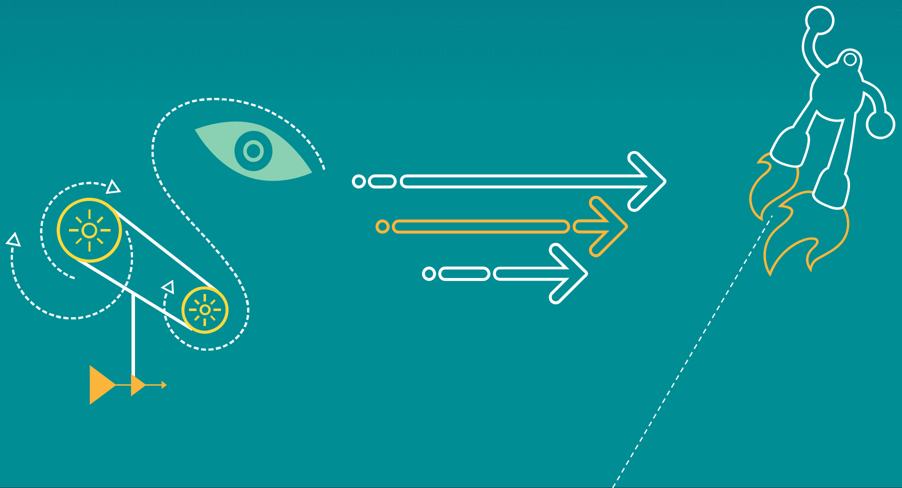
App Optimization Guide Based on Tile Decoding for Android N&O

Qualcomm Technologies, Inc.

80-PD503-1 A



Revision History

Revision	Date	Description
A	August 2017	Initial release

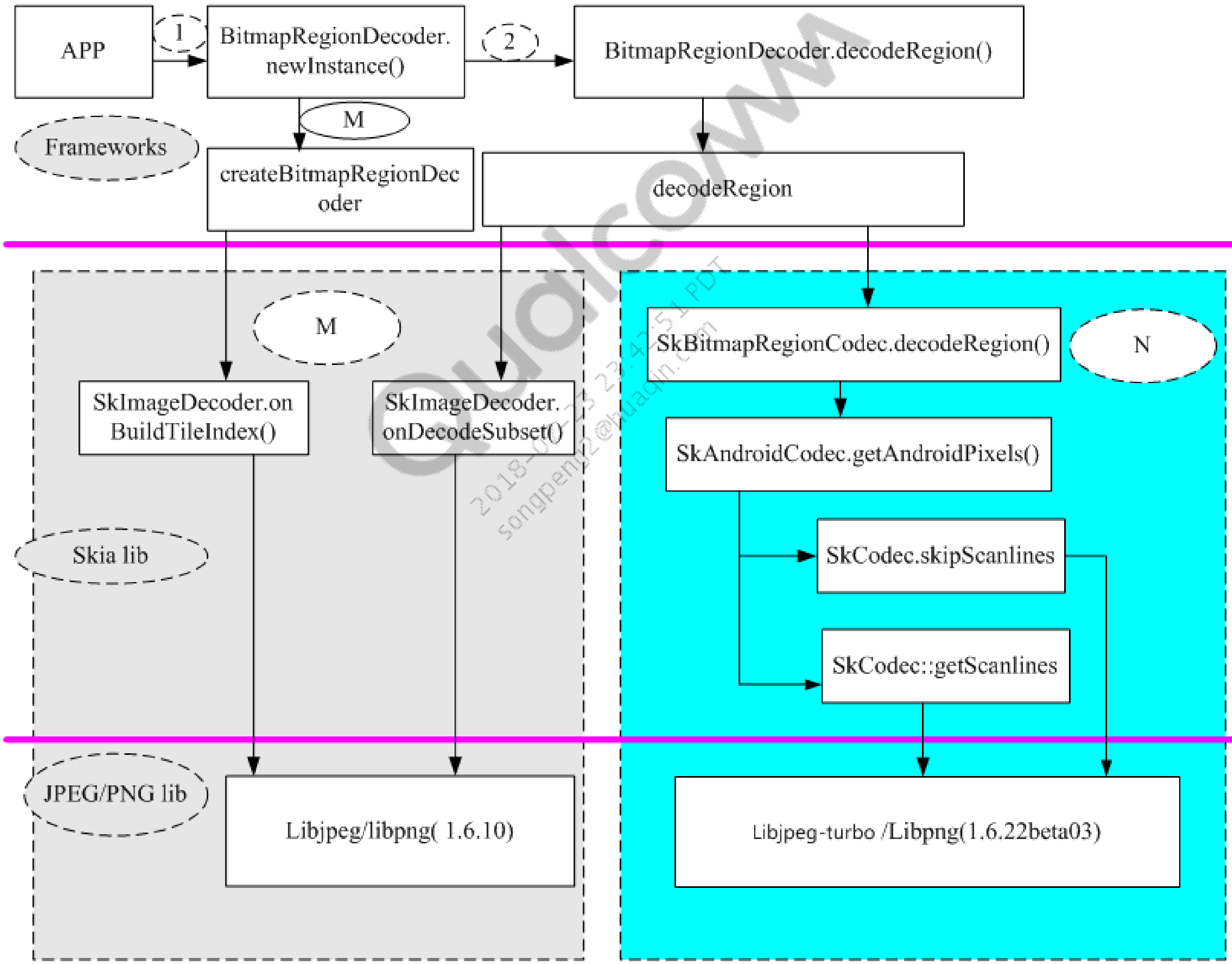
Issue Description

Issue Description (cont.)

- Many applications use the decodeRegion() to perform picture tile decode. It is very simple and convenient. It can be used as follows:
    - Create the BitmapRegionDecoder object by calling frameworks API
    - Call the decodeRegion() of BitmapRegionDecoder
    - Its performance and user experience is good in android M. While it has a performance regression for picture region decoding since Android N, especially for big pictures, the tile of picture in the bottom-right corner is decoded. This causes poor user experience.
- The main reason is that Google has made many changes to the decodeRegion() that includes frameworks, Skia Lib and Codec Lib since Android N.

Issue Analysis

Issue Analysis


APP
1
BitmapRegionDecoder.newInstance()
2
BitmapRegionDecoder.decodeRegion()
M
Frameworks
createBitmapRegionDecoder
decodeRegion
M
SkImageDecoder.onBuildTileIndex()
SkImageDecoder.onDecodeSubset()
SkBitmapRegionCodec.decodeRegion()
N
SkAndroidCodec.getAndroidPixels()
Skia lib
SkCodec.skipScanlines
SkCodec::getScanlines
JPEG/PNG lib
Libjpeg/libpng(1.6.10)
Libjpeg-turbo /Libpng(1.6.22beta03)

Issue Analysis (cont.)

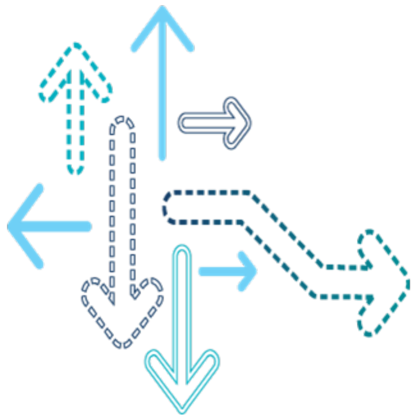
- There are three main changes as shown in the image present in Slide 7.
  - Frameworks
    - In Android M, it calls onbuildTileIndex() when the BitmapRegionDecoder() object is created. The TileIndex() can increase the performance of the decoding region. Starting from Android N release, it does not call the buildTileIndex() anymore. This function is removed from the frameworks and the codec libs does not support to get Tileindex().
  - Skia lib
    - In android M, it calls the onDecodeSubset() when it performs the decodeRegion. It is very fast as TileIndex() is present for the decode region and it can jump to the desired bitstream location through the TileIndex() to perform region decode. Starting from Android N, it always analyses the bitstream from the beginning instead of jumping to the desired bitstream as TileIndex() is not present.
    - From the functional implement perspective, starting from Android N release, it needs to call skipScanlines to skip the unnecessary content one by one from the top of the image to the decoding region and call getScanlines to get desired content and decode. It takes time to perform this as TileIndex() is not present in Android N
    - Due to this, the duration of tile decoding heavily depends on the location of the tile in the image and image size. As the tile moves towards the bottom-right corner, the duration of the tile decoding increases.
  - Codec Lib
    - In Android M, Libjpeg and Libpng (1.6.10) is used. Starting from Android N release, Libjpeg-turbo and Libpng (1.6.22beta03) are used. These codec Libs do not support the functions which can get the image TileIndex() and have better performance than Android M for image region decoding.

Solution

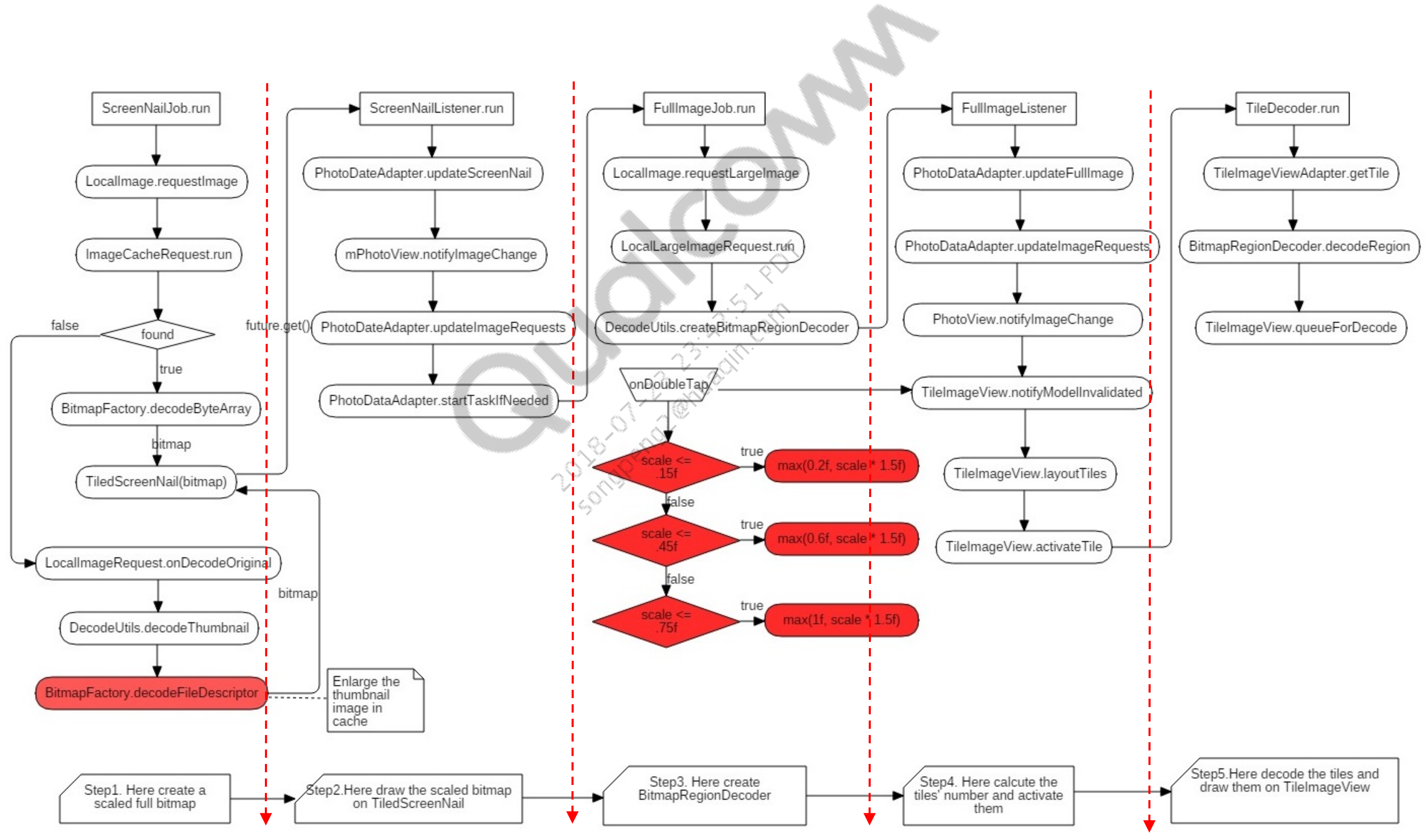
Solution

- To avoid the performance regression of decodeRegion, the following proposals are specified for the application design and implementation:
  - If full image decode is needed and the image size is not big, then decoding the full image is preferred.
  - If full image decode is needed and the image size is big, it might cause poor user experience. In such cases, the following way is adopted:
    - Downscale the full image to device's resolution size and save it as cache image when the user visits the image for the first time. Decode the cache image in subsequent visits.
    - Perform tile decoding of the full image in the background and replace the relevant region when the relevant tile decoding is finished. This process is repeated until the complete image decoding is finished.
- Tile decoding: Reducing number of regions/tiles improves speed.
- Typically, four to nine regions have good performance in Android N if the application wants to do tile decoding to save memory.

SnapdragonGallery app optimization



SnapdragonGallery app optimization



ScreenNailJob.run
LocalImage.requestImage
ImageCacheRequest.run
found
false
true
BitmapFactory.decodeByteArray
bitmap
TiledScreenNail(bitmap)
LocalImageRequest.onDecodeOriginal
DecodeUtils.decodeThumbnail
BitmapFactory.decodeFileDescriptor
bitmap
Enlarge the thumbnail image in cache
future.get()
ScreenNailListener.run
PhotoDateAdapter.updateScreenNail
mPhotoView.notifyImageChange
PhotoDateAdapter.updateImageRequests
PhotoDataAdapter.startTaskIfNeeded
FullImageJob.run
LocalImage.requestLargeImage
LocalLargeImageRequest.run
DecodeUtils.createBitmapRegionDecoder
onDoubleTap
scale <= .15f
true
max(0.2f, scale * 1.5f)
false
scale <= .45f
true
max(0.6f, scale * 1.5f)
false
scale <= .75f
true
max(1f, scale * 1.5f)
FullImageListener
PhotoDataAdapter.updateFullImage
PhotoDataAdapter.updateImageRequests
PhotoView.notifyImageChange
TileImageView.notifyModelInvalidated
TileImageView.layoutTiles
TileImageView.activateTile
TileDecoder.run
TileImageViewAdapter.getTile
BitmapRegionDecoder.decodeRegion
TileImageView.queueForDecode
Step1. Here create a scaled full bitmap
Step2.Here draw the scaled bitmap on TiledScreenNail
Step3. Here create BitmapRegionDecoder
Step4. Here calcute the tiles' number and activate them
Step5.Here decode the tiles and draw them on TileImageView

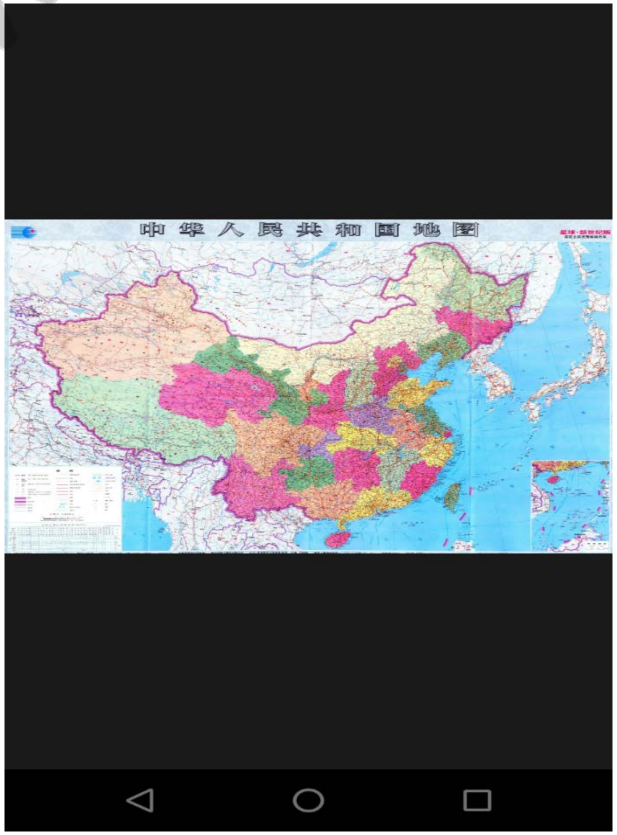
SnapdragonGallery app optimization flow diagram

SnapdragonGallery app optimization (cont.)

- SnapdragonGallery is optimized. Red color shows the main change in SnapdragonGallery application optimization flow diagram.
- Enlarge the tile size and reduce the number of regions/tiles of big image.
  - From 512 to 2048 for HighResolution(DisplayMetrics.heightPixels > 2048 || DisplayMetrics.widthPixels > 2048)
  - From 256 to 1024 for non-HighResolution device
- Select the variety of the scale according to the image size and device resolution size after double tap
  - scale <= .15f, the image will be downscale to max(0.2f, scale * 1.5f) to decode
  - scale <= .45f, the image will be downscale to max(0.6f, scale * 1.5f) to decode
  - scale <= .75f, the image will be downscale to max(1.0f, scale * 1.5f) to decode
  - Other scale, the image will be upscale to 1.5 to decode
- Increase the thumbnail target size to avoid fuzzy image after thumbnail is enlarged after double tap.

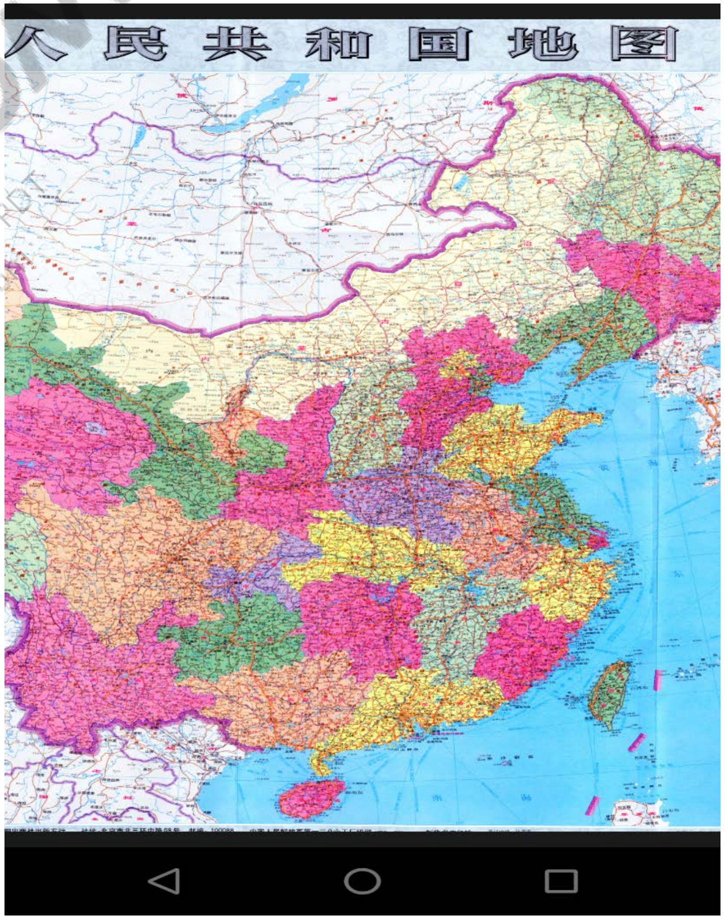
SnapdragonGallery app optimization (cont.)

- For example, consider the image of China placed here – (11935*8554: pixel size of the image) to show the flow and optimization.
  - Select the image to show in SnapdragonGallery
  - The application gets thumbnail by BitmapFactory.decodeByteArray from the application BlobCache if the thumbnail exists.
  - The application decodes the thumbnail by using BitmapFactory.decodeFileDescriptor if the thumbnail does not exist.
  - The application caches the thumbnail after decoding it
  - The thumbmail size is set to the maximum device resolution (width, height)
  - Meanwhile another thread is decoding the tiles.



SnapdragonGallery app optimization (cont.)

- After double tapping the image, the thumbnail is enlarged to show the following:
  - If the thumbnail is big enough then user does not see the fuzzy image.
  - The tiles decoded thread begins to decode the tile by BitmapRegionDecoder.decodeRegion
  - The decoded tiles showed one by one to replace the thumbnail when the tile decoder finished to decode.



SnapdragonGallery App Code Change

SnapdragonGallery app code change

```
--- a/src/com/android/gallery3d/data/MediaItem.java
+++ b/src/com/android/gallery3d/data/MediaItem.java
@@ -130,7 +130,7 @@ public abstract class MediaItem extends
MediaObject {
  }

     public static void setThumbnailSizes(int size, int microSize) {
-        sThumbnailTargetSize = size;
+        sThumbnailTargetSize = Math.max(size, sThumbnailTargetSize);
         if (sMicrothumbnailTargetSize != microSize) {
             sMicrothumbnailTargetSize = microSize;
         }
```

SnapdragonGallery app code change (cont.)

```
--- a/src/com/android/gallery3d/glrenderer/TiledTexture.java
+++ b/src/com/android/gallery3d/glrenderer/TiledTexture.java
@@ -36,7 +36,7 @@ import java.util.ArrayList;
 // upload the whole bitmap but we reduce the time of uploading each
tile
 // so it make the animation more smooth and prevents jank.
 public class TiledTexture implements Texture {
-    private static final int CONTENT_SIZE = 254;
+    private static final int CONTENT_SIZE = 510;
     private static final int BORDER_SIZE = 1;
     private static final int TILE_SIZE = CONTENT_SIZE + 2 *
BORDER_SIZE;
     private static final int INIT_CAPACITY = 8;
```

SnapdragonGallery app code change (cont.)

```
--- a/src/com/android/gallery3d/ui/PhotoView.java
+++ b/src/com/android/gallery3d/ui/PhotoView.java
@@ -18,7 +18,6 @@ package com.android.gallery3d.ui;

 import android.content.Context;
 import android.content.res.Configuration;
-//import android.drm.DrmHelper;
 import android.graphics.Color;
 import android.graphics.Matrix;
 import android.graphics.Rect;
@@ -1054,8 +1053,16 @@ public class PhotoView extends GLView {
             // onDoubleTap happened on the second ACTION_DOWN.
             // We need to ignore the next UP event.
             mIgnoreUpEvent = true;
-            if (scale <= .75f || controller.isAtMinimalScale()) {
-                controller.zoomIn(x, y, Math.max(1.0f, scale * 1.5f));
+            Log.d(TAG, "onDoubleTap scale=" + scale);
+            if (scale <= .15f) {
+                //mPictures.get(0).setScreenNail(mBigSNail);
+                controller.zoomIn(x, y, Math.max(0.2f, scale * 1.5f));
+            } else if (scale <= .45f) {
+                controller.zoomIn(x, y, Math.max(0.6f, scale * 1.5f));
+            } else if (scale <= .75f) {
+                controller.zoomIn(x, y, Math.max(1f, scale * 1.5f));
+            } else if (controller.isAtMinimalScale()) {
+                controller.zoomIn(x, y, scale * 1.5f);
             } else {
                 controller.resetToFullView();
             }
```

SnapdragonGallery app code change (cont.)

```
--- a/src/com/android/gallery3d/ui/TileImageView.java
+++ b/src/com/android/gallery3d/ui/TileImageView.java
@@ -154,9 +154,10 @@ public class TileImageView extends GLView {
         mTileDecoder = mThreadPool.submit(new TileDecoder());
         if (sTileSize == 0) {
             if (isHighResolution(context.getAndroidContext())) {
-                sTileSize = 512 ;
+                // Need to tuning
+                sTileSize = 2048;
             } else {
-                sTileSize = 256;
+                sTileSize = 1024;
             }
         }
     }
@@ -184,7 +185,6 @@ public class TileImageView extends GLView {
             mLevelCount = mModel.getLevelCount();
         }
         layoutTiles(mCenterX, mCenterY, mScale, mRotation);
-        invalidate();
     }

     @Override
@@ -358,7 +358,6 @@ public class TileImageView extends GLView {
         mScale = scale;
         mRotation = rotation;
         layoutTiles(centerX, centerY, scale, rotation);
-        invalidate();
         return true;
     }
```

SnapdragonGallery app code change (cont.)

```
--- a/src/com/android/gallery3d/ui/TiledScreenNail.java
+++ b/src/com/android/gallery3d/ui/TiledScreenNail.java
@@ -213,6 +213,6 @@ public class TiledScreenNail implements ScreenNail
{
     }

     public static void setMaxSide(int size) {
-        sMaxSide = size;
+        sMaxSide = Math.max(size, sMaxSide);
     }
 }
```

SnapdragonGallery app code change (cont.)

```
--- a/src/com/android/gallery3d/util/GalleryUtils.java
+++ b/src/com/android/gallery3d/util/GalleryUtils.java
@@ -93,10 +93,9 @@ public class GalleryUtils {

     private static void initializeThumbnailSizes(DisplayMetrics metrics,
Resources r) {
         int maxPixels = Math.max(metrics.heightPixels, metrics.widthPixels);
-
         // For screen-nails, we never need to completely fill the screen
-        MediaItem.setThumbnailSizes(maxPixels / 2, maxPixels / 5);
-        TiledScreenNail.setMaxSide(maxPixels / 2);
+        MediaItem.setThumbnailSizes(maxPixels, maxPixels / 5);
+        TiledScreenNail.setMaxSide(maxPixels);
     }
```

SnapdragonGallery app optimization

Test Result

SnapdragonGallery app optimization Test Result

- Test Method:
  - Launch app and open the image once
  - Force stop app and clear data
  - Launch app, open the image
- For JPG
  - Wait a moment (~5s)
  - Double click to zoom in the image
  - Measure the latency from touch up to image clearly displayed
- For PNG
  - Wait a moment (~15s)
  - Double click at the beginning of the image to zoom in the pic
  - Taking high FPS video.
  - Measure the latency from scroll done (no moving) to image clearly displayed in the video.
  - Repeat "Force stop app and clear data" and "Launch app, open the image" get avg data of 3 times.
- Memory usage test
  - Launch app and test the app PSS
  - Open image and test the application's proportional set size (PSS).
  - Double click to zoom in the image and test the PSS
  - Test 10 images one after another.

SnapdragonGallery app optimization Test Result (cont.)

❑ Test Result

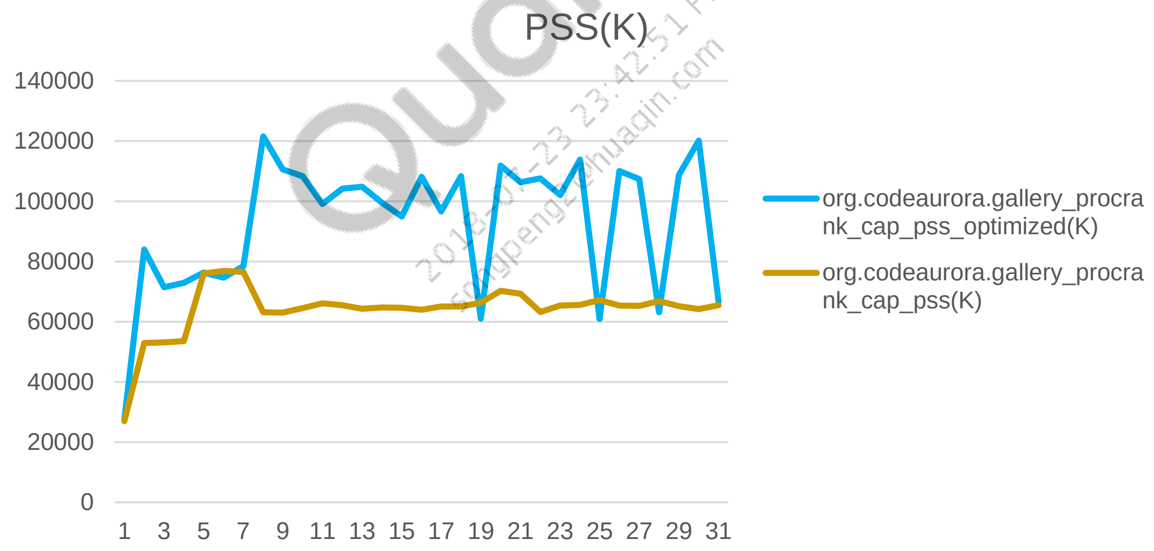
- Response time

  It improves much for the latency with the optimization version.

Image	SnapdragonGallery	SnapdragonGallery Optimized
	Time(s)	Time(s)
JPG(6240x8320)	4.56	~0.1s(Not Observed)
PNG(1080x6558)	5.73	0.36

SnapdragonGallery app optimization Test Result (cont.)

- Test Result
  - Memory usage PSS(KByte)
    - Memory usage is higher in optimization version and it is excepted that it has bigger region decoder and cached thumbnail.



Questions?

https://createpoint.qti.qualcomm.com