user is interacting with directly (e.g., a content layer) is considered to be the top layer on a Z axis, that is, the layer that is closest to the user. Other layers are considered to be lower layers on a Z axis, that is, further away from the user. Examples of a parallax effects can be seen in FIG. **5** and in FIGS. **6**A-**6**D.

[0143] In this detailed example, a top layer reacts directly to the gesture, and the other layers move at increasingly lower speeds the further they are from the top layer along the Z axis. Mathematically speaking, that can be accomplished by applying a scaling factor to the delta between an initial gesture position and an updated gesture position. The updated gesture position can be obtained directly from user interaction (e.g., in a finger tracking gesture such as a panning gesture) or from a gesture with simulated inertia (e.g., a flick gesture). If k_P is the constant parallax scaling factor to be applied for a particular layer L at initial position $p_L=(x_L, y_L)$, then the parallaxed position $p_P=(x_P, y_P)$ can be computed according to the following equation:

$$p_P=p_L+k_P(q-q_0) \qquad \text{(Eq. 15)}$$

where q is the (x, y) vector that represents the current, post-gesture position (e.g., after the gesture and application of any simulated inertia), and q_0 is the (x_0, y_0) vector that represents the touch contact position at the beginning of the gesture. The parallax constant k_P can vary depending on the application, scenario and/or content of the control. For example, layers with different lengths can have different parallax constants.

[0144] Alternatively, parallax effects can be presented in different ways. For example, parallel layers can move according to the model shown in Equation 18 for some movements or parts of a movement and move according to other models in other movements or parts of a movement. Referring again to FIGS. **4**A-**4**C, parallel layers that exhibit parallax effects can move according to the model shown in Equation 18 in transitions from FIG. **4**A to FIG. **4**B, and from FIG. **4**B to **4**C, and then move according to a specialized wrapping animation if a gesture to the right from the state shown in FIG. **4**C, or inertia motion from an earlier gesture, causes a wrap back to the state shown in FIG. **4**A. As another alternative, parallax effects can be omitted.

Optional Motion Features: First Example Boundary Feedback Model

[0145] When the boundary feedback motion feature is applied, a boundary feedback effect can be applied whenever a gesture would move the UI element past a boundary, either directly (e.g., by a dragging or panning gesture) or indirectly (e.g., by inertia motion generated by a flick gesture). In this first example boundary feedback model, once the UI element hits a boundary the content is compressed in the direction of the motion (e.g., a vertical compression for a vertical motion) up to a certain threshold. If the compression is caused by inertia, the content compresses up to a certain amount based on the velocity at the time the boundary is hit, then decompresses to the original size. If the compression is caused directly (e.g., by dragging), the compression can be held as long as the last touch contact point is held and decompress when the user breaks contact, or decompress after a fixed length of time.

[0146] In this first example boundary feedback model, the compression effect is achieved by applying a scale factor and dynamically placing a compression point to ensure that the effect looks the same regardless of the size of the list. In order to properly compute the motion and scale for the boundary effect, the first step is to identify that a boundary has been crossed and by how much. The boundary motion rule described above illustrates how to compute a boundary position in this first example boundary feedback model, and in the second example boundary feedback model described below.

[0147] Let $q=(x_q, y_q)$ be the unmodified, post-gesture position resulting from an active finger tracking gesture (e.g., a dragging gesture) or from simulated inertia (e.g., from a flick gesture), let x_L be the left boundary, let x_R be the right boundary, let y_T be the top boundary, and let y_B be the bottom boundary. Let $r=(r_x, r_y)$ represent how far the post-gesture position exceeds the boundaries with respect to $\{x_L, x_R, y_T, y_B\}$:

$$r_x=\max(x_L-x_q,0,x_q-x_R) \qquad \text{(Eq. 16)}$$

$$r_y=\max(y_T-y_q,0,y_q-y_B) \qquad \text{(Eq. 17)}$$

In cases where only a vertical or horizontal boundary applies (e.g., in axis-locked elements), r may be calculated in only a vertical or horizontal dimension, as appropriate, while omitting a calculation of the other dimension of r.

[0148] Let S_c be the compressible area with dimensions (w_c, h_c), which is some area equal to or greater than the visible area, depending on the value of coefficients k % and k_+, where $k_\%$ is the compression percentage coefficient (e.g., MotionParameter_CompressPercent ($k_\%\geqq 0$)), and k_+ is the compression offset coefficient (e.g., MotionParameter_CompressOffset{X,Y} ($k_+\geqq 0$)). If $k_\%=0$, then the compressible area matches the size of the visible area and the visual result is that only the visible part of the control is being compressed. If $k_\%=1$, the compressible area matches the entire control area. $k_+=(w_+, h_+)$ allows an increase in the compressible area by a fixed amount, regardless of the control area size. In this detailed example, the compressible area can be calculated according to the following equation:

$$S_c=S_v+k_\%\cdot(S_A-S_V)+k_+ \qquad \text{(Eq. 18)},$$

where S_A is the control area with dimensions (W_A, h_A), and S_V, is the visible area with dimensions (w_V, h_V). In one implementation, the compression percentage coefficient is 0.0 and the compression offset coefficient is $0.5{*}S_V$.

[0149] If the user is actively dragging the content, the compression scale factor $s_{comp}=(s_{compx}, s_{compy})$ to apply to the target UI element can be computed according to the following equations:

$$s_{comp}=\frac{S_c-k_s\cdot r}{S_c} \qquad \text{(Eq. 19)}$$

$$s_{compx}=\frac{w_c-k_s\cdot r_x}{w_c} \qquad \text{(Eq. 20)}$$

$$s_{compy}=\frac{h_c-k_s\cdot r_y}{h_c} \qquad \text{(Eq. 21)}$$

where k_s is the compression factor coefficient (e.g., MotionParameter_CompressFactor ($0<k_s\leqq 1$)), and $r\leqq S_v$. In one implementation, the compression factor coefficient is 0.2. Alternatively, the scale factor and/or the compressible area can be calculated in different ways. For example, different ranges of compression coefficients can be used.

[0150] In words, what is being done here is to find the difference between the compressible area (e.g., in the horizontal or vertical dimensions) and the amount by which the gesture is compressing the compressible area, then calculating the scale factor based on that difference. The compression factor k_s, if it is less than 1, limits how much the value of r (the amount by which the post-gesture position has exceeded the boundary) will cause the compressible area to be compressed. A UI system can then place a distortion point (which can also be referred to as a “squeeze point” or “compression point” when applying compression effects) at the other side of the compressible area (i.e., the side of the compressible area opposite the side where the gesture is being made) and apply that scale factor, resulting in a compression effect.

[0151] Once the user ends the dragging gesture (e.g., by lifting a finger from the touchscreen), and if no wrap-around functionality is available or if the threshold for wrap-around hasn't been reached, the content in the compressible area returns to a decompressed state. In this first example boundary effects model, decompression proceeds according to the appropriate equations set forth below.

[0152] In this first example boundary effects model, if a boundary is exceeded during inertia motion, the following equations are used to compute how far off the boundaries the current position is (r) over time, based on the velocity at the time the boundary was crossed (v_h) and how far off the boundary the position is (r_I) when the following equations are applied:

$$r_0 = \frac{\min(r_I, S_v)}{k_s} \quad \text{(Eq. 22)}$$

$$a = \frac{v_h^2}{2 \cdot \max\left(\frac{S_v}{k_s} r_I, 0.001\right)} \quad \text{(Eq. 23)}$$

$$r = r_0 + |v_h| \cdot t - \frac{a \cdot t^2}{2}. \quad \text{(Eq. 24)}$$

If r<0, the motion is complete. Note that r_I can come either from inertia or from an active drag, such as when a user drags the content into a compressed state, then flicks, generating inertia.

[0153] The compression scale factor $s_{inertiacomp}=(s_{inertiacompx}, s_{inertiacompy})$ to apply during inertia compression can be computed according to the following equations:

$$s_{inertiacomp} = \frac{s_c - r}{s_c} \quad \text{(Eq. 25)}$$

$$s_{inertiacompx} = \frac{w_c - r_x}{w_c} \quad \text{(Eq. 26)}$$

$$s_{inertiacompy} = \frac{h_c - r_y}{h_c} \quad \text{(Eq. 27)}$$

Note that these equations are similar to the case when dragging the content (see Equations 19-21, above), except that the coefficient k_s (the compression factor coefficient) has already been applied in this case in Equations 22 and 23. Alternatively, the scale factor can be calculated in a different way. For example, constants such as the compression factor coefficient k_s or the value 0.001 in Equation 23 can be replaced with other constants depending on implementation.

[0154] In this first example boundary effects model, in addition to computing the scale factor to apply to the target UI element, a compression point $C_{comp}=(c_{compx}, c_{compy})$ is calculated in order to generate the expected visual effect. In practice, a compression point can be at different positions in a UI element. For example, a compression point can be located at or near the center of a UI element, such that half (or approximately half) of the content in the UI element will be compressed. As another example, a compression point can be located at or near a border of UI element, such that all (or approximately all) of the content in the UI element will be compressed. The compression point can vary for different UI elements. Using different compression points can be helpful for providing a consistent amount of distortion in the content of UI elements of different sizes. The compression point position can be computed according to the following equations:

$$c_{compx} = \begin{cases} \text{left} \Rightarrow 1 - \frac{w_c}{w_A} \\ \text{right} \Rightarrow \frac{w_c}{w_A} \\ \text{none} \Rightarrow 0.5 \end{cases} \quad \text{(Eq. 28)}$$

$$c_{compy} = \begin{cases} \text{top} \Rightarrow 1 - \frac{h_c}{h_A} \\ \text{bottom} \Rightarrow \frac{h_c}{h_A} \\ \text{none} \Rightarrow 0.5 \end{cases} \quad \text{(Eq. 29)}$$

Alternatively, compression points can be calculated in a different way, or the calculation of compression points can be omitted.

Optional Motion Features: Second Example Boundary Feedback Model

[0155] In this second example boundary feedback model, the appearance of the boundary feedback can be controlled in finer detail by using more coefficients. Also, regardless of whether the compression is caused directly (e.g., by dragging) or by inertia, the same calculations are used for the compression effects

[0156] Let $q=(x_q, y_q)$ be the unmodified, post-gesture position resulting from an active finger tracking gesture (e.g., a dragging gesture) or from simulated inertia (e.g., from a flick gesture), let x_L be the left boundary, let x_R be the right boundary, let y_T be the top boundary, and let y_B be the bottom boundary. Let $r=(w_r, h_r)$ represent how far the post-gesture position exceeds the boundaries with respect to $\{x_L, x_R, y_T, y_B\}$:

$$w_r=\max(x_L-x_q,0,x_q-x_R) \quad \text{(Eq. 30)}$$

$$h_r=\max(y_T-y_q,0,y_q-y_B) \quad \text{(Eq. 31)}$$

In cases where only a vertical or horizontal boundary applies (e.g., in axis-locked elements), r may be calculated in only a vertical or horizontal dimension, as appropriate, while omitting a calculation of the other dimension of r.

[0157] As in the first example boundary effects model, S_c is the compressible area with dimensions (w_c, h_c), calculated as shown in Equation 18. However, in this second example boundary effects model, given $r=(w_r, h_r)$ and a compressible area $S_c=(w_c, h_c)$, the compression scale factor $s_{comp}=(s_{compx}, s_{compy})$ to apply to the target UI element is computed according to the following equations:

$$d = (\min(w_r, w_c), \min(h_r, h_c)) \quad \text{(Eq. 32)}$$

$$F_s = k_s \cdot d^{k_e} \quad \text{(Eq. 33)}$$

$$r' = d - F_s \cdot (1 - k_d) \cdot \Delta t \quad \text{(Eq. 34)}$$

$$r'' = \max(0, \min(k_L, r')) \quad \text{(Eq. 35)}$$

-continued

$$s_{comp} = \frac{s_c - r''}{s_c} \tag{Eq. 36}$$

$$s_{compx} = \frac{w_c - w_r''}{w_c} \tag{Eq. 37}$$

$$s_{compy} = \frac{h_c - h_r''}{h_c} \tag{Eq. 38}$$

where k_s is a spring factor coefficient (e.g., MotionParameter_SpringFactor ($k_s>0$)), k_e is a spring power coefficient (e.g., MotionParameter_SpringPower ($k_e>0$)), k_d is a damper factor coefficient (e.g., MotionParameter_DamperFactor ($0 \leqq k_d \leqq 1$)), k_L is a compression limit coefficient (e.g., MotionParameter_CompressionLimit ($k_L>0$)), and Δt is the time interval since the last iteration of the simulation ($\Delta t \geqq 0$). The equation for r" imposes limits on the movement in the UI element during boundary feedback. If r"=0, the motion is considered to be complete.

[0158] In this second example boundary effects model, the spring factor coefficient k_s is a number that specifies how much resistance will counteract the inertia force, and the spring power coefficient k_e shapes the curve of the resistance. For example, a spring power coefficient of 1 indicates linear resistance, where resistance increases at a constant rate as compression increases. A spring power coefficient greater than 1 means that the resistance will increase at an increasing rate at higher compression, and less than 1 means that the resistance will increase, but at a decreasing rate, at higher compression. The damper factor coefficient k_d represents a percentage of energy absorbed by the system and taken away from the inertia. The damper factor coefficient can be used to smooth out the boundary effect and avoid a repeated cycle of compression and decompression. The time interval Δt can vary depending on the number of frames per second in the animation of the boundary feedback, hardware speed, and other factors. In one implementation, the time interval is about 16 ms between each update. Varying the time interval can alter the effect of the boundary effect. For example, a smaller time interval can result in more fluid motion.

[0159] Alternatively, the scale factor and/or the compressible area can be calculated in different ways. For example, different ranges or values of coefficients can be used.

[0160] FIG. **10** is a graph of position changes in a UI element over time according to the second example boundary effects model. According to the graph shown in FIG. **10**, a compression effect occurs during the time that the position of the UI element exceeds the boundary position indicated by the dashed line **1010** in FIG. **10**). The compression line can indicate the position of a boundary in a UI element.

[0161] The shape of the position curve **1020** can be modified in different ways, such as by adjusting coefficients. For example, by adjusting the spring power coefficient, the uppermost tip of the boundary effect curve **1020** can be made to go higher (e.g., up to a configurable limit) or lower for a particular initial velocity. A higher tip of the curve can indicate a greater compression effect, and a lower tip can indicate a lesser compression effect. As another example, by adjusting the spring factor coefficient, the duration of the compression can be adjusted to be shorter or longer. In FIG. **10**, the duration is represented by the distance between the points at which the line **1010** is crossed by the curve **1020**. As another example, by adjusting the damper factor coefficient the right-hand tail of the curve (e.g., the part of the curve **1020** after the boundary position line **1010** is crossed for the second time) can be moved up or down, resulting in a more gradual or more abrupt end to the compression effect. Coefficients can be adjusted in combination or independently, and other values besides those indicated can be adjusted as well, to cause changes in position. Different combinations of adjustments can be used to obtain specific shapes in the position curve **1020**.

[0162] In this second example boundary effects model, a current inertia velocity v_c and a current touch contact position q_c can be updated to reflect the physics interaction of the boundary effect. For example, the updated velocity v'_c and updated touch contact position q'_c are calculated according to the following equations:

$$v_n = \max\left(0,\, v_c - \left(F_s \cdot k_d + \max\left(0,\, \frac{r' - k_L}{\Delta t}\right)\right)\right) \tag{Eq. 39}$$

$$v'_c = \begin{cases} v_n, & r'' > 0 \\ 0, & r'' \le 0 \end{cases} \tag{Eq. 40}$$

$$q'_c = \begin{cases} q_c - r', & r'' \ge 0 \\ q_c - d, & r'' < 0 \end{cases} \tag{Eq. 41}$$

[0163] Various alternatives to the boundary feedback models described above are possible. For example, if wrapping beyond a boundary (e.g., wrapping back to the beginning of a list after the end of the list has been reached) is permitted, if the compression is caused by dragging, the list can wrap around once a threshold compression has been reached. As another alternative, boundary effects can be omitted.

Reference Values:

[0164] A UI system can provide programmatic access to system-wide values e.g., (inertia values, boundary effect values). Using system-wide values can help in maintaining consistent UI behavior across components and frameworks, and can allow adjustments to the behavior in multiple UI elements at once. For example, inertia effects in multiple UI elements can be changed by adjusting system-wide inertia values.

[0165] In one implementation, in order to provide frameworks with access to the reference values of each coefficient, an API is included the ITouchSession module (HRESULT GetMotionParameterValue(IN MotionParameter ID, OUT float*value)). In one implementation, the identifiers and default values for the coefficients whose values are accessible through the ITouchSession::GetMotionParameterValue( ) API are as follows:

```
enum MotionParameter
{
    MotionParameter_Friction,           // default: 0.4f
    MotionParameter_ParkingSpeed,       // default: 60.0f
    MotionParameter_MaximumSpeed,       // default: 20000.0f
    MotionParameter_SpringFactor,       // default: 48.0f
    MotionParameter_SpringPower,        // default: 0.75f
```

-continued

```
    MotionParameter__DamperFactor,         // default: 0.09f
    MotionParameter__CompressLimit,        // default: 300.0f
    MotionParameter__CompressPercent,      // default: 0.0f
    MotionParameter__CompressOffsetX,      // default: 720.0f
    MotionParameter__CompressOffsetY,      // default: 1200.0f
};
```

The values that are accessible through the API can vary depending on implementation. For example, a UI system that uses the first example boundary effects model described above can omit values such as spring factor, spring power, and damper factor values. Or, a UI system can use additional values or replace the listed default values with other default values. Values can be fixed or adjustable, and can be updated during operation of the system (e.g., based on system settings or user preferences).

Example 10

UI System

[0166] FIG. **11** is a system diagram showing an example UI system **1100** that presents a UI on a device (e.g., a smartphone or other mobile computing device). In this example, the UI system **1100** is a multi-layer UI system that presents motion feedback (e.g., parallax effects, boundary effects, etc.). Alternatively, the system **1100** presents motion feedback in UIs that do not have multiple UI layers. The system **1100** can be used to implement functionality described in other examples, or other functionality.

[0167] In this example, the system **1100** includes a hub module **1110** that provides a declarative description of a hub page to UI control **1120**, which controls display of UI layers. UI control **1120** also can be referred to as a "panorama" or "pano" control in a multi-layer UI system. Such a description can be used, for example, when the UI layers move in a panoramic, or horizontal, fashion. Alternatively, UI control **1120** controls UI layers that move vertically, or in some other fashion. UI control **1120** includes markup generator **1130** and motion module **1140**.

[0168] The declarative description of the hub page includes information that defines UI elements. In a multi-layer UI system, UI elements can include multiple layers, such as a background layer, a title layer, a section header layer, and a content layer. The declarative description of the hub page is provided to markup generator **1130**, along with other information such as style information and/or configuration properties. Markup generator **1130** generates markup that can be used to render the UI layers. Motion module **1140** accepts events (e.g., direct UI manipulation events) generated in response to user input and generates motion commands. The motion commands are provided along with the markup to a UI framework **1150**. In the UI framework **1150**, the markup and motion commands are received in layout module **1152**, which generates UI rendering requests to be sent to device operating system (OS) **1160**. The device OS **1160** receives the rendering requests and causes a rendered UI to be output to a display on the device. System components such as hub module **1110**, UI control **1120**, and UI framework **1150** also can be implemented as part of device OS **1160**. In one implementation, the device OS **1160** is a mobile computing device OS.

[0169] A user (not shown) can generate user input that affects how the UI is presented. In the example shown in FIG. **11**, the UI control **1120** listens for direct UI manipulation events generated by UI framework **1150**. In UI framework **1150**, direct UI manipulation events are generated by interaction module **1154**, which receives gesture messages (e.g., messages generated in response to panning or flick gestures by a user interacting with a touchscreen on the device) from device OS **1160**. Interaction module **1154** also can accept and generate direct UI manipulation events for navigation messages generated in response to other kinds of user input, such as voice commands, directional buttons on a keypad or keyboard, trackball motions, etc. Device OS **1160** includes functionality for recognizing user gestures and creating messages that can be used by UI framework **1150**. UI framework **1150** translates gesture messages into direction UI manipulation events to be sent to UI control **1120**.

[0170] The system **1100** can distinguish between different gestures on the touchscreen, such as drag gestures, pan gestures and flick gestures. The system **1100** can also detect a tap or touch gesture, such as where the user touches the touchscreen in a particular location, but does not move the finger, stylus, etc. before breaking contact with the touchscreen. As an alternative, some movement is permitted, within a small threshold, before breaking contact with the touchscreen in a tap or touch gesture.

[0171] The system **1100** interprets an interaction as a particular gesture depending on the nature of the interaction with the touchscreen. The system **1100** obtains one or more discrete inputs from a user's interaction. A gesture can be determined from a series of inputs. For example, when the user touches the touchscreen and begins a movement in UI element in a horizontal direction while maintaining contact with the touchscreen, the system **1100** can fire a pan input and begin a horizontal movement in the UI element. The system **1100** can continue to tire pan inputs while the user maintains contact with the touchscreen and continues moving. For example, the system **1100** can fire a new pan input each time the user moves N pixels while maintaining contact with the touch screen. In this way, a continuous physical gesture on a touchscreen can be interpreted by the system **1100** as a series of pan inputs. The system **1100** can continuously update the contact position and rate of movement. When the physical gesture ends (e.g., when user breaks contact with the touchscreen), the system **1100** can determine whether to interpret the motion at the end as a flick by determining how quickly the user's finger, stylus, etc., was moving when it broke contact with the touchscreen, and whether the rate of movement exceeds a threshold.

[0172] The system **1100** can render motion (e.g., motion in a layer, list, or other UI element) on the display differently depending on the type of gesture. For example, in the case of a horizontal drag gesture (in which the user is currently maintaining contact with the touchscreen) on a content layer in a multi-layer UI system, the system **1100** moves the content layer in a horizontal direction by the same distance as the horizontal distance of the drag. In a parallax effect, the title layer and background layer also move in response to the drag. As another example, in the case of a pan gesture (in which the user has ended the gesture) on the content layer, the system **1100** can move the content layer in the amount of the pan, and determine whether to perform an additional movement in the content layer. For example, the system **1100** can perform a locking animation (i.e., an animation of a movement in the content layer to snap to a lock point) and move the content layer to a left or right lock point associated with an item in the content layer. The system **1100** can determine which lock point associated with the current pane is closer, and transition to the closer lock point. As another example, the system **1100** can move the content layer in order to bring an item in the content layer that is in partial view on the display area into full view. Alternatively, the system **1100** can maintain the current position of the content layer. As another example, in the case of a flick gesture (e.g., where the user was moving more rapidly when the user broke contact with the touchscreen) on the content layer, the system **1100** can use simulated inertia to determine a post-gesture position for the content layer. Alternatively, the system **1100** can present some other kind of motion, such as a wrapping animation or other transition animation. The threshold velocity for a flick to be detected (i.e., to distinguish a flick gesture from a pan gesture) can vary depending on implementation.

[0173] The system **1100** also can implement edge tap functionality. In an edge tap, a user can tap within a given margin of edges of the display area to cause a transition (e.g., to a next or previous item in a content layer, a next or previous list element, etc.). This can be useful, for example, where an element is partially in view in the display area. The user can tap near the element to cause the system to bring that element completely into the display area.

V. Extensions and Alternative Implementations

[0174] Various extensions and alternatives to the embodiments described herein are possible.

[0175] For example, described examples show different positions of UI elements (e.g., layers, lists, etc.) that may be of interest to a user. A user can begin navigation of an element at the beginning of an element, or use different entry points. For example, a user can begin interacting in the middle of a content layer, at the end of a content layer, etc. This can be useful, for example, where a user has previously exited at a position other than the beginning of a layer (e.g., the end of a layer), so that the user can return to the prior location (e.g., before and after a user uses an application (such as an audio player) invoked by actuating a content image).

[0176] As another example, other models can be used to model inertia and movement. For example, although some equations are provided in some examples that approximate motion according to Newtonian physics, other equations can be used that model other kinds of motion (e.g., non-Newtonian physics).

[0177] As another example, although controls can share global parameters, such as a global friction coefficient for inertia motion, parameters can be customized. For example, friction coefficients can be customized for specific controls or content, such as friction coefficients that result in more rapid deceleration of inertia motion for photos or photo slide shows.

[0178] As another example, boundary feedback can be applied to pinch and stretch gestures. Such boundary feedback can useful, for example, to indicate that a border of the UI element has been reached.

[0179] As another example, additional feedback on gestures can be used. For example, visual feedback such as a distortion effect can be used to alert a user that a UI element with zoom capability (e.g., a map or image) has reached a maximum or minimum zoom level.

[0180] As another example, boundary effects such as compression effects can themselves produce inertia movement. For example, when a vertically scrolling list is compressed upon reaching the end of the list, and breaking contact with the touchscreen causes the list decompress, the decompression can be combined with a spring or rebound effect, causing the list to scroll in the opposite direction of the motion that originally caused the compression. In this way, the spring effect could provide boundary feedback to indicate that the end of list had been reached, while also providing an alternative technique for navigating the list. The spring effect could be used to cause a movement in the list similar to a flick in the opposite direction. Inertia motion can applied to motion caused by the spring effect.

VI. Example Computing Environment

[0181] FIG. **12** illustrates a generalized example of a suitable computing environment **1200** in which several of the described embodiments may be implemented. The computing environment **1200** is not intended to suggest any limitation as to scope of use or functionality, as the techniques and tools described herein may be implemented in diverse general-purpose or special-purpose computing environments.

[0182] With reference to FIG. **12**, the computing environment **1200** includes at least one CPU **1210** and associated memory **1220**. In FIG. **12**, this most basic configuration **1230** is included within a dashed line. The processing unit **1210** executes computer-executable instructions and may be a real or a virtual processor. In a multi-processing system, multiple processing units execute computer-executable instructions to increase processing power. FIG. **12** shows a second processing unit **1215** (e.g., a GPU or other co-processing unit) and associated memory **1225**, which can be used for video acceleration or other processing. The memory **1220**, **1225** may be volatile memory (e.g., registers, cache, RAM), non-volatile memory (e.g., ROM, EEPROM, flash memory, etc.), or some combination of the two. The memory **1220**, **1225** stores software **1280** for implementing a system with one or more of the described techniques and tools.

[0183] A computing environment may have additional features. For example, the computing environment **1200** includes storage **1240**, one or more input devices **1250**, one or more output devices **1260**, and one or more communication connections **1270**. An interconnection mechanism (not shown) such as a bus, controller, or network interconnects the components of the computing environment **1200**. Typically, operating system software (not shown) provides an operating environment for other software executing in the computing environment **1200**, and coordinates activities of the components of the computing environment **1200**.

[0184] The storage **1240** may be removable or non-removable, and includes magnetic disks, magnetic tapes or cassettes, CD-ROMs, DVDs, memory cards, or any other medium which can be used to store information and which can be accessed within the computing environment **1200**. The storage **1240** stores instructions for the software **1280** implementing described techniques and tools.

[0185] The input device(s) **1250** may be a touch input device such as a keyboard, mouse, pen, trackball or touchscreen, an audio input device such as a microphone, a scanning device, a digital camera, or another device that provides input to the computing environment **1200**. For video, the input device(s) **1250** may be a video card, TV tuner card, or similar device that accepts video input in analog or digital form, or a CD-ROM or CD-RW that reads video samples into the computing environment **1200**. The output device(s) **1260**

may be a display, printer, speaker, CD-writer, or another device that provides output from the computing environment **1200**.

[0186] The communication connection(s) **1270** enable communication over a communication medium to another computing entity. The communication medium conveys information such as computer-executable instructions, audio or video input or output, or other data in a modulated data signal. A modulated data signal is a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media include wired or wireless techniques implemented with an electrical, optical, RF, infrared, acoustic, or other carrier.

[0187] The techniques and tools can be described in the general context of computer-readable media. Computer-readable media are any available media that can be accessed within a computing environment. By way of example, and not limitation, with the computing environment **1200**, computer-readable media include memory **1220**, **1225**, storage **1240**, and combinations thereof.

[0188] The techniques and tools can be described in the general context of computer-executable instructions, such as those included in program modules, being executed in a computing environment on a target real or virtual processor. Generally, program modules include routines, programs, libraries, objects, classes, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The functionality of the program modules may be combined or split between program modules as desired in various embodiments. Computer-executable instructions for program modules may be executed within a local or distributed computing environment. Any of the methods described herein can be implemented by computer-executable instructions encoded on one or more computer-readable media (e.g., computer-readable storage media or other tangible media).

[0189] For the sake of presentation, the detailed description uses terms like "interpret" and "squeeze" to describe computer operations in a computing environment. These terms are high-level abstractions for operations performed by a computer, and should not be confused with acts performed by a human being. The actual computer operations corresponding to these terms vary depending on implementation.

VII. Example Implementation Environment

[0190] FIG. **13** illustrates a generalized example of a suitable implementation environment **1300** in which described embodiments, techniques, and technologies may be implemented.

[0191] In example environment **1300**, various types of services (e.g., computing services **1312**) are provided by a cloud **1310**. For example, the cloud **1310** can comprise a collection of computing devices, which may be located centrally or distributed, that provide cloud-based services to various types of users and devices connected via a network such as the Internet. The cloud computing environment **1300** can be used in different ways to accomplish computing tasks. For example, with reference to described techniques and tools, some tasks, such as processing user input and presenting a user interface, can be performed on a local computing device, while other tasks, such as storage of data to be used in subsequent processing, can be performed elsewhere in the cloud.

[0192] In example environment **1300**, the cloud **1310** provides services for connected devices with a variety of screen capabilities **1320**A-N. Connected device **1320**A represents a device with a mid-sized screen. For example, connected device **1320**A could be a personal computer such as desktop computer, laptop, notebook, netbook, or the like. Connected device **1320**B represents a device with a small-sized screen. For example, connected device **1320**B could be a mobile phone, smart phone, personal digital assistant, tablet computer, and the like. Connected device **1320**N represents a device with a large screen. For example, connected device **1320**N could be a television (e.g., a smart television) or another device connected to a television or projector screen (e.g., a set-top box or gaming console).

[0193] A variety of services can be provided by the cloud **1310** through one or more service providers (not shown). For example, the cloud **1310** can provide services related to mobile computing to one or more of the various connected devices **1320**A-N. Cloud services can be customized to the screen size, display capability, or other functionality of the particular connected device (e.g., connected devices **1320**A-N). For example, cloud services can be customized for mobile devices by taking into account the screen size, input devices, and communication bandwidth limitations typically associated with mobile devices.

VIII. Example Mobile Device

[0194] FIG. **14** is a system diagram depicting an exemplary mobile device **1400** including a variety of optional hardware and software components, shown generally at **1402**. Any components **1402** in the mobile device can communicate with any other component, although not all connections are shown, for ease of illustration. The mobile device can be any of a variety of computing devices (e.g., cell phone, smartphone, handheld computer, personal digital assistant (PDA), etc.) and can allow wireless two-way communications with one or more mobile communications networks **1404**, such as a cellular or satellite network.

[0195] The illustrated mobile device can include a controller or processor **1410** (e.g., signal processor, microprocessor, ASIC, or other control and processing logic circuitry) for performing such tasks as signal coding, data processing, input/output processing, power control, and/or other functions. An operating system **1412** can control the allocation and usage of the components **1402** and support for one or more application programs **1414**. The application programs can include common mobile computing applications (e.g., email applications, calendars, contact managers, web browsers, messaging applications), or any other computing application.

[0196] The illustrated mobile device can include memory **1420**. Memory **1420** can include non-removable memory **1422** and/or removable memory **1424**. The non-removable memory **1422** can include RAM, ROM, flash memory, a disk drive, or other well-known memory storage technologies. The removable memory **1424** can include flash memory or a Subscriber Identity Module (SIM) card, which is well known in GSM communication systems, or other well-known memory storage technologies, such as smart cards. The memory **1420** can be used for storing data and/or code for running the operating system **1412** and the applications **1414**. Example data can include web pages, text, images, sound files, video data, or other data sets to be sent to and/or received from one or more network servers or other mobile devices via one or more wired or wireless networks. The memory **1420** can be used to store a subscriber identifier, such as an International Mobile Subscriber Identity (IMSI), and an equipment identifier, such as an International Mobile Equipment Identifier (IMEI). Such identifiers can be transmitted to a network server to identify users and equipment.

[0197] The mobile device can support one or more input devices **1430**, such as a touchscreen **1432**, microphone **1434**, camera **1436**, physical keyboard **1438** and/or trackball **1440** and one or more output devices **1450**, such as a speaker **1452** and a display **1454**. Other possible output devices (not shown) can include a piezoelectric or other haptic output device. Some devices can serve more than one input/output function. For example, touchscreen **1432** and display **1454** can be combined in a single input/output device.

[0198] Touchscreen **1432** can accept input in different ways. For example, capacitive touchscreens detect touch input when an object (e.g., a fingertip or stylus) distorts or interrupts an electrical current running across the surface. As another example, touchscreens can use optical sensors to detect touch input when beams from the optical sensors are interrupted. Physical contact with the surface of the screen is not necessary for input to be detected by some touchscreens.

[0199] A wireless modem **1460** can be coupled to an antenna (not shown) and can support two-way communications between the processor **1410** and external devices, as is well understood in the art. The modem **1460** is shown generically and can include a cellular modem for communicating with the mobile communication network **1404** and/or other radio-based modems (e.g., Bluetooth or Wi-Fi). The wireless modem **1460** is typically configured for communication with one or more cellular networks, such as a GSM network for data and voice communications within a single cellular network, between cellular networks, or between the mobile device and a public switched telephone network (PSSTN).

[0200] The mobile device can further include at least one input/output port **1480**, a power supply **1482**, a satellite navigation system receiver **1484**, such as a Global Positioning System (GPS) receiver, an accelerometer **1486**, a transceiver **1488** (for wirelessly transmitting analog or digital signals) and/or a physical connector **1490**, which can be a USB port, IEEE 1494 (firewall) port, and/or RS-232 port. The illustrated components **1402** are not required or all-inclusive, as components can be deleted and other components can be added.

[0201] In view of the many possible embodiments to which the principles of the disclosed invention may be applied, it should be recognized that the illustrated embodiments are only preferred examples of the invention and should not be taken as limiting the scope of the invention. Rather, the scope of the invention is defined by the following claims. We therefore claim as our invention all that comes within the scope and spirit of these claims.

We claim:

1. In a computer system, a method comprising:

receiving gesture information corresponding to a gesture on a touch input device;

calculating simulated inertia motion for a movable user interface element based at least in part on the gesture information;

based at least in part on the gesture information and on the simulated inertia motion, calculating a post-gesture position of the movable user interface element;

determining that the post-gesture position exceeds a gesture boundary of the movable user interface element; and

calculating a distortion effect in the movable user interface element to indicate that the gesture boundary has been exceeded.

2. The method of claim **1** wherein calculating the distortion effect comprises:

determining an extent by which the gesture boundary has been exceeded;

determining a compressible area of the movable user interface element; and

determining a scale factor for the distortion effect based at least in part on the compressible area and the extent by which the gesture boundary has been exceeded.

3. The method of claim **2** further comprising scaling the compressible area according to the scale factor.

4. The method of claim **2** wherein calculating the distortion effect further comprises determining a distortion point for the distortion effect.

5. The method of claim **4** further comprising scaling the compressible area according to the scale factor and the distortion point.

6. The method of claim **1** wherein the distortion effect is a squeeze effect.

7. The method of claim **1** further comprising:

displaying a portion of the movable user interface element to indicate availability of a wrapping feature in the movable user interface element.

8. The method of claim **1**, wherein the gesture information comprises gesture coordinates.

9. The method of claim **1** wherein calculating the post-gesture position comprises interrupting the simulated inertia motion when new gesture information corresponding to a new gesture is received.

10. The method of claim **1** wherein calculating simulated inertia motion is further based on inertia information comprising an inertia velocity.

11. The method of claim **10**, wherein the inertia velocity is based at least in part on a friction coefficient.

12. The method of claim **10** wherein calculating simulated inertia motion comprises:

comparing the inertia velocity with a parking speed coefficient; and

determining whether to stop the inertia motion based on the comparing.

13. The method of claim **1** wherein the movable user interface element is an axis-locked user interface element.

14. In a computer system, a method comprising:

receiving user input that indicates movement in a graphical user interface element having plural movable layers;

based at least in part on inertia information and the user input, calculating a first motion having a first movement rate in a first layer of the plural movable layers; and

calculating a parallax motion in a second layer of the plural movable layers, wherein the parallax motion is based at least in part on the first motion, and wherein the parallax motion comprises a movement of the second layer at a second movement rate that differs from the first movement rate.

15. The method of claim **14** wherein calculating the parallax motion is based at least in part on a parallax constant for the second layer.

16. The method of claim **14** wherein calculating the parallax motion is based at least in part on an amount of displayable data in the second layer.

17. The method of claim **14** wherein calculating the first motion comprises applying simulated inertia motion based at least in part on the inertia information.

18. The method of claim **14** wherein the user input is a gesture on a touch screen.

19. The method of claim **18** wherein the inertia information comprises a velocity of the gesture.

20. A computer readable medium having stored thereon computer-executable instructions operable to cause a computer to perform a method comprising:

receiving gesture information corresponding to a gesture on a touch input device, the gesture information indicating a movement of a user interface element having a movement boundary;

based at least in part on the gesture information, computing a new position of the user interface element;

based at least in part on the new position, determining that the movement boundary has been exceeded;

determining an extent by which the movement boundary has been exceeded;

determining a compressible area of the user interface element;

determining a scale factor for a distortion effect based at least in part on the compressible area and the extent by which the movement boundary has been exceeded; and

presenting a distortion effect in the user interface element, wherein the distortion effect comprises a visual compression of content in the compressible area according to the scale factor, wherein the visual compression is in a dimension that corresponds to the movement of the user interface element.

* * * * *