Debugging Android UI

Tomasz Cielecki

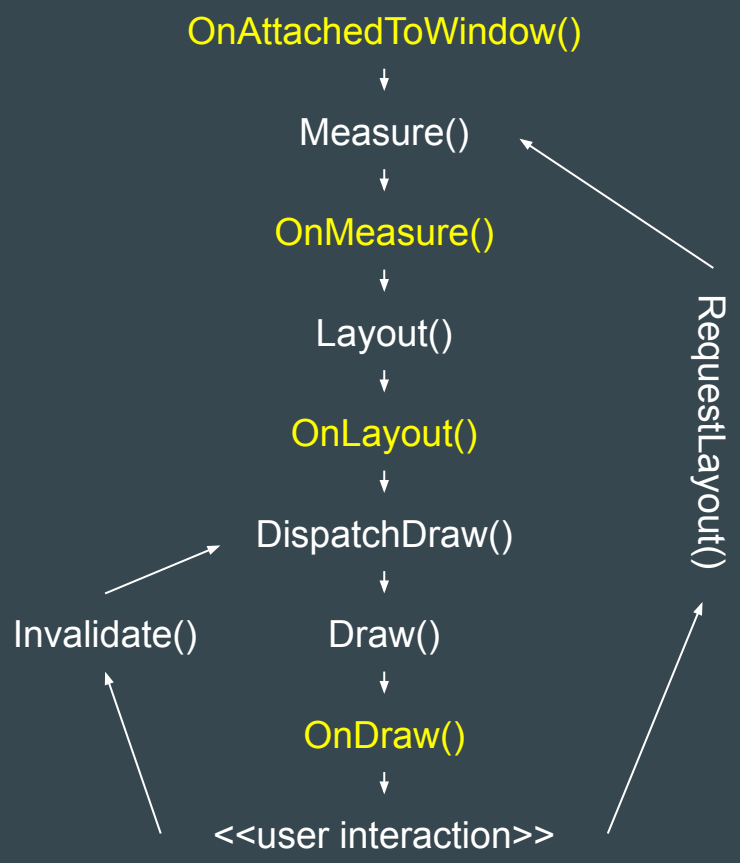
@Cheesebaron

What am I going to talk about?

- Introduction to how *Android* layouts and draws views
  - OnMeasure
  - OnLayout
  - OnDraw
- Examples of views with bad layout performance
  - Overdraw
  - Expensive Layouts
- Tools to diagnose issues + common fixes
- What to avoid doing in OnDraw methods

🙋‍♂️ Feel free to interrupt me and ask questions during the presentation 🙋‍♀️

View / ViewGroup lifecycle



How does Android draw views?

- Bottom up approach
  - Parents drawn before (behind) children
- Each *ViewGroup* is responsible to tell children to be drawn
- Each *View* is responsible of drawing itself

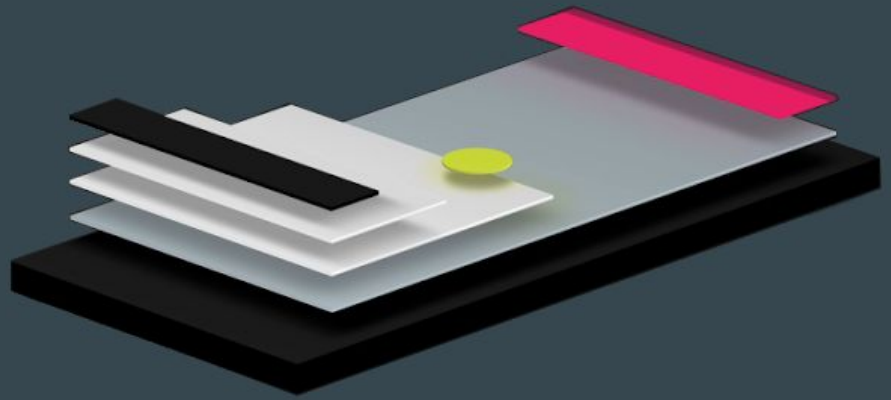


Image Source:
https://blogs.adobe.com/creativecloud/xd-essentials-layered-interface-techniques-for-mobile-apps/

Two Pass Process

- Measure pass implemented in *OnMeasure(widthSpec, heightSpec)*
  - Uses two classes to communicate dimensions
    - Pushes ***MeasureSpec*** requirements down the tree from parent to child
      - *Unspecified*, *Exactly*, *At Most*
    - ***LayoutParams*** allows the developer to request size
      - Exact number (px, dp, sp, pt), *match_parent*, *wrap_content*
  - At end every View has stored its measurements
    - *GetMeasuredWidth()* and *GetMeasuredHeight()* values ***must*** *be set also for descendants*
- Layout pass implemented in *OnLayout(left, top, right, bottom)*
  - Takes measurements found in Measure pass and calls *OnLayout* on children
  - Each parent is responsible for positioning children

DEMO

OnMeasure, OnLayout and OnDraw in action

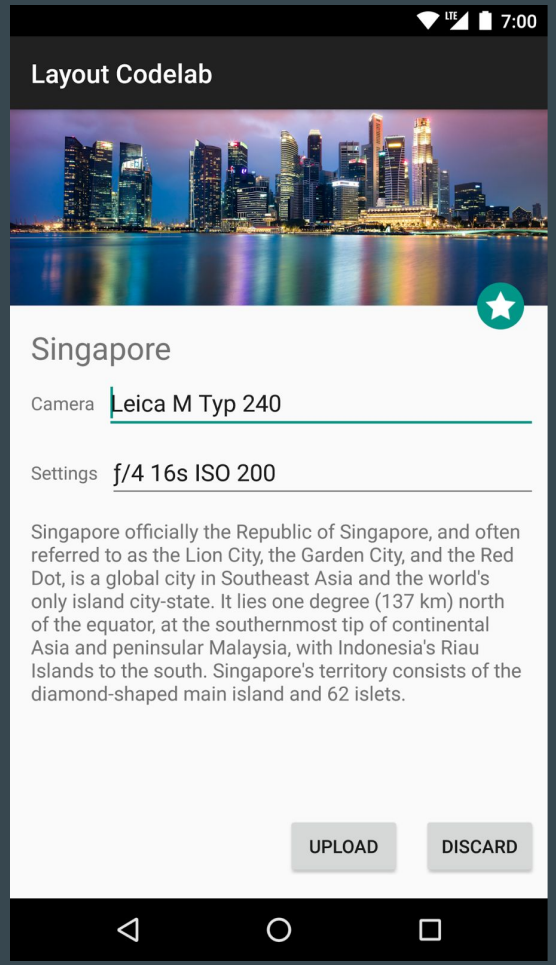
Expensive layouts

- RelativeLayout
  - always runs 2 measure passes
- LinearLayout
  - at least 2 measure passes when:
    - using *layout_weight*
    - orientation is set to horizontal
- GridLayout
  - *layout_gravity* set to *fill* requires 2 measure passes

- Avoid deeply nested hierarchies

```
<RelativeLayout>
  <ImageView />
  <ImageView />
  <RelativeLayout>
    <TextView />
    <LinearLayout>
      <TextView />
      <RelativeLayout>
        <EditText />
      </RelativeLayout>
    </LinearLayout>
    <LinearLayout>
      <TextView />
      <RelativeLayout>
        <EditText />
      </RelativeLayout>
    </LinearLayout>
    <TextView />
  </RelativeLayout>
  <LinearLayout >
    <Button />
    <Button />
  </LinearLayout>
</RelativeLayout>
```

```
<ConstraintLayout>
  <ImageView />
  <ImageView />
  <TextView />
  <EditText />
  <TextView />
  <TextView />
  <EditText />
  <Button />
  <Button />
  <TextView />
</ConstraintLayout>
```



Source: https://android-developers.googleblog.com/2017/08/understanding-performance-benefits-of.html

DEMO

Comparing nested layout to a constraint layout

What is overdraw?

- Drawing the same pixel multiple times in the same frame of rendering

- Happens when
  - Having a number of stacked layouts
  - Each layout hides a portion of the one below it

How to fix overdraw?

- Removing unneeded backgrounds
- Flattening the view hierarchy
- Reducing transparency

DEMO

Visualizing overdraw and common fixes

Why custom views?

- Reduce complex view hierarchy
- Desired view does not exist
- Existing views are not good enough or cannot be extended

DEMO

Look at a custom view

What to keep in mind in OnDraw?

- Don't take too long
  - A frame at 60 FPS is around 16ms
- Stay away from allocating new paints and drawing objects in OnDraw
  - Reuse existing ones
- Do not use large Paths if you are modifying them next frame. Use line, circle and other methods on canvas
- Do not draw bitmaps larger than the canvas
- If you are using bitmaps for caching. Call *SetLayerType(LayerType.Hardware)*
- *ClipPath()* is super expensive!

Resources

- https://developer.android.com/guide/topics/ui/how-android-draws.html
- http://www.vogella.com/tutorials/AndroidCustomViews/article.html
- https://developer.android.com/topic/performance/rendering/optimizing-view-hierarchies.html
- https://developer.android.com/studio/profile/inspect-gpu-rendering#profile_rendering
- https://developer.android.com/topic/performance/vitals/render.html
- https://android-developers.googleblog.com/2017/08/understanding-performance-benefits-of.html