Application à un système vidéo (suite)

Romain Raveaux

Sommaire :

But du TP 1
Travail à faire 1

But du TP

1°) Mise en œuvre des appels JNI (Java Native Interface)
2°) Effectuer un traitement d'image (filtrage) en avec une implémentation en C++
3°) Monitorer l'exécution en calculant le nombre de Frame Per Second

Travail à faire

NDK est un compilateur C/C++ pour le système d'exploitation Android.

3°) Reprenez le projet du TP précédent.
4°) Créer un répertoire nommé 'jni' à la racine de votre projet.
5°) Créer un répertoire nommé 'libs' à la racine de votre projet.
6°) Pour NDK, créer un fichier de configuration s'appelant Application.mk dans votre répertoire « jni ». Ce fichier doit contenir les informations suivantes :

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi-v7a
```

APP_ABI correspond à l'architecture cible, arm ou x86. *armeabi* correspond à une architecture ARM bi-processeurs. *V7a* correspond au type d'architecture ARM, ici un cortex A7.

7°) Pour NDK, créer un fichier de configuration s'appelant Android.mk dans votre répertoire « jni ». Ce fichier doit contenir les informations suivantes :

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)


LOCAL_MODULE    := native_sample
LOCAL_SRC_FILES := jni_part.cpp
LOCAL_LDLIBS +=  -llog -ldl

include $(BUILD_SHARED_LIBRARY)
```

LOCAL_MOULDE : le nom de la librairie qui sera générée.
LOCAL_SRC_FILES : le nom des fichiers c/cpp à compiler.
LOCAL_LDLIBS : Les dépendances…. Libraires pour les logs dans notre cas.
8°) Dans votre répertoire jni, créer un fichier cpp, **jni_part.cpp**
9°) Voici la souche de votre fichier :

```
#include <jni.h>
#include <math.h>
```

```
using namespace std;


extern "C" {
	JNIEXPORT void JNICALL Java_epu_android_FrameProcessing_ProcessFast(JNIEnv* env,
	jobject thiz, jint width, jint height, jbyteArray data, jbyteArray out)
	{
	    jbyte* _data  = env->GetByteArrayElements(data, 0);
	    jbyte* _out  = env->GetByteArrayElements(out, 0);


	//stuf todo here

	    env->ReleaseByteArrayElements(data, _data, 0);
	    env->ReleaseByteArrayElements(out, _out, 0);

	}

}
```

Le nom de la fonction est très long, il correspond à une syntaxe précise :

```
JNIEXPORT void JNICALL Java_nom des packages_nom de la classe_nom de la fonction
```

`JNIEnv* env` :

Correspond à l'environnement JNI permettant certaines conversions de types JAVA vers C/C++

`jobject thiz`

Correspond à la classe qui a appelé la méthode.

`jint width, jint height`

Largeur et Hauteur de l'image

`jbyteArray data`

Le tableau de données issues de la caméra

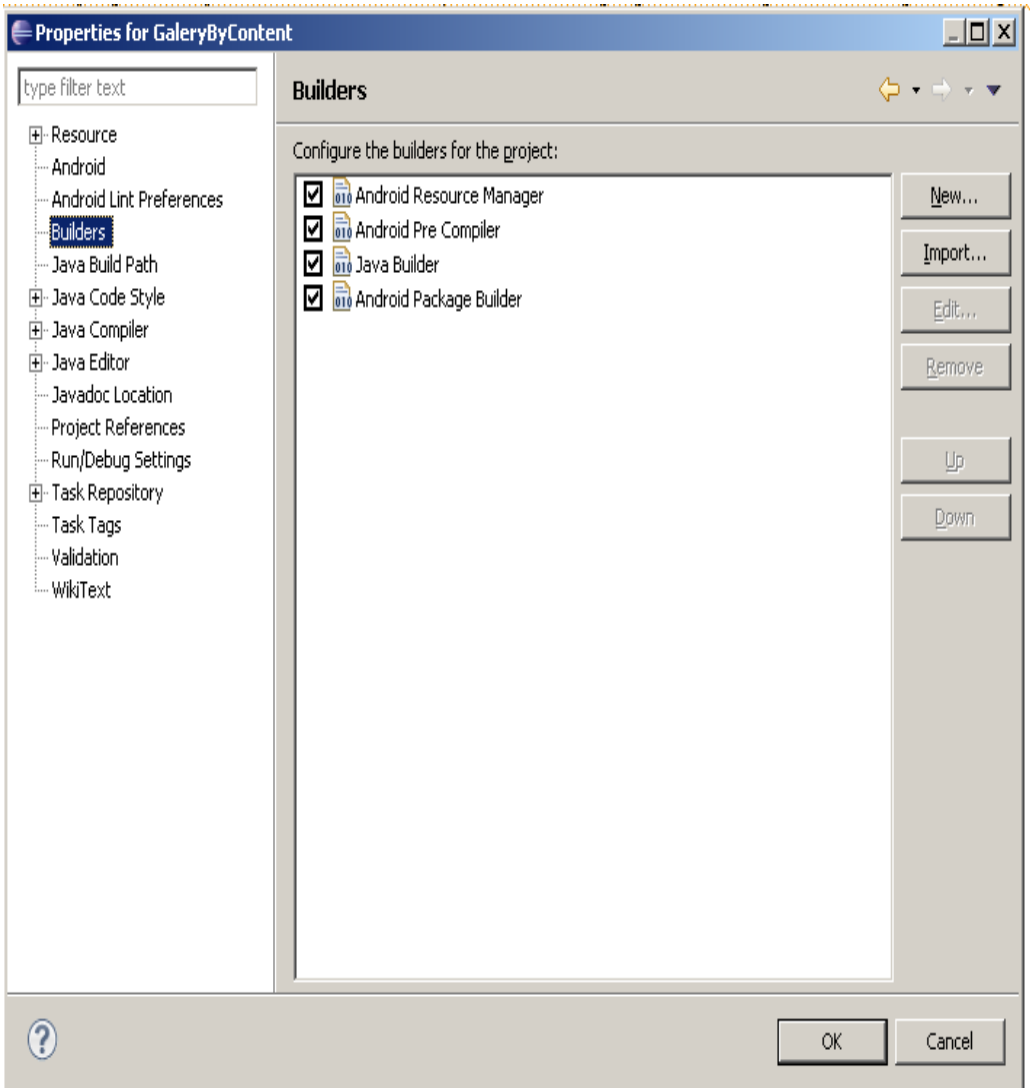
Les appels suivants :

```
jbyte* _data  = env->GetByteArrayElements(data, 0);
```

Ces appels servent à obtenir des pointeurs sur les objets issus du framework JAVA (dans le but de les modifier). A la fin il faut libérer les objets alloués :

```
env->ReleaseIntArrayElements(data, _data, 0);
```

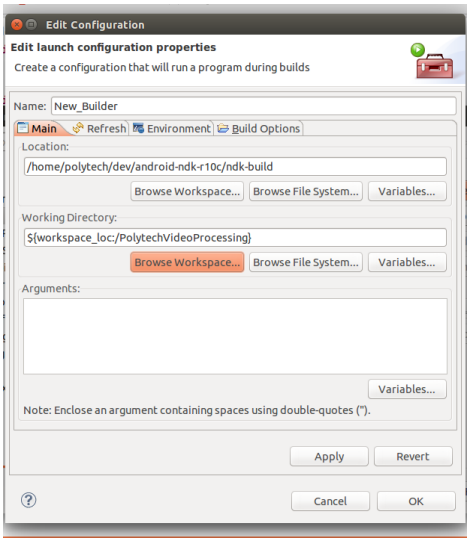
10°) Quelle est le nom de la méthode dans notre cas ?
11°) Quelle est le nom la classe possédant cette méthode ?
12°) Modifier le nom de la méthode JNI afin de correspondre à votre arborescence de projet.
13°) Pour compiler le code C/C++, créer un nouveau Builder pour votre projet.
Clic droit sur votre projet. Ensuite Propriété du projet :



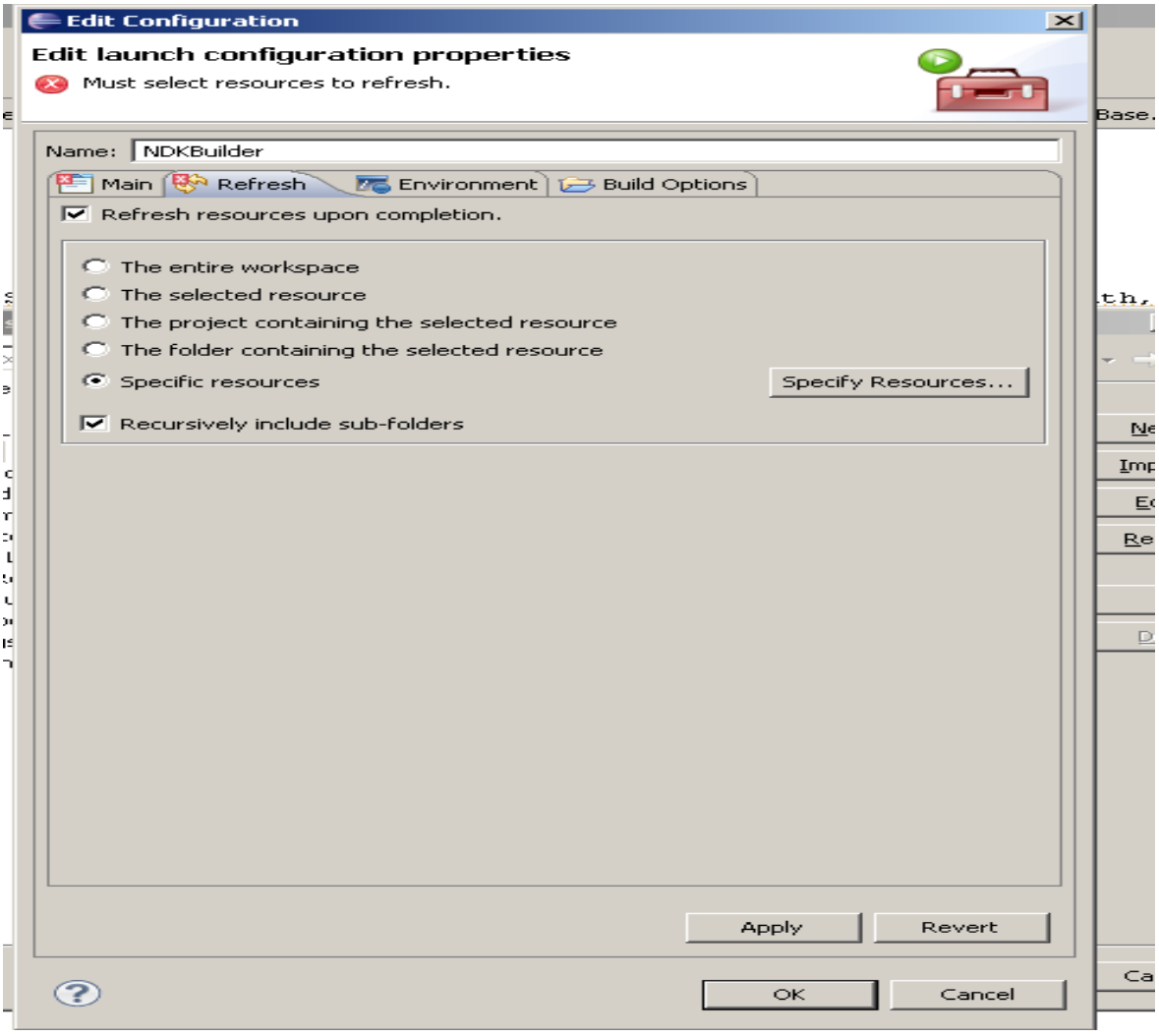
- New
- Configuration de type « program »
- Configurer votre Builder comme suit :

    Le champ Location est égal à <repertoire NDK>/NDKBuilder

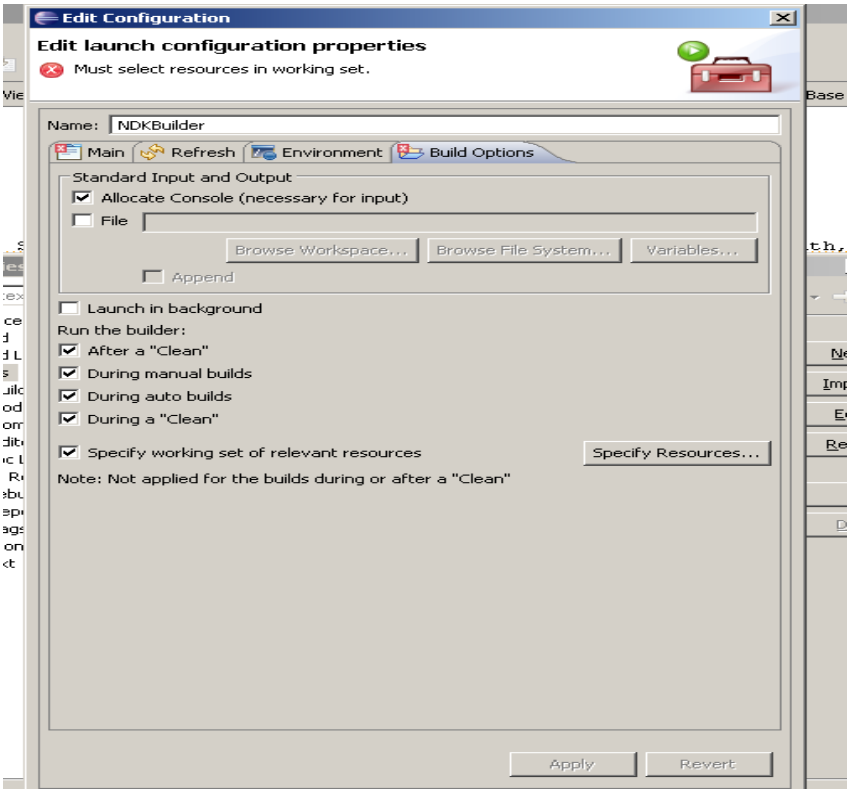
    Le nom du builder sera NDKBuilder



Ensuite passer à l'onglet Refresh :



- Cliquer sur *Specify Ressources* et choisissez le répertoire « libs ».
- Cliquer sur l'onglet Build Option :



- Cliquer sur les cases à cocher comme dans l'image ci-dessus.
- Cliquer sur Specify Ressources et choisissez le répertoire JNI.
- Voilà votre compilateur NDK est prêt.

15°) Dans votre classe de traitement vidéo, placer le code suivant :

```
public native void ProcessFast(int width, int height, byte input[], byte output[]);

static {
        System.loadLibrary("native_sample");
}
```

16°) Que fait le code ci-dessus ?

17°) Dans la fonction onCameraFrame, commenter le code JAVA écrit précédemment (dans le TP précédent) et faites un appel à la fonction ProcessFast.

19°) Compiler et exécuter sur votre tablette/emulateur.

20°) Que conclure sur le nombre de Frame par seconde ?

21°) ajouter le code C/C++ dans la méthode ProcessFast permettant de calculer les gradient de l'image ? Aidez vous du code JAVA écrit au TP précédent.

22°) Compiler et exécuter sur un émulateur/tablette.

23°) Que conclure sur le nombre de Frame par seconde ?

24°) Au lieu de calculer le gradient calculer le filtre de Sobel.

25°) Que conclure sur le nombre de Frame par seconde ?