Visual View

Programming Guide

Version 1.0

Table of Contents

1. OVERVIEW 3

1.1. ARCHITECTURE 3
1.2. CLASS DIAGRAM 4
1.3. SUPPORTED PLATFORMS 5
1.4. SUPPORTED FEATURES 5
1.5. COMPONENTS 5
1.6. INSTALLING THE PACKAGE FOR ECLIPSE 5

2. HELLO VISUALVIEW 7

3. USING THE SV CLASS 9

3.1. USING THE INITIALIZE() METHOD 9
3.2. HANDLING SSDKUNSUPPORTEDEXCEPTION 10
3.3. CHECKING THE AVAILABILITY OF VISUAL VIEW PACKAGE FEATURES 10

4. USING THE VISUAL VIEW PACKAGE 11

4.1. CREATING A SLIDE AND SETTING ITS PROPERTIES 11
4.1.1. Creating an SVSlide Instance 11
4.1.2. Applying Properties 12
4.1.3. Adding an Image to the Slide 12
4.1.4. Adding an Image Reference to a Slide 12
4.2. CREATING AND ATTACHING ANIMATIONS 13
4.2.1. Using Implicit Animation 13
4.2.2. Using Explicit Animations 14
4.3. USING ANIMATION LISTENERS 20
4.3.1. Using Implicit Animation Listeners 20
4.3.2. Using Explicit Animation Listeners 20

COPYRIGHT 22

1. Overview

Visual View allows you to easily add various rich visual effects to your applications. This package supports more than 85 animation effects. Your applications can specify geometry, lighting, and projection methods, and rendering properties such as blending and anti-aliasing. You can use Visual View to animate images implicitly or explicitly.

You can use Visual View to add the following animation features to your application:

- Basic Animation – Set the from and to property values.
- KeyFrame Animation – Set properties for different key frames.
- Transition Animation – Apply almost 40 transition effects.
- Sprite Animation – Use a texture atlas to make animations.
- Animation Set - Use a group of animations.

1.1. Architecture

The following figure shows the Visual View architecture.

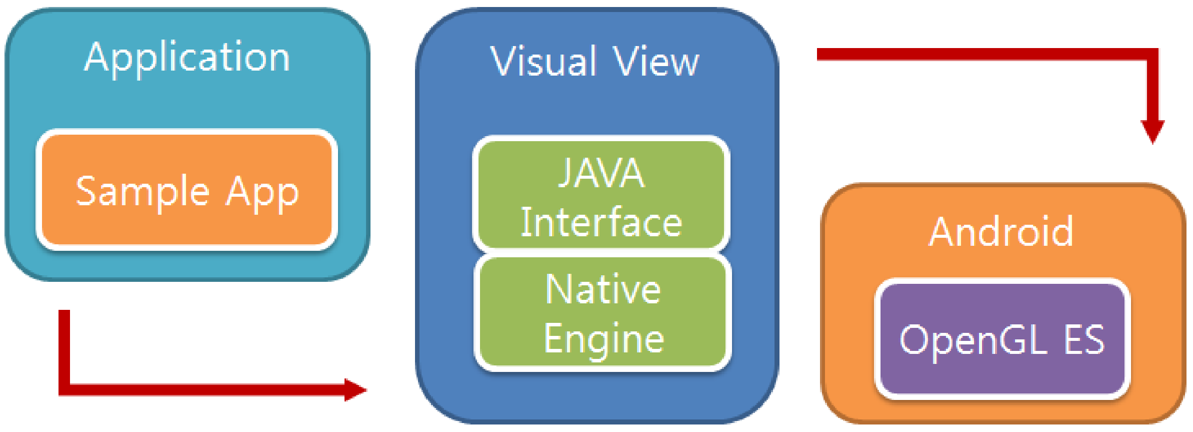


Figure 1: Visual View architecture

The architecture consists of:

- **Applications:** One or more Android Activities that create views with SVGLSurface to render contents .They use Visual View to create animations.
- **Native Engine:** Handles the draw events generated by the Android Framework and renders the Slide instance with OpenGL ES 2.0.
- **Android Framework:** Android SurfaceFlinger composes the rendered SVGLSurface for final display.

1.2. Class Diagram

The following figure shows the Visual View classes and interfaces that you can use in your application.

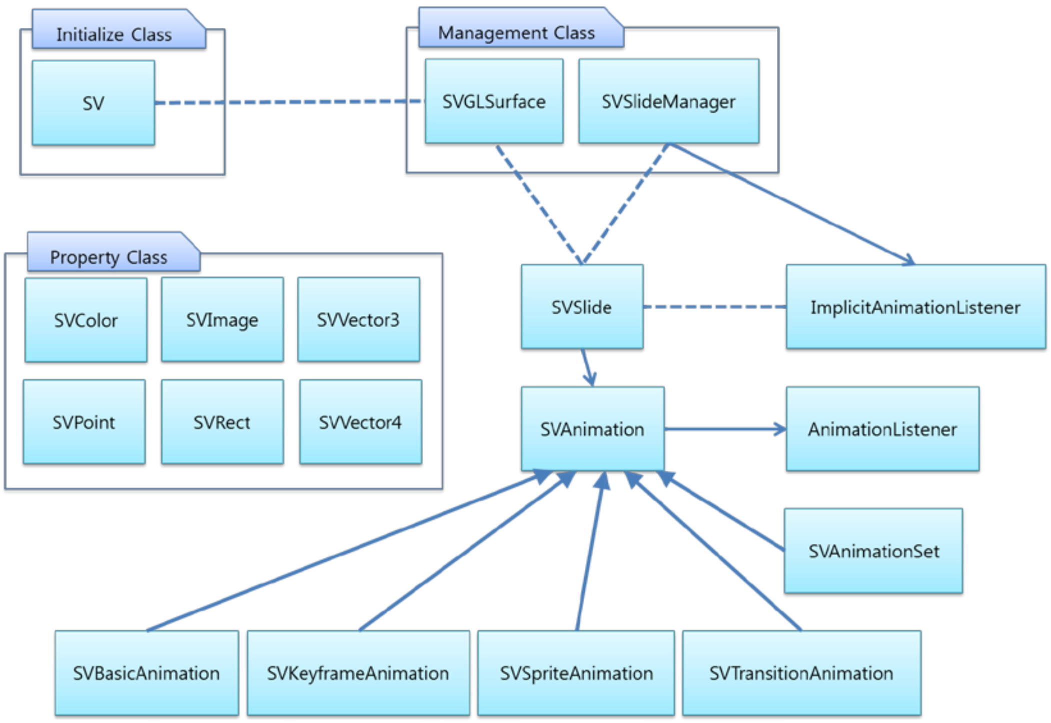


Figure 2: Visual View classes and interfaces

The Visual View classes and interfaces include:

- **SVGLSurface:** Provides the drawing surface for Visual View applications. This class extends Android's GLSurfaceView to enable Visual View to render with OpenGL ES 2.0.
- **SVSlide:** Encapsulates geometry, timing, and visual properties. Provides the content that is displayed. The actual display is not the Slide instance's responsibility.
- **SVAnimation:** Defines animation that can be attached to a Slide instance. This is the root class for all Visual View animations.
- **SVBasicAnimation:** Animates changes one property at a time.
- **SVAnimationSet:** Implements composite animations by combining different types of animations, including other SVAnimationSet instances. Multiple animations created for the same Slide instance must belong to same animation set.
- **SVKeyFrameAnimation:** Implements KeyFrame animations by allowing animations to change frame-by-frame between a starting and end point.
- **SVSpriteAnimation:** Implements a sprite animation created out of a texture atlas. Animations are created by moving frame-by-frame along the texture atlas.
- **SVTransitionAnimation:** Implements a transition animation to display transition effects between initial and final frames based on the specified type.
- **SVImage:** Holds image data that can be set to a slide.

- **SVSlideManager:** Controls implicit animation.

Visual View follows the Model-View-Controller (MVC) pattern.

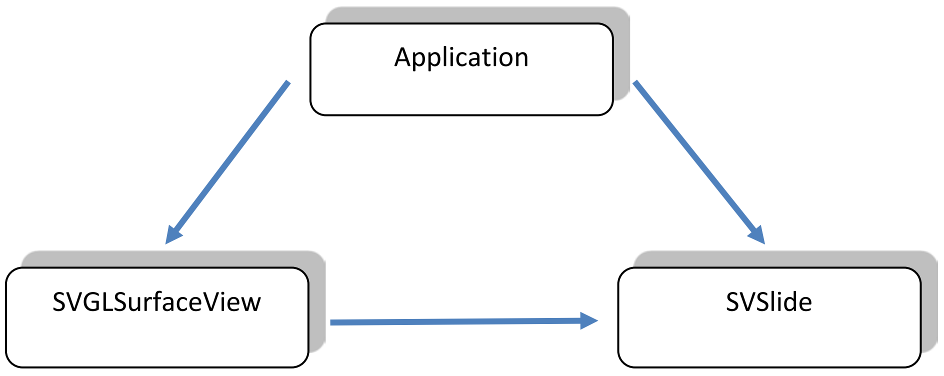


Figure 3: VisualView MVC model

1.3. Supported Platforms

- Android 1.6 or above support Visual View.
- All devices with OpenGL ES 2.0 support Visual View.

1.4. Supported Features

Visual View supports the following features:

- Adding smooth animations seamlessly.
- 41 interpolators for the animations.
- Adding animations implicitly or explicitly.
- Creating a group of animations and attaching it to a Slide instance.

1.5. Components

Components:

- visualview-v1.0.0.jar
- sdk-v1.0.0.jar
- libSVIEngine.so

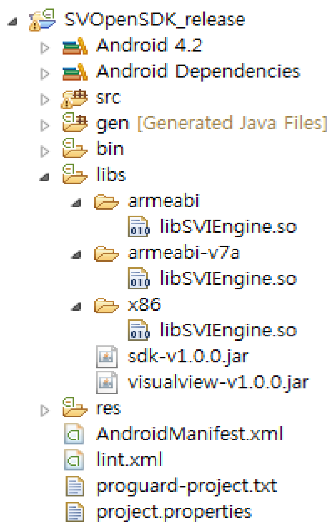
Imported package:

- com.samsung.android.sdk.visualview

1.6. Installing the Package for Eclipse

To install VisualView for Eclipse:

1. Add the visualview-v1.0.0.jar, sdk-v1.0.0.jar, and libSVIEngine.so files to the libs folder in Eclipse.



```
SVOpenSDK_release
    Android 4.2
    Android Dependencies
    src
    gen [Generated Java Files]
    bin
    libs
        armeabi
            libSVIEngine.so
        armeabi-v7a
            libSVIEngine.so
        x86
            libSVIEngine.so
        sdk-v1.0.0.jar
        visualview-v1.0.0.jar
    res
    AndroidManifest.xml
    lint.xml
    proguard-project.txt
    project.properties
```

Figure 4: libs folder in Eclipse

2. Hello VisualView

Hello VisualView is a simple application that:

- initializes the Visual View package
- applies animation effects

```
public class HelloVisualView extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);

		//Initialize SV
			SV sv = new SV();
		try {
			sv.initialize(this);
		} catch (SsdkUnsupportedException e) {
			e.printStackTrace();
			return;
		}

		//Initialize Surface
		mSurface = new SVGLSurface(this);

		//Set the Activity content
			setContentView(mSurface);

		//Create the slide
			mSlide = new SVSlide(null, 50.0f, 50.0f, 800.0f, 800.0f, null);
		//Attach root slide to surface
			mSurface.addSlide(mSlide);
		mSlide.setBackgroundColor(new SVColor(1.0f, 1.0f, 1.0f, 1.0f));
		mSubSlide  =  new  SVSlide(mSlide,  40.0f,  40.0f,  600.0f,  600.0f,  new
SVColor(1.0f, 1.0f, 1.0f, 1.0f));

		mSubSlide.setProjectionType(SVSlide.PROJECTION_PERSPECTIVE);
					InputStream is;
		is = getResources().openRawResource(R.drawable.icon);
		Bitmap bitmap = BitmapFactory.decodeStream(is);

		SVImage svImage = new SVImage(true);
		svImage.setBitmap(bitmap);

		mSubSlide.setImage(svImage);

	}

	SVGLSurface mSurface = null;
	SVSlide mSlide = null;
	SVSlide mSubSlide = null;
```

Initializing Visual View

The sample application:

- Initializes Visual View to check the package availability on the device. If the package is not available or if the device is not a valid Samsung device, the initialization method throws an exception.

Initializes the SVGLsurface instance for rendering before creating the Visual View Slide and attaching animation. @Override
```
protected void onCreate(Bundle savedInstanceState) {
	// TODO Auto-generated method stub
	super.onCreate(savedInstanceState);
	SV sv = new SV();

	try {
		sv.initialize(this);
	} catch (SsdkUnsupportedException e) {
		e.printStackTrace();
		return;
	}
	mSurface = new SVGLSurface(this);
	/*
	 * To create your own surface:
	 *	mSurface = new SVGLSurface(context, translucent, depth, stencil)
	 *
	 */
    …
```

Setting Content View

The Surface instance is a GLSurfaceView instance. You can set the surface view as an Activity content view explicitly. If you do not set the surface as the view of the activity, the output screen remains blank.

```
	// Set the view to Android content view
	setContentView(mSurface);
```

3. Using the SV Class

You need to initialize a SV before you can use it. Samsung Mobile SDK provides a base class with an initialize() method for each package.

The SV can run only on Samsung Smart Devices. Some Samsung Smart Device models do not support some *of the* packages.

You can use a `initialize()` method to initialize it and also to check if the device supports the SV. If the device does not support the SV, the method throws an SsdkUnsupportedException exception. You should handle this exception. If an SsdkUnsupportedException exception is thrown, you can check the exception type with **`SsdkUnsupportedException.getType()`**. If the device is not a Samsung device, the exception type is **SsdkUnsupportedException.VENDOR_NOT_SUPPORTED**. If the device is a Samsung model that does not support the SV, the exception type is **SsdkUnsupportedException.DEVICE_NOT_SUPPORTED**.

The SV class provides the following methods:

- `initialize()` initializes Visual View. You need to initialize the Visual View package before you can use it. If the device does not support Visual View, SsdkUnsupportedException is thrown.
  - VENDOR_NOT_SUPPORTED : if the device is not Samsung Device
  - DEVICE_NOT _SUPPORTED : if the device doesn't have the OPENGL ES 2.0 support.
- `getVersionCode()` gets the Visual View version number as an integer.
- `getVersionName()` gets the Visual View version name as a string.

```
SV sv = new SV();
try {
   // Initialize an instance of SV.
   sv.initialize(this);
    } catch (SsdkUnsupportedException e) {
        // Error Handling
    }

int versionCode = sv.getVersionCode();
String versionName = sv.getVersionName();
```

3.1. Using the initialize() Method

The SV.initialize() method:

- initializes the Visual View package
- checks if the device is a Samsung device
- checks if the Samsung device supports the Visual View package
- checks if the Visual View libraries are installed on the device

```
void initialize(Context context) throws SsdkUnsupportedException
```

If the Visual View package fails to initialize, the initialize() method throws an SsdkUnsupportedException exception. To find out the reason for the exception, check the exception message.

3.2. Handling SsdkUnsupportedException

If an SsdkUnsupportedException exception is thrown, check the exception message type using SsdkUnsupportedException.getType().

The following two types of exception messages are defined in the SV class:

- **VENDOR_NOT_SUPPORTED:** The device is not a Samsung device.
- **DEVICE_NOT_SUPPORTED:** The device does not support OpenGL ES 2.0.

3.3. Checking the Availability of Visual View Package Features

You can check if a Visual View package feature is supported on the device with the isFeatureEnabled() method. The feature types are defined in the SV class. Pass the feature type as a parameter when calling the isFeatureEnabled() method. The method returns a Boolean value that indicates the support for the feature on the device.

```
boolean isFeatureEnabled(int type);
```

4. Using the Visual View Package

Visual View uses SVSlide, similar to the Android View, to which you can attach animations. The basic approach to using Visual View is to create a Slide instance, or a set of Slide instances, and to apply the desired animation or animations to it. A Slide instance is much like an Android View.

4.1. Creating a Slide and Setting Its Properties

4.1.1. Creating an SVSlide Instance

To create a Slide instance:

1. Create a new SVSlide instance.

The container slide is instantiated when the attached SVGLSurface is initialized.

```
/*
* Create the new slide with the desired width, height, and color
* SVSlide(Parent slide, Start X Point, Start Y Point, Width, Height, Slide's Color)
*/
SVColor color = new SVColor(1.0f, 0.0f, 0.0f, 1.0f);
mSlide = new SVSlide(null, 100.0f, 100.0f, 500.0f, 500.0f,color );
mSurface.addSlide(mSlide);
```

The following figure shows the output of the above code.

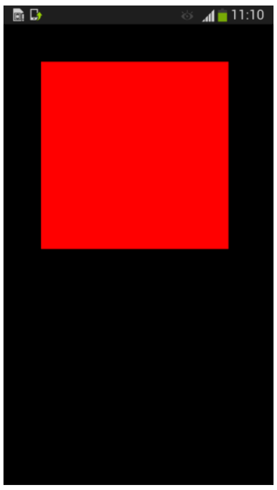


Figure 5: SVSlide instance

You can use the default constructor and then set the Slide properties with the Visual View methods. Use the addSlide(SVSlide) method to attach your first slide to the root slide because you cannot directly access the root slide. This enables the created slide tree to be part of the root tree during rendering. Add other slides as subslides with the SVSlide.addSubSlide(SVSlide) method.

4.1.2. Applying Properties

Set the properties of your Slide instance.

- To set the position of the slide, call `SVSlide.setPosition()`.
- To add a border, call `SVSlide.setBorderWidth()`.
- To rotate the slide, call `SVSlide.setRotation()`.

```
mSlide.setPosition(new SVPoint(100.0f, 100.0f));
mSlide.setBorderWidth(20.0f);
mSlide.setRotation(new SVVector3( 0.0f, 0.0f, 90.0f));
```

For more information on the available properties for SVSlide, see VisualView API Reference.

4.1.3. Adding an Image to the Slide

To add an image to a slide:

1. Decode the bitmap resource.
2. Create an SVImage instance, add the bitmap to it, and add the SVImage instance to your Slide instance.

```
/*
* Decode the Image Resource
*/
int Image = R.drawable.svimage;
Bitmap bitmap = null;
SVImage mSVImage = new SVImage();
InputStream is;
Activity activity = (Activity) context;

is = activity.getResources().openRawResource(Image);
bitmap = BitmapFactory.decodeStream(is);
mSVImage.setBitmap(bitmap);
/*
* Add the Image Reource
*/
mSlide.setImage(mSVImage);
```

4.1.4. Adding an Image Reference to a Slide

Visual View allows you to add images to a Slide instance by using a referenced image. The image is referenced on the Java side and the Visual View Engine only holds the image pointer. Consider the Java Garbage Collection properties before using this approach.

```
/*
* Decode the Image Reource
*/
int Image = R.drawable.svimage;
```

```
Bitmap bitmap = null;
SVImage mSVImage = new SVImage(true);
InputStream is;
Activity activity = (Activity) context;
is = activity.getResources().openRawResource(Image);
bitmap = BitmapFactory.decodeStream(is);
mSVImage.setBitmap(bitmap);
/*
* Add the Image Reource
*/
mSlide.setImage(mSVImage);
```

The following image shows the output of the above code.

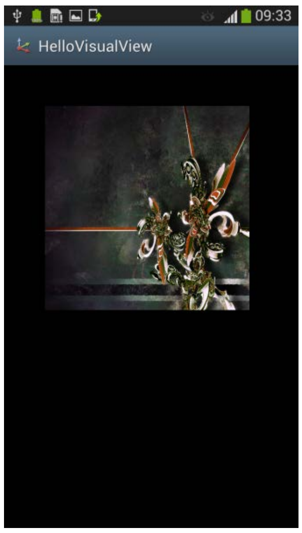


Figure 6: Adding an Image to the Slide instance

4.2. Creating and Attaching Animations

Visual View allows you to create two kinds of animations:

- implicit animations
- explicit animations

4.2.1. Using Implicit Animation

You can create implicit animations in two ways.

The simplest method for creating implicit animations is to set the properties of a Slide instance with an overloaded method that accepts the duration as the second parameter.

```
/* Implicit animation
* Without using checkout and checkin methods
*/
mSlide.setRotation(new SVVector3( 0.0f, 0.0f, 90.0f), 1000);
```

You can use the `checkOutAnimation()` method to add implicit animation without setting the duration separately.

Only use the `checkInAnimation()` method for checked out animations.

```
/*
* Implicit animation
* If interpolator is not set, it takes the default interpolator
*/
SVSlideManager.checkOutAnimation();
SVSlideManager.setDuration(3000);
SVSlideManager.setInterpolatorType(SV.INTERPOLATOR_BOUNCE);
mSlide.setRotation(new SVVector3( 0.0f, 0.0f, 90.0f));
mSlide.setScale(new SVVector3(2.0f, 2.0f, 1.0f));
SVSlideManager.checkInAnimation();
```

You can set a global duration with the `SVSlideManager.setDuration(duration)` method. You can set a global interpolator with the `SVSLideManager.setInterpolatorType()` method. Rotation and scale take effect after the `checkInAnimation()` is executed. In the above example, both scale and rotation use the same duration value and interpolator. For more information on the available methods for implicit animations, see the Visual View API Reference.

You can hold and resume implicit animations with the `holdImplicitAnimation()` and `resumeImplicitAnimation()` methods.

The following figure shows the output of the code.

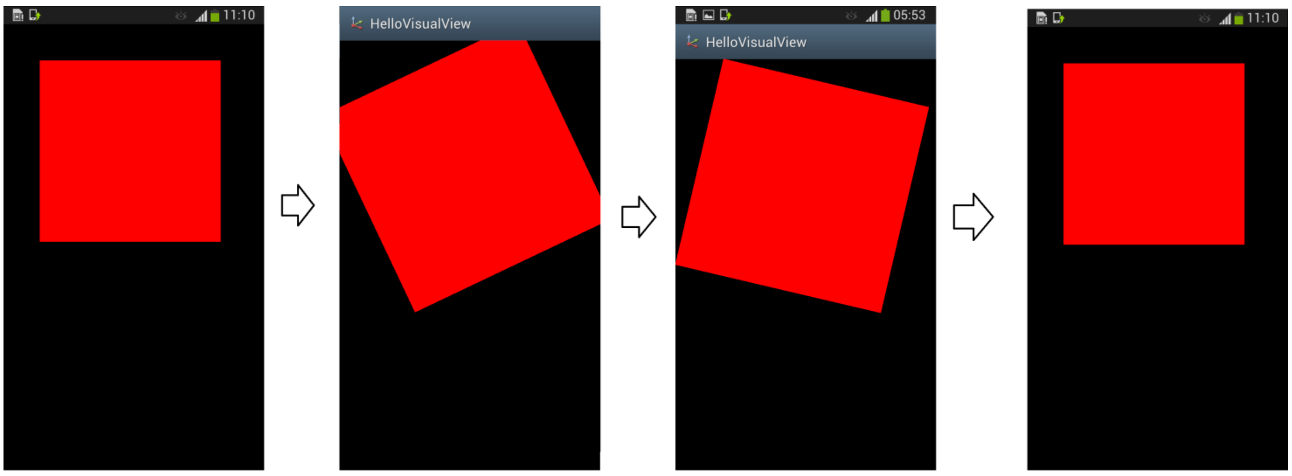


Figure 7: Implicit animations

4.2.2. Using Explicit Animations

You can use the following animation types for explicit animation:

- Basic animation

- KeyFrame animation
- Transition animation
- Sprite animation

4.2.2.1 Basic Animation

You can use explicit animation for all the property types included in the Animation class, such as border, opacity, shadow, and position.

In the sample code below, the Slide instance is set to rotate 180 degree in 1000ms with z as its axis.

Basic animation has two overloaded constructors: one with an array input and the other with a single value input. Use the constructor with a single float value input for properties such as opacity and radius. Use the constructor with the array input for properties such as rotation.

```
/*
* Explicit Animation ===> Basic animation
*/
SVAnimation basicAnimation = null;
int type = SVAnimation.ROTATION;
float[] from = new float[]{0.0f, 0.0f, 0.0f};
float[] to = new float[]{0.0f, 0.0f, 180.0f};
basicAnimation = new SVBasicAnimation(type, from, to);
basicAnimation.setDuration(1000);
mSlide.startAnimation(basicAnimation);
```

Changing Animation Properties

The following table lists the properties that can be changed for an animation.

Methods	Parameters	Description
`setDuration`	int duration	Set the animation duration for this animation. Duration is the animation running time in milliseconds.
`setInterpolator`	int interpolator	Set the animation interpolator for this animation. The valid values are 0 to 41.
`setRepeatCount`	int repeatCount	Set the number of times the animation repeats. You can set the animation to restart automatically after completion of each cycle. Use this method to set the number of times the animation repeats.
`setOffset`	int offset	Set the offset for this animation in milliseconds. The offset time is the time that expires before the animation starts.
`setAutoReverseEnabled`	boolean enabled	Set whether this animation resets to its initial values. Set this to reset the animation properties to their initial values when the animation ends.

In the sample code below, the animation has an offset time of 800. This means that the animation starts after 800 ms. The animation repeats twice because the repeat count is set to 2. The whole animation duration time is set to 1000ms. The animation does not return to its original values because the autoReverse property is set to false.

```
/*
* Explicit Animation ===> Basic animation
*/
SVAnimation basicAnimation = null;
int type = SVAnimation.ROTATION;
float[] from = new float[]{0.0f, 0.0f, 0.0f};
float[] to = new float[]{0.0f, 0.0f, 180.0f};
basicAnimation = new SVBasicAnimation(type, from, to);
basicAnimation.setDuration(1000);
basicAnimation.setInterpolator(SV.INTERPOLATOR_BOUNCE);
basicAnimation.setRepeatCount(2);//Repeats the same animation twice
basicAnimation.setOffset(800);//Animation starts after 800 ms
basicAnimation.setAutoReverseEnabled(false);//Animation  doesn’t  go  back  to  its
original values
mSlide.startAnimation(basicAnimation);
```

4.2.2.2 KeyFrame Animation

You can use KeyFrame animation to set values for the Slide instance on a frame-by-frame basis.

Controlling KeyFrame Properties

You can add properties with the addkeyProperty(time,property) method. Start the animation by setting the time to 0.0f (start time) with the startKeyProperty(value) method and end the animation by setting the time to 1.0f (end time) with the endKeyProperty(value) method to avoid an exception being thrown. Add the intervening values between these two frames with the addKeyProperty(time, value) method. You must take care that the time values that are passed to addKeyProperty() are in ascending order as the frames with a lower time value are played before frames with a higher time value.

The Sample code below shows how to create a KeyFrame Animation.

```
/*
* Explicit Animation ===> KeyFrame animation
*/
SVKeyFrameAnimation scaleAni = new SVKeyFrameAnimation(SVAnimation.SCALE);
scaleAni.startKeyProperty(new float[]{1.0f, 1.0f, 1.0f});
scaleAni.addKeyProperty(0.2f, new float[]{3.0f, 3.0f, 1.0f});
scaleAni.addKeyProperty(0.5f, new float[]{2.0f, 2.0f, 1.0f});
scaleAni.addKeyProperty(0.65f, new float[]{0.5f, 0.5f, 1.0f});
scaleAni.endKeyProperty(new float[]{1.0f, 1.0f, 1.0f});
scaleAni.setDuration(3000);
mSlide.startAnimation(scaleAni);
```

In the sample code, the KeyFrame animation controls the Slide instance scaling at different time values. The Slide instance scales from 1.0f ->3.0f -> 2.0f -> 0.5f -> 1.0f at different key frames.

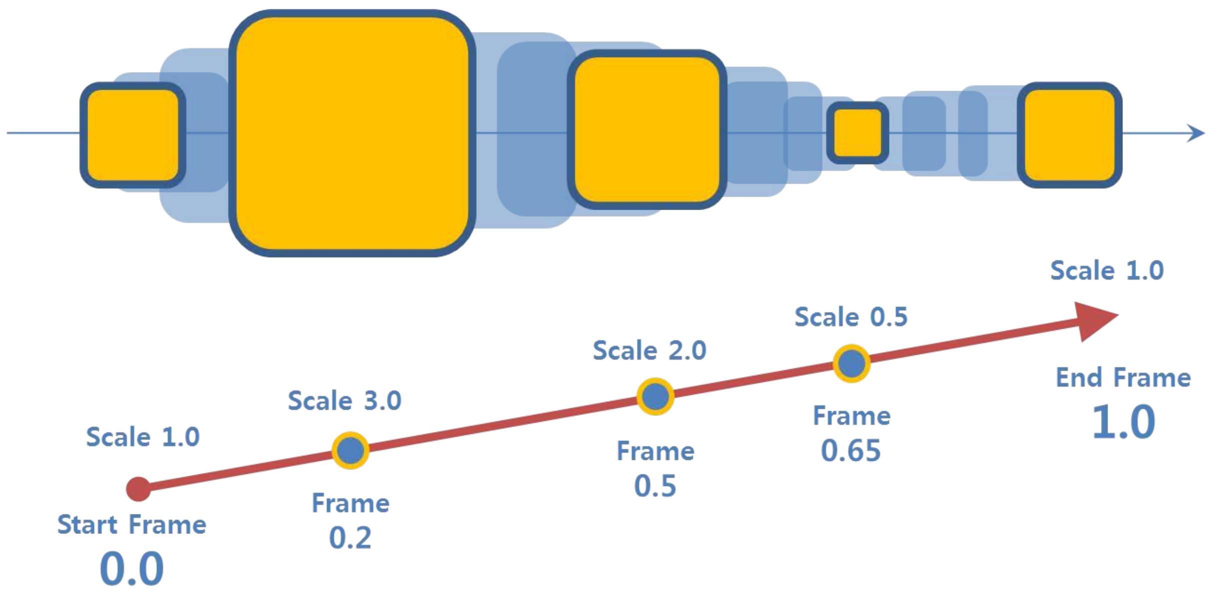


Figure 8: KeyFrame animation

4.2.2.3 Sprite Animation

Sprite animation uses a texture atlas to create an animation from image frames in the atlas. The texture atlas shown below contains animation frames for an explosion. Visual View takes the texture atlas as input and animates these frames.

You can calculate the width and height of each frame as shown in the sample code.

If the number of frames is less than the grid value (for example, 5x4 = 20) or if you are displaying only a few frames, set the interval explicitly using the `setinterval(start,end)` method, where start is the frame value where the animation starts and end is where the animation stops.

The Sample code below shows how to create a Sprite Animation.

```
/*
* Explicit Animation ===> Sprite animation
*/
int width = image.getWidth() /4; // As the grid is 4x4
int height = image.getHeight()/4;
SVSpriteAnimation spriteAnimation= new SVSpriteAnimation(mSpriteImage, width,height);
spriteAnimation.setDuration(3000);
spriteAnimation.setinterval(0,15);//(0, totalcount -1)
mSlide.startAnimation(spriteAnimation);
```

You can set the interval to any value; for example, `setInterval(4,9)`. In this case, the sprite animation runs from frame 4 to frame 9.

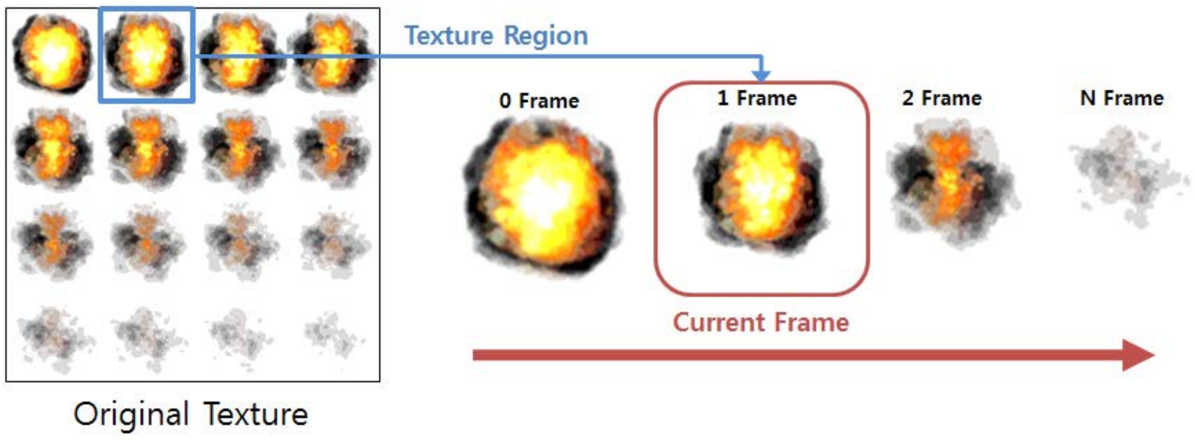


Figure 9: Sprite animation

4.2.2.4 Transition Animation

Transition animation is scene to scene animation. You can apply this animation while going from one rendered scene to another scene. Visual View provides numerous transition animations. You can set the direction for each transition.

In the sample code below, a break transition is used to go from one image to another. If the animation is applied to a child slide, then the transition occurs when going from the child slide to the parent slide.

```
int transitiontype = SVTransitionAnimation.TRANSITION_BREAK;
SVTransitionAnimation transitionanimation = new SVTransitionAnimation(transitiontype,
SVTransitionAnimation.DIRECTION_RIGHT);
transitionanimation.setInterpolator(SV.INTERPOLATOR_ACCELERATE);
transitionanimation.setDuration(5000);

mSlide.startAnimation(transitionanimation);
```

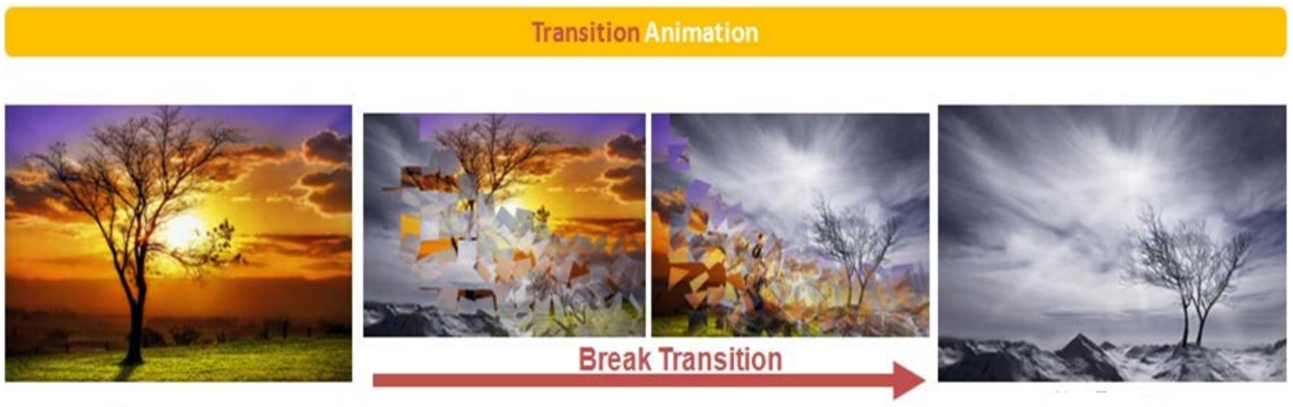


Figure 10: Transition animation

4.2.2.5 Using Animation Sets

Visual View allows you to create a set of explicit animations that you can apply to a Slide instance.

Sharing Animation Information

You can enable animation information sharing with the `setSharedAnimationInfoEnabled(boolean)` method. When you enable animation information sharing, all animations of the animation set use the same animation properties. If you disable animation information sharing, you have to set the properties for each animation in the animation set. Animation set considers the added animation to be a box and applies its own properties on top of the box.

The sample code below shows how to create an animation set:

```
SVAnimationSet aniSet = new SVAnimationSet();
aniSet.setDuration(5000);
aniSet.setSharedAnimationInfoEnabled(true);

int type = SVAnimation.ROTATION;
float[] from1 = new float[]{0.0f, 0.0f, 0.0f};
float[] to1 = new float[]{0.0f, 0.0f, 90.0f};
SVBasicAnimation ani1 = new SVBasicAnimation(type, from1, to1);

type = SVAnimation.BACKGROUND_COLOR;
float[] from2 = new float[]{1.0f, 1.0f, 1.0f, 1.0f};
float[] to2 = new float[]{1.0f, 0.0f, 0.0f, 0.5f};
SVBasicAnimation ani2 = new SVBasicAnimation(type, from2, to2);

aniSet.addAnimation(ani1);
aniSet.addAnimation(ani2);
mSlide.startAnimation(aniset)
```

The sample application uses a rotation animation and a background color animation for a Slide instance. The slide rotates from 0 to 90 degrees and the background color changes.

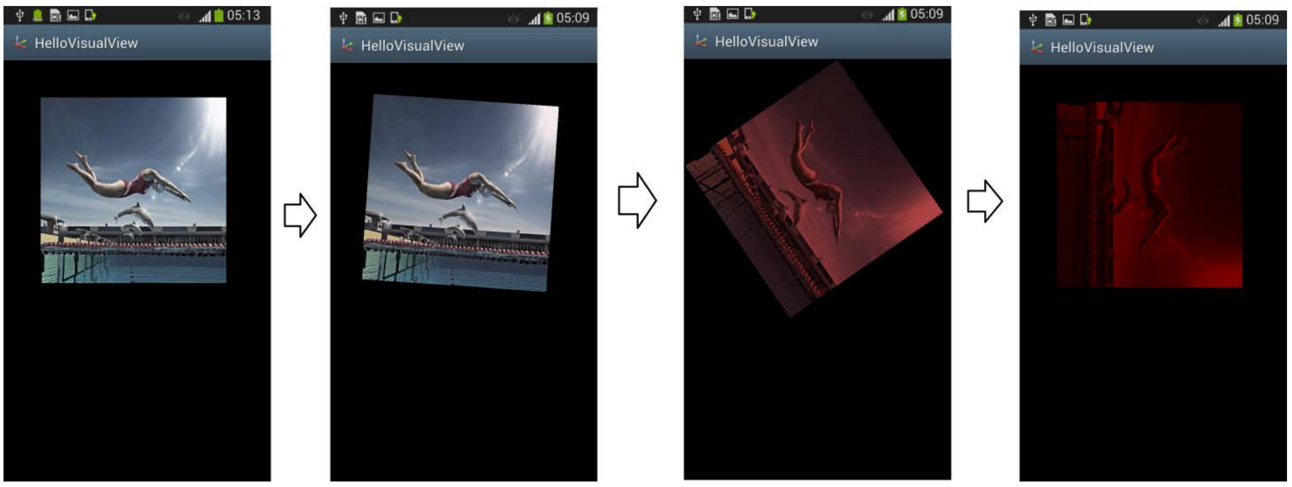


Figure 11: Animation set

4.3. Using Animation Listeners

The Visual View package provides animation listeners for receiving slide animation status change events.

4.3.1. Using Implicit Animation Listeners

You can set the implicit animation listener by implementing the ImplicitAnimationListener interface and registering the listener as shown below.

```
SVSlideManager.checkOutAnimation();
SVSlideManager.setImplicitListener(mTestSlide, new MyImplicitAnimationListener());
SVSlideManager.setInterpolatorType(SV.INTERPOLATOR_BOUNCE);
SVSlideManager.setDuration(1000);

mTestSlide.setRotation(new SVVector3( 0.0f, 0.0f, 90.0f));
mTestSlide.setScale(new SVVector3(2.0f, 2.0f, 1.0f));

SVSlideManager.checkInAnimation();
```

The listener provides callback methods for start, repeat, and end events. The callback methods take the Slide pointer handle as a parameter.

```
private class MyImplicitAnimationListener implements ImplicitAnimationListener{
	@Override
	public void onImplicitAnimationEnded(int slideHandle) {
		// TODO Auto-generated method stub
	}
	@Override
	public void onImplicitAnimationRepeated(int slideHandle) {
		// TODO Auto-generated method stub
	}
	@Override
	public void onImplicitAnimationStarted(int slideHandle) {
		// TODO Auto-generated method stub
	}
}
```

4.3.2. Using Explicit Animation Listeners

You can set the explicit animation listener by implementing the AnimationListener interface and registering the listener.

```
animation.setRepeatCount(mRepeatCount);
animation.setDuration(DURATION);
animation.setAnimationListener(new MyExplicitAnimationListener());
mSlide.startAnimation(animation);
```

The listener provides callback methods for start, repeat, and end events. The callback methods take the animation pointer handle as a parameter.

```
private class MyExplicitAnimationListener implements AnimationListener{
       @Override
       public void onAnimationEnded(int aniHandle) {
              // TODO Auto-generated method stub
       }
       @Override
       public void onAnimationRepeated(int aniHandle) {
              // TODO Auto-generated method stub
       }
       @Override
       public void onAnimationStarted(int aniHandle) {
              // TODO Auto-generated method stub
       }
}
```

For more information on the available Visual View methods, see the Visual View API Reference.

Copyright