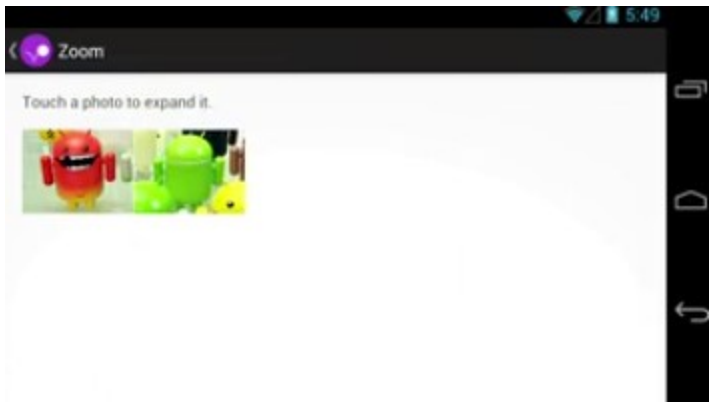
使用缩放动画放大视图

本课演示了如何执行轻触缩放动画，通过动画将视图从缩略图过渡到填满整个屏幕的完整尺寸图片，这对于照片图库等应用来说十分有用。

下方演示了轻触缩放动画如何展开图片缩略图以填满整个屏幕：



缩放动画

如果您想要抢先查看完整的可运行示例，请参阅 GitHub 上的 Wear 扬声器示例 (https://github.com/android/wear-os-samples/tree/master/WearSpeakerSample)中的 **`UIAnimation`** 类。

创建视图

创建一个布局文件，使其包含要缩放的内容的小版本和大版本。以下示例为可点击的图片缩略图创建了一个 **`ImageButton`** (/reference/android/widget/ImageButton)，并且创建了一个显示图片放大视图的 **`ImageView`** (/reference/android/widget/ImageView)：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

        <ImageButton
            android:id="@+id/thumb_button_1"
            android:layout_width="100dp"
            android:layout_height="75dp"
            android:layout_marginRight="1dp"
            android:src="@drawable/thumb1"
            android:scaleType="centerCrop"
            android:contentDescription="@string/description_image_1" />

    </LinearLayout>

    <!-- This initially-hidden ImageView will hold the expanded/zoomed version of
         the images above. Without transformations applied, it takes up the entire
         screen. To achieve the "zoom" animation, this view's bounds are animated
         from the bounds of the thumbnail button above, to its final laid-out
         bounds.
         -->

    <ImageView
        android:id="@+id/expanded_image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="invisible"
        android:contentDescription="@string/description_zoom_touch_close" />

</FrameLayout>
```

设置缩放动画

应用布局后，设置触发缩放动画的事件处理脚本。 以下示例向 **ImageButton** (/reference/android/widget/ImageButton) 添加 **View.OnClickListener** (/reference/android/view/View.OnClickListener)，以在用户点按图片按钮时执行缩放动画：

KOTLIN (#KOTLIN)**JAVA**

```
public class ZoomActivity extends FragmentActivity {
    // Hold a reference to the current animator,
    // so that it can be canceled mid-way.
    private Animator currentAnimator;

    // The system "short" animation time duration, in milliseconds. This
    // duration is ideal for subtle animations or animations that occur
    // very frequently.
    private int shortAnimationDuration;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_zoom);

        // Hook up clicks on the thumbnail views.

        final View thumb1View = findViewById(R.id.thumb_button_1);
        thumb1View.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                zoomImageFromThumb(thumb1View, R.drawable.image1);
            }
        });

        // Retrieve and cache the system's default "short" animation time.
        shortAnimationDuration = getResources().getInteger(
                android.R.integer.config_shortAnimTime);
    }
    ...
}
```

缩放视图

现在，您需要根据情况通过动画从正常大小视图过渡到缩放视图。通常，您需要在正常大小视图的边界与放大视图的边界之间添加动画。以下方法介绍了如何通过执行以下操作，实现一个从图片缩略图缩放到放大视图的缩放动画：

1. 将高分辨率图片分配给隐藏的“放大”**ImageView** (/reference/android/widget/ImageView)。为简单起见，以下示例将大图片资源加载到界面线程。您需要在单独的线程中进行加载以防止界面线程上出现阻塞，然后在界面线程上设置位图。理想情况下，位图不应大于屏幕尺寸。
2. 计算 **ImageView** (/reference/android/widget/ImageView) 的起始边界和结束边界。
3. 从起始边界到结束边界，同时为四个定位和大小调整属性 **X** (/reference/android/view/View#X)、**Y** (/reference/android/view/View#Y)、**SCALE_X** (/reference/android/view/View#SCALE_X) 和 **SCALE_Y** (/reference/android/view/View#SCALE_Y) 添加动画。这四个动画会添加到 **AnimatorSet** (/reference/android/animation/AnimatorSet) 中，以便可以同时启动。
4. 当用户在图片放大的情况下轻触屏幕时，通过运行类似但是相反的动画重新缩小视图。要完成此操作，您可以通过向 **ImageView** (/reference/android/widget/ImageView) 添加 **View.OnClickListener** (/reference/android/view/View.OnClickListener)。点按后，**ImageView** (/reference/android/widget/ImageView) 会重新最小化为图片缩略图的大小，并将其可见性设置为 **GONE** (/reference/android/view/View#GONE) 以将其隐藏。

KOTLIN (#KOTLIN)**JAVA**

```
private void zoomImageFromThumb(final View thumbView, int imageResId) {
    // If there's an animation in progress, cancel it
    // immediately and proceed with this one.
    if (currentAnimator != null) {
        currentAnimator.cancel();
    }

    // Load the high-resolution "zoomed-in" image.
    final ImageView expandedImageView = (ImageView) findViewById(
            R.id.expanded_image);
    expandedImageView.setImageResource(imageResId);

    // Calculate the starting and ending bounds for the zoomed-in image.
    // This step involves lots of math. Yay, math.
    final Rect startBounds = new Rect();
    final Rect finalBounds = new Rect();
    final Point globalOffset = new Point();

    // The start bounds are the global visible rectangle of the thumbnail,
    // and the final bounds are the global visible rectangle of the container
    // view. Also set the container view's offset as the origin for the
    // bounds, since that's the origin for the positioning animation
    // properties (X, Y).
    thumbView.getGlobalVisibleRect(startBounds);
    findViewById(R.id.container)
            .getGlobalVisibleRect(finalBounds, globalOffset);
    startBounds.offset(-globalOffset.x, -globalOffset.y);
    finalBounds.offset(-globalOffset.x, -globalOffset.y);

    // Adjust the start bounds to be the same aspect ratio as the final
    // bounds using the "center crop" technique. This prevents undesirable
    // stretching during the animation. Also calculate the start scaling
    // factor (the end scaling factor is always 1.0).
    float startScale;
    if ((float) finalBounds.width() / finalBounds.height()
            > (float) startBounds.width() / startBounds.height()) {
        // Extend start bounds horizontally
        startScale = (float) startBounds.height() / finalBounds.height();
        float startWidth = startScale * finalBounds.width();
        float deltaWidth = (startWidth - startBounds.width()) / 2;
        startBounds.left -= deltaWidth;
        startBounds.right += deltaWidth;
    } else {
        // Extend start bounds vertically
        startScale = (float) startBounds.width() / finalBounds.width();
        float startHeight = startScale * finalBounds.height();
        float deltaHeight = (startHeight - startBounds.height()) / 2;
        startBounds.top -= deltaHeight;
        startBounds.bottom += deltaHeight;
    }

    // Hide the thumbnail and show the zoomed-in view. When the animation
    // begins, it will position the zoomed-in view in the place of the
    // thumbnail.
    thumbView.setAlpha(0f);
    expandedImageView.setVisibility(View.VISIBLE);

    // Set the pivot point for SCALE_X and SCALE_Y transformations
    // to the top-left corner of the zoomed-in view (the default
    // is the center of the view).
    expandedImageView.setPivotX(0f);
    expandedImageView.setPivotY(0f);

    // Construct and run the parallel animation of the four translation and
    // scale properties (X, Y, SCALE_X, and SCALE_Y).
    AnimatorSet set = new AnimatorSet();
    set
            .play(ObjectAnimator.ofFloat(expandedImageView, View.X,
                    startBounds.left, finalBounds.left))
            .with(ObjectAnimator.ofFloat(expandedImageView, View.Y,
```

```
                startBounds.top, finalBounds.top))
        .with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE_X,
                startScale, 1f))
        .with(ObjectAnimator.ofFloat(expandedImageView,
                View.SCALE_Y, startScale, 1f));
set.setDuration(shortAnimationDuration);
set.setInterpolator(new DecelerateInterpolator());
set.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        currentAnimator = null;
    }

    @Override
    public void onAnimationCancel(Animator animation) {
        currentAnimator = null;
    }
});
set.start();
currentAnimator = set;

// Upon clicking the zoomed-in image, it should zoom back down
// to the original bounds and show the thumbnail instead of
// the expanded image.
final float startScaleFinal = startScale;
expandedImageView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (currentAnimator != null) {
            currentAnimator.cancel();
        }

        // Animate the four positioning/sizing properties in parallel,
        // back to their original values.
        AnimatorSet set = new AnimatorSet();
        set.play(ObjectAnimator
                    .ofFloat(expandedImageView, View.X, startBounds.left))
                    .with(ObjectAnimator
                            .ofFloat(expandedImageView,
                                    View.Y,startBounds.top))
                    .with(ObjectAnimator
                            .ofFloat(expandedImageView,
                                    View.SCALE_X, startScaleFinal))
                    .with(ObjectAnimator
                            .ofFloat(expandedImageView,
                                    View.SCALE_Y, startScaleFinal));
        set.setDuration(shortAnimationDuration);
        set.setInterpolator(new DecelerateInterpolator());
        set.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    thumbView.setAlpha(1f);
                    expandedImageView.setVisibility(View.GONE);
                    currentAnimator = null;
                }

                @Override
                public void onAnimationCancel(Animator animation) {
                    thumbView.setAlpha(1f);
                    expandedImageView.setVisibility(View.GONE);
                    currentAnimator = null;
                }
            });
            set.start();
            currentAnimator = set;
        }
    });
}
```

Last updated 2020-06-05 UTC.