

# **Animations dans Android**

## **Introduction**

L'animation est une technique qui donne l'illusion que des objets inanimés peuvent bouger.

Elle se sert d'une série d'images qui changent avec le temps.

En parcourant rapidement ces images, mises ensemble, elles donnent l'illusion que les objets bougent avec le temps.

### **Exemple :**

Folioscope (Flip Book)

[https://youtu.be/p3q9MM\\_h-M](https://youtu.be/p3q9MM_h-M)

Comment faire un folioscope

<https://youtu.be/4IRMBRUIRB4>

La vitesse à laquelle ces images apparaissent à l'écran s'appelle la cadence. Idéalement, votre fréquence d'images correspond étroitement à la fréquence de rafraîchissement de l'écran pour éviter des artefacts visuels déplaisants.



[Si le nombre d'images par seconde au moment de la projection est supérieur à celui du tournage, on obtient un accéléré. À l'inverse, si le nombre d'images par seconde au moment de la projection est inférieur à celui du tournage, on obtient un ralenti.]

[https://fr.wikipedia.org/wiki/Images\\_par\\_seconde](https://fr.wikipedia.org/wiki/Images_par_seconde)

Le système Android gère la fréquence d'images pour vous. Il n'est donc pas nécessaire, dans la plupart des cas, de gérer la fréquence d'images de vos animations.

## Types d'animation pour Android

Android fournit les systèmes d'animation suivants:

**View animation** : elle permet d'animer des vues. Il est relativement facile à configurer et offre suffisamment de fonctionnalités pour répondre aux besoins de nombreuses applications (API 1).

**Property animation** : introduit dans Android 3.0 (API niveau 11). Il permet d'animer les propriétés de n'importe quel objet. Le système est extensible et vous permet d'animer les propriétés des types personnalisés. Il est préféré à « View animation ».

**Animation basée sur la physique** : elle se sert des lois de la physique pour manifester un haut degré de réalisme dans l'animation. Par exemple, une animation est entraînée par la force et l'animation s'immobilise lorsque la force atteint l'équilibre.

## **View animation**

C'est la manière la plus simple pour faire des animations.

Elles sont limitées aux vues.

Elles ne peuvent donc animer que les vues comme : ImageView, TextView ou des boutons.

Elles permettent d'animer que certains aspects de la vue.

La vue en elle-même n'est pas modifiée, on change uniquement la manière avec laquelle elle est dessinée.

Elles utilisent les fichiers « xml » définis dans le répertoire « res/anim ».

Il y a deux sous-catégories :

## **Tween animation (animation interpolée)**

Une animation peut-être de 4 types : alpha (disparition graduelle), scale (mise à l'échelle), translate (mouvement) et rotate (rotation).

Le fichier XML peut contenir les balises :

Le fichier doit avoir obligatoirement une balise « set ».

Cette balise représente un groupe d'animations parmi les 4 types mentionnées précédemment.

On peut imbriquer plusieurs balises « set », afin de créer plusieurs animations.

Le fichier XML peut contenir aussi une des balises, « alpha », « scale », « translate » et « rotate ».

[Examiner l'exemple « AndroidAnimations »](#)

Exemple la rotation, fichier « rotate.xml »

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <rotate android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="600"
    android:repeatMode="restart"
    android:repeatCount="infinite"
    android:interpolator="@android:anim/cycle_interpolator"/>
</set>
```

`android:fromDegrees="0"` : 0o est l'angle de début de la rotation.

`android:toDegrees="360"` : 360o est l'angle de fin de la rotation.

`android:pivotX="50%"` et `android:pivotY="50%"` : est la coordonnée du centre de rotation de l'axe X (ou Y), en pourcentage par rapport à la gauche de la vue. Le nombre 50% correspond au milieu de la vue. 100% au bord droit.

**android:duration="600"** : la durée de la rotation est fixée à 1000 millisecondes.

**android:repeatMode="restart"** : on repart du début

**android:repeatCount="infinite"** : à l'infini

**android:interpolator="@android:anim/cycle\_interpolator"** : la vitesse de l'effet varie dans le temps de manière cyclique.

```
// Rotate
btnRotate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent(MainActivity.this, RotateActivity.class);
        startActivity(i);
    }
});
```

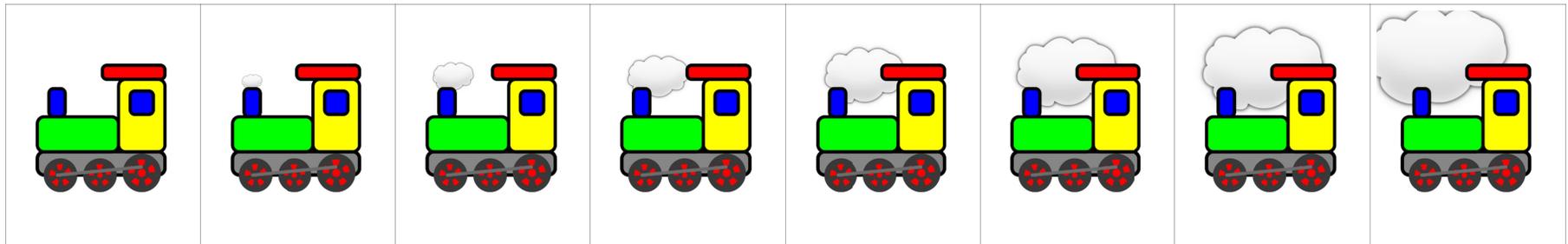
```
ImageView imgLogo;  
Button btnStart;  
  
// Animation  
Animation animRotate;  
  
imgLogo = findViewById(R.id.imgLogo);  
btnStart = findViewById(R.id.btnStart);  
  
// load the animation  
animRotate = AnimationUtils.loadAnimation(getApplicationContext(),  
    R.anim.rotate);  
  
// set animation listener  
animRotate.setAnimationListener(this);  
// button click event  
btnStart.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        imgLogo.setVisibility(View.VISIBLE);  
        // start the animation  
        imgLogo.startAnimation(animRotate);  
    }  
});
```

## Frame animation (ou Drawable animation)

Elle consiste à afficher les ressources Drawable les unes après les autres, image après image, comme un rouleau de film. Cette méthode d'animation est utile si vous souhaitez animer des éléments plus faciles à représenter avec des ressources Drawable (API 1).

[Examiner l'exemple « FrameAnimation »](#)

On commence par copier dans le répertoire « Drawable » la séquence d'images.



```
private AnimationDrawable myFrameAnimation;  
View myFrameAnimationImageView;  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    myFrameAnimationImageView = findViewById(R.id.frameAnimationImageView);  
    myFrameAnimationImageView.setBackgroundResource(R.drawable.frame_animation);  
    myFrameAnimation = (AnimationDrawable)  
        myFrameAnimationImageView.getBackground();  
}  
  
/*clicking the image starts the frame animation.  
if it is running, it first stops and then starts it*/  
    public void showAnimation(View view) {  
        myFrameAnimation.stop();  
        myFrameAnimation.start();  
    }  
  
<ImageView  
    android:id="@+id/frameAnimationImageView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:onClick="showAnimation"  
/>
```

## **Property animation (animation de propriétés)**

Les animations interpolées modifient l'apparence de vue sans modifier la vue. Les animations de propriétés modifient les propriétés de l'objet.

Elles permettent d'animer tous les objets, vues et non-vues (objets « java »).

On peut modifier n'importe quelle propriété de l'objet pour une durée spécifiée : couleur, la taille, la position. « View animation » ne permet pas de modifier par exemple la couleur de l'arrière-plan (« background ») de l'objet.

Par exemple, on peut grossir un cercle en modifiant uniquement son rayon pour une durée déterminée.

« View animation » permet de modifier uniquement l'endroit où l'objet est dessiné. Si on anime un bouton pour qu'il se déplace dans un écran, il faut implémenter votre propre logique pour trouver l'endroit sur l'écran où il faut cliquer sur l'objet. L'animation de propriétés ne pose pas ce genre de difficultés.

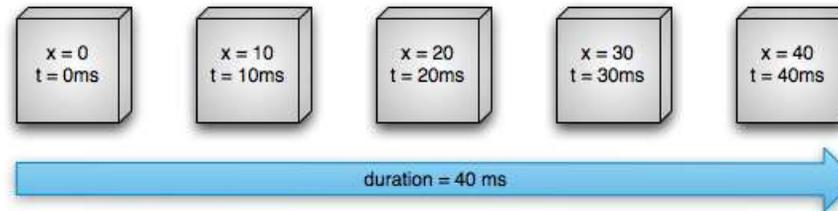
On retrouve donc globalement les mêmes techniques définies dans « View animation ». La façon de le faire est par contre différente.

De ce fait, « Property animation » est préférable au « View animation », car elle est plus récente, elle couvre l'ensemble des objets et elle permet d'offrir plus de techniques d'animation.

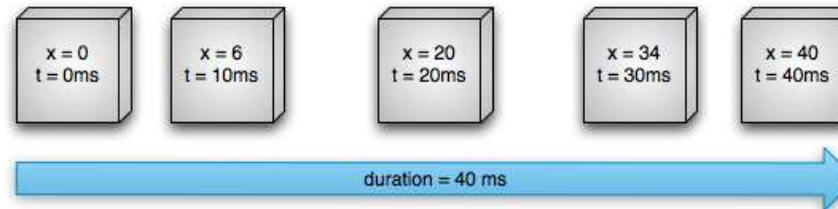
Cependant, « Property animation » n'est pas aussi simple à mettre en place que « View animation ».

## Fonctionnement des animations de propriétés

Durée : 40ms distance : 40 pixels



### Animation linéaire / même distance

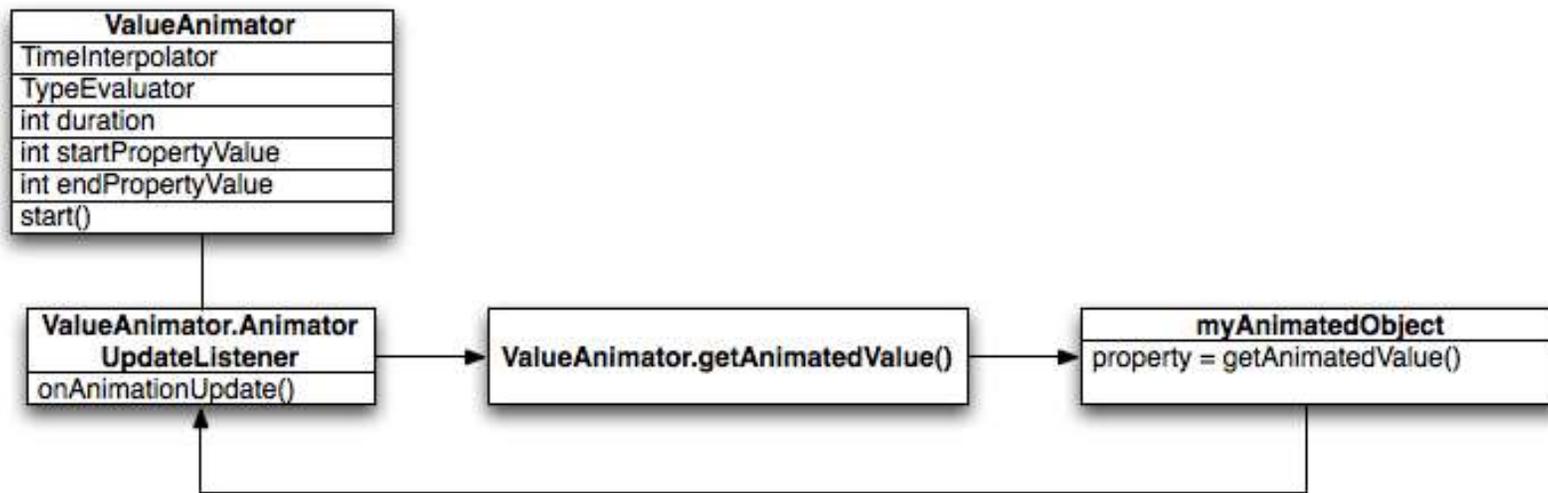


Animation non linéaire / accélération au début, décélération à la fin

## Animation des propriétés d'un objet

On a besoin des éléments suivants :

- Un « Animator » qui gère l'animation. La classe « Animator » fournit la classe de base pour la création d'animation. Cette classe fournit des fonctionnalités minimales. De ce fait, il est préférable d'utiliser à la place la sous-classe « ObjectAnimator ». Elle permet de mettre à jour les propriétés de l'objet à chaque changement détecté.
- Un « TypeEvaluator ». Il permet d'indiquer au système d'animation des propriétés comment calculer une valeur pour une propriété donnée. À partir des données fournies par la classe « Animator », le début et la fin de l'animation, il calcule les propriétés de l'objet. On peut utiliser un des fournisseurs IntEvaluator, FloatEvaluator ou ArgbEvaluator, sinon la sous-classe TypeEvaluator pour les autres types.
- Un « TimeInterpolator ». Il permet de définir la fréquence de changement d'une animation, comme l'accélération ou la décélération.



Comment, calculer l'animation

Examiner l'exemple « PropertyAnimation »

## Fichier XML de l'activité

```
<com.example.android.propertyanimation.PulseAnimationView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

Il permet de charger le fichier « PulseAnimationView.java »

On a besoin de déclarer deux constructeurs :

// si l'appel est provoqué par l'activité

```
public PulseAnimationView(Context context) {  
    this(context, null);  
}
```

// si l'appel est provoqué par le fichier XML

```
public PulseAnimationView(Context context, AttributeSet attrs) {  
    super(context, attrs);  
}
```

On déclare la méthode « `onSizeChanged` », responsable de changer la taille de l'objet.

```
public void onSizeChanged(int w, int h, int oldw, int oldh);
```

Pour cet exemple, elle est appelée uniquement au démarrage ou redémarrage de l'application.

Créer une instance « `ObjectAnimator` », en fournissant les arguments appropriés :

```
ObjectAnimator growAnimator =  
    ObjectAnimator.ofFloat(this, "radius", 0, getWidth());
```

Les arguments : instance, nom de la propriété, valeur de début, valeur de fin.

On peut préciser de manière optionnelle l'interpolateur. Dans cet exemple, il est linéaire (constant) :

```
growAnimator.setInterpolator(new LinearInterpolator());
```

On spécifie la durée de l'animation en millisecondes :

```
growAnimator.setDuration(ANIMATION_DURATION);
```

On démarre l'animation :

```
growAnimator.start();
```

On peut définir des écouteurs, utiles pour réaliser une tâche donnée après la fin de l'animation.

On peut combiner plusieurs animations :

```
// Play the expanding circle, wait, then play the shrinking circle.
```

```
mPulseAnimatorSet.play(growAnimator).before(shrinkAnimator);
```

```
mPulseAnimatorSet.play(repeatAnimator).after(shrinkAnimator);
```

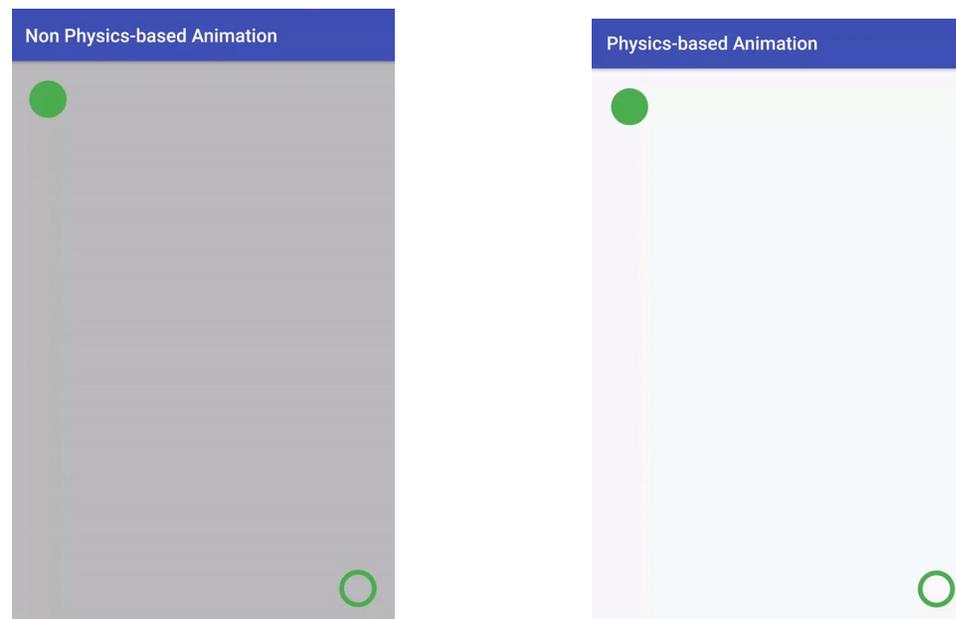
Dans cet exemple, « onDraw » est la méthode en charge de dessiner un cercle.

## Animation basée sur la physique

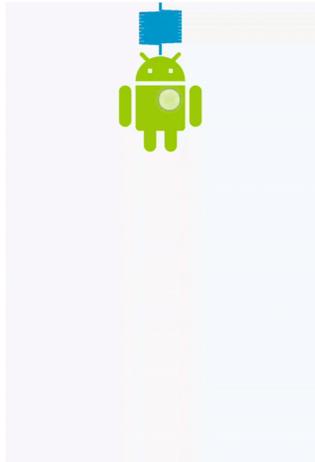
Utilise les fondements de la physique pour réaliser l'animation.

Le système pourra effectuer ainsi un rendu plus lisse et plus réaliste.

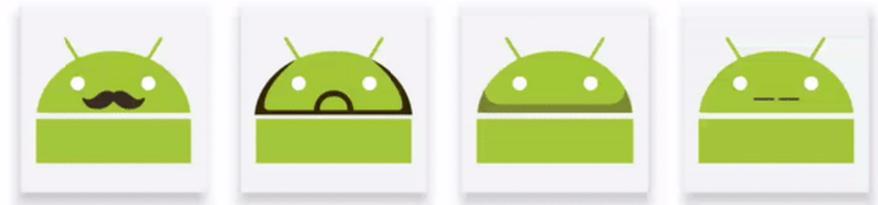
Introduite dès l'API 25. Par contre, il faut intégrer la version 28 de la librairie.



spring animation (simuler un bond)



fling animation (simuler un lancer)



Examiner l'exemple « [PhysicsAnimation](#) »

Il faut inclure dans le fichier « build.gradle » du module « app »

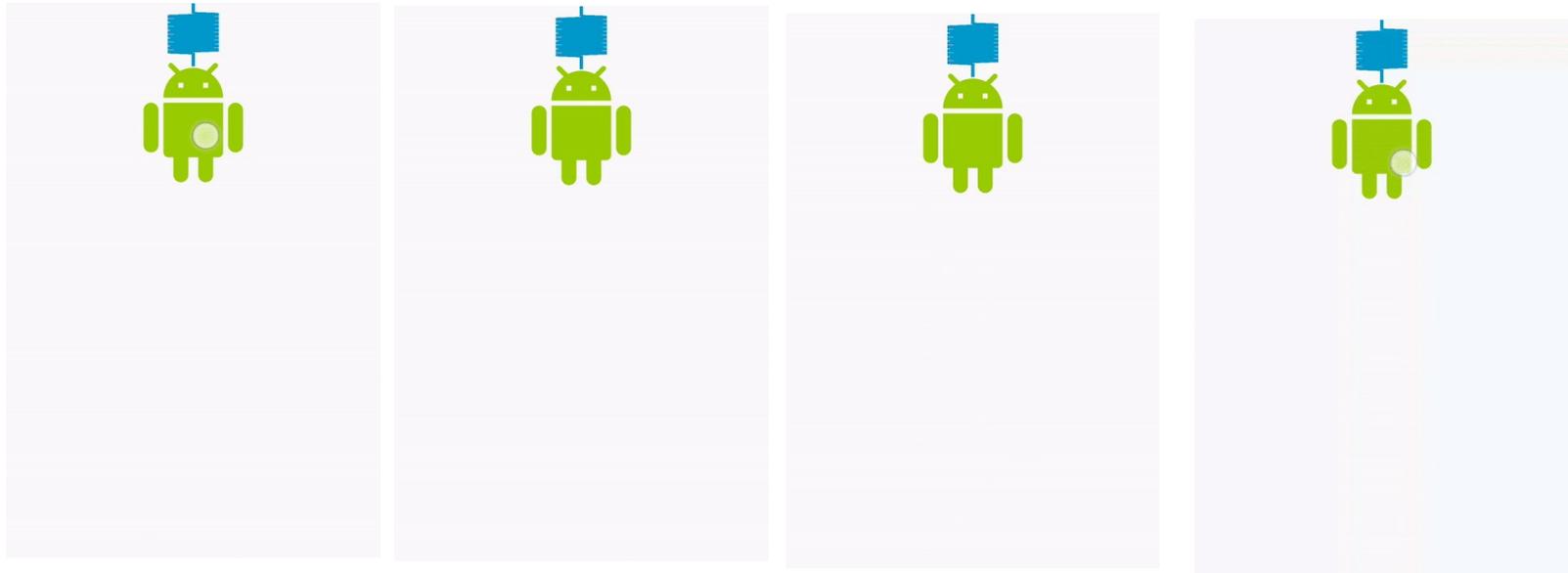
```
dependencies {  
    ...  
    implementation "com.android.support:support-dynamic-animation:28.0.0"  
    ...  
}
```

## Fling animations

```
// création d'une animation et rotation sur l'axe X.  
  
FlingAnimation fling = new FlingAnimation(this, DynamicAnimation.ROTATION_X);  
  
// on fixe les paramètres et contraintes de l'animation.  
// on réalise une rotation presque complète  
  
fling.setStartVelocity(150) // Pixels par seconde.  
    .setFriction(0.11f) // ajouter des frictions pour ralentir l'animation.  
    .start();
```

## Spring animations

```
// on fait un bond sur l'axe Y.  
  
final SpringAnimation anim = new SpringAnimation(this,  
    (FloatPropertyCompat) DynamicAnimation.Y, this.getY())  
    .setStartVelocity(10000); // Pixels par seconde.  
  
// on fixe la raideur de l'objet. Plus la valeur est faible,  
// plus l'objet peut bondir plus rapidement.  
  
anim.getSpring().setStiffness(STIFFNESS_LOW);  
anim.start();
```



Niveau de raideur

élevé	moyen	faible	très faible
-------	-------	--------	-------------

## **Animation de transition**

Elle est utilisée dans 3 situations

- Animer le contenu de la vue quand on passe d'une activité à une autre.
- Animer un élément graphique partagé entre plusieurs écrans.
- Animer le changement de la vue dans la même activité.

Examiner l'exemple « [MaterialAnimationMaster](#) »

## **Bibliographie**

Exemple Tween animation

<https://www.androidhive.info/2013/06/android-working-with-xml-animations/>

Exemple Frame animation

<https://www.101apps.co.za/articles/frame-by-frame-animation-tutorial.html>

Exemple Property Animation

<https://codelabs.developers.google.com/codelabs/advanced-android-training-animations/index.html?index=..%2F..advanced-android-training#0>

<https://google-developer-training.github.io/android-developer-advanced-course-practicals/unit-5-advanced-graphics-and-views/lesson-12-animations/12-1-p-property-animation/12-1-p-property-animation.html>

<https://openclassrooms.com/en/courses/2023346-creez-des-applications-pour-android/2025243-les-autres-ressources#r-2025242>

<https://developer.android.com/guide/topics/graphics/view-animation.html>

<https://developer.android.com/guide/topics/graphics/prop-animation>

Animation basée sur la physique

<https://developer.android.com/training/animation/overview#physics-based>

<https://developer.android.com/guide/topics/graphics/flip-animation.html>

<https://developer.android.com/guide/topics/graphics/spring-animation.html>

Animation de transition

<https://github.com/Igvalle/Material-Animations>