Android multitoque

Toque perfeito

Com a versão 2.0 do Android, os desenvolvedores passaram a ter acesso a recursos multitoque anteriormente reservados a aplicativos do sistema. Mas atenção: é preciso ter cuidado ao manusear a API.

por Markus Junginger

O sucesso do iPhone demonstra que os controles multitoque oferecem muitos benefícios práticos. Não é de se admirar que concorrentes do iPhone trabalhem duro no desenvolvimento de suas próprias tecnologias multitoque. O Google estendeu a versão 2.0 do Android API para que desenvolvedores e usuários pudessem se beneficiar de recursos que envolvem toques e gestos.

Uma infinidade de touchpads

Ao contrário do iOS, o Android precisa lidar com uma variedade de combinações de hardware e software. Primeiro, é preciso trabalhar com uma API que suporte múltiplas entradas por toque. No caso do Android, isto é o equivalente a uma API nível 5 (Android 2.0 ou mais recente). O hardware também influencia no nível de suporte multitoque, e a verdadeira extensão deste suporte nem sempre é transparente. Por exemplo, embora o HTC originalmente suportasse Android 1.0 em seu modelo G1 (antes mesmo do recurso multitoque ter sido anunciado), o hardware já era capaz de identificar diversos tipos de gestos.

Listagem 1: onTouchListener

```
01 public boolean onTouch(View
➥ v, MotionEvent event) {
02   if (event.getAction() ==
➥ MotionEvent.ACTION_DOWN) {
03     int x = event.getX();
04     int y = event.getX();
05     doSomething(x,y);
06   }
07 }
```

No entanto, o reconhecimento de gestos não é necessariamente o mesmo que suporta multitoque. Dispositivos com telas resistentes ao toque, em particular, frequentemente detectam apenas um único toque – isto se aplica, por exemplo, ao HTC Tattoo.

Outros dispositivos retornam apenas uma caixa delimitadora: embora estes dispositivos possam identificar uma caixa entre dois dedos, não são capazes de identificar toques individuais. A **figura 1** ilustra estas restrições de hardware: o desenho mostra dois dedos posicionados de forma distinta em uma caixa delimitadora. Se os dedos do usuário se moverem de uma posição a outra, o dispositivo frequentemente confunde as posições dos dedos e só identificará a caixa com segurança. Muitos aparelhos HTC, como o Desire, usam esta tecnologia.

O primeiro smartphone com Android 2.0, o Motorola Droid, detectava as posições dos dois dedos individualmente. O Galaxy S é capaz de identificar e processar até quatro toques simultâneos. Os desenvolvedores não devem assumir que a tecnologia multitoque estará disponível onde quer que Android esteja presente. Para aplicativos que dependem de múltiplas entradas por toque para recursos críticos, o desenvolvedor deve adicionar uma nota ao manifesto para garantir que somente os dispositivos multitoque sejam capazes de instalar o aplicativo no Google Play:

```
<uses-feature android:name = \
"android.hardware.touchscreen.
➥ multitouch" />
```

Descoberta das entradas multitoque

A API multitoque do Android é baseada no recurso de toque, introduzido no Android 1.0. A extensão principal para multitoque está relacionada

com a classe `MotionEvent`, que inclui recursos adicionais suportados pelo Android 2.0 [1].

Para começar, aceite objetos `MotionEvent`. Um desenvolvedor normalmente instala um recurso `listener` para uma visualização usando `setOnTouchListener()` ou substitui as classes `Activity` ou `View` em `onTouchEvent()`. A **listagem 1** mostra um esqueleto para o `onTouchListener`.

Como é possível notar na **listagem 1**, o desenvolvedor acessa todos os dados relevantes, tais como a ação realizada ou coordenadas *x* e *y*, através dos métodos de acesso a objetos `MotionEvent`. Na sequência de um evento `ACTION_DOWN`, desencadeado pelo Android quando o usuário toca na tela pela primeira vez, o sistema operacional envia eventos `ACTION_MOVE` para movimentação de dedos.

Os eventos continuam até que o usuário pare de tocar na tela. O Android então informa ao aplicativo que o dedo deixou a tela, e envia um `ACTION_UP`. Como indica o método de assinatura `onTouch()`, é possível registrar um `OnTouchListener` com múltiplas visualizações.

Na base desta curta atualização `MotionEvent`, o desenvolvedor poderá criar extensões de múltiplas entradas por toque. Se atualizarmos um aplicativo Android existente, não há necessidade de modificar a estrutura do aplicativo pois os dados das entradas multitoque utilizam o mesmo caminho. Isto fornece um significado adicional ao código do método `getAction()`: o número inteiro de 32-bit codifica dois valores. Podemos isolar esses valores usando bitmasks apropriadas e deslocamentos de bits:

```
int action = event.getAction() &
➥ MotionEvent.ACTION_MASK;
int pointerIndex = (event.getAction()
➥ & MotionEvent.ACTION_POINTER_ID_
➥ MASK) >> MotionEvent.ACTION_
➥ POINTER_ID_SHIFT;
```

A `action` é, portanto, equivalente e familiar ao código premultitoque. A variável `pointerIndex` contém um índice para o indicador responsável pelo evento.

Liberdade para os dedos

Se precisar manter cada dedo na tela por um período de tempo, será necessário utilizar indicadores de ID. O `getPointerId()`, que pressupõe o indicador de index como parâmetro, cuida desta tarefa. O exemplo a seguir demonstra o uso de IDs: o indicador e o dedo médio tocam na tela, um após o outro. Se o usuário manter um dedo da tela, o programa não consegue identificar o dedo sem conhecer o ID do indicador.

No exemplo, o dedo indicador fornece um indicador ID de `0` por ser o primeiro a tocar na tela, e ao dedo médio é atribuído o próximo indicador ID de `1`. Com base nos IDs, o programa pode identificar cada dedo individualmente e, assim, sabe qual dedo ainda está tocando a tela.

A utilização de indicadores múltiplos introduz dois novos códi-

Listagem 2: onTouchListener recebendo Motion Events

```
01 public boolean onTouch(View v, MotionEvent e) {
02 int action = e.getAction() &
03 MotionEvent.ACTION_MASK;
04 int pointerIndex = (e.getAction() &
➥ MotionEvent.ACTION_POINTER_ID_MASK) >>
➥ MotionEvent.ACTION_POINTER_ID_SHIFT;
05 int actionId = e.getPointerId(pointerIndex);
06 pointerCount = e.getPointerCount();
07 if (actionId < MAX_POINTERS) {
08 lastActions[actionId] = action;
09 }
10 for (int i = 0; i < pointerCount; i++) {
11 int pointerId = e.getPointerId(i);
12 if (pointerId < MAX_POINTERS) {
13 points[pointerId] = new PointF(e.getX(i),
14       e.getY(i));
15   if (action == MotionEvent.ACTION_MOVE) {
16     lastActions[pointerId] = action;
17 }
18 }
19 }
20 touchView.invalidate();
21 return true;
22 }
```

Listagem 3: View mostra os valores

```
01 class TouchView extends View {
02   public TouchView(Context context) {
03     super(context);
04     setBackgroundColor(Color.WHITE);
05   }
06
07   protected void onDraw(Canvas canvas) {
08     super.onDraw(canvas);
09     for (int i = 0; i < MAX_POINTERS; i++) {
10       PointF point = points[i];
11       if (point != null) {
12         paint.setColor(getColor(i));
13         canvas.drawCircle(point.x, point.y,
14  radius, paint);
15 String text = getActionText(i);
16 float width = paint.measureText(text);
17 canvas.drawText(text, point.x - width / 2,
18    point.y - radius - calcDevicePixels(8),
19     paint);
20 }
21     }
22    canvas.drawText("Pointer: " + pointerCount,
23       10, calcDevicePixels(30), paintInfoText);
24   }
25 }
```

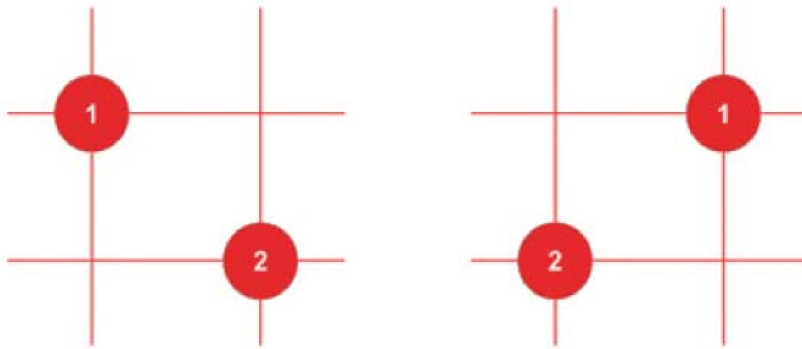

Figura 1 Dedos cruzados causam problemas, a caixa de vínculo para as posições dos dedos é a mesma para ambas as variantes, o que significa que o dispositivo não pode identificar precisamente as posições.

gos de ação: `ACTION_POINTER_DOWN` e `ACTION_POINTER_UP` correspondem a `ACTION_DOWN` e `ACTION_UP`; a diferença é que o Android não os utiliza até que um segundo dedo toque na tela.

O exemplo demonstra a sequência de ações. Primeiro o dedo 1 toca a tela, desencadeando assim uma `ACTION_DOWN`. Quando o dedo 2 toca a tela, é disparado um evento `ACTION_POINTER_DOWN`. Se os dedos se moverem em seguida, o Android registrará um `ACTION_MOVE`. Se o dedo 1 então se erguer para fora da tela, o código fornecerá um `ACTION_POINTER_UP`, considerando que este se trata de um evento `ACTION_UP` para o dedo 2.

O aplicativo *Multitouch Test*, disposto na **figura 2**, resume os principais aspectos relacionados às múltiplas entradas por toque no Android. O aplicativo, disponível no Google Play, serve para visualizar os eventos multitoque e é útil como ponto de partida para experimentação de extensões próprias.

Teste de múltiplos toques

O recurso oferecido pelo aplicativo de teste é limitado: ajuda a visualizar cada indicador a partir de um círculo, que assume diferentes cores dependendo do código utilizado para chamar a ação. O círculo fica azul quando o usuário toca a tela, verde quando o usuário move um dedo, ou cinza quando acionado o código para um dedo sendo levantado fora da tela. O aplicativo mostra também nos círculos o ID do indicador e o último código utilizado para chamar a ação.

É preciso implementar o código do aplicativo no contexto de uma atividade. A **listagem 2** mostra a implementação do `OnTouchListener`, que vincula-se à exibição usando o `setOnTouchListener()`, conforme ilustrado na **listagem 3**. Para economizar espaço, deixamos de fora as definições de membros, tais como as arrays `points[]` e `lastActions[]`. O código-fonte completo está disponível online [2].

As primeiras linhas da **listagem 2** resumem o código da ação e o índice do indicador. Após este passo, o código da ação e da posição do indicador estão descritos para as arrays `lastActions[]` ou `points[]`. O índice de ambas as arrays corresponde à identificação do indicador, em cada caso. O número de indicadores é limitado pelas constantes `MAX_POINTERS`: o aplicativo gerencia um número máximo de 20 indicadores.

Um caso especial ocorre quando são definidos os códigos para uma determinada ação. O `MotionEvent` para o `ACTION_POINTER_DOWN` retorna múltiplos valores, desde que o `ACTION_POINTER_DOWN` refira-se apenas a um único indicador ID. O único caso em que é possível usar os valores `lastActions()` para todos os indicadores no `MotionEvent` é o `ACTION_MOVE`. Por fim, o `touchView.invalidate()` na **linha 20** dispara uma atualização de status e mostra os novos valores.

A visualização neste aplicativo é uma classe separada, diretamente derivada da `View`, como mostra a **listagem 3**. O objeto `Canvas` fornece um método simples de desenhar círculos em um loop. Os códigos para

Quadro 1: Indicadores

Nos círculos multitoque do Android, um *indicador* é representado por um dedo que interage com a tela. Em outras palavras, um indicador pode ser alguma outra fonte de entrada, como um cursor do mouse. Os desenvolvedores enumeram indicadores usando o `getPointerCount()`. Ao invés de métodos de acesso sem parâmetros do `MotionEvent`, temos agora métodos sobrecarregados, cada um com o indicador de índice, como o `getX(int pointerIndex)`. Dependendo do aplicativo, o desenvolvedor pode não precisar de mais nada além de informações para projetar um aplicativo que suporte múltiplas entradas por toque.

Quadro 2: Máscaras e deslocamentos

O Google adicionou recursos que facilitam a estimativa do código de chamada de ação e indicadores de índice. Os desenvolvedores não precisam mais de bitmasks e deslocamentos de bits. Em vez disso, o `getActionMasked()` e o `getActionIndex()` retornam os valores exigidos pelo `MotionEvent`.

Incidentalmente, o Android 2.2 modifica as constantes `ACTION_POINTER_ID_MASK` e `ACTION_POINTER_ID_SHIFT` que ocorrem nos exemplos mostrados neste artigo. Porque eles se relacionam com o indicador de índice mais do que com o ID, são agora mais corretamente entitulados de `ACTION_POINTER_INDEX_MASK` e `ACTION_POINTER_INDEX_SHIFT`.

coordenadas e chamadas de ação necessários para desenhar círculos são coletados pelo `onTouchListener` da **listagem 2** nas arrays `points[]` e `lastActions[]`. A cor e o texto de um círculo são definidos pelos métodos `getColor()` e `getActionText()`. A múltipla escolha determina o valor de retorno, que depende do código da última ação atribuída ao indicador.

Finalmente, o aplicativo também desenha o número de indicadores contidos no `MotionEvent` em formato texto. O método `calcDevicePixels()` é auxiliar e verifica o número específico de pixels do dispositivo, que por sua vez depende da profundidade de pixels dispostos na tela.

Pinch e zoom

Possivelmente, a novidade mais interessante é o `ScaleGestureDetector`, que detecta e amplia gestos. Três etapas são necessárias para utilizá-lo: primeiro, é preciso instanciar um `ScaleGestureDetector`; em seguida, passar cada `MotionEvent` para o `ScaleGesture-Detector`; por último, é necessário um `listener` especial para coletar os novos dados.

A **listagem 4** mostra estas três etapas nos métodos `init()` e `onTouchEvent()`, e na classe `MyScaleListener`. O método `onScale()` escalona a visualização ao multiplicar a propriedade `ScaleFactor` pelo `ScaleGestureDetector`.

Com os métodos `getFocusX()` e `getFocusY()` é possível verificar o foco do dimensionamento.

Conclusão

Hardware e software em dispositivos Android possuem uma enorme influência no suporte multitoque. Telas que suportam múltiplas entradas por toque dependem do Android 2.0 ou mais recente e muitas vezes apresentam capacidades limitadas – um fato que os desenvolvedores precisam ter em mente na construção de seus aplicativos.

A API multitoque, com suas `MotionEvents` e diversos indicadores, asseguram um lugar no pódio para o Android no que se refere a dispositivos dotados deste recurso. ■

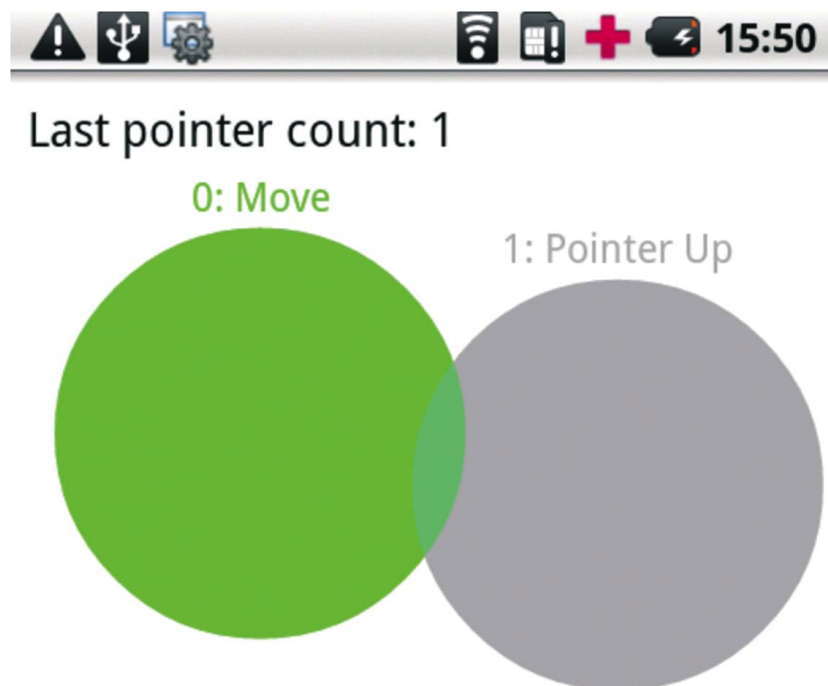

Figura 2 O aplicativo Multitouch Test permite que programadores depurem eventos e os exibam em cores.

Mais informações

[1] API multitoque para Android: http://developer.android.com/reference/android/view/MotionEvent.html

[2] Código-fonte para este artigo: http://smart-developer.com/Resources/Article-Code

Listagem 4: Pinch e Zoom

```
01 // called in constructor, for example
02 private void init(Context context) {
03   scaleDetector = new ScaleGestureDetector(context,
➥ new MyScaleListener());
04 }
05
06 public boolean onTouchEvent(MotionEvent ev) {
07   scaleDetector.onTouchEvent(ev);
08  // ...
09 }
10 private class MyScaleListener
11 extends ScaleGestureDetector.SimpleOnScaleGestureListener {
12   public boolean
13   onScale(ScaleGestureDetector detector) {
14     skalierung *= detector.getScaleFactor();
15     invalidate();
16     return true;
17   }
18 }
```

Gostou do artigo?

Queremos ouvir sua opinião. Fale conosco em cartas@linuxmagazine.com.br

Este artigo no nosso site: http://lnm.com.br/article/7648