Android Media Basics: Images, Audio, and Video

By Lauren Darcey, Shane Conder, Carmen Delessio

Nov 19, 2013

Bitmaps and Canvas

The `Bitmap (android.graphics.Bitmap)` class represents a bitmap image. You create bitmaps via the `BitmapFactory (android.graphics.BitmapFactory)` class.

Using a `BitmapFactory`, you can create bitmaps in three common ways: from a resource, a file, or an `InputStream`. To create a bitmap from a resource, you use the `BitmapFactory` method `decodeResource()`:

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.someImage);
```

To create a bitmap from a file or `InputStream`, you use the `decodeFile()` and `decodeStream()` methods, respectively.

Handling Large Images

There are techniques for avoiding the dreaded OOM (out of memory) exception. Large images can have a significant impact on memory use in your app. To demonstrate this, in this section, you create an unrealistically large image to display in an `ImageView`. When the image is just loaded into the `ImageView`, the app fails with an out-of-memory error. A `java.lang.OutOfMemory` exception occurs. You'll learn to fix the memory error for this case.

You want to display the image at an appropriate size and resolution for the device. No point exists in showing a 10-foot mural in a 6-inch frame. Similarly, no point exists in showing a 20-inch image on a 3-inch phone screen. You can manipulate the image to display well and save memory.

The details of your app will influence your memory usage and the techniques that will work best in your case. This example shows how to handle a single large image.

As a demonstration, start with an image and increase it to an unrealistic size. This example uses a photo that is 72 inches × 54 inches and that has a 28Mb file size.

The image is in the `drawable` resource folder and has the id `R.drawable.largeimage`.

You can cause the app to fail with an out-of-memory error by trying to set an `ImageView` to this resource. You have an `ImageView` named `largeImage`. This line of code causes the app to fail:

```
largeImage.setImageResource(R.drawable.largeimage);
```

Some work is required to fix this, but handling an image this large is possible. In all cases, working with appropriately sized images would be ideal, but that does not always happen.

The goal is to get the dimensions of the underlying bitmap without actually rendering it. Getting those dimensions is not a memory-intensive activity. After you have the bitmap, you can determine an appropriate size for the bitmap that will fit in your display. If you have a 20-inch image and a 4-inch display, you request that the bitmap that is created in memory be created at a size and resolution that is appropriate for the 4-inch display.

Using BitmapFactory.Options

You use the `BitmapFactory.Options` class with the `BitmapFactory` class; it is essential to how you handle large `Bitmaps`.

You use the following options from the BitmapFactoryOptions class:

- inJustDecodeBounds: If set to true, this option indicates that the bitmap dimensions should be determined by the BitmapFactory, but that the bitmap itself should not be created. This is the key to getting the bitmap dimensions without the memory overhead of creating the bitmap.
- outWidth: The width of the image set when you use inJustDecodeBounds.
- outHeight: The height of the image set when you use inJustDecodeBounds.
- inSampleSize: This integer indicates how much the dimensions of the bitmap should be reduced. Given an image of 1000×400, an inSampleSize of 4 will result in a bitmap of 250×100. The dimensions are reduced by a factor of 4.

Listing 21.3 shows the code to address handling large images. We'll step through the approach and the code. The code is in the Hour21LargeImage project in MainActivity.java.

Listing 21.3 Displaying a Large Image

```
 1:  ImageView largeImage = (ImageView) findViewById(R.id.imageView1);
 2:  Display display = getWindowManager().getDefaultDisplay();
 3:  int displayWidth = display.getWidth();
 4:  BitmapFactory.Options options = new BitmapFactory.Options();
 5:  options.inJustDecodeBounds = true;
 6:  BitmapFactory.decodeResource(getResources(), R.drawable.largeimage, options);
 7:  int width = options.outWidth;
 8:  if (width > displayWidth) {
 9:    int widthRatio = Math.round((float) width / (float) displayWidth);
10:    options.inSampleSize = widthRatio;
11:  }
12:  options.inJustDecodeBounds = false;
13:  Bitmap scaledBitmap =  BitmapFactory.decodeResource(getResources(),
14:  R.drawable.largeimage, options);
15:  largeImage.setImageBitmap(scaledBitmap);
```

Lines 2 and 3 get the size of the device display. You use this as the target size for reducing the image size.

In lines 4–7, you determine the size of the current bitmap. You do that by creating a BitmapFactory.Options class and setting the inJustDecodeBounds value to true. On line 6, the bitmap is decoded to get the dimensions. This is where you get the dimensions without the memory overhead of creating the bitmap. The result is available in options.outWidth. On line 7, you assign options.outWidth to the int variable width.

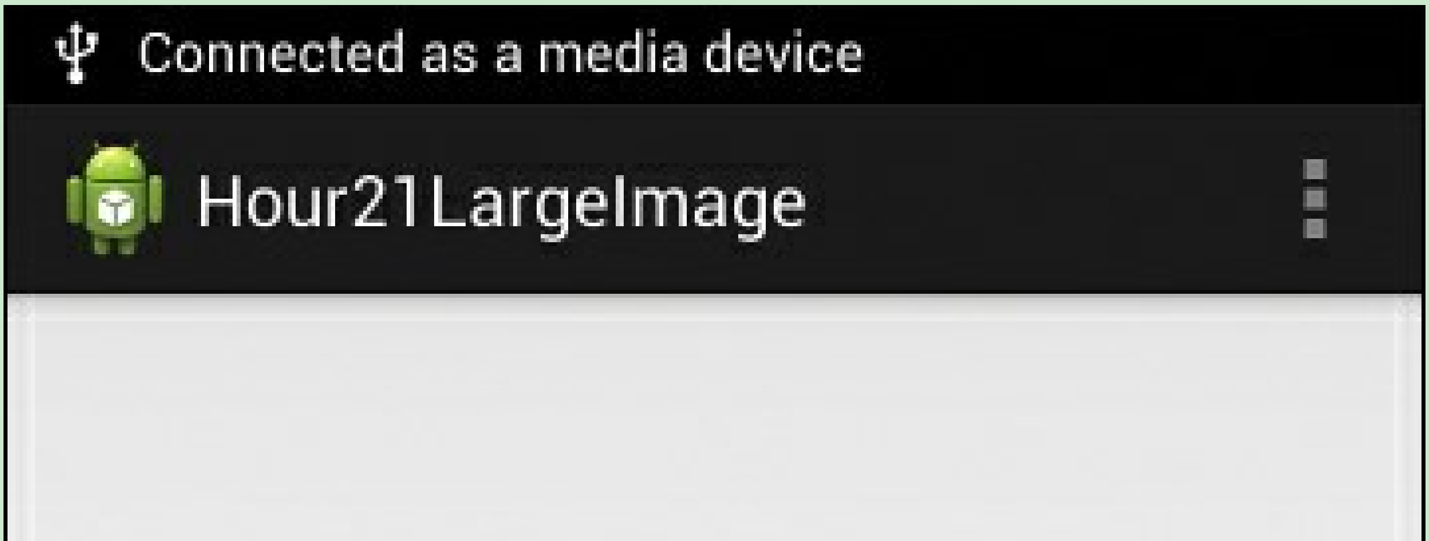
This example uses a simple test for the size of the image. Line 8 checks whether the width of the bitmap is greater than the size of the display. If that is the case, you determine the inSampleSize to use. You do that on lines 9 and 10. If the width of the bitmap is 1000 pixels and the size of the display is 250 pixels, you get an inSampleSize of 4 by dividing the width of the bitmap by the width of the display. For simplicity, this example does not check the height, which could theoretically leave you exposed to a bitmap that was 20 inches tall and 2 inches wide.

With the imSampleSize set to an appropriate value, you can render the image.

On line 12, the inJustDecodeBounds value is set to false. You want to decode the image and create the Bitmap object.

Lines 13 and 14 use the BitmapFactory.decodeResource() method to create a Bitmap and assign it to the variable scaledBitmap. Note that in this call, the BitmapFactory.Options variable options is passed as a parameter. That is how you indicate to the BitmapFactory what inSampleSize to use.

Displaying a 72-inch image on a phone is certainly not recommended, but Figure 21.5 shows that it can be done.



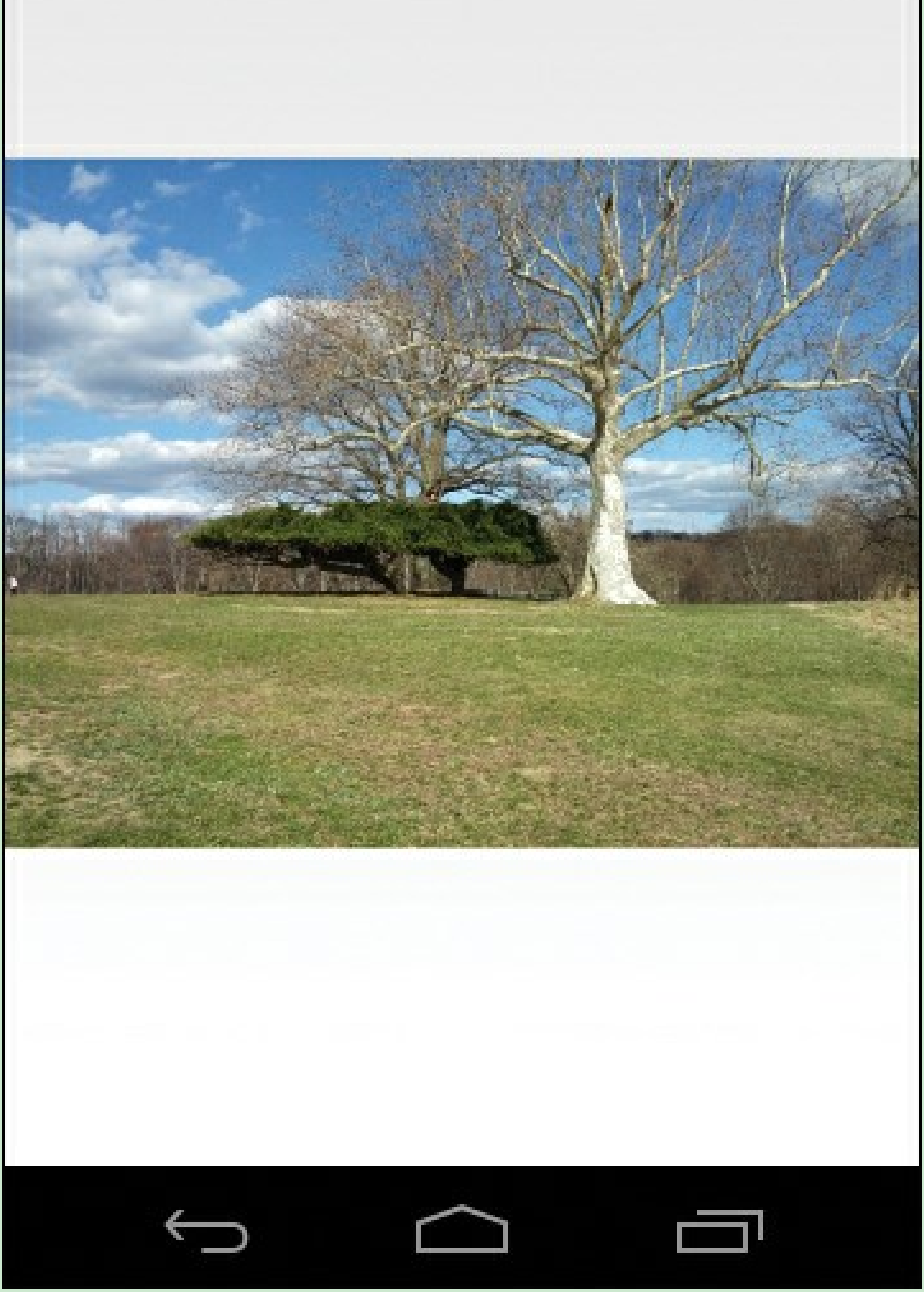


Figure 21.5 Very large photo displayed on device

Drawing Directly on a Canvas

You can do one more thing with an `ImageView` and bitmap. You can create a bitmap and draw directly on the `Canvas(android.graphics.Canvas)` that is associated with the bitmap. A `Canvas` is an object that you can draw on by calling drawing commands.

To see this, add another `Button` to the MainActivity activity from the Hour21ImageView project. Listing 21.4 shows the `onClickListener()` method for the draw button. The code creates a bitmap, gets the canvas, and draws the word “`Hello`” on the canvas. The resulting bitmap appears in the `ImageView`.

You create a new `Bitmap` on lines 3 and 4 by using the method `Bitmap.createBitmap()`. You set the width and height of the bitmap to that of the `ImageView` and use the `Bitmap.Config (android.graphics.Bitmap.Config) set` to `Bitmap.Config.ARGB_8888`.

The documentation for the `Bitmap` class offers a number of `createBitmap()` methods that take different parameters. These methods may return a mutable or an immutable `Bitmap`, but only a mutable `Bitmap` can be used for drawing.

On line 5, a `Canvas` is instantiated based on the bitmap that you created.

You apply simple drawing commands to the `Canvas` on lines 6–10. You create a `Paint` object, set the `Color` to blue, and set the text size. Line 6 gets the density of the display, which is used to get the text size you want. Hour 4 covered converting density independent pixels to pixels. Line 10 draws the word “Hello” in the center of the `ImageView`.

Line 11 updates the `ImageView` to show the generated bitmap.

Listing 21.4 Drawing on a Canvas

```
 1: draw.setOnClickListener(new OnClickListener() {
 2:   public void onClick(View v) {
 3:     Bitmap imageBitmap = Bitmap.createBitmap(imageView.getWidth(),
 4:     imageView.getHeight(), Bitmap.Config.ARGB_8888);
 5:     Canvas canvas = new Canvas(imageBitmap);
 6:     float scale = getResources().getDisplayMetrics().density;
 7:     Paint p = new Paint();
 8:     p.setColor(Color.BLUE);
 9:     p.setTextSize(24*scale);
10:     canvas.drawText("Hello", imageView.getWidth()/2,imageView.getHeight()/2,
p);
11:     imageView.setImageBitmap(imageBitmap);
12:   }
13: });
```

+ Share This   Save To Your Account

< Back **Page 2** of 9 Next >