Custom View, скроллинг и жесты в Android на примере простого вьювера картинок [из песочницы](https://habrahabr.ru/sandbox/)

[Разработка под Android](https://habrahabr.ru/hub/android_dev/)\*

В статье описываются детали реализации простого вьювера картинок и показываются некоторые тонкости имплементации скроллинга и обработки жестов.  
  
И так, начнем. Ми будем разрабатывать приложения для просмотра картинок. Готовое приложение выглядит так (хотя скриншоты, конечно, слабо передают функционал):  
  
Установить приложение можно либо из [Маркета](https://market.android.com/details?id=com.riilabs.app.SimpleImageViewer), либо установив вручную [отсюда](http://riilabs.com/files/SimpleImageViewer.apk). Исходный код доступен [здесь](http://riilabs.com/files/SimpleImageViewer_src.zip).  
  
Главным элементом нашего приложения является класс ImageViewer который мы и будем разрабатывать. Но нужно также отметить, что для выбора файла для просмотра я не стал изобретать велосипед и взял готовый «компонент» [здесь](http://code.google.com/p/android-file-dialog/).  
  
Компонент представляет собой activity, который вызывается при старте из главного activity. После выбора файла, мы его загружаем и показываем на экране с помощью класса ImageViewer. Рассмотрим класс более подробно.  
  
Класс является наследником класса View и переопределяет только один его метод onDraw. Также класс содержит конструктор и метод загрузки изображения:

**public** **class** **ImageViewer** **extends** **View**

{

**private** Bitmap image = **null**;

**public** **ImageViewer**(Context context)

{

**super**(context);

}

**@Override**

**public** **void** **onDraw**(Canvas canvas)

{

**if** (image != **null**) canvas.drawBitmap(image, 0, 0, **null**);

}

**public** **void** **loadImage**(String fileName)

{

image = BitmapFactory.decodeFile(fileName);

}

}

Если мы загрузим картинку по размерам больше чем экран смартфона, то отобразится только часть ее и у нас не будет способа ее подвинуть или уменьшить.  
  
Добавим теперь возможность скроллинга. Скроллинг по своей сути представляет собой жест, при котором пользователь дотрагивается пальцем к экрану, передвигает его не отрывая, и отпускает. Для того чтоб иметь возможность обрабатывать события связанные с тач-скрином, нужно переопределить метод onTouchEvent. Метод принимает один параметр типа MotionEvent и должен возвратить true в случае обработки события. Через этот метод можно реализовать поддержку любого жеста, включая скроллинг.  
Для распознавания скроллинга нам нужно зафиксировать момент дотрагивания, перемещения и отпускания. К счастью нету необходимости делать это вручную так как в Android SDK есть класс делающий всю работу за нас. Таким образом для того чтоб распознать жест скроллинга, нужно добавить в наш класс поле типа GestureDetector которое инициализируется объектом реализующим интерфейс OnGestureListener (именно этот объект будет получать события скроллинга). Также нужно переопределить метод onTouchEvent в классе ImageViewer и передавать обработку событий из него в наш объект типа OnGestureListener. Измененный класс ImageViewer (без неизмененных методов) представлен ниже:

**public** **class** **ImageViewer** **extends** **View**

{

**private** Bitmap image = **null**;

**private** **final** GestureDetector gestureDetector;

**public** **ImageViewer**(Context context)

{

**super**(context);

gestureDetector = **new** GestureDetector(context, **new** MyGestureListener());

}

**@Override**

**public** **boolean** **onTouchEvent**(MotionEvent event)

{

**if** (gestureDetector.onTouchEvent(event)) **return** **true**;

**return** **true**;

}

**private** **class** **MyGestureListener** **extends** **SimpleOnGestureListener**

{

**@Override**

**public** **boolean** **onScroll**(MotionEvent e1, MotionEvent e2, **float** distanceX, **float** distanceY)

{

scrollBy((**int**)distanceX, (**int**)distanceY);

**return** **true**;

}

}

}

Как видно на самом деле ми наследуем MyGestureListener не от OnGestureListener, а от SimpleOnGestureListener. Последний класс просто реализует интерфейс OnGestureListener с помощью пустых методов. Этим мы избавляем себя от реализации всех методов, выбирая только те, что нужно.  
  
Теперь если загрузить большую картинку, мы, по крайней мере, сможем скролить ее. Но: во первых мы можем проскроллить за рамки картинки, во вторых нету скроллбаров, которые бы подсказали нам где мы находимся и сколько еще осталось до краев.  
  
Решим для начала вторую проблему. Поиск в Интернет приводит нас к тому, что нужно переопределить методы computeHorizontalScrollRange и computeVerticalScrollRange. Эти методы должны возвратить реальные размеры картинки (на самом деле есть еще методы которые имеют отношение к скроллбарам – это методы computeHorizontalScrollExtent, computeHorizontalScrollOffset и такая же пара для вертикального скроллбара. Если переопределить и их, то тогда возвращать можно более произвольные значения). Но этого оказывается недостаточно – скроллбары в первых нужно включить, во вторых проинициализировать. Включаются они методами setHorizontalScrollBarEnabled и setVerticalScrollBarEnabled, инициализируются методом initializeScrollbars. Но вот незадача – последний метод принимает немного непонятный параметр типа TypedArray. Этот параметр должен содержать в себе набор стандартных для View атрибутов. Список можно увидеть [здесь](http://developer.android.com/reference/android/view/View.html) в таблице «XML Attributes». Если бы мы создавали наш view из XML, Android runtime бы автоматически составил такой список. Но так как мы создаем класс программно, нужно также создать этот список программно. Для этого нужно создать файл attrs.xml в каталоге res\values с таким содержимым:

**<?xml version="1.0" encoding="utf-8"?>**

<resources>

<declare-styleable name="View">

<attr name="android:background"/>

<attr name="android:clickable"/>

<attr name="android:contentDescription"/>

<attr name="android:drawingCacheQuality"/>

<attr name="android:duplicateParentState"/>

<attr name="android:fadeScrollbars"/>

<attr name="android:fadingEdge"/>

<attr name="android:fadingEdgeLength"/>

<attr name="android:fitsSystemWindows"/>

<attr name="android:focusable"/>

<attr name="android:focusableInTouchMode"/>

<attr name="android:hapticFeedbackEnabled"/>

<attr name="android:id"/>

<attr name="android:isScrollContainer"/>

<attr name="android:keepScreenOn"/>

<attr name="android:longClickable"/>

<attr name="android:minHeight"/>

<attr name="android:minWidth"/>

<attr name="android:nextFocusDown"/>

<attr name="android:nextFocusLeft"/>

<attr name="android:nextFocusRight"/>

<attr name="android:nextFocusUp"/>

<attr name="android:onClick"/>

<attr name="android:padding"/>

<attr name="android:paddingBottom"/>

<attr name="android:paddingLeft"/>

<attr name="android:paddingRight"/>

<attr name="android:paddingTop"/>

<attr name="android:saveEnabled"/>

<attr name="android:scrollX"/>

<attr name="android:scrollY"/>

<attr name="android:scrollbarAlwaysDrawHorizontalTrack"/>

<attr name="android:scrollbarAlwaysDrawVerticalTrack"/>

<attr name="android:scrollbarDefaultDelayBeforeFade"/>

<attr name="android:scrollbarFadeDuration"/>

<attr name="android:scrollbarSize"/>

<attr name="android:scrollbarStyle"/>

<attr name="android:scrollbarThumbHorizontal"/>

<attr name="android:scrollbarThumbVertical"/>

<attr name="android:scrollbarTrackHorizontal"/>

<attr name="android:scrollbarTrackVertical"/>

<attr name="android:scrollbars"/>

<attr name="android:soundEffectsEnabled"/>

<attr name="android:tag"/>

<attr name="android:visibility"/>

</declare-styleable>

</resources>

В файле просто перечислены все атрибуты, которые были указаны в таблице, упомянутой выше (кроме некоторых на которые указывает компилятор как на ошибку – видимо в документации список приведен самый последний). Измененный класс ImageViewer (кроме неизменных методов):

**public** **class** **ImageViewer** **extends** **View**

{

**private** Bitmap image = **null**;

**private** **final** GestureDetector gestureDetector;

**public** **ImageViewer**(Context context)

{

**super**(context);

gestureDetector = **new** GestureDetector(context, **new** MyGestureListener());

*// init scrollbars*

setHorizontalScrollBarEnabled(**true**);

setVerticalScrollBarEnabled(**true**);

TypedArray a = context.obtainStyledAttributes(R.styleable.View);

initializeScrollbars(a);

a.recycle();

}

**@Override**

**protected** **int** **computeHorizontalScrollRange**()

{

**return** image.getWidth();

}

**@Override**

**protected** **int** **computeVerticalScrollRange**()

{

**return** image.getHeight();

}

}

Не хотелось бы на этом останавливаться, поэтому давайте добавим поддержку жеста «бросок» (fling). Этот жест есть просто дополнение к жесту скроллинга, но учитывается скорость перемещения пальца в последние моменты (перед отпусканием), и если она не нулевая, скроллинг продолжается с постепенным затуханием. Поддержка этого жеста уже заложена в GestureDetector – поэтому нам нужно всего лишь переопределить метод onFling в классе MyGestureListener. Отловив это событие нам нужно еще некоторое время изменять положение скроллинга. Конечно, это можно сделать «вручную» с помощью таймеров или еще как, но опять же в Android SDK уже есть класс, реализующий нужный функционал. Поэтому нужно добавить в класс ImageViewer еще одно поле типа Scroller, которое и будет заниматься «остаточным» скроллингом – для старта скроллинга нужно вызвать его метод fling. Также нужно показать скроллбары (они ведь прячутся когда не нужны) вызовом метода awakenScrollBars. И последнее что нужно сделать – это переопределить метод computeScroll, который должен непосредственно делать скроллинг с помощью метода scrollTo (класс Scroller сам не занимается скроллингом – он просто работает с координатами). Код измененного класса ImageViewer представлен ниже:

**public** **class** **ImageViewer** **extends** **View**

{

**private** Bitmap image = **null**;

**private** **final** GestureDetector gestureDetector;

**private** **final** Scroller scroller;

**public** **ImageViewer**(Context context)

{

**super**(context);

gestureDetector = **new** GestureDetector(context, **new** MyGestureListener());

scroller = **new** Scroller(context);

*// init scrollbars*

setHorizontalScrollBarEnabled(**true**);

setVerticalScrollBarEnabled(**true**);

TypedArray a = context.obtainStyledAttributes(R.styleable.View);

initializeScrollbars(a);

a.recycle();

}

**@Override**

**public** **void** **computeScroll**()

{

**if** (scroller.computeScrollOffset())

{

**int** oldX = getScrollX();

**int** oldY = getScrollY();

**int** x = scroller.getCurrX();

**int** y = scroller.getCurrY();

scrollTo(x, y);

**if** (oldX != getScrollX() || oldY != getScrollY())

{

onScrollChanged(getScrollX(), getScrollY(), oldX, oldY);

}

postInvalidate();

}

}

**private** **class** **MyGestureListener** **extends** **SimpleOnGestureListener**

{

**@Override**

**public** **boolean** **onFling**(MotionEvent e1, MotionEvent e2, **float** velocityX, **float** velocityY)

{

scroller.fling(getScrollX(), getScrollY(), -(**int**)velocityX, -(**int**)velocityY, 0, image.getWidth() - getWidth(), 0, image.getHeight() - getHeight());

awakenScrollBars(scroller.getDuration());

**return** **true**;

}

}

}

В завершения разговора о жесте fling надо сделать одну мелочь – при прикосновении пальцем во время скроллинга от броска, нужно остановить скроллинг. На этот раз мы это сделаем «вручную» в методе onTouchEvent. Измененный метод представлен ниже:

**@Override**

**public** **boolean** **onTouchEvent**(MotionEvent event)

{

*// check for tap and cancel fling*

**if** ((event.getAction() & MotionEvent.ACTION\_MASK) == MotionEvent.ACTION\_DOWN)

{

**if** (!scroller.isFinished()) scroller.abortAnimation();

}

**if** (gestureDetector.onTouchEvent(event)) **return** **true**;

**return** **true**;

}

Уже можно любоваться достаточно интересной физикой, но можно увидеть некоторые «глюки» при скроллинге за пределы картинки. Это происходит из-за того, что fling работает только в пределах картинки, а скроллинг без броска работает везде. Т.е. мы сможем выйти за рамки картинки только если очень плавно скролить (чтоб не срабатывал fling). Исправить этот «косяк» можно путем введения ограничение на обработку в метод onFling и обрабатывать бросок только если он не выходит за границы картинки. Измененный метод представлен ниже:

**@Override**

**public** **boolean** **onFling**(MotionEvent e1, MotionEvent e2, **float** velocityX, **float** velocityY)

{

**boolean** scrollBeyondImage = ((getScrollX() < 0) || (getScrollX() > image.getWidth()) || (getScrollY() < 0) || (getScrollY() > image.getHeight()));

**if** (scrollBeyondImage) **return** **false**;

scroller.fling(getScrollX(), getScrollY(), -(**int**)velocityX, -(**int**)velocityY, 0, image.getWidth() - getWidth(), 0, image.getHeight() - getHeight());

awakenScrollBars(scroller.getDuration());

**return** **true**;

}

Теперь мы опять можем беспрепятственно скролить за рамки картинки. Кажется, эту проблему мы уже вспоминали… У нее есть элегантное решение, лежащее в том, что при отпускании пальца (при завершении скроллинга за рамками картинки) нужно картинку плавно вернуть в «положенное» место. И опять мы это сделаем «вручную» в методе onTouchEvent:

**@Override**

**public** **boolean** **onTouchEvent**(MotionEvent event)

{

*// check for tap and cancel fling*

**if** ((event.getAction() & MotionEvent.ACTION\_MASK) == MotionEvent.ACTION\_DOWN)

{

**if** (!scroller.isFinished()) scroller.abortAnimation();

}

**if** (gestureDetector.onTouchEvent(event)) **return** **true**;

*// check for pointer release*

**if** ((event.getPointerCount() == 1) && ((event.getAction() & MotionEvent.ACTION\_MASK) == MotionEvent.ACTION\_UP))

{

**int** newScrollX = getScrollX();

**if** (getScrollX() < 0) newScrollX = 0;

**else** **if** (getScrollX() > image.getWidth() - getWidth()) newScrollX = image.getWidth() - getWidth();

**int** newScrollY = getScrollY();

**if** (getScrollY() < 0) newScrollY = 0;

**else** **if** (getScrollY() > image.getHeight() - getHeight()) newScrollY = image.getHeight() - getHeight();

**if** ((newScrollX != getScrollX()) || (newScrollY != getScrollY()))

{

scroller.startScroll(getScrollX(), getScrollY(), newScrollX - getScrollX(), newScrollY - getScrollY());

awakenScrollBars(scroller.getDuration());

}

}

**return** **true**;

}

Вот теперь с уверенностью можно сказать что со скроллингом мы разобрались. Можем переходить к последнему жесту который хотелось бы реализовать – это жест pinch zoom.  
  
Со стороны жест выглядит как растягивание или сжатие чего-то воображаемого на экране смартфона двумя пальцами. Пошагово жест происходит так: нажатие одним пальцем, нажатие вторым пальцем, изменение положения одного или двух пальцев не отпуская, отпускание второго пальца. Для определения величины масштабирования нужно вычислить соотношение между расстояниями между пальцами в момент начала жеста и в момент окончания жеста. Расстояние между пальцами находится по формуле sqrt(pow(x2 – x1, 2) + pow(y2 – y1, 2)). Также нужно отметить некоторое положение скроллинга которое нужно сохранять – ведь если жестом увеличить картинку, то положение скроллинга изменится (из-за измененного размера картинки). Это положение – а точнее точка, положение которой нужно сохранить, в терминологии Android SDK называется фокальной точкой, и находиться она посередине между двумя пальцами.  
Реализовать жест как всегда можно самому, но и это к счастью уже реализовано в Android SDK (правда, только начиная с версии 2.2). Поможет в этом класс ScaleGestureDetector, инстанс которого добавим в наш класс. ScaleGestureDetector инициализируется обьектом, поддерживающим интерфейс OnScaleGestureListener, поэтому создадим также внутренний класс MyScaleGestureListener, который реализует методы onScaleBegin, onScale и onScaleEnd. Не забываем передать управление ScaleGestureDetector из метода onTouchEvent. Ну и самое главное – нужно как-то использовать данные масштабирования: их нужно учитывать во всех местах, где раньше фигурировали ширина и высота картинки (т.е. фактически нужно умножить эти параметры на коэффициент масштабирования). Финальный код класса ImageViewer можно посмотреть в исходниках.  
На этом все. Надеюсь статься окажется полезной.