Drawing 2D and 3D Graphics and Multimedia

Learning Objective

After studying this module student should be able to:

- Understand about 2D & 3D graphics Animation
- How to create graphics using Library
- Understand about how to play multimedia file in multimedia players or other way
- Will be able to create 2d or 3D graphics object component

Introduction

Android provides a huge set of 2D-drawing APIs that allow you to create graphics.

Android has got visually appealing graphics and mind-blowing animations.

The Android framework provides a rich set of powerful APIS for applying animation to UI elements and graphics as well as drawing custom 2D and 3D graphics.

The android.graphics.Canvas can be used to draw graphics in android. It provides methods to draw oval, rectangle, picture, text, line etc.

The android.graphics.Paint class is used with canvas to draw objects. It holds the information of color and style.

➢ **Canvas**

Android graphics provides low level graphics tools such as canvases, colour, filters, points and rectangles which handle drawing to the screen directly.

The Android framework provides a set of 2D-DRAWING APIs which allows user to provide own custom graphics onto a canvas or to modify existing views to customize their look and feel.

There are two ways to draw 2D graphics,

1. Draw your animation into a View object from your layout.

2. Draw your animation directly to a Canvas.

Some of the important methods of Canvas Class are as follows

I. drawText()
II. drawRoundRect()
III. drawCircle()
IV. drawRect()
V. drawBitmap()
VI. drawARGB()

You can use these methods in onDraw() method to create your own custom user interface.

Drawing an animation with a View is the best option to draw simple graphics that do not need to change dynamically and are not a part of a performance-intensive game. It is used when user wants to display a static graphic or predefined animation.

Drawing an animation with a Canvas is better option when your application needs to re-draw itself regularly. For example video games should be drawing to the Canvas on its own.

Drawing 2D

Android comes along with strong open-source API libraries which support customized 2D and 3D graphics in addition to animations.

The Android framework APIs as well makes available a set of 2D-drawing APIs which gives you room to customize graphics onto a canvas or to alter current Views to change their appearance and feel.

When drawing 2D graphics, you will characteristically do that in two ways. API makes available 2D drawing APIs for simple animation that does not have any need for key alterations changes. These two ways of carrying this out using API are:

- To draw to a View
- To draw on a Canvas

DRAWING A CIRCLE TO VIEW : Drawing to view is a preferred option when your UI does not require dynamic alterations in the application. The most suitable aspect of doing so is that the Android framework will make available for you a pre-defined Canvas to which you will put your drawing calls.

This can be fulfilled merely simply by extending the View category and define an onDraw() callback technique.

Inside your View constituent's onDraw(), Make use of the Canvas offered to you for everyone of drawing, make use of lot of Canvas.draw…() techniques (Ex: canvas.drawCircle(x / 2, y / 2, radius, paint);). onDraw() is a callback technique called when the view is at first drawn.

DRAWING TO A CANVAS: This is the preferred option when your application requires to constantly re-draw itself. Applications like video games ought to be drawing to the Canvas by itself. Although, there are other ways this could be achieved.

HOW TO DRAW 2D OBJECTS ON A CANVAS: To draw 2D graphics in a place in your application that requires to constantly re draw itself, the best option for you is to draw on a canvas. A Canvas functions for you as an interface, to the real surface on which your graphics will be drawn.

If you are required to produce a fresh Canvas, then you ought to specify the bitmap on which drawing will in reality be out. The Bitmap is at all times needed for a Canvas.

DRAWABLES: Android provides a customized 2D graphics files for drawing shapes and images. The android.graphics.drawable file is the location where the regular categories used for drawing in two-dimensions can be found.

We have provided here the fundamentals of making use of Drawable objects to draw graphics and how to make use of a few subclasses of the Drawable category.

A Drawable is a common abstraction for "something that can be drawn." You'll find out that the Drawable category extends to define a lot of particular forms of drawable graphics, which consists of BitmapDrawable, ShapeDrawable, PictureDrawable, LayerDrawable, and many others. You can as well extend these to specify your own customized Drawable objects that act in particular ways.

There are three ways to specify and initiate a Drawable: Through the utilization of an image saved in your project resources; through the use of an XML file that specifies the Drawable features; or the of standard category constructors.

GENERATING FROM RESOURCE IMAGES: An easy way to incorporate graphics to your application is by referring to an image file from your project resources.

The file types that are supported are PNG (which is the most preferred option), JPG (which is an acceptable option) and GIF (which should not be used at all). This method would clearly be preferred for application icons, logos, or other graphics like those made use of in a game.

To make use of an image resource, you merely require to incorporate your file to the res/drawable/ directory of your project.

You can the refer it from your code or your XML layout. Whichever one you choose it is termed making use of a resource ID, which is the file name without the extension of the file type extension like my_image.png is referenced as my_image.

In other scenarios, you may want to take care of your image resource as a Drawable object. To be able to achieve this, build a Drawable from the resource such as:

```
Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);
```

Every singular resource in your project can sustain just a unique state, irrespective of the number of various objects you may initiate for it.

For instance, if you initiate two Drawable objects from an equivalent image resource, then alter a property (like the alpha) for one of the Drawables, then it will as well affect the other.

Thus, anytime you are handling a lot of examples of an image resource, rather than unswervingly changing the Drawable, you ought to carry out a tween animation.

Example XML

The XML code below illustrates how to add a resource Drawable to an ImageView in the XML layout (with a few red tints merely to offer fun).

```
<ImageView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:tint="#55ff0000"
android:src="@drawable/my_image"/>
```

CREATING FROM RESOURCE XML: You ought to have at this stage be able to create a User Interface. Therefore, you should know the strength and flexibility intrinsic in specifying objects in XML.

This is transferred from Views to Drawables. If there is a Drawable object that you'd wish to produce, which is not at first reliant on variables specified by your application code or user interaction, then specifying the Drawable in XML is an excellent option.

Even if you expect your Drawable to alter its features during the user's experience with your application, you ought to take into consideration the specification of the object in XML, as you can at all times alter properties immediately it is initiated.

Immediately you've specified your Drawable in XML, store the file in the res/drawable/directory of your project and after that retrieve and initiate the object by calling Resources.getDrawable(), transferring to it the resource ID of your XML file.

Any Drawable subcategory that supports the inflate() technique can be specified in XML and started by your application. Each Drawable that supports XML inflation makes use of particular XML characteristics that assist you to define the object properties. See the category documentation for every Drawable subcategory for information on how to specify it in XML.

Example: Below are a few XML that specifies a TransitionDrawable:

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/image_expand">
<item android:drawable="@drawable/image_collapse">
</transition>
```

With this XML stored in the file res/drawable/expand_collapse.xml, the code will kick off the TransitionDrawable and set it as the content of an ImageView:

```
Resources res = mContext.getResources();
```

```
	TransitionDrawable transition = (TransitionDrawable)
	res.getDrawable(R.drawable.expand_collapse);
	ImageView image = (ImageView) findViewById(R.id.toggle_image);
	image.setImageDrawable(transition);
```

At this point the transition can be run forward (for 1 second) with:

```
	transition.startTransition(1000);
```

SHAPE DRAWABLE:

Anytime you intend to draw a few 2D graphics dynamically, a ShapeDrawable object will possibly be what you need to achieve this.

A ShapeDrawable, allows you c to draw as a program primeval shapes and design them in any way you can think of.

A ShapeDrawable is an expansion of Drawable, that allows you to make use of it anywhere a Drawable is should be used like for the background of a View, set with setBackgroundDrawable().

Of course, you can as well draw your shape as its own customized View, to be incorporated to your layout no matter the way it pleases you.

Due to the fact that ShapeDrawable possess its own draw() technique, you can produce a subcategory of View that draws the ShapeDrawable during the View.onDraw() technique

See below the main expansion of the View category that draw a ShapeDrawable as a View:

```
public class CustomDrawableView extends View {
	private ShapeDrawable mDrawable;
	public CustomDrawableView(Context context) {
		super(context);
		int x = 10;
		int y = 10;
		int width = 300;
		int height = 50;
```

```
        mDrawable = new ShapeDrawable(new OvalShape());
        mDrawable.getPaint().setColor(0xff74AC23);
        mDrawable.setBounds(x, y, x + width, y + height);

    }

    protected void onDraw(Canvas canvas) {
        mDrawable.draw(canvas);
    }
}
```

In the constructor, a ShapeDrawable is specified as an OvalShape. It's then offered a color and the limits of the shape are set. If you do not set the limits, then the shape will not be drawn, while if you fail to set the color, it will change to black color by default.

With the customized View specified, it can be drawn in any form that pleases you. With the sample above, we can draw the shape as a program in an Activity:

```
CustomDrawableView mCustomDrawableView;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mCustomDrawableView = new CustomDrawableView(this);
    setContentView(mCustomDrawableView);
}
```

If you wish to draw this customized drawable from the XML layout rather than from the Activity, then the CustomDrawable category ought to override the View (Context, characteristic Set) constructor which is invoked during the start of a View through inflation from XML. After this incorporate a CustomDrawable factor to the XML, such as:

```
    <com.example.shapedrawable.CustomDrawableView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

The ShapeDrawable category such as a lot of other Drawable types in the android.graphics.draw+able package permits you to specify a lot of properties of the drawable with public techniques.

A few properties you may wish to alter are alpha transparency, color filter, dither, opacity and color.

You can as well specify primordial drawable shapes with the use of XML.

NINE-PATCHDRAWABLE GRAPHIC: A NinePatchDrawable graphic is a bitmap image that can be stretched, which Android will routinely adjust contain the contents of the View in which you have put in it as the background.

One instance that shows the use of a NinePatch is the backgrounds used by typical Android buttons — buttons ought to stretch to contain strings of varying lengths.

The Draw 9-patch tool presents an exceptionally practical way to build your NinePatch pictures, with the use of a WYSIWYG graphics editor. It even increases warnings if the area you’ve specified for the extendable region is at risk of producing drawing artifacts due to the pixel duplication.

Example XML: Below is a few instance of sample layout XML that shows how to add a NinePatch image to a group of buttons. The NinePatch image is stored in the form res/drawable/my_button_background.9.png

```
<Button id="@+id/tiny"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentTop="true"
android:layout_centerInParent="true"
android:text="Tiny"
android:textSize="8sp"
android:background="@drawable/my_button_background"/>

<Button id="@+id/big"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:layout_centerInParent="true"
android:text="Biiiiiiig text!"
```

```
        android:textSize="30sp"
        android:background="@drawable/my_button_background"/>
```

Observe that the width and height are set to “wrap_content” to see to it that the button fit precisely about the text.

3D Graphics

Almost every Android phone available in the market today has a graphics processing module, or GPU for short. As its name suggests, this is a hardware module dedicated to handling calculations that are usually related to 3D graphics. As an app developer, you can make use of the GPU to create complex graphics and animations that run at very high frame rates.

There are currently two different APIs you can use to interact with an Android device's GPU: Vulkan and OpenGL ES. While Vulkan is available only on devices running Android 7.0 or higher, OpenGL ES is supported by all Android versions.

In this block, we will try to understand and started with using OpenGL ES 2.0 in Android apps.

Prerequisites:

- The latest version of Android Studio
- an Android device that supports OpenGL ES 2.0 or higher
- a recent version of Blender, or any other 3D modeling software

What Is OpenGL ES?

OpenGL, which is short for Open Graphics Library, is a platform-independent API that allows you to create hardware-accelerated 3D graphics. OpenGL ES, short for OpenGL for Embedded Systems, is a subset of the API.

OpenGL ES is a very low-level API. In other words, it doesn't offer any methods that allow you to quickly create or manipulate 3D objects. Instead, while working with it, you are expected to manually manage tasks such as creating the individual vertices and faces of 3D objects, calculating various 3D transformations, and creating different types of shaders.

It is also worth mentioning that the Android SDK and NDK together allow you to write OpenGL ES-related code in both Java and C.

In this Block, lets we understand, how to create 3D graphics using OpenGL in android.

Basic description of Underlying algorithm in step by step form:

1. Create a Project Graphics3d.
2. Put an image in res/drawable.
3. Create a custom view

Multimedia

MediaPlayer overview: The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using MediaPlayer APIs.

This document shows you how to write a media-playing application that interacts with the user and the system in order to obtain good performance and a pleasant user experience.

In android, by using MediaPlayer class we can easily fetch, decode and play both audio and video files with minimal setup.

The android media framework provides a built in support for playing a variety of common media types, such as audio or video. We have a multiple ways to play audio or video but the most important component of media framework is MediaPlayer class.

Android MediaPlayer Class: In android, by using MediaPlayer class we can access audio or video files from application (raw) resources, standalone files in file system or from a data stream arriving over a network connection and play audio or video files with the multiple playback options such as play, pause, forward, backward, etc.

Following is the code snippet, to play an audio that is available in our application's local raw resource (res/raw) directory.

```
MediaPlayer mPlayer = MediaPlayer.create(this, R.raw.baitikochi_chuste);
mPlayer.start();
```

The second parameter in create() method is the name of the song that we want to play from our application resource directory (res/raw). In case if raw folder not exists in your application, create a new raw folder under res directory and add a properly encoded and formatted media files in it.

In case, if we want to play an audio from a URI that is locally available in the system, we need to write the code like as shown below.

```
Uri myUri = ....; // initialize Uri here
MediaPlayer mPlayer = new MediaPlayer();
mPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mPlayer.setDataSource(getApplicationContext(), myUri);
mPlayer.prepare();
mPlayer.start();
```

If we want to play an audio from a URL via HTTP streaming, we need to write the code like as shown below.

```
String url = "http://........"; // your URL here
MediaPlayer mPlayer = new MediaPlayer();
mPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mPlayer.setDataSource(url);
mPlayer.prepare(); // might take long! (for buffering, etc)
mPlayer.start();
```

If you observe above code snippets, we create an instance of MediaPlayer class and added required audio source, streaming type, audio file path, etc. to play an audio from our application.

Apart from above methods, MediaPlayer class provides a different type of methods to control audio and video files based on requirements.

Method	Description
getCurrentPosition()	It is used to get the current position of song in milliseconds.
getDuration()	It is used to get the total time duration of song in milliseconds.
isPlaying()	It returns true / false to indicate whether song playing or not.
pause()	It is used to pause the song playing.
setAudioStreamType()	it is used to specify the audio streaming type.
setDataSource()	It is used to specify the path of audio / video file to play.
setVolume()	It is used to adjust media player volume either up / down.
seekTo(position)	It is used to move song to particular position in milliseconds.
getTrackInfo()	It returns an array of track information.
start()	It is used to start playing the audio / video.
stop()	It is used to stop playing the audio / video.
reset()	It is used to reset the MediaPlayer object.
release()	It is used to releases the resources which are associated with MediaPlayer object.

Table: Methods of MediaPlayer Class

Now we will see how to implement media playing application using MediaPlayer to play a song or audio with multiple playback options, such as play, pause, forward, backward in android application with examples.

Let us Sum Up

In this block we understand about Drawing 2D/3D Graphics, canvas, Draw animation into a View object from your layout. and animation directly to a Canvas.

Methods: drawText(), drawRoundRect(), drawCircle(), drawRect(), drawBitmap(), drawARGB()

and Drawing an animation with a View is the best option to draw simple graphics that do not need to change dynamically and are not a part of a performance-intensive game.

Further Reading

- Android Application Development for Dummies by Donn Felker
- Professional Android 4th Edition by Reto Meier (Author), Ian Lake (Author) ISBN-13: 978-1118949528 ISBN-10: 9781118949528
- Hello, Android: Introducing Google’s Mobile Development Platform by Ed Burnette
- Android Programming by Nicolas Gramlich.
- Thinking in Java (4th Edition) 4th Edition by Bruce Eckel ISBN-13: 978-0131872486 ISBN-10: 0131872486 Android Programming for Beginners: Learn all the Java and Android skills you need to start making powerful mobile applications ISBN-10: 1785883267 ISBN-13: 978-1785883262
- Learning Java by Building Android Games: Explore Java Through Mobile Game Development ISBN-10: 1784398853 ISBN-13: 978-1784398859
- Beginning Android Application Development by Wei-Meng Lee
- Java: A Beginner's Guide, Sixth Edition 6th Edition by Herbert Schildt ISBN-13: 978-0071809252 ISBN-10: 0071809252
- Android Programming: The Big Nerd Ranch Guide (3rd Edition) (Big Nerd Ranch Guides) 3rd Edition by Bill Phillips , Chris Stewart , Kristin Marsicano ISBN-13: 978-0134706054 ISBN-10: 0134706056
- Android Programming: Pushing the Limits 1st Edition by Erik Hellman ISBN-13: 978-1118717370 ISBN-10: 1118717376

- Head First Android Development: A Brain-Friendly Guide 1st Edition by Dawn Griffiths ISBN-13: 978-1449362188 ISBN-10: 1449362184
- Pro Android by Sayed Y. Hashimi and Satya Komatineni, Springer, New York, 2009.

ACTIVITIES

- Create an android application using 2D or 3D animation-based game as per your knowledge

Acknowledgement: “The content in this module is modifications based on work created and shared by the Android Open-Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.”