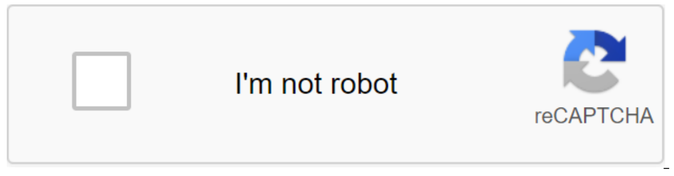

Continue

Android blur background programmatically

My team at GG Network added a very good effect to Open FM - an online radio player that we rebuilt from the ground in the second half of 2014. We used a blurry version of the content image as the app's background. I want to share with you the steps we have taken to implement this. At first it seemed quite a challenge. Android doesn't use blur anywhere in the user interface system (at least not with Froyo), nor make any of the AOSP or Google apps, so it seemed there would be no example of code to rely on. In the end it wasn't that hard - there can be a blur () function in any of the framework classes, but what's available is almost as simple. Blur blur this image seems to be the most important part of the job. However, there are already quite a few writeups on how to do this on Android. I gloss over this part and instead describe the whole process to use this method in practice to achieve the full effect. If you're only interested in blur, then you can find out all about it by following these links: I warn you that we use Picasso to upload images and caching in our apps. You'll see that this makes it easier to implement this effect. If you don't use Picasso, I would advise you to start using it. Otherwise, you'll have to figure out how to adapt our code, but I'm sure you'll be able to. Step 1: Change the background of the app with Picasso. We started by tweaking the app's background with Picasso. In our BaseActivity code, we've added a method similar to something like this: 1 2 3 4 5 private invalid updateWindowBackground () Picasso.with (it).load (url) .error (R.drawable.background_default). To do this, we needed to implement the target interface in our BaseActivity class: 1 2 3 4 5 6 8 9 9 10 11 12 14 15 @Override public void on TheBitmapFailed (Drawable drawable) - getWindow ().setWindback @Override public void onBitmapLoaded (Bitmap bitmap, LoadedFrom from) - getWindow (.setBackgroundDrawable (new BitMapDrawable (getResources(, bitmap)); - @Override public void onPrepareLoad (Drawable placeHolderDrawable) - / Just prepare mentally - Step 2: Connect to Picasso's custom transformation. But it's a powerful tool for custom effects too. This allows you to apply conversions when uploading images, so we can do something like this: 1 2 Picasso.with (it).load (url).transform (new BlurTransformation ()R.drawable.background_default). (Actually, it's a simplified example. Picasso doesn't come with a BlurTransformation class out of the box, so we needed It. Transformation is an interface with two methods that you should So we started with something in that direction: 1 2 3 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 public class BlurTransformation implements Conversion - private context mContext; private float mRadius; public blurTransformation (Context context, float radius) - mContext - context; mRadius - radius; - @Override the public conversion bitmap (Bitmap bitmap) - return zero; TODO: Actually make some transformations - @Override public string key () - return blur. It's pretty simple. In the conversion () method you get passed Bitmap and you have to return Bitmap. There are two important things to remember: If you return the newly created Bitmap (as opposed to the mutated Bitmap input) from the conversion () method, then you should call recycling () at the Bitmap input. You'll see an example of this in the next phase. In the key method, you should consider all the variables that affect the appearance of the Bitmap received. In this example, we added a blur radius to the key. Thus, if you blur the same Bitmap source with different radii, Picasso can cache both of them and correctly return each one when needed. Step 3: Spot that Bitmahp already! If you look at the writeups and gists I connected at the beginning, then you'll find a code that is pretty much willing to just drop into the conversion () method, and that's what we just did: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 @Override public transformation Bitmap (Bitmap bitmap) - Bitmap blurredBitmap - Bitmap.createBitmap (bitmap.getWidth(),), bitmap.getHeight(, Bitmap.Config.ARGB_8888); Initiate RenderScript and the script to be used by RenderScript renderScript and RenderScript (mContext); ScriptIntrinsicBlur - ScriptIntrinsicBlur.create (renderScript, Element.U8_4 (renderScript)); Highlighting memory for Renderscript to work with distribution input - Allocation.createFromBitmap (renderScript, bitmap); Distribution - Distribution .createFromBitmap (renderScript, blurredBitmap); script.setInput (entry); script.setRadius (mRadius); script.forEach output.copyTo (blurredBitmap); renderScript.destroy(); bitmap.recycle(); Return is blurred It looks pretty complete. But there are still some details and problems to iron out. The first is that ScriptIntrinsicBlur takes a maximum radius of 25. Try to pass him a larger radius and he will clearly crash. However, if you decide that you need more blurring there are some workarounds. We could think of two: You could run ScriptIntrinsicBlur several times. You can blur the abbreviated version of the bit card and then stretch it again to its original size. It also stretches the final blur radius, but you will obviously have to watch out for the loss of quality. But let's move on to some more Issues. Step 4: Make it run a preview of API 17 Above the code only works starting with API 17 and that's what be too high for most applications. Renderscript was previously available, but ScriptIntrinsicBlur was only added to API 17. Fortunately, there is lib support for this. Here's an official blog about this Wait, what? Eclipse? What about Android Studio and Gradle? No problem, fortunately, there is dSL for that! Just add this to your build.gradle: 1 2 3 4 5 defaultConfig... renderscriptTargetApi 21 renderscriptSupportModeEnable true - Now you can go back to API 8 if you like (or need) to! (If you haven't read the related blog, for which I don't blame you because it's about Ehclipse, a little hint: you should also change the import.) Step 5: Fighting artifacts that looks like we did (we aaaalmost are). It looks great on a test device, at least on a few test images. But when we launched it on a wide variety of images or on specific devices (like the Samsung Galaxy S3), we started to see some nasty artifacts. It took us precious time to find a solution to this problem, so I'm just going to save you this time and point you right to Roman Nurik's G e post, which is a great sticking point. Thanks Roman! So, let's just trim some pixels if our width isn't multiples of four: 1 2 3 4 5 int bitmapWidth and bitmap.getWidth (); if (bitmapWidth % 4 ! ' 0) - bitmap - BitmapUtils.resize (bit map, bitWimapdth - (bitmapWidth % 4), bitmap.getHeight (), ScaleType.CENTER); For the actual size I use here is our internal BitmapUtils class, which does the hard work (including recycling the original bit card as it returns a cropped copy). To make it simple, I won't go into exactly how we do it. You probably already have a similar util in the code. If not there are probably a lot of great resources on how to do this (AOSP code for the Gallery app can be one place to start). Step 6: Release prod, see it crash OK, now we really did. At least that's what we thought. We couldn't find any more problems with this code in our internal testing. So we sent him. And prod back at us with accidents, as it always does. Fortunately, we use Crashlytics and this has helped us to quickly find problems. On some devices (judging by their names they were quite exotic) the Bitmap entrance was Config.RGB_565. If you look back at the code from Step 3, you'll see that we've created a weekend bit card with Config.ARGB_8888. So when Renderscript tried to copy the blurry pixels to the Bitmap outlet it crashed because of this incompatibility. So the obvious solution would be to create a Bitmap outlet with the same config as the entry one. However, if you tried it (and if you managed to find a device that produces these RGB_565 bitmaps), or if you just researched you'll find that renderscript doesn't play well with this config. If you're not in the results, like in this SO stream. We thought it would be a great idea for a hallucinogen modeling app, but it's not to our music player. What we've decided to do is try to convert the input into ARGB_8888and if it fails, we throw an exception inside the conversion. Picasso catches it and processes it, causing a callback error. We decided that we really enjoy displaying some static background by default than these trippy images, so that worked for us. Here's the code we use to convert: 1 2 3 4 5 6 7 8 9 10 11 12 14 15 private static Bitmap convertBitmap (Bitmap bitmap, Config config) - if (bitmap.getConfig)) - Bitmap argbBitmap; argbBitmap - bitmap.copy (config, false); bitmap.recycle (); if (argbBitmap) - zero) - throw a new Unsupported OperationIndence, failed to convert a bit card from a confiscated return argbBit; The question we call this before finally passing it to Renderscript: 1 Bitmap argbBitmap - convertBitmap (croppedBitmap, Config.ARGB_8888); Bonus: Changing the background smoothly is the last thing that was added later by my fabulous teammates. It didn't really have anything to do with blur, but it was a 10% awesomeness touch that made 90% effect. There's only one problem with gorgeous blurry images that cover the user's screen from edge to edge and dynamically switches while the app is used - changing one to the other in an instant just doesn't cut it. It just makes for a really jarring experience when the entire screen flashes from one background to another (and even worse on some devices that sometimes show a solid black background for a few frames). The solution is obvious, we have to smoothly animate from one background to another. TransitionDrawable comes to the rescue. There's also a problem that the box doesn't have a getter for the current drawable background, but you can get around it by reaching deeper - in DecorView. Here's our code: 1 2 3 4 5 6 7 8 private invalid changeBackground (Drawable drawable) - View decorView and getWindow (.getDecorView (); Drawable oldBackgroundDrawable - decorView.getBackground (); TransitionDrawable transitionDrawable - new TransitionDrawable (new Drawable'oldBackgroundDrawable, drawable); decorView.setBackground (transition Drawn); transitionDrawable.startTransatation (1000); Check out the full code for a running demo application. Below is the preliminary effect. And of course, the best live demo you can see on your own phone is the open FM app in the Play Store. Give it a try - download it and play some songs. The images for this preview and demo code were taken from Marie Schweitz's Lorem Ipsum Illustration. Thank you, Marie! Marie! how to blur background image in android programmatically