Utilisation de l’API caméra sous Android

Dans ce document, nous vous expliquerons comment créer une application Android permettant de prendre des photos à partir de la caméra de votre téléphone. Par la suite nous vous expliquerons comment modifier les pixels d’une image.

**Comment prendre une photo :**

Sous Android, il existe la classe Camera qui permet de manipuler le hardware de la caméra de votre téléphone. Mais avant tout chose, vous devez soit utiliser votre téléphone en mode développeur (activer l’ «option pour les développeurs» dans les paramètres de votre smartphone), soit télécharger une machine virtuel (Genymotion). En effet, l’émulateur d’Android Studio et le SDK d’Eclipse ne gèrent pas la caméra. Avec Genymotion, c’est la webcam de votre ordinateur qui simule la caméra de votre smartphone.

Je vous ai parlé de la classe camera (android.hardware.Camera), c’est celle-ci que nous allons aborder par la suite. Mais il faut savoir qu’il existe la classe Camera2 (android.hardware.Camera2). Celle-ci a été créée suite à la mise à jour d’Android fin 2014, pour les nouveaux system Android, et va tendre à remplacer la première.

**Les autorisations :**

Lorsque vous créez le projet, la première chose à faire est de donner les **droits d’accès** à la camera et ses différentes fonctionnalités.

Pour cela, il faut que vous ajoutiez dans le manifeste les permissions d’accès suivantes sans quoi vous ne pourrez pas utiliser l’objet :

<uses-feature android:name="android.hardware.camera" /> accès à la caméra

<uses-feature android:name="android.hardware.camera.autofocus" /> accès à l’option d’autofocus

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

La dernière ligne, ne concerne pas la caméra mais l’espace de stockage. Elle va vous donner la permission d’écrire dans la mémoire du téléphone.

**L’outil Camera Preview :**

L'outil camera preview est un outil java qui comprend les classes SurfaceView, SurfaceHolder et SurfaceTexture qui permettent de visualiser les images de la caméra et de les récupérer.

Si on veut faire des captures d'images en continue, il faut utiliser les méthodes startPreview et stopPreview afin de débuter et de finir la capture en continu. La méthode setPreviewDisplay, elle, permet de définir où nous voulons récupérer les images.

Ces trois méthodes appartiennent à la classe SurfaceHolder.

Sous java, nous pouvons récupérer les images sous la forme de matrices de pixels dans divers formats tels que les formats jpeg ou PNG.

Un SurfaceTexture peut également être utilisé à la place d'un SurfaceHolder lorsque vous spécifiez la destination de sortie de l'API de l'appareil photo. Tout flux d'image doit être envoyé à l'objet de SurfaceTexture plutôt qu’à l'écran de l’appareil.

**Premiers pas avec Camera Preview :**

Afin de tester le code que vous allez faire, il peut être intéressant d’insérer dans le layout un bouton permettant de lancer la prise d’une photo et une SurfaceView permettant d’avoir le point de vue de la camera et d’afficher l’image dans cette SurfaceView.

Votre code sera composé de deux classes. La première sera une fille de l’activité de votre application, elle permettra notamment de gérer la prise la photo. La seconde sera une fille de la classe SurfaceView et elle implémentera l’interface SurfaceHolder.Callback (interface de rappel de SurfaceHoder)

Commençons par étudier la classe ShowCamera. Cette classe a besoin de deux attributs : une SurfaceHolder et un objet Camera.

Il faut ensuite initialiser les objets de type ShowCamera grâce à un constructeur du type :

public ShowCamera (Context context,Camera camera). Un context est une classe renseignant la classe SurfaceView sur l’environnement de l’application. Elle permet également l’accès aux ressources de l’application.

Ce constructeur doit initialiser la classe mère ( SurfaceView), et initialiser les attributs de la classe.

Exemple de code :

Soit les attributs holdMe et theCamera respectivement de type SurfaceHolder et Camera, le constructeur peut être :

public ShowCamera(Context context,Camera camera) {

super(context);

theCamera = camera;

holdMe = getHolder();

holdMe.addCallback(this);

}

Ensuite, il ne vous reste plus qu’à implémenter les trois méthodes de l’interface SurfaceHolder.Callbak qui sont:

public void surfaceCreated(SurfaceHolder holder) ;

public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) ;

// width est la largeur et height la hauteur de l’image

public void surfaceDestroyed(SurfaceHolder holder) ;

Exemple d’implémentation:

public void surfaceCreated(SurfaceHolder holder) {

try {

theCamera.setPreviewDisplay(holder);

theCamera.startPreview();

} catch (IOException e) {

System.out.println(e.setStackTrace());

}

}

public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {

Camera.Parameters parameters = theCamera.getParameters();

parameters.setPreviewSize(width, height);

theCamera.setParameters(parameters);

theCamera.startPreview();

}

public void surfaceDestroyed(SurfaceHolder holder) {

theCamera.stopPreview();

theCamera = null;

}

Dans la classe fille de votre activité, vous remplissez les méthodes concernant l’activité de votre application comme pour les autres. Cependant dans la méthode onCreated vous initialisez des attributs (que vous aurez au préalable déclarés) de types Camera et ShowCamera.

L’objet de type Camera, nous pouvons l’initialiser grâce à une méthode retournant une Camera qui l’ouvre et qui lance la fonction startPreview() ;

Vous pouvez ensuite créer une méthode qui appelle la méthode takePicture(Camer.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg) de votre objet Camera. C’est cette méthode qui va déclencher la prise de photo. **Attention**, lors de l’appel à cette méthode la preview de la caméra s’arrête, il faudra donc la redémarrer. Les arguments appellent des méthodes de rappel pour récupérer les images.

Maintenant, il faut donc créer une méthode de rappel. Dans cette méthode il faut que vous récupériez les pixels qui sont en rgb dans un Bitmap grâce à la méthode BitmapFactory. Dans cette méthode vous pouvez également redémarrer la preview de la caméra.

Exemple de code :

public class Smart extends ActionBarActivity {

private Camera cameraObject;

private ShowCamera showCamera;

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_smart);

cameraObject = isCameraAvailiable();

showCamera = new ShowCamera(this, cameraObject);

}

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.menu\_smart, menu);

return true;

}

public boolean onOptionsItemSelected(MenuItem item) {

int id = item.getItemId();

if (id == R.id.action\_settings) {

return true;

}

return super.onOptionsItemSelected(item);

}

// Méthode de rappel afin de récupérer les pixels de l’image dans // un bitmap

private PictureCallback capturedIt = new PictureCallback() {

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

Bitmap btm = BitmapFactory.decodeByteArray(data, 0, data.length);

bitmap = btm;

if (bitmap == null) {

Toast.makeText(getApplicationContext(), "not taken", Toast.LENGTH\_SHORT).show();

} else {

Toast.makeText(getApplicationContext(), "taken", Toast.LENGTH\_SHORT).show();

}

cameraObject.startPreview();

}

};

// initialisation de la camera

public static Camera isCameraAvailiable() {

Camera object = null;

try {

object = Camera.open();

object.startPreview();

} catch (Exception e) {

}

return object;

}

public void takePicture(View view) {

cameraObject.takePicture(null, null, capturedIt);

// **Attention la preview de la camera s’arrête**

}

}

Afin de tester la prise de l’image, vous pouvez créer une méthode permettant la sauvegarde de l’image grâce à la méthode compress(Bitmap.CompressFormat format, int quality, OutputStream outputStream) des Bitmaps (exemple : bitmap.compress(Bitmap.CompressFormat.PNG, 100, fos) avec bitmap le nom du Bitmap que vous voulez enregistrer.

**Manipulation des Pixels:**

Pour manipuler les pixels les objets bitmaps possèdent les méthodes getPixel(int x, int y) et setPixel(int x, int y, Color color) ou getPixels(int[] pixels, int offset, int stride, int x ,int y, int width, int height) et setPixels(int[] pixels, int offset, int stride, int x ,int y, int width, int height) qui eux prennent en argument un tableaux . Color est la classe permettant de changer la couleur du pixel. Mais attention si vous utiliser setPixel ou setPixels, il faut que le Bitmap soit « mutable » (modifiable). Si vous essayez de modifier des valeurs d’un bitmap « immutable », une IllegalStationException est renvoyée. Pour cela, vous pouvez utiliser la méthode copy de la classe Bitmap (renvoie un Bitmap) prenant en argument une config de bitmap (souvent celle du bitmap de départ), qui sera celle du nouveau bitmap, et un booléen qui est true si on veut que le Bitmap renvoyé soit « mutable ».

**Paramètre de l’objet Camera :**

Vous pouvez obtenir et modifier les paramètres des objets de type Camera grâce aux méthodes de la classe Camera.Parametre. Vous pouvez par exemple modifier les paramètres de l’autofocus, du flash, du zoom. Mais aussi, vous pouvez modifier les paramètres de la qualité d’image, voire les paramètres des formats de la preview (NV1 ou YUV1 formats recommandés).

Cependant, nous ne pouvons pas modifier la fréquence de la preview et pour Camera2 c’est la même chose.