Programming in Android

Session: 6

Media Handling

Objectives

- Explain the use of graphics in Android
- Explain animation
- Explain OpenGL and OpenGL rendering
- Explain playing of audio and video with Media Player
- Explain the process of capturing image and video using camera
- Explain the process of creating a live wallpaper

Introduction

- Android supports multimedia capability
- Multimedia capabilities play a significant role for customers
- Android's Multimedia API provides the necessary functionality
- Camera API provides functionality to interact with the camera
- OpenGL support is available for making games and multimedia applications




Graphics

- Android provides basic level of graphics tools
- Basic Graphical tools are:
  - Canvases
  - Colors filters
  - Points
  - Rectangles
- The technique varies depending upon a 2D and 3D, static and dynamic, and so on

Basic Graphics Concepts

- Views are known to be the visible elements in an Android UI
- Each View is associated with the Canvas
- When the View is displayed, its onDraw() method is automatically invoked by Android
- The developer can be creating their own View with the onDraw() method to display basic objects using the Canvas
- Canvas has methods for drawing Arcs, Circles, Lines, and so on
- Paint has the method for setting the alpha, color, and so on

onDraw() and onMouseEvent() Methods

- Following Code Snippet demonstrates an example overriding the onDraw() method to create custom graphic output:

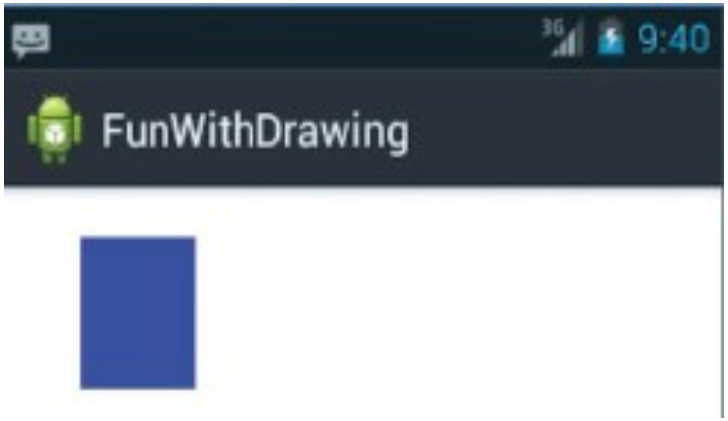
```
protected void onDraw(Canvas canvas) {

        Paint paint = new Paint();

        paint.setColor(Color.YELLOW);

        canvas.drawRect(40, 20, 90, 80, paint);
}
```

- Following Code Snippet demonstrates an example overriding the onMouseEvent() method to respond to user touch events:

```
public boolean onTouchEvent(MotionEvent event) {
        // if it's an up ("release") action
        if (event.getAction() == MotionEvent.ACTION_UP) {
        // Handle Click Event
        if (x >= 40 && x <= 90 && y >= 20 && y <= 80) {
          // redraw the View... this calls onDraw again!
                        invalidate();
                }
        }
```

Graphics Example Application

- Using the code, an application for demonstrating graphics is created as shown in the following figure:



Animation

- Android provides a set of API for applying animation to UI controls
- Android 3.0 introduces the Properties Animation API
- There are multiple types of animations:
  - Property Animation
  - View Animation
  - Drawable Animation
- The super class of the animation API is the Animator class
- The object of Animator class is used to modify the attributes of an object

Base Animation State and Transition Animation State

- Following Code Snippet demonstrates an example of creating the base state for the animation:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/ android" >
<translate
        android:duration="500"
        android:fromXDelta="100%p"
        android:fromYDelta="0%p"
        android:toXDelta="0%p"
        android:toYDelta="0" />
</set>
```

- Following Code Snippet demonstrates an example of creating the Transition state for the animation:

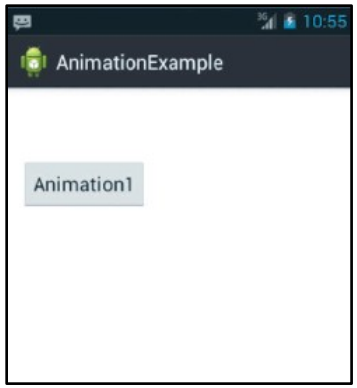
```
<?xml version="1.0" encoding="utf-8"?>
<set>
<translate
        xmlns:android="http://schemas.android.com/apk/res/ android"
        android:duration="500"
        android:fromXDelta="0"
        android:interpolator="@android:anim/accelerate_ interpolator"
        android:toXDelta="0" />
</set>
```

Starting the Animation

- Following Code Snippet demonstrates an example of starting the animation using the xml resources:

```
…
Intent intent = new Intent(getApplicationContext(), FirstActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.anim_in, R.anim. hold);
…
```

- Using the code, an application for demonstrating animation is created as shown in the following figure:



- By clicking the Animation1 button, the title animation is displayed

OpenGL

- OpenGL is a cross platform 2D and 3D graphics rendering library
- Android supports the Open GL for Embedded Systems (OpenGL ES)
- The OpenGL ES library is used to develop multimedia applications and games

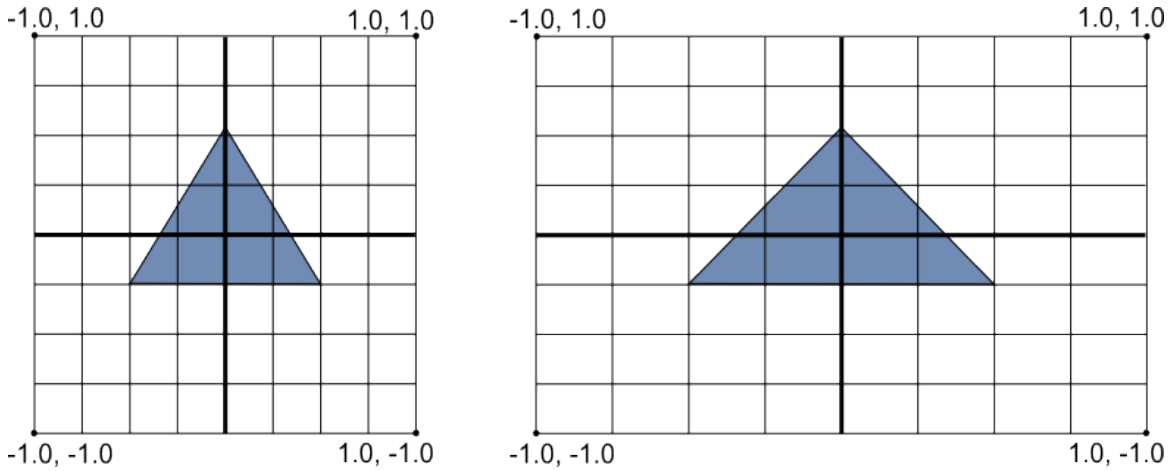
OpenGL History and Support

- Following table shows support for OpenGL ES version by different versions of Android:

OpenGL Version	Android Version
OpenGL ES 1.0	Android 1.0
OpenGL ES 2.0	Android 2.2
OpenGL ES 3.0	Android 4.3
OpenGL ES 3.1	Android 5.0

OpenGL Co-Ordinate System

- OpenGL assumes the center to be at the center of the screen
- The co-ordinates are mentioned in fractions varying from 0 to 1
- The extreme end of the screen is specified by the value of 1f and the center by 0f
- A point half way in between is specified by 0.5f
- OpenGL does not account for varying screen sizes and aspect ratio
- On rectangular displays, the canvas is 'stretched', as shown in the following figure:



OpenGL Views

- Views allow the object to be 'perceived' in a different way

- **Projection View**
  - Allows the object to be 'projected' according to the size and aspect ratio of the screen
  - Responsible for transforming the co-ordinates from a square system to the rectangular varying screen sizes
  - Projection Matrix is used for this

- **Camera View**
  - It is where the object appears to be perceived from
  - In order to move the object on the screen, we can move the camera away from the object
  - Transformation are of two types Translation and Rotation

OpenGL Shaders, OpenGL Vertices, and Draw Order

- A Shader is a user defined code that runs at some stage in the rendering process
- The Shader code is specified using a String literal
- The code to achieve this is shown in the following Code Snippet:

```
private final String vertexShaderCode =
        "uniform mat4 uMVPMatrix;" +
                "attribute vec4 vPosition;" +
                "void main() {" +
                "  gl_Position = uMVPMatrix * vPosition;" +
                "}";
```

- Colors in OpenGL are represented as a matrix of four units
  Programmer is responsible for specifying the order in which the vertices are drawn
- A square can be specified as drawing two triangles
- A → B → C → A → C → D

OpenGL Colors

- Colors in OpenGL are represented as a matrix of four units:
  - Red
  - Green
  - Blue
  - Alpha
- Alpha value specifies the transparency of the color
- All the values are specified in a range of 0f to 1f

Steps to Render an OpenGL Object 1-2

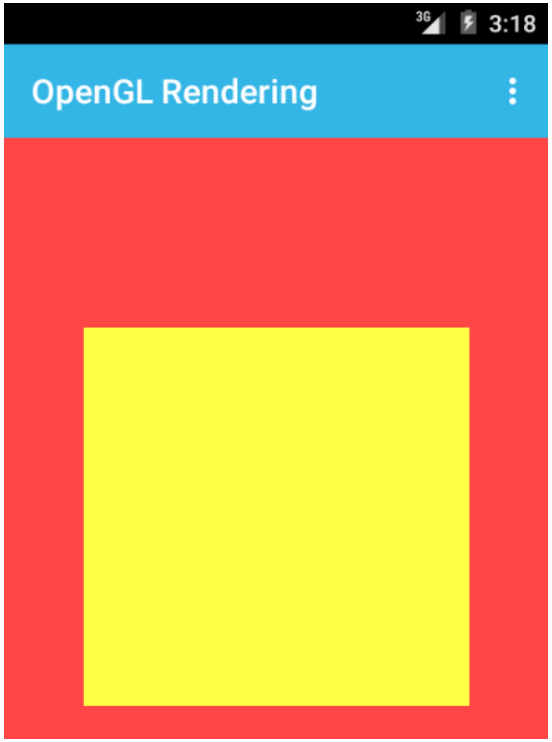
- **Create a View**:
  - Extend the class GLSurfaceView from the package android.opengl.GLSurfaceView
  - View itself does not render the objects
  - Purpose of a view is only to display the rendered objects

- **Creating a Renderer**:
  - In order to implement a renderer, extend the class GLSurfaceview.Renderer
  - Following methods need to be implemented:
    - onSurfaceCreated()
    - onSurfaceChanged()
    - onDrawFrame()

Steps to Render an OpenGL Object 2-2

- **Creating Objects**:
  - Objects can be rendered directly on the onDrawFrame() method
  - However, this process is not recommended
  - Hence, we create OpenGL objects
  - A valid OpenGL object consists of:
    - Shader Code
    - Draw Order
    - Draw Method

- An application demonstrating OpenGL rendering is created as shown in the figure.



Audio/Video Playback

- Android uses OpenCore as its core component of media framework
- OpenCore supports different file formats such as MP3, AAC, MPEG-4, and so on
- Following classes are used to play audio and video in Android framework:
  - MediaPlayer
  - AudioManager

MediaPlayer Basics 1-2

- MediaPlayer class is the most important component of media framework
- Media can be played from:
  - Resource folder
  - File System path
  - URL
- States of MediaPlayer:
  - Creation and initialization
  - Preparation
  - Star of playback
  - Pause or Stop
  - Termination

MediaPlayer Basics 2-2

- Permissions may be required based on the source of the media
- Commonly used permissions are:
  - Internet Permission
  - Wake Lock Permission

MediaPlayer Initialization and Playback

- Following Code Snippet demonstrates an example for initializing the MediaPlayer Object:

```
…
        mediaPlayer = new MediaPlayer();
        mediaPlayer.setAudioStreamType(AudioManager.STREAM_ MUSIC);
        mediaPlayer.setDataSource(url);
        mediaPlayer.setOnPreparedListener(this);
        mediaPlayer.setWakeMode(this, PowerManager.PARTIAL_ WAKE_LOCK);
        mediaPlayer.prepareAsync();
…
```

- Following Code Snippet demonstrates an example playing audio using a resource id:

```
…
        mediaPlayer = new MediaPlayer();
        mediaPlayer.setAudioStreamType(AudioManager. STREAM_MUSIC);
        mediaPlayer.setDataSource(fileDesc. getFileDescriptor(),
        fileDesc.getStartOffset(), fileDesc. getLength());
        fileDesc.close();
        mediaPlayer.prepare();
        mediaPlayer.start();
```

Audio Playback Example Application

- Using the code, an application for demonstrating Audio Playback is created as shown in the following figure:

Video Playback

- Following Code Snippet demonstrates an example playing video using a VideoView:

```
…
        VideoView myVideoView = (VideoView) findViewById(R. id.myvideoview);
        myVideoView.setVideoPath(SrcPath);
        myVideoView.setMediaController(new MediaController(this));
        myVideoView.requestFocus();
        myVideoView.start();
…
```

- Using the code, an application for demonstrating Video Playback is created as shown in the following figure:

MediaStore API

- Android provides the MediaScanner Service which searches and maintains a record of all the media available on the device
- The scanned information is available through the MediaStore API
- MediaStore is a content provider and uses the same API

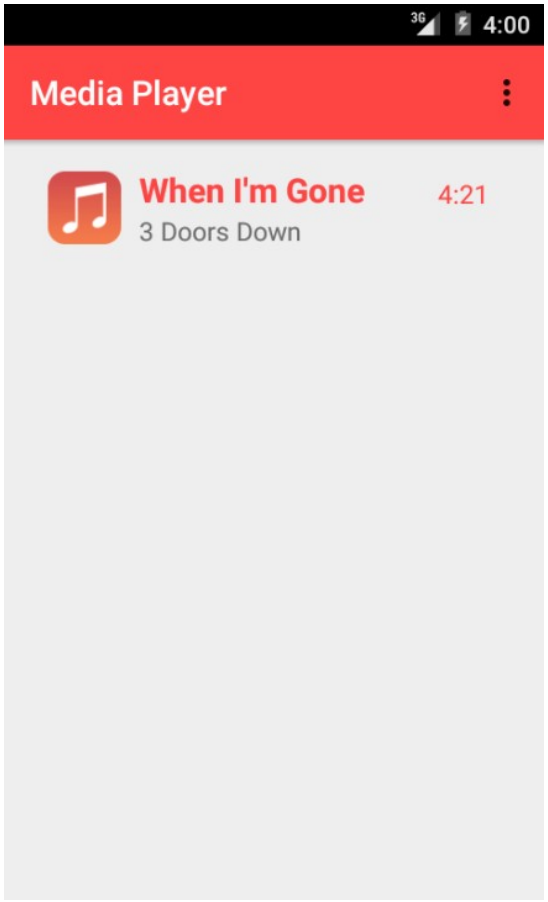
MediaStore Example Application 1-3

- Following Code Snippet demonstrates an example of using the MediaStore API to retrieve audio and video files:

```
…
 Cursor videoCursor = getContentResolver().query(
MediaStore.Video.Media.EXTERNAL_CONTENT_URI,projection,selection, null,
null);
…

…
Cursor = getContentResolver().query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection,
selection,                  null, null);
…
```

MediaStore Example Application 2-3

- Using the code, an application for demonstrating MediaStore API is created as shown in the following figure:



- By selecting a track, the audio player is displayed as shown in the following figure:

MediaStore Example Application 3-3

- The album art is extracted from the track and displayed to the user as shown in the following figure:

- The REWIND and FORWARD buttons move the song behind or forward by five seconds
- In addition to that the user can change the progress using the seek bar as shown in the following figure:

Camera

- Android framework provides support for various cameras and features
- The relevant classes for working with the Camera are:
  - Camera
  - SurfaceView
  - MediaRecorder

Camera Capture

- Following Code Snippet demonstrates an example capturing the camera output and displaying it:

```
…
 Intent captureIntent = new
Intent(MediaStore. ACTION_IMAGE_CAPTURE);
//we will handle the returned data in
//onActivityResult
  startActivityForResult(captureIntent,
CAMERA_CAPTURE);

…
…
        Bitmap thePic =
extras.getParcelable("data");
 //retrieve a reference to the ImageView
        ImageView picView =
(ImageView)findViewById(R. id.picture);
 //display the returned cropped image
 picView.setImageBitmap(thePic);
…
```

- Using the code, an application for demonstrating Camera functionality is created as shown in the following figure:

Camera2 API

- New API introduced in Lollipop
- New features include multiple cameras, Color Correction, Auto focus, and RAW Image capture
- Camera2 API uses a CameraManager to interact with the physical hardware
- The Camera objects are replaced with CameraDevice objects, and the functionality of the object is reduced to just representing the camera device

Capture Process 1-2

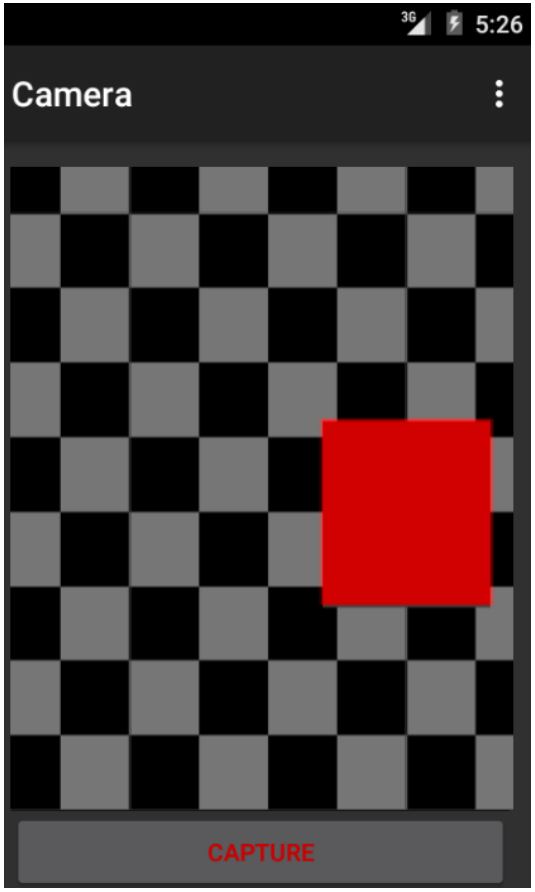
- Get a reference to the camera manager instance using the getSystemService(Context.CAMERA_SERVICE) method
- Use the getCameraIdList() method to retrieve an id list of connected camera devices of the camera manager
- Retrieve the camera characteristics using getCameraCharacteristics() method of the camera manager
- Save the supported capture sizes and initialize the preview resolution
- Open the camera using the openCamera() method of the camera manager
- Get a reference to the CameraDevice object from the onOpened() callback once the camera is opened
- Create a new MediaRecorder object
- Initialize the media recorder object with the appropriate settings
- Retrieve the MediaRecorder surface using the getSurface() method

Capture Process 2-2

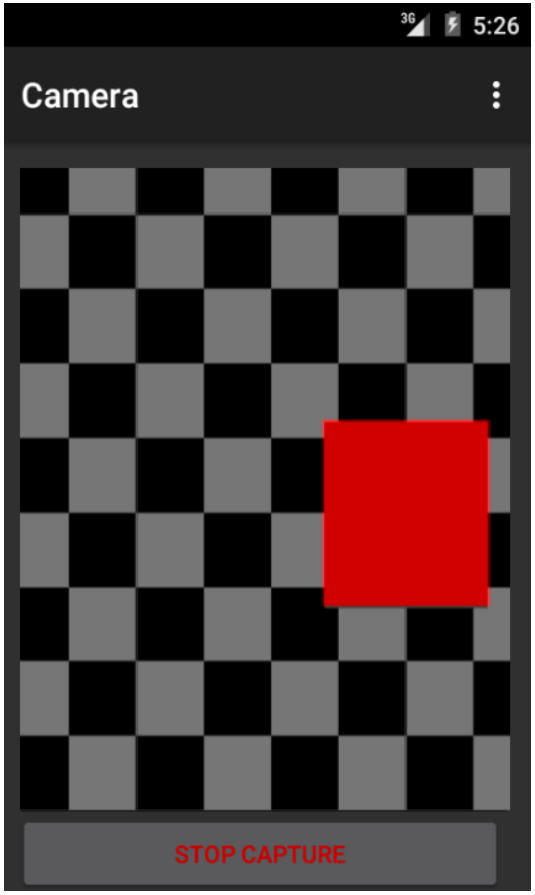
- (Optional) Get the surface of preview target so that the user can know what is being captured
- Create a capture session request by passing the list of surfaces containing the surfaces of the media recorder and the preview target. This is done using the createCaptureSession() method of the camera device. The camera data will be sent to these surfaces
- Start recording video when needed using the MediaRecorder's start() method
- The recording can be stopped using the stop() method
- (Optional) A background thread and handler can be used to not block the UI while waiting for IO and the camera to open, and so on

Camera2 API Example Application 1-3

- Using the steps, an application for demonstrating Camera2 API functionality is created as shown in the following figure:

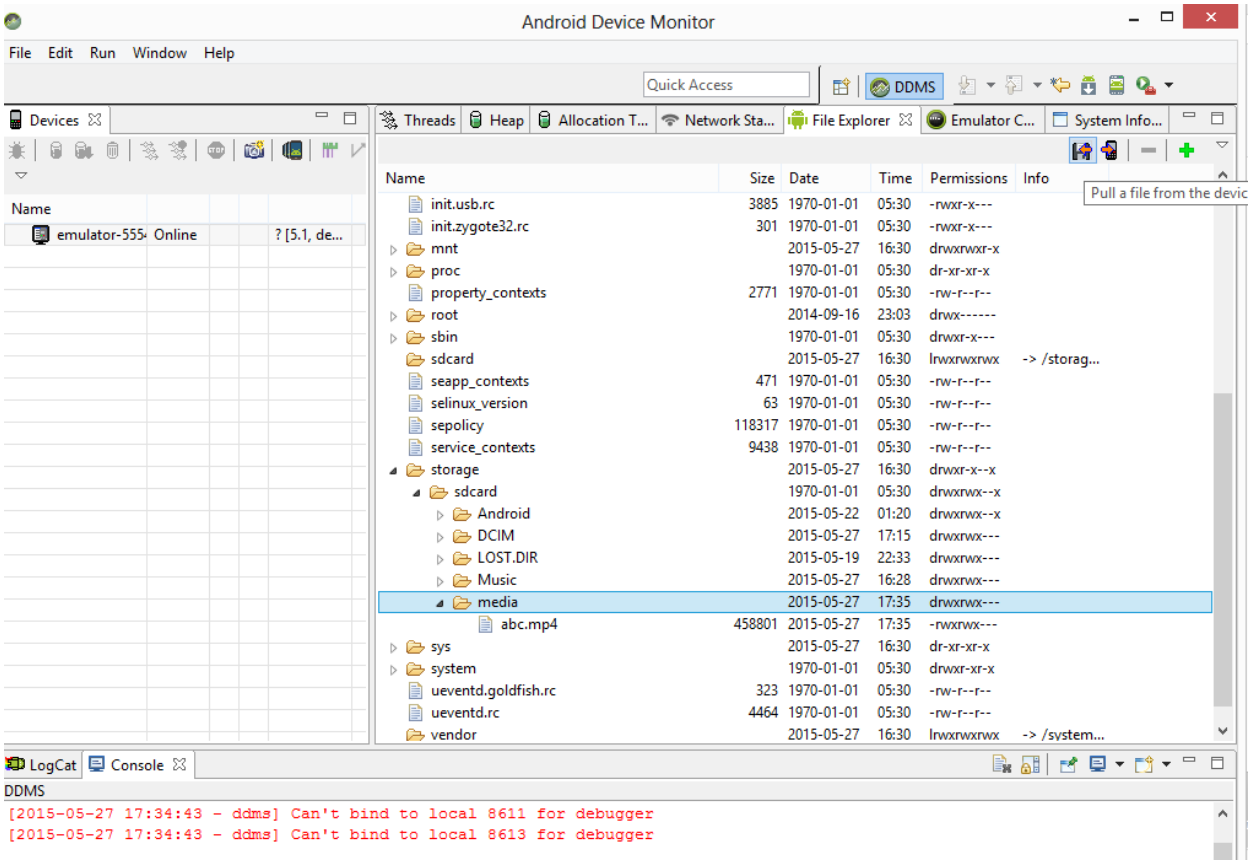


- By clicking the capture button, the recording starts as shown in the following figure:



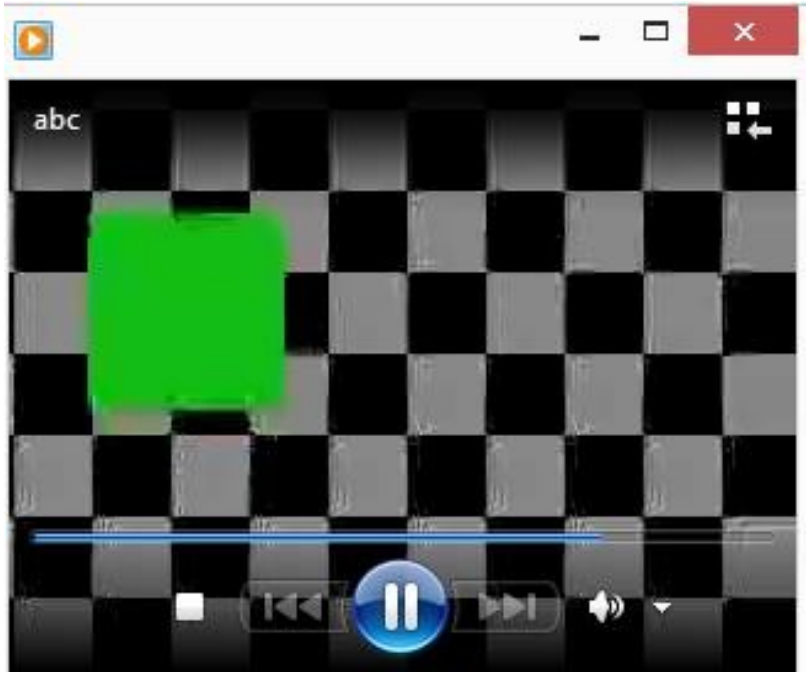
Camera2 API Example Application 2-3

- To stop the recording, the Stop Capture button can be clicked
- The recorded file can be accessed from DDMS as shown in the following figure:



Camera2 API Example Application 3-3

- The file can be pulled from the device to view on the PC as shown in the following figure:



Live Wallpapers

- Live wallpapers are animated and sometimes interactive wallpapers
- Live wallpaper can be created using:
  - Drawing and Animations
  - OpenGL
  - Animated GIFs

Steps to Create a Live Wallpaper 1-2

- **Manifest and Permissions**:
  - The manifest file needs to contain a service extending the Wallpaper service with the permission android.permission.BIND_WALLPAPER
- **Settings Activity**:
  - A settings activity needs to be specified which saves the preferences and settings to configure the live wallpaper.onSurfaceChanged()
- **Metadata Description**:
  - An xml file which holds the details of the live wallpaper such as the name of the wallpaper, a thumbnail, description, and so on

Steps to Create a Live Wallpaper 2-2

- **Wallpaper Service**:
  - A class extending the WallpaperService class needs to exist in the project
  - The onCreateEngine() method needs to be overridden to return an engine object
- **Wallpaper Engine**:
  - The rendering of the wallpaper is handled by the Wallpaper engine
  - In order to create an engine, a new inner class extending the class WallpaperService.Engine needs to be created
  - Following methods need to be implemented:
    - onSurfaceCreated()
    - onSurfaceChanged()
    - onDrawFrame()

Live Wallpaper Example Application 1-4

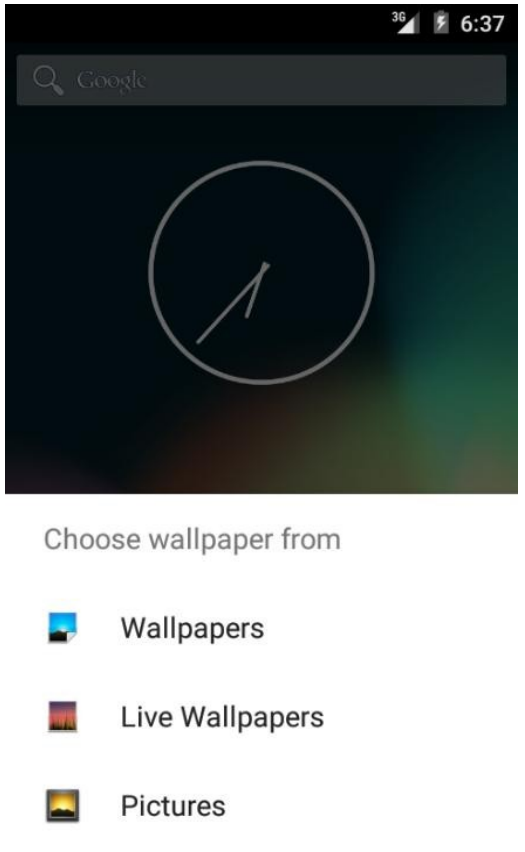
- Following Code Snippet demonstrates an example for creating a Live Wallpaper:

```
public class AnimatedWallpaperService extends WallpaperService {

    private class WallpaperEngine extends WallpaperService.Engine {

        …
                public WallpaperEngine() {
                …
                }
        …
        private void draw() {
            …
        }

        @Override
        public void onTouchEvent(MotionEvent event) { …   }
        }
        …
        public Engine onCreateEngine() { …
        }
}
```

Live Wallpaper Example Application 2-4

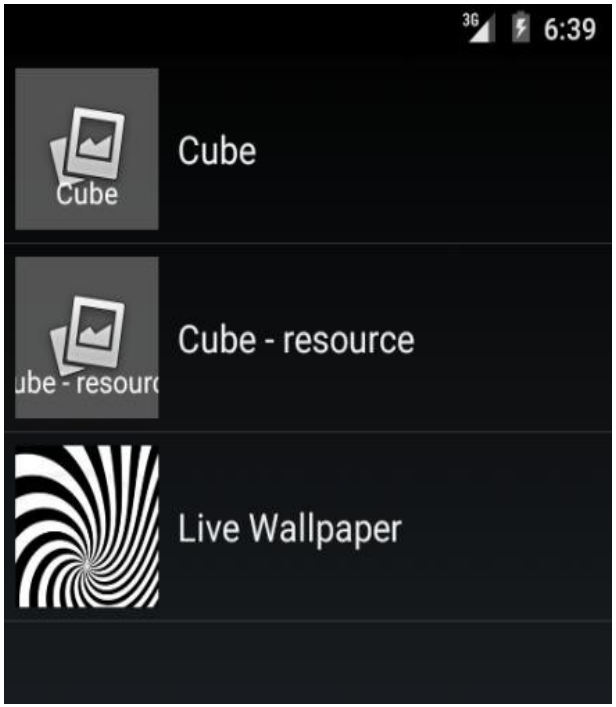
- Using the code and steps, an application for demonstrating Live Wallpapers is created as shown in the following figure:

- In order to set the wallpaper, click and hold on the main screen as shown in the following figure:

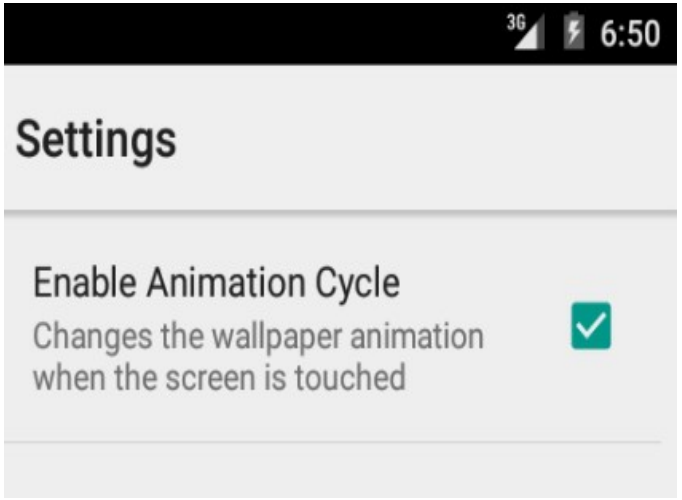


Live Wallpaper Example Application 3-4

- Select Live Wallpaper from the list as shown in the following figure:



- Click Settings to display the wallpaper setting as shown in the following figure:



Live Wallpaper Example Application 4-4

- If the Cycle setting is enabled, the animation changes whenever the user touches the screen as shown in the following figure:

Summary

- Android provides basic level of graphics tools such as canvases, colors filters, points, and rectangles that allows the developer to handle drawing on the screen directly
- Android framework provides two animation types namely property animation and view animation. Property Animation creates an animation by modifying an object's property values over a set period of time with an Animator. View Animation creates an animation by performing a series of transformations on the contents of a View object
- OpenGL is the 2D/3D rendering library supported by Android. It can be used to develop rich multimedia applications and games
- Android multimedia framework includes support for playing variety of common media types, so that one can easily integrate audio, video and images into the applications. The MediaStore API allows searching for media content
- Android framework supports capturing images and video through the Camera API or camera Intent
- Lollipop introduced the new Camera2 API which allows low level access to camera functionality with more features