Android Camera

Outline

- Using external Camera app
- Building a custom Camera app
  - Preview
  - Capturing photos
  - Capturing videos
  - Processing the live feed

Using the camera on Android

- There are two ways of using the camera
- Using an external application to take images or videos
  - Use intents!
- Create your custom camera application
  - To take a picture
  - To take a video
  - To process the camera stream in real time

Using intents

- Easy way to take videos and pictures without developing a new application

1. Create a camera intent according to the type of action
   - `MediaStore.ACTION_IMAGE_CAPTURE`
     for requesting an image from an existing camera application.
   - `MediaStore.ACTION_VIDEO_CAPTURE`
     for requesting a video from an existing camera application.
2. Start the camera intent `startActivityForResult`()
3. Receive the intent result by overriding `onActivityResult`()
   1. Typically, retrieve the reference to the taken image or video for later use or processing

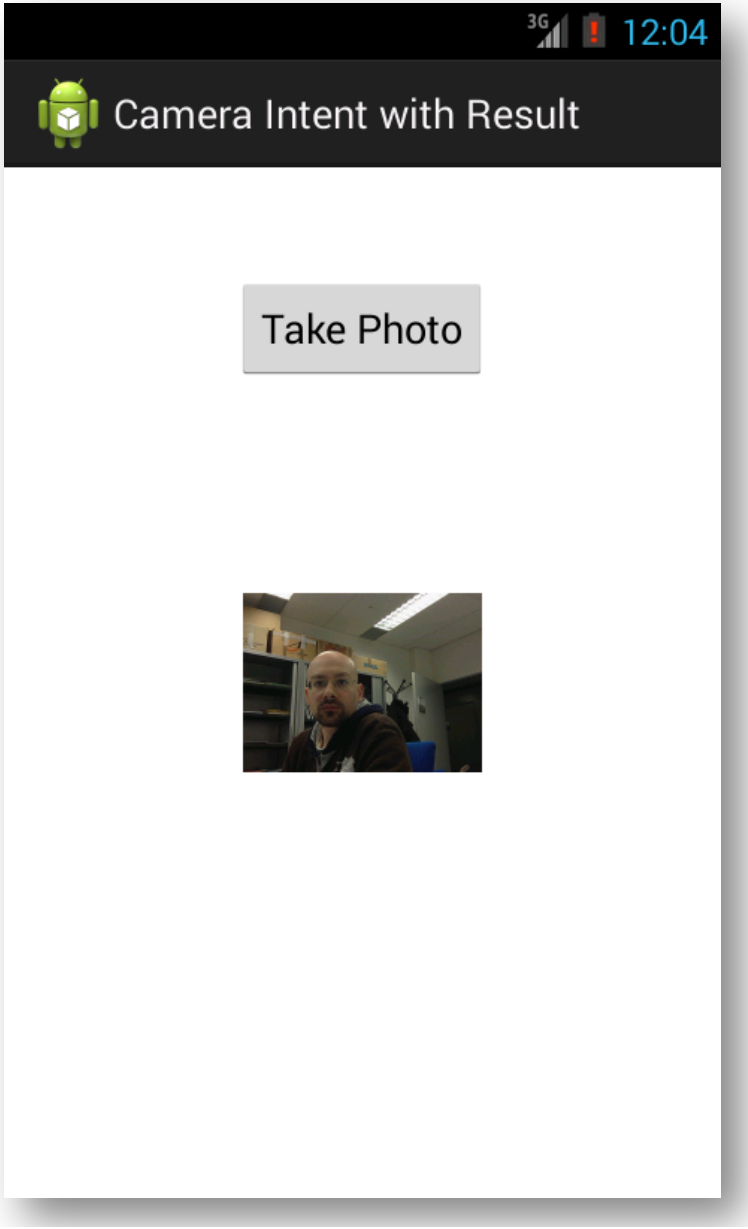
Back to the Camera intent with result

- The result was a thumbnail of the taken picture
- To get the full image we need to ask to save the picture and get back the reference to the file
- Expensive to pass the whole image (or large data in general)!

```
@Override
public void onClick(View v)
{
      Intent newInt = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
      startActivityForResult(newInt, CAMERA_RESULT);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent)
{
      super.onActivityResult( requestCode, resultCode, intent );
      // if the request code is the same as the one used to start the activity
      // and the user has actually taken a picture
      if ( resultCode == RESULT_OK && requestCode == CAMERA_RESULT )
      {
            // get the image from the "extras" of the result intent
            Bundle extras = intent.getExtras();
            Bitmap bmp = (Bitmap) extras.get( "data" );
            // set the result image to the ImageView widget
            imv = (ImageView) findViewById( R.id.ReturnedImageView );
            imv.setImageBitmap( bmp );
```

Preview with a thumbnail



Content resolver – Content provider

- The android.content package contains classes for accessing and publishing data.
- Android enforces a **robust** and **secure** data sharing model.
  - Applications are *not* allowed direct access to other application's internal data.
- Two classes in the package help enforce this requirement:
  - the `ContentResolver`
  - the `ContentProvider`

Content resolver

- A single, global instance providing access to your (and other applications') content providers.
- It accepts requests from clients, and *resolves* these requests by directing them to the content provider
- It includes the CRUD (create, read, update, delete) methods
  - the abstract methods (insert, query, update, delete) in the Content Provider class.
- It provides an abstraction over the content provider

Content Provider

- It provides an abstraction from the underlying data source (i.e. a SQLite database).
- It offers a standard interface that connects data in one process with code running in another process.
- Content Providers provide an interface for publishing and consuming data
- based on URI addressing model using the `content://` schema.
  - Ex. `content://media/external/video/media/44`

A new version

```
import android.provider.MediaStore.Images.Media;

public class MainActivity extends Activity implements OnClickListener
{
    ...
    private Uri imageFileUri;
    ...

    @Override
    public void onClick(View v)
    {
        // first get a new filename from the media provider
        imageFileUri = getContentResolver().insert( Media.EXTERNAL_CONTENT_URI,
                                                    new ContentValues() );

        // create the intent
        Intent newInt = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        // add the filename to assign to the taken picture as extra
        newInt.putExtra( android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri );

        startActivityForResult(newInt, CAMERA_RESULT);
    }
```

A new version

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent)
{
    super.onActivityResult( requestCode, resultCode, intent );
    // if the request code is the same as the one used to start the activity
    // and the user has actually taken a picture
    if ( resultCode == RESULT_OK && requestCode == CAMERA_RESULT )
    {

        // load the image into a bitmap
        Bitmap bmp = BitmapFactory.decodeStream(
                     getContentResolver().openInputStream( imageFileUri ),
                     null,
                     new BitmapFactory.Options());

        // set the result image to the ImageView widget
        imv = (ImageView) findViewById( R.id.ReturnedImageView );
        imv.setImageBitmap( bmp );
```

- We get the full size image
- Ok, but we also need to scale it to fit to the screen...

A new version

```
if ( resultCode == RESULT_OK && requestCode == CAMERA_RESULT )
{
     // easy way to get the image visualized at screen full width
     // we get the display
     Display curr = getWindowManager().getDefaultDisplay();

     // options to pass when opening the image
     BitmapFactory.Options bmpOptions = new BitmapFactory.Options();

     // set the scaled image width to the size of the display
     bmpOptions.outWidth = curr.getWidth();

     try
     {
          // load the image into a bitmap
          Bitmap bmp = BitmapFactory.decodeStream(
               getContentResolver().openInputStream( imageFileUri ),
               null, bmpOptions );

          // set the result image to the ImageView widget
          imv = (ImageView) findViewById( R.id.ReturnedImageView );
          imv.setImageBitmap( bmp );
     }
     catch( FileNotFoundException e )
      ...
```

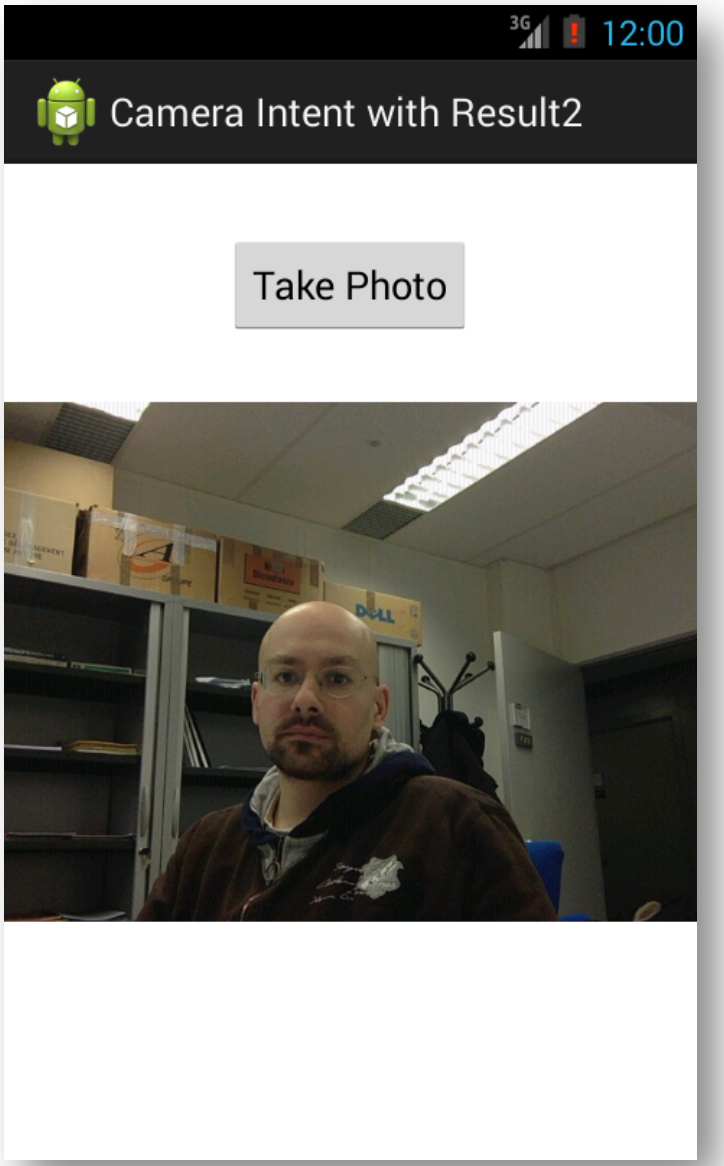
Are we missing something?

- Ooops...
- This new version requires the permission to write on the external storage as we are explicitly creating a new file in our application!!
- Because of the

  `getContentResolver().insert(...)`

  we are creating a new data record in the store
- Add `android.permission.WRITE_EXTERNAL_STORAGE` to the manifest file

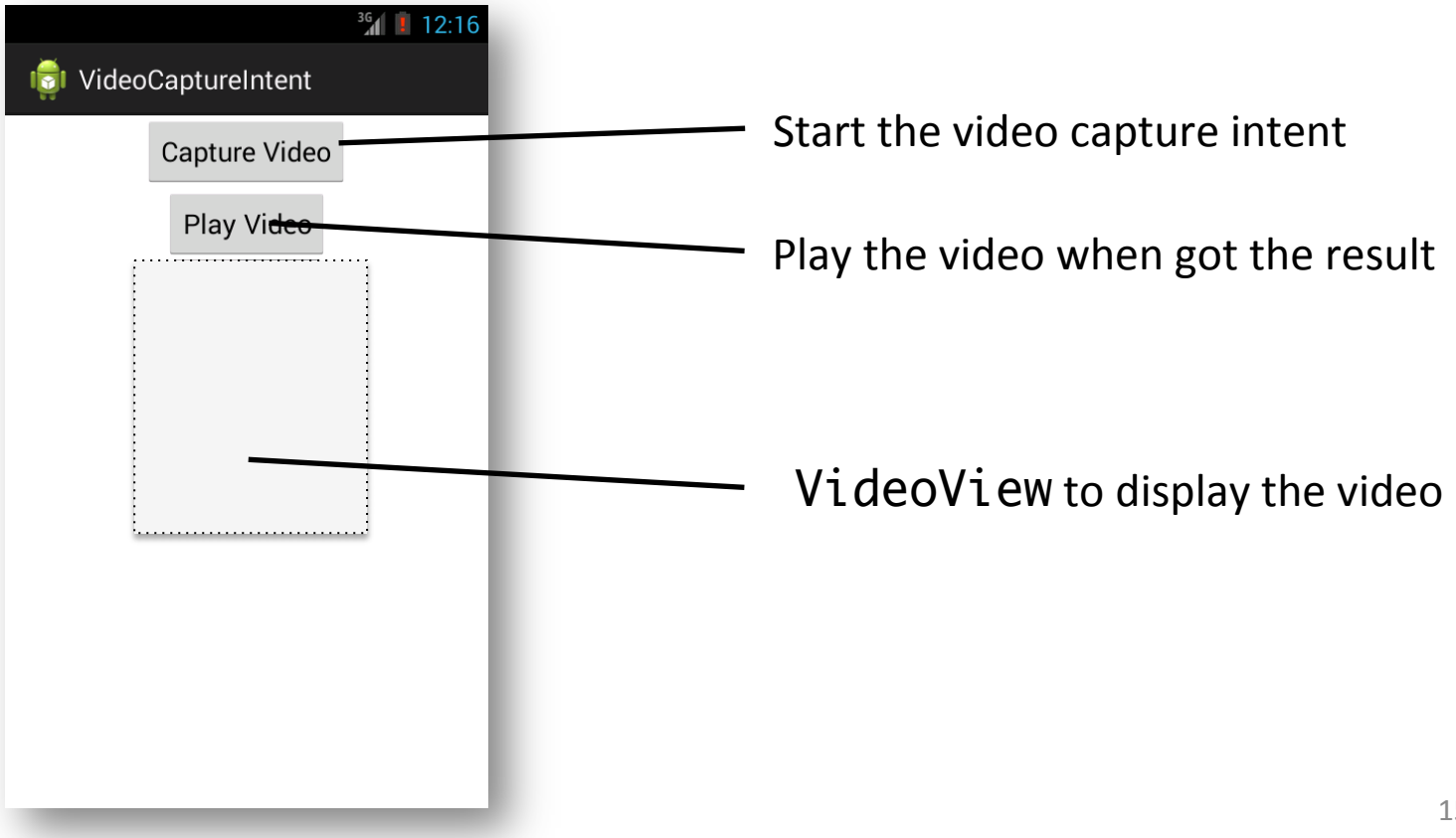
A better preview...



Exercise: manage the different layouts of the picture (landscape, portrait) so it always fit the screen

What about videos?

- It's similar but it is not necessary to use the content provider
- The intent will return the Uri of the video file in data
- Just use that to retrieve the video



The UI...

```
protected void onCreate( Bundle savedInstanceState )
{
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_main );

    // retrieve all the buttons and set the listener
    captButton = (Button) this.findViewById( R.id.CaptureVideoButton );
    playButton = (Button) this.findViewById( R.id.PlayVideoButton );

    captButton.setOnClickListener( this );
    playButton.setOnClickListener( this );

    // at the beginning the play button is disabled
    playButton.setEnabled( false );

    vidView = (VideoView) this.findViewById( R.id.VideoView );
```

The video capture intent

```
@Override
public void onClick( View v )
{
    if( v.getId() == captButton.getId() )
    {

        Intent captureVideoIntent = new Intent(
                android.provider.MediaStore.ACTION_VIDEO_CAPTURE );

        startActivityForResult( captureVideoIntent, CAMERA_RESULT);
    }
 ...
```

Just use the action **android.provider.MediaStore.*ACTION_VIDEO_CAPTURE***

Get the result from the intent...

```
@Override
protected void onActivityResult( int requestCode, int resultCode,
        Intent data )
{
    if( resultCode == RESULT_OK && requestCode == CAMERA_RESULT )
    {
        videoFileUri = data.getData();

        // now the button can be used
        playButton.setEnabled( true );
    }
}
```

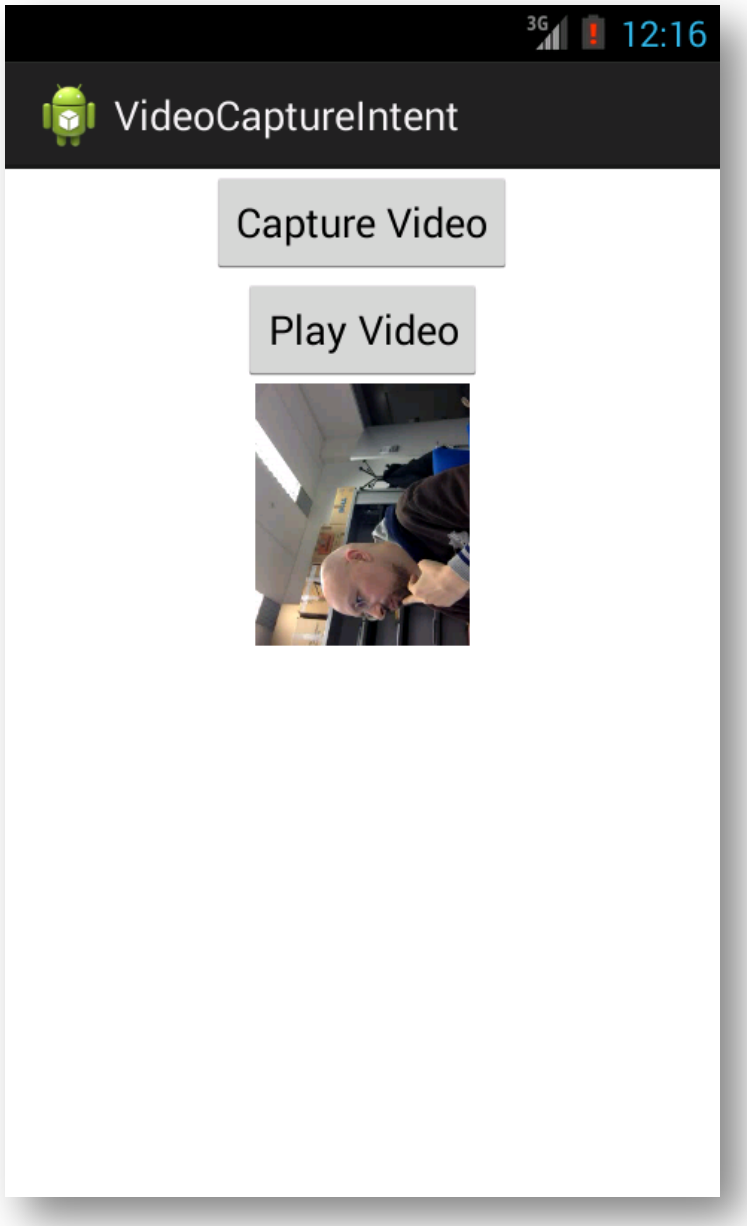
… And play the video

```
@Override
public void onClick( View v )
{
    if( v.getId() == captButton.getId() )
    {
        Intent captureVideoIntent = new Intent(
              android.provider.MediaStore.ACTION_VIDEO_CAPTURE );

        startActivityForResult( captureVideoIntent, CAMERA_RESULT );
    }
    else if( v.getId() == playButton.getId() )
    {
        vidView.setVideoURI( videoFileUri );
        vidView.start();
    }

}
```

Video capture with intent and preview



Other extras for the video intent

MediaStore.EXTRA_VIDEO_QUALITY

- This value can be 0 for lowest quality and smallest file size or 1 for highest quality and larger file size.

MediaStore.EXTRA_DURATION_LIMIT

- Set this value to limit the length, in seconds, of the video being captured.

MediaStore.EXTRA_SIZE_LIMIT

- Set this value to limit the file size, in bytes, of the video being captured.

Manifest – uses-feature

- When using the camera through an intent you **don't need the permission** to use the camera.
- The application must declare the use of camera feature though!

```
<uses-feature android:name="android.hardware.camera" />
```

- Google Play prevents your application from being installed to devices not supporting the camera
- If your application *can use* a camera but does not *require* it, set `android:required` attribute to false:

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

Manifest – uses-feature

- You can specify other camera sub-features your application may need or require (mostly for a custom camera app)

Feature	Description
`android.hardware.camera`	With multiple cameras, it uses the camera facing away from the screen.
`android.hardware.camera.autofocus`	The application uses the autofocus.
`android.hardware.camera.flash`	The application uses the flash.
`android.hardware.camera.front`	The application uses a front-facing camera.
`android.hardware.camera.any`	The application uses at least one camera facing in any direction, or an external camera device if one is connected.
`android.hardware.camera.external`	The application uses an external camera device if one is connected.

To resume

- Using external Camera app -- > intent
- Image
  - Action `MediaStore.ACTION_IMAGE_CAPTURE`
  - Pre-create URI to pass to the intent
  - Get the image from URI once `onActivityResult()` is called
- Video
  - Action `MediaStore.ACTION_VIDEO_CAPTURE`
  - Get the URI from the result intent in `onActivityResult()`
  - `VideoView` for quick playback

Outline

- Using external Camera app
- Building a custom Camera app
  - Camera Preview
  - Capturing photos
  - Capturing videos
  - Processing the live feed

Custom camera application

- Customized camera app requires more code…

`Camera`

- API for controlling device cameras.
- Allows to take pictures or videos

`SurfaceView`

- This class is used to present a live camera preview to the user.

`MediaRecorder`

- This class is used to record video from the camera.

Custom camera application

- **Detect and Access Camera**: check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class extending SurfaceView to preview the live feed from the camera.
- **Build a Preview Layout** - Create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for capturing images or videos
- **Capture and Save Files** - Setup the code for saving the output.
- **Release the Camera** - After using the camera, properly release it for use by other applications.

Detect the camera

- Camera hardware is a shared resource
- It must be carefully managed to avoid collisions with other apps.
- If camera is not required in the manifest you need to check at **runtime**

```
/** Check if this device has a camera */
private boolean checkCameraHardware(Context context)
{
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA))
    {
        // this device has a camera
        return true;
    }
    else
    {
       // no camera on this device
       return false;
    }
}
```

PackageManager.hasSystemFeature()

Access the camera

- Get an instance of `Camera` using the static method `Camera.open()`

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance()
{
    Camera c = null;
    try
    {
        c = Camera.open(); // attempt to get a Camera instance
    }
    catch (Exception e){
        // Camera is not available (in use or does not exist)
    }
    return c; // returns null if camera is unavailable
}
```

This will provide access to the first, back-facing camera on a multiple camera device.

Back-facing = facing of the camera is opposite to that of the screen (ie NOT the selfie camera).

Access the camera (multi-camera device)

```
public static int findBackFacingCamera()
{
    int cameraId = -1;
    // get the number of the cameras available on the device
    int numberOfCameras = Camera.getNumberOfCameras();
    // check the info for each of them and the return the front one
    for( int i = 0; i < numberOfCameras; i++ )
    {
        CameraInfo info = new CameraInfo();
        Camera.getCameraInfo( i, info );
        if( info.facing == CameraInfo.CAMERA_FACING_BACK )
        {
            cameraId = i;
            break;
        }
    }
    return cameraId;
}
```

Access the camera (multi-camera device)

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance()
{
    Camera c = null;
    int camId = findBackFacingCamera();
    if( camId != -1 )
    {
        try
        {
            // attempt to get a Camera instance
            c = Camera.open( camId );
        }
        catch (Exception e){
            // Camera is not available (in use or does not exist)
        }
    }
    return c; // returns null if camera is unavailable
}
```

Custom camera application

- **Detect and Access Camera**: check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class extending SurfaceView to preview the live feed from the camera.
- **Build a Preview Layout** - Create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for capturing images or videos
- **Capture and Save Files** - Setup the code for saving the output.
- **Release the Camera** - After using the camera, properly release it for use by other applications.

Creating a preview class

- We need to display the feed of the camera
- A camera preview class is a `SurfaceView` that can display the live image data coming from a camera
- This class implements `SurfaceHolder.Callback` :
  - It defines the callback events for creating and destroying the view
  - needed for assigning the camera preview input.

`public static interface` **`SurfaceHolder.Callback`**

Public Methods	
abstract void	surfaceChanged (SurfaceHolder holder, int format, int width, int height) This is called immediately after any structural changes (format or size) have been made to the surface.
abstract void	surfaceCreated (SurfaceHolder holder) This is called immediately after the surface is first created.
abstract void	surfaceDestroyed (SurfaceHolder holder) This is called immediately before a surface is being destroyed.

Creating a preview class

```
/** A basic Camera preview class */
public class CameraPreview extends SurfaceView
                           implements SurfaceHolder.Callback
{
    private SurfaceHolder mHolder;
    private Camera mCamera;

    public CameraPreview(Context context, Camera camera)
    {
        super(context);
        mCamera = camera;

        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions <3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
```

Creating a preview class

Implementing the SurfaceHolder callbacks

```
@Override
public void surfaceCreated(SurfaceHolder holder)
{
    // The Surface has been created, now tell the camera where to
    draw the preview.
    try
    {
        mCamera.setPreviewDisplay(holder);
        mCamera.startPreview();
    }
    catch (IOException e)
    {
        Log.d(TAG, "Error setting camera preview: " + e.getMessage());
    }
}
```

Creating a preview class

```
@override
public void surfaceDestroyed(SurfaceHolder holder)
{
    // empty. Nothing to do here
    // Take care of releasing the Camera preview in your activity.
}
```

Creating a preview class

```
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
    // If your preview can change or rotate, take care of those events here.
    // Make sure to stop the preview before resizing or reformatting it.

    if (mHolder.getSurface() == null){
      // preview surface does not exist
      return;
    }

    // stop preview before making changes
    try {
        mCamera.stopPreview();
    } catch (Exception e){
      // ignore: tried to stop a non-existent preview
    }

    // set preview size and make any resize, rotate or
    // reformatting changes here

    // start preview with new settings
    try {
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();

    } catch (Exception e){
        Log.d(TAG, "Error starting camera preview: " + e.getMessage());
    }
}
```

Preview size

- The `surfaceChanged` method is called at least once after its creation
- Then every time there is a change of layout (orientation etc.)
- In `surfaceChanged` you can set the preview size with `setPreviewSize()` using `getSupportedPreviewSizes()`

Preview size

```
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
    ...
    // set preview size and make any resize, rotate or
    // reformatting changes here
     Camera.Parameters myParameters = mCamera.getParameters();
     Camera.Size myBestSize = getBestPreviewSize( w, h, myParameters );

     myParameters.setPreviewSize( myBestSize.width, myBestSize.height );
     mCamera.setParameters( myParameters );

    // start preview with new settings
   ...
}
```

Preview size

```
private Camera.Size getBestPreviewSize(int width, int height, Camera.Parameters parameters)
{
   Camera.Size bestSize = null;
   // for all supported sizes
   for (Camera.Size size : parameters.getSupportedPreviewSizes())
   {
      // if sizes fits the screen size
      if (size.width <= width && size.height <= height)
      {
         // the first one found is automatically the best
         if (bestSize == null)
         {
            bestSize = size;
         }
         else
         {
            // check if this solution cover more area wrt the current best
            int resultArea = bestSize.width * bestSize.height;
            int newArea = size.width * size.height;

            if (newArea > resultArea)
            {
               bestSize = size;
            }
         }
      }
   }
   return bestSize;
}
```

Find the best matched area

Custom camera application

- **Detect and Access Camera**: check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class extending SurfaceView to preview the live feed from the camera.
- **Build a Preview Layout** - Create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for capturing images or videos
- **Capture and Save Files** - Setup the code for saving the output.
- **Release the Camera** - After using the camera, properly release it for use by other applications.

Build preview layout

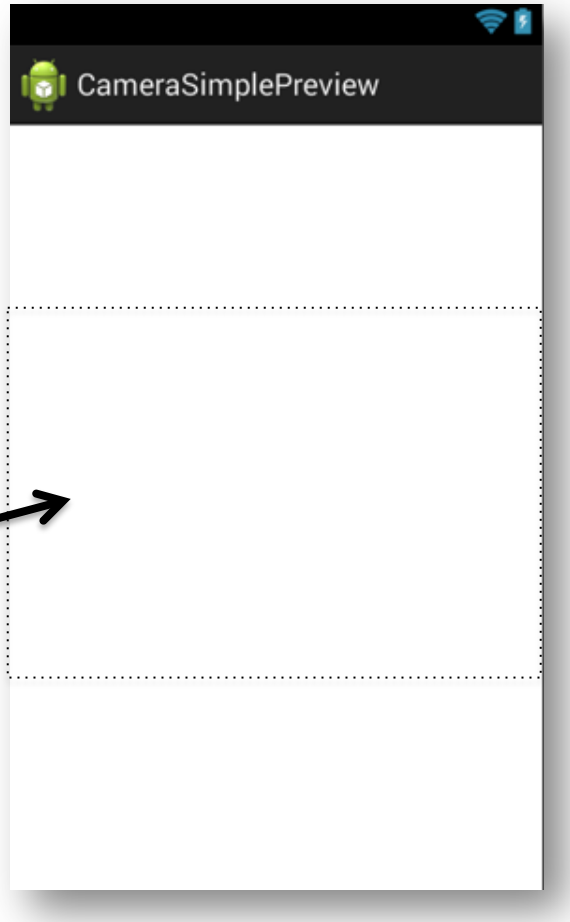
- A camera preview class must be placed in the layout
- The FrameLayout can be the container for the camera preview class.

```
<RelativeLayout
     ...
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${packageName}.${activityClass}" >

    <FrameLayout
        android:id="@+id/cameraPreview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" >

    </FrameLayout>

</RelativeLayout>
```



Force landscape mode

- To force use the landscape mode
- Also no need to stop and restart camera in surface change

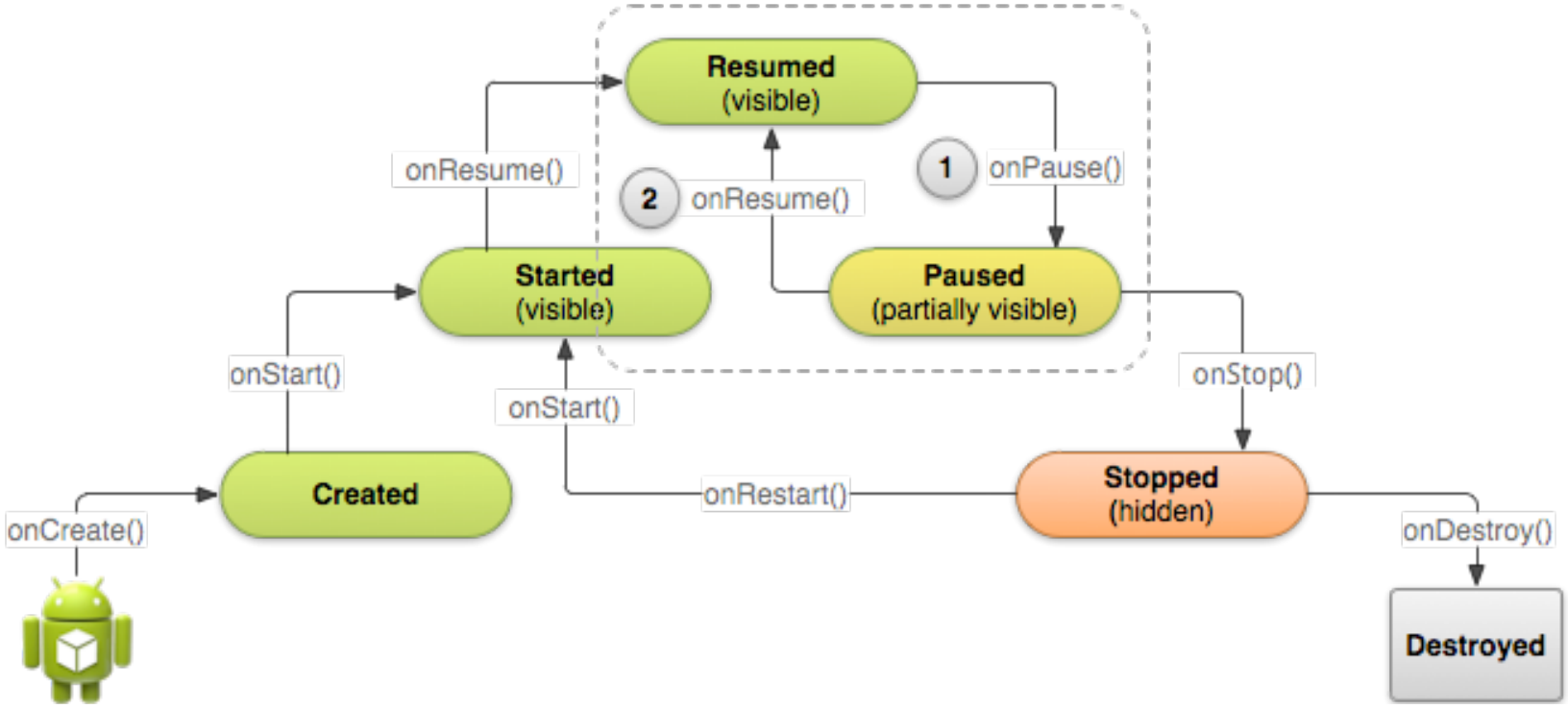
```
<activity android:name=".MainActivity"
      android:label="@string/app_name"
      android:screenOrientation= "landscape">
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
```

Custom camera application

- **Detect and Access Camera**: check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class extending SurfaceView to preview the live feed from the camera.
- **Build a Preview Layout** - Create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for capturing images or videos
- **Capture and Save Files** - Setup the code for saving the output.
- **Release the Camera** - After using the camera, properly release it for use by other applications.

Build preview layout

- In the activity add the preview class to the FrameLayout element
- Activity must also ensure that it **releases the camera** when it is paused or shut down.
- Override:
  - `onResume()` → get the camera and create the `CameraPreview`
  - `onPause()` → release the camera



Build preview layout

```
@Override
public void onCreate( Bundle savedInstanceState )
{
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_main );
}

@Override
protected void onPause()
{
    super.onPause();
    if( mCamera != null )
    {
        mCamera.release(); // release the camera for other applications
        mCamera = null;
    }
}

@Override
protected void onResume()
{
    super.onResume();
    mCamera = getCameraInstance();
    mPreview = new CameraPreview( this, mCamera );
    FrameLayout preview = (FrameLayout) findViewById( R.id.cameraPreview );
    preview.addView( mPreview );
}
```

Custom camera application

- **Detect and Access Camera**: check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class extending SurfaceView to preview the live feed from the camera.
- **Build a Preview Layout** - Create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for capturing images or videos
- **Capture and Save Files** - Setup the code for saving the output.
- **Release the Camera** - After using the camera, properly release it for use by other applications.

Capturing pictures

- Set up listeners for your user interface controls to respond to a user action by taking a picture.
- To retrieve a picture, use the Camera.takePicture() method.
- It takes 3 or 4 parameters which receive data from the camera.
- In order to receive data in a JPEG format, you must implement Camera.PictureCallback interface to receive the image data and write it to a file.

takePicture()

```
public final void takePicture (Camera.ShutterCallback shutter,
Camera.PictureCallback raw, Camera.PictureCallback postview, Camera.PictureCallback
jpeg)
```

- The camera service will initiate a series of callbacks to the application as the image capture progresses.
- The **shutter** callback occurs after the image is captured, ie trigger a sound when image has been captured.
- The **raw** callback occurs when the raw image data is available
- The **postview** callback occurs when a scaled, fully processed postview image is available (NOTE: not all hardware supports this)
- The **jpeg** callback occurs when the compressed image is available

- Only valid when preview is active (after startPreview()). Preview will be stopped after the image is taken; callers must call startPreview() again if they want to re-start preview or take more pictures. This should not be called between start() and stop().

- After calling this method, you must not call startPreview() or take another picture until the JPEG callback has returned.

Camera.PictureCallback

```
private PictureCallback mPicture = new PictureCallback() {

    @Override
    public void onPictureTaken(byte[] data, Camera camera) {

        File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
        if (pictureFile == null){
            Log.d(TAG, "Error creating media file, check storage permissions: " +
                e.getMessage());
            return;
        }

        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        } catch (FileNotFoundException e) {
            Log.d(TAG, "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.d(TAG, "Error accessing file: " + e.getMessage());
        }
    }
};
```

More on this later...

Wire the button

If we add a capture button...

```
// get the button reference
Button captureButton = (Button) findViewById(id.button_capture);

//Add a listener to the Capture button
captureButton.setOnClickListener( new View.OnClickListener() {
     @Override
     public void onClick(View v) {
          // get an image from the camera
          mCamera.takePicture(null, null, mPicture);
          // the same as
         //mCamera.takePicture(null, null, null, mPicture);

     }
 }
);
```

Capturing videos

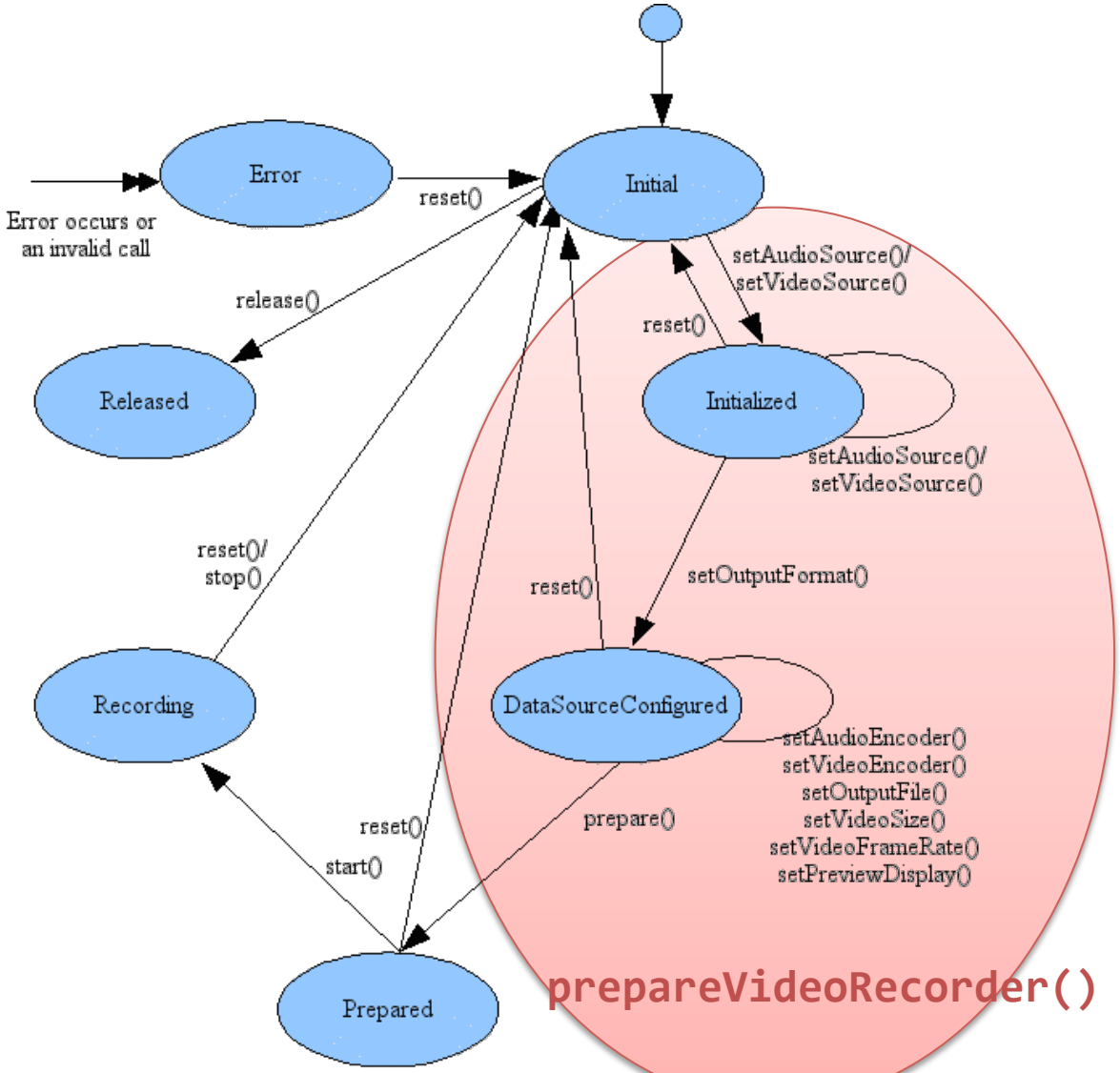
- Video capture using the Android framework requires careful management of the Camera object and coordination with the MediaRecorder class.
- Manage the resource with Camera.lock() and Camera.unlock() to allow MediaRecorder access to the camera hardware
- **Note:** Starting with Android 4.0 (API level 14), the Camera.lock() and Camera.unlock() calls are managed for you automatically.

Using MediaRecorder

1. **Start Recording Video** - The following steps must be completed *in order*:
   a. **Unlock the Camera** - Unlock the camera for use by MediaRecorder by calling Camera.unlock().
   b. **Configure MediaRecorder –** series of calls to setup the MediaRecorder
   c. **Prepare MediaRecorder** - call MediaRecorder.prepare().
   d. **Start MediaRecorder** - Start recording video with MediaRecorder.start().

2. **Stop Recording Video** - Call the following methods *in order*:
   a. **Stop MediaRecorder** - Stop recording video by calling MediaRecorder.stop().
   b. **Reset MediaRecorder –** [*Optional*], remove settings MediaRecorder.reset().
   c. **Release MediaRecorder** - Release it with MediaRecorder.release().
   d. **Lock the Camera** - Lock the camera so that future MediaRecorder sessions can use it by calling Camera.lock().
      - Starting with Android 4.0 (API level 14), this call is not required unless the MediaRecorder.prepare() call fails.

MediaRecorder state machine



MediaRecorder state diagram

Recording the video (1)

```
private boolean isRecording = false;

//Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
     captureButton.setOnClickListener(
      new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              if (! isRecording) {
                  // initialize video camera
                   if (prepareVideoRecorder()) {
                       // Camera is available and unlocked, MediaRecorder is prepared,
                       // now you can start recording
                       mMediaRecorder.start();

                       // inform the user that recording has started
                       setCaptureButtonText("Stop");
                       isRecording = true;
                   } else {
                       // prepare didn't work, release the camera
                       releaseMediaRecorder();
                       // inform user
                   }
               } else {
                    ...
```

Steps 1.a-c

Step 1.d

Recording the video (2)

```
private boolean isRecording = false;

//Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
 new View.OnClickListener() {
     @Override
     public void onClick(View v) {
         if (! isRecording)
         {
              ...
         }
         else
         {    // stop recording and release camera
             mMediaRecorder.stop();  // stop the recording
             mMediaRecorder.reset();   // clear recorder configuration
             mMediaRecorder.release(); // release the recorder object
             mMediaRecorder = null;

             mCamera.lock();         // take camera access back from MediaRecorder

             // inform the user that recording has stopped
             setCaptureButtonText("Capture");
             isRecording = false;
         }
     } });
```

Step 2.a

Steps 2.b-c

Step 2.d

Configure MediaRecorder

1. setCamera() - Set the camera to be used for video capture, use your application's current instance of Camera.
2. setAudioSource() - Set the audio source, use
   - MediaRecorder.AudioSource.DEFAULT.
3. setVideoSource() - Set the video source, use
   - MediaRecorder.VideoSource.CAMERA.
4. setOutputFile() - Set the output file
5. setAudioEncoder() and setVideoEncoder() – Set the a/v encoders
6. setPreviewDisplay() - Specify the SurfaceView preview layout element for your application.

Caution: You must call these MediaRecorder configuration methods *in this order*, otherwise your application will encounter errors and the recording will fail.

Preparing the media recorder

```
private boolean prepareVideoRecorder(){

     mMediaRecorder = new MediaRecorder();

    // Step 1: Unlock and set camera to MediaRecorder
    mCamera.unlock();
    mMediaRecorder.setCamera(mCamera);

    // Step 1: Set sources
    mMediaRecorder.setAudioSource( MediaRecorder.AudioSource.DEFAULT );
    mMediaRecorder.setVideoSource( MediaRecorder.VideoSource.CAMERA );

    // Step 3: Set a output and formats
    mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
    mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
    mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H264);

     // Step 4: Set output file
    mMediaRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toString());

    // Step 5: Set the preview output
    mMediaRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());

    // Step 6: Prepare configured MediaRecorder
    try {
        mMediaRecorder.prepare();
    } catch (IllegalStateException e) { ...
```

Not there yet!

- When we record a video we use the camera to record the images
- But we also use the microphone to record the audio
- The app need the permission for recording the audio!

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

Custom camera application

- **Detect and Access Camera**: check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class extending SurfaceView to preview the live feed from the camera.
- **Build a Preview Layout** - Create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for capturing images or videos
- **Capture and Save Files** - Setup the code for saving the output.
- **Release the Camera** - After using the camera, properly release it for use by other applications.

Saving media files on the SD CARD

Two standard locations for media files

- **Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)**
  - the standard, shared and recommended location for saving pictures and videos.
  - shared (public) directory, other apps read/write/delete
  - media files not removed after your app is uninstalled.
  - Good idea to create a sub-directory for your app

- **Context.getExternalFilesDir(Environment.DIRECTORY_PICTURES)**
  - A standard location for saving pictures and videos
  - media files removed after your app is uninstalled.
  - other applications may read, change and delete them.

Saving media files on the SD CARD

```
public static final int MEDIA_TYPE_IMAGE = 1;
public static final int MEDIA_TYPE_VIDEO = 2;

/** Create a File for saving an image or video */
private File getOutputMediaFile(int type){
    // To be safe, you should check that the SDCard is mounted
    // using Environment.getExternalStorageState() before doing this...

    File mediaStorageDir = new File(Environment.getExternalStoragePublicDirectory(
              Environment.DIRECTORY_PICTURES), "MyCameraApp");

    // Create the storage directory if it does not exist
    if (! mediaStorageDir.exists()){
        if (! mediaStorageDir.mkdirs()){
            Log.d(TAG, "failed to create directory");
            return null;
        }
    }
    // Create a media file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    File mediaFile;
    if (type == MEDIA_TYPE_IMAGE){
        mediaFile = new File(mediaStorageDir.getPath() + File.separator +
        "IMG_"+ timeStamp + ".jpg");
    } else if(type == MEDIA_TYPE_VIDEO) {
        mediaFile = new File(mediaStorageDir.getPath() + File.separator +
        "VID_"+ timeStamp + ".mp4");
    } else {
        return null;
    }
    return mediaFile;
}
```

Checking SD Card availability

```
/** Create a File for saving an image or video */
private static File getOutputMediaFile(int type)
{

    boolean isSDPresent =
        android.os.Environment.getExternalStorageState().equals(
                                        android.os.Environment.MEDIA_MOUNTED);
    if(!isSDPresent)
    {
        int duration = Toast.LENGTH_LONG;

        Toast toast = Toast.makeText(context, "card not mounted", duration);
        toast.show();

        Log.d("ERROR", "Card not mounted");
    }
    else
    {
        File mediaStorageDir ...
    ...
```

Processing the frame of the live feed

- What if we want to process the frame from the live feed?
- Eg. apply some image processing filter etc?
- Install a callback to be invoked for every preview frame

```
mCamera.setPreviewCallbackWithBuffer( new PreviewCallback()
{

    @Override
    public void onPreviewFrame( byte[] data, Camera camera )
    {
        // TODO Auto-generated method stub
        // process the data buffer here
    }
} );
```

From Android >= 2.2

Processing the frame of the live feed

`public static interface` **`Camera.PreviewCallback`**

Public Methods

`public abstract void onPreviewFrame (byte[] data, Camera camera)`

- Called as preview frames are displayed. This callback is invoked on the event thread open(int) was called from.
- If using the YV12 format, refer to the equations in setPreviewFormat(int) for the arrangement of the pixel data in the preview callback buffers.

Parameters

- `data` the contents of the preview frame in the format defined by ImageFormat, which can be queried with getPreviewFormat(). If setPreviewFormat(int) is never called, **the default will be the YCbCr_420_SP** (NV21) format.

Processing the frame of the live feed

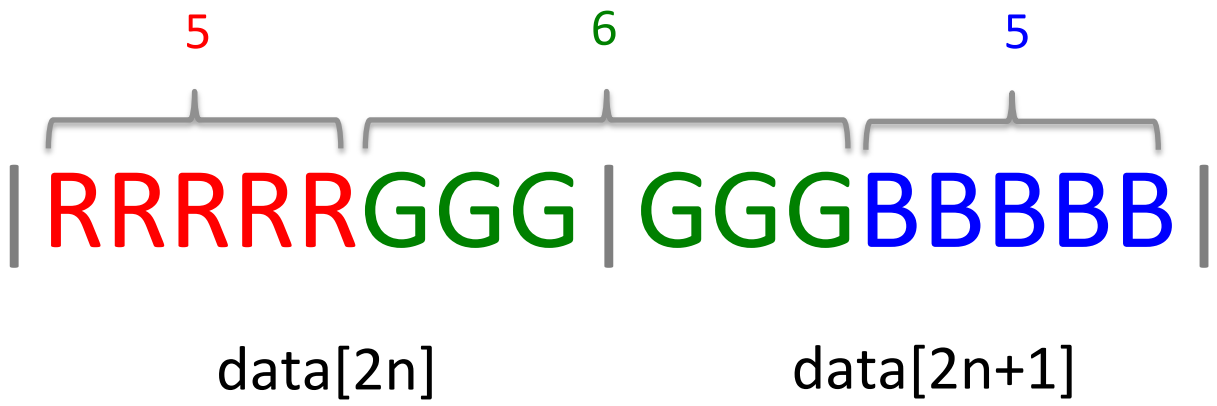
ImageFormat

Constants		
int	JPEG	Compressed JPEG format.
int	NV16	YCbCr format, used for video.
int	NV21	YCrCb format used for images, which uses the NV21 encoding format.
int	RAW10	Android 10-bit raw format This is a single-plane, 10-bit per pixel, densely packed (in each row), unprocessed format, usually representing raw Bayer-pattern images coming from an image sensor.
int	RAW_SENSOR	General raw camera sensor image format, usually representing a single-channel Bayer-mosaic image.
int	RGB_565	RGB format used for pictures encoded as RGB_565.
int	UNKNOWN	
int	YUV_420_888	Multi-plane Android YUV format This format is a generic YCbCr format, capable of describing any 4:2:0 chroma-subsampled planar or semiplanar buffer (but not fully interleaved), with 8 bits per color sample.
int	YUY2	YCbCr format used for images, which uses YUYV (YUY2) encoding format.
int	YV12	Android YUV format.

Processing the frame of the live feed

Example **RGB565**

- 16 bits, 2 bytes per pixel

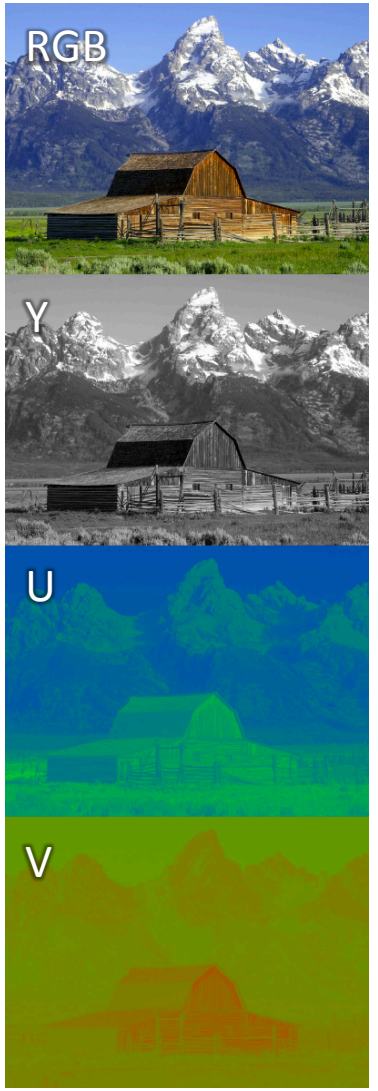


- The image buffer is a 1D array, image lines are stacked together
- n is the “linear” position of the pixel,

Processing the frame of the live feed

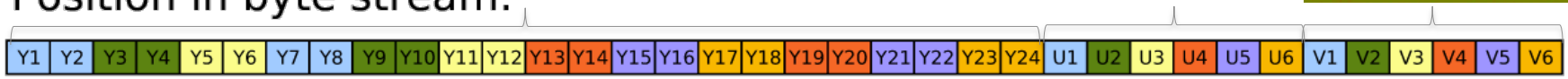
Example **YUV (YCbCr 420)**

- values are grouped together instead of interspersed.
- the image becomes much more compressible.
- Y gives a full resolution greyscale image

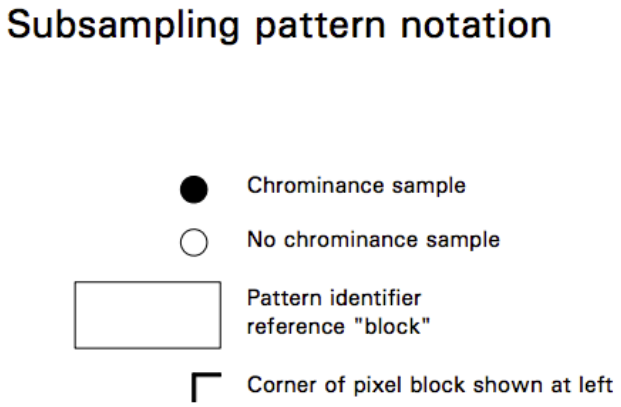


Single Frame YUV420: For a 6x4 image

Position in byte stream:

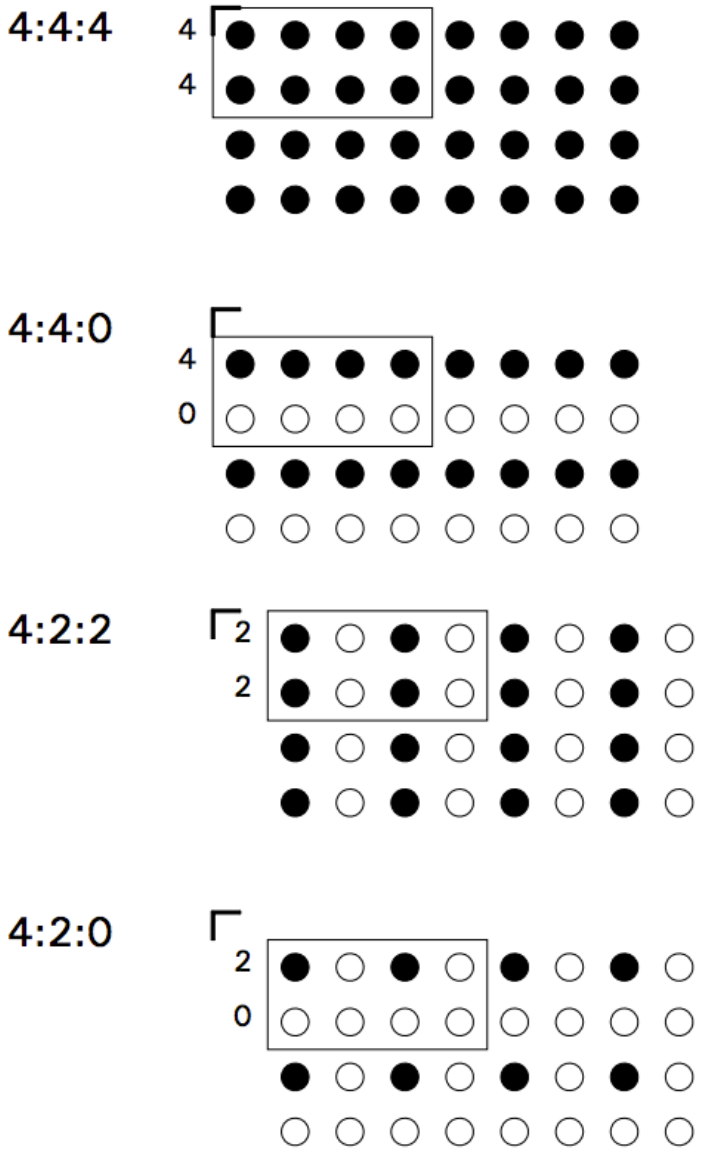


Digression



J:a:b

- **J**: horizontal sampling reference Usually, 4.
- **a**: number of chrominance samples (Cr, Cb) in the first row of J pixels.
- **b**: number of changes of chrominance samples (Cr, Cb) between first and second row of J pixels.



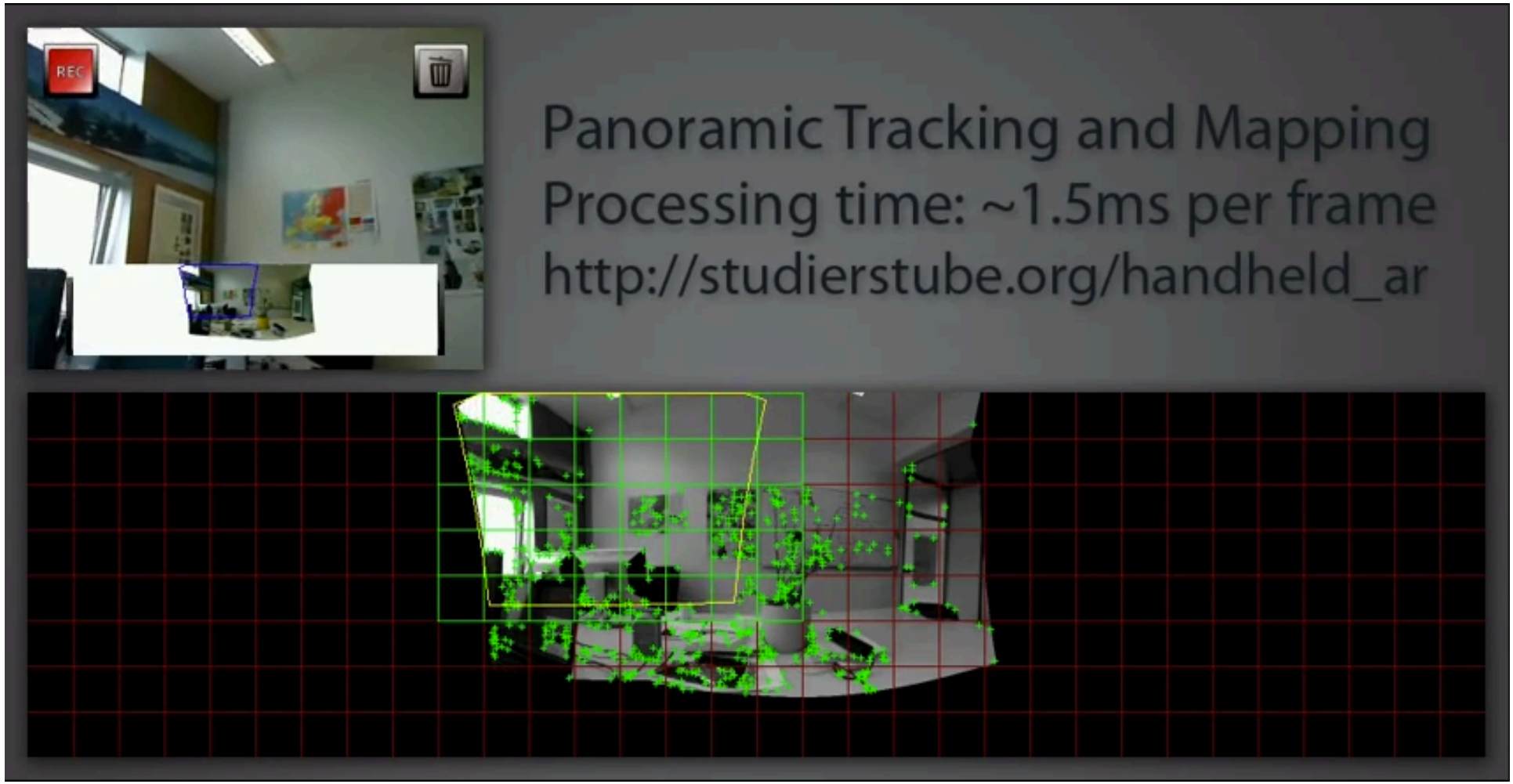
Processing the frame of the live feed

- Usually not a good idea to process the image in Java if real-time is a constraint (for live feed)
- Java on mobile device is still slow for real-time intensive processing
- Use JNI instead
- Use other libraries (eg. OpenCV)

Camera2

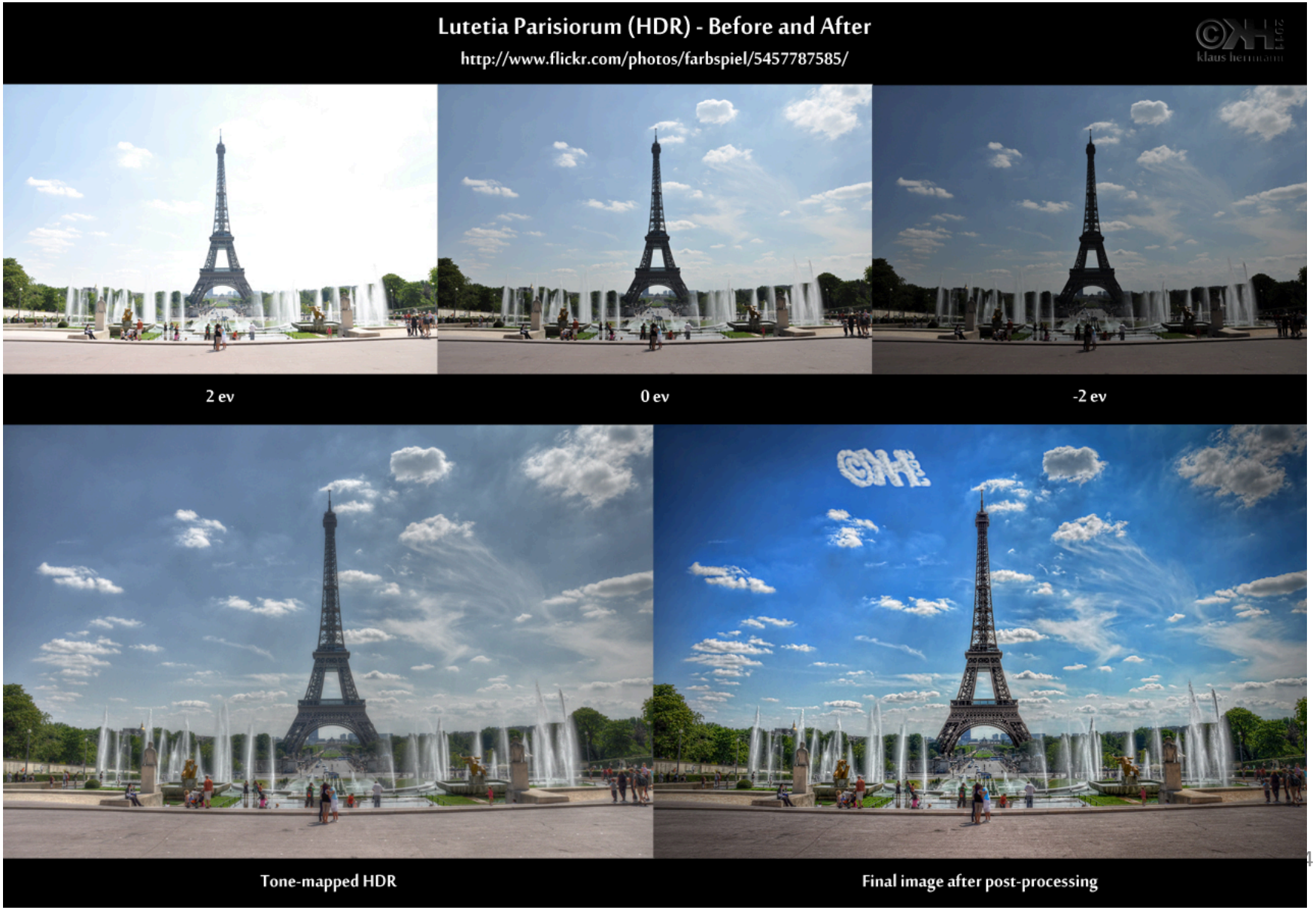
- Starting with Lollipop API 21 a new Camera interface is introduced
- Camera is deprecated
- `Camera2` will be the reference in the future
- Change of paradigm to exploit all the possibility of modern devices and have a finer control over image
- Based on the idea of Frankencamera @Stanford
- **Computational Photography**:
  - Eg. Panorama images, HDR images, depth from focus
  - In general use the full capability of the combination of optical and digital devices: bring the processing directly on the camera chip

Panorama images



http://youtu.be/W-mJG3peIXA

HDR Images


Lutetia Parisiorum (HDR) - Before and After
http://www.flickr.com/photos/farbspiel/5457787585/
klaus hermann
2 ev
0 ev
-2 ev
Tone-mapped HDR
Final image after post-processing

HDR Images


Rien ne va plus! (HDR) - Before and After
http://www.flickr.com/photos/farbspiel/5086360077/
klaus hermann
+2 ev
0 ev
-2 ev
Tone-mapped HDR
Final image after post-processing

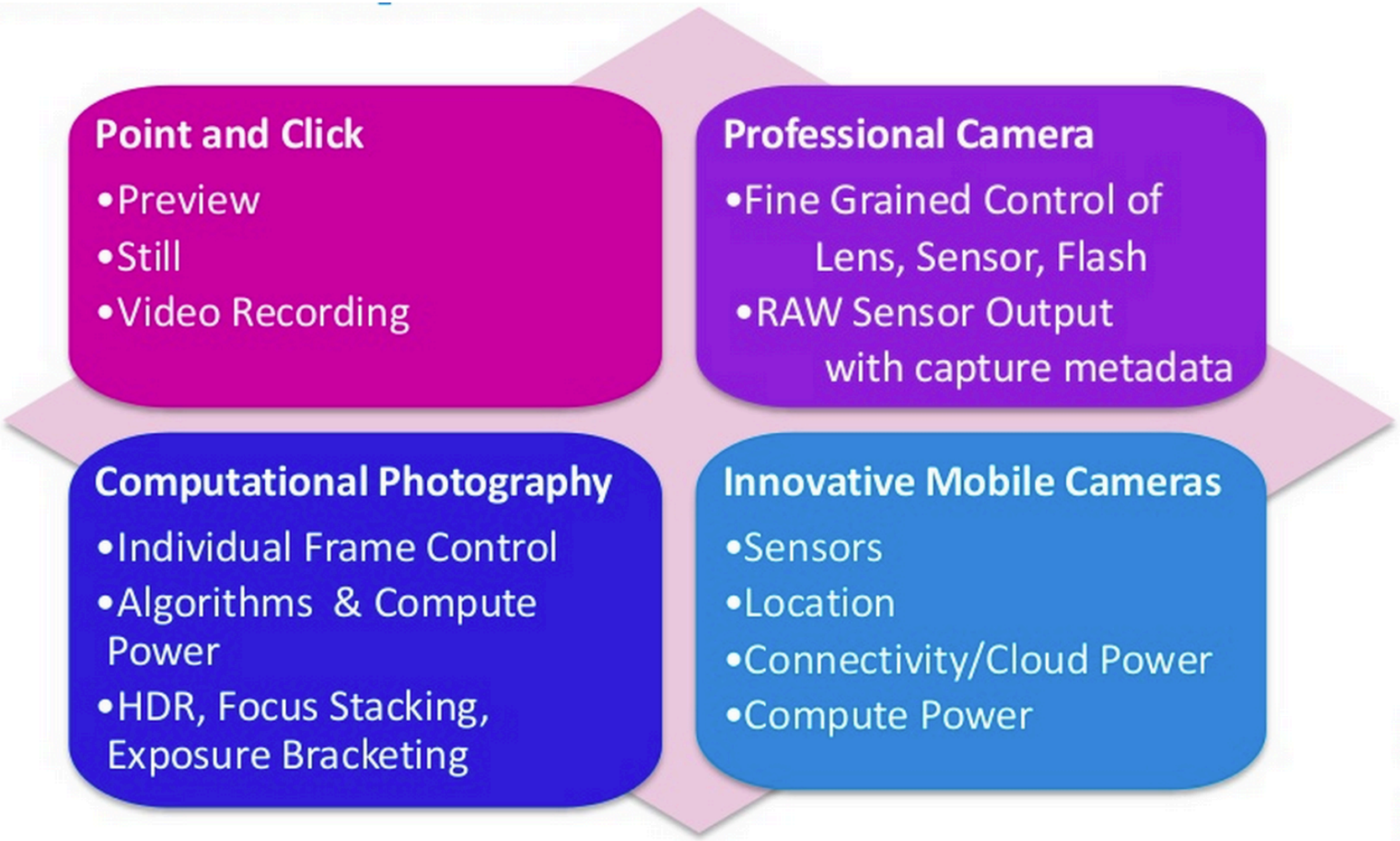
Depth from defocus

Images taken with different focal settings

Estimated depth image

All-focus image

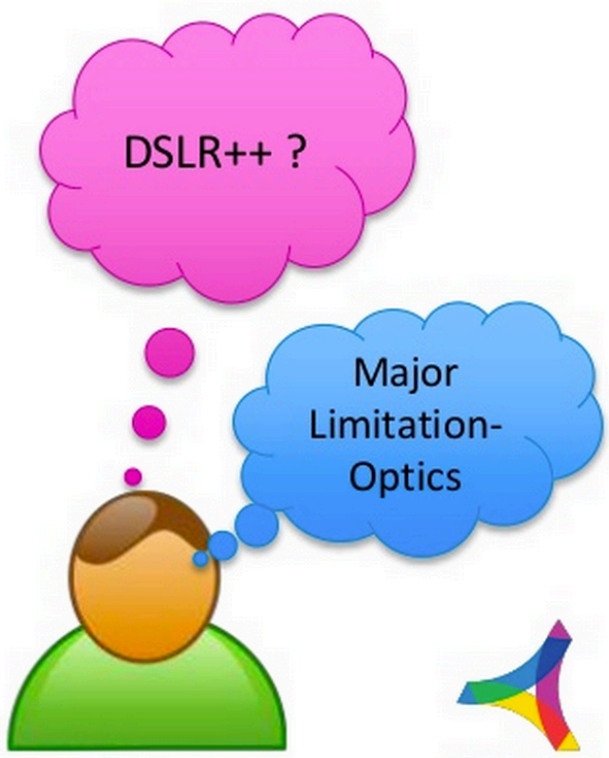
android.hardware.camera2



android.hardware.camera2

- Enables Professional Quality Photography
  - Think DSLR instead of Point-and-Shoot
- Enables Access to RAW Images
- Enables On-device processing of Camera Data
  - High Dynamic Range, Focus Stacking
- Enables New Use Cases combining Imaging with
  - Rich Sensor input
    - Inertial sensors, altitude et. al.
  - Compute Power
    - Multi-core CPU & GPU
  - Connectivity
    - Power of the Cloud & Access to Proximity (BLE, NFC)
  - Context
    - Location & User History

Computational Photography

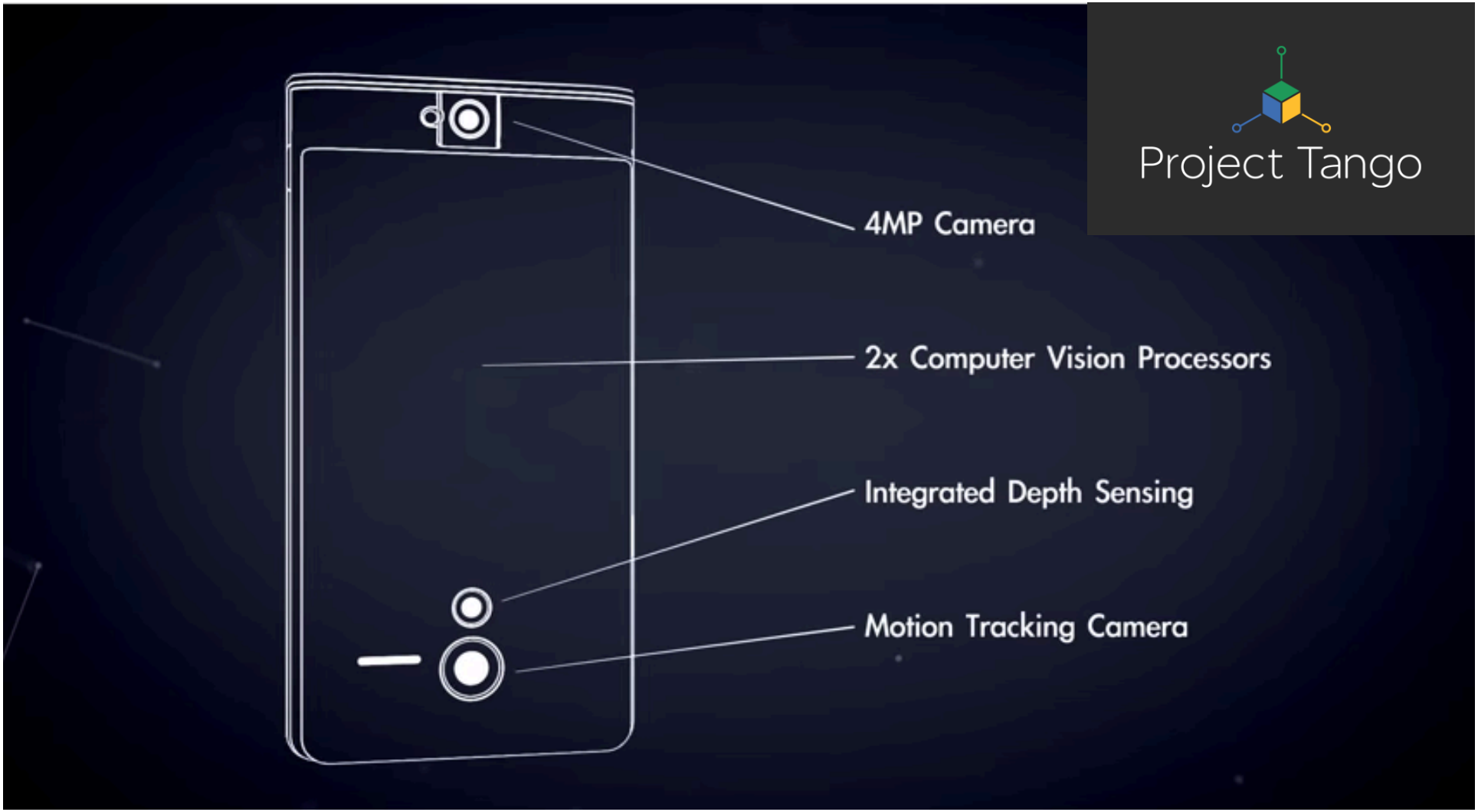


android.hardware.camera2

- Fine grained control of the
  - Sensor
  - Flash
  - Lens
  - Image Signal Processing Pipeline
- Control on a per-frame basis, and deterministic behavior
- Processing Images at full resolution and full frame rate(30 fps)
- Meta data
  - Every frame is returned with the actual settings that it was taken with, and requested for.

Professional Camera

and more to come...



https://www.youtube.com/watch?v=Qe10ExwzCqk

Project Tango Homepage