HOW TO native code with Android

Installation SDK :

La doc d'android est pas mal faite et tous les sujets sont couverts. Je ne vais pas tout réécrire mais donner des astuces sur les points bloquants et les concepts.

Cela se passe ici pour obtenir le SDK et eclipse :

http://developer.android.com/sdk/index.html

Puis les détails de l'installation :

http://developer.android.com/sdk/installing.html

Le plugin ADT et Eclipse :

Eclipse est un IDE écris en Java qui supporte l'ajout de plugin dont ADT (Android Development Tools) Ce plugin permet de lancer la compilation de programme java pour Android (car ce n'est pas du Java mais un dérivée utilisant la machine virtuel Dalvik). Si je dis des âneries pendant ce tuto arrêtez moi.

Android et la compatibilité descendante – ascendante

C'est là qu'Android est tout simplement génial (et prise de tête aussi). On vous demande pendant l'installation de choisir quelle plateforme de compilation téléchargée. Je recommande de télécharger ceux par défaut proposée lors de l'installation avec l'installeur windows (le dernie sdk tools, sdk platform-tools, sdk platform). Mais aussi de prendre le sdk platform de la version cible de votre programme Android. On utilise l'outil Android SDK manager présent dès l'install.

Explication : lorsque l'on compile une application on définit 2 params. Le « min SDK version » qui correspond à la platforme minimale sur laquelle l'application fonctionne et le « target SDK version » qui correspond à la plateforme de compilation (et donc testé éprouvé). On peut définir aussi un « max SDK version » mais ce n'est pas recommandé sauf si l'on souhaite par exemple faire une version pour les 2.x et une autre pour les 3.x ou 4.x.

Ensuite si vous souhaitez tester via l'émulateur vous pouvez créer un AVD via le AVD manager présent dans Eclipse. On peux ne pas trouver un AVD fraichement créé lors du lancement de l'appli si la bonnes version des tools correpsondant à la version de compilation n'est pas présent. Bref retour au SDK manager.

Android NDK (support du langage natif)

Android supporte le C++ mais il faut un peu bidouiller, rassurez-vous beaucoup moins qu'au début d'Android. Pour information on peux écrire du tout natif depuis la v2.3 d'Android mais du coup seul les possesseurs > 2.3 peuvent utiliser ces applications.

Donc pour ratisser plus large je préconise d'écrire une classe Java héritant d'Activity. Cette classe lance alors le thread de rendu. Je reviens dessus un peu plus loin.

Installation :

http://developer.android.com/sdk/ndk/index.html

Suivre les instructions sauf que vous n'avez pas besoin d'aller créer les répertoires à la main. Nous allons rajouter un plugin : Sequoyah project donc voici le lien vers la page expliquant comment installer et configurer le debug pas à pas du code natif. J'en reparle plus loin mais installé déjà le plugin

http://www.eclipse.org/sequoyah/documentation/native_debug.php

url du plugin 1.0.1 : Sequoyah - http://download.eclipse.org/sequoyah/updates/1.0.1/

Je n'ai pas testé avec les nouvelles versions (actuellement en 2.0.1):
http://www.eclipse.org/sequoyah/downloads/index.php

Créer un projet Android. Puis click droit sur le projet et Add Native Support. Voilà le projet supporte le code natif, ou presque.

Un bug apparait et il faut rajouter à la main le fichier default.properties à la racine du projet. Copier coller le fichier projet.properties qui contient la version target d'Android. Si vous ne le faites pas vous aurez une erreur d'inclusion des fichiers opengl es. (rappelez-vous en ca peut éviter des heures de recherche)

Un dossier jni est donc créé à la racine du projet. Jni signifie Java Native Interface et cela permet de faire communiquer le Java et un autre langage comme le C/C++. Il faut utiliser du code C un peu spécial dans un fichier d'interface (par défaut /jni/monprojet.c). je vous laisse voir les exemples dans le NDK, le hello-jni et le san-angleles devrait permettre de faire un rendu opengl rapidement.

Compatibilité :

Attention on ne peut pas écrire du code natif pour gérer l'audio avant Android 2.3. Donc l'astuce consiste à appeler via une fonction JNI la fonction dans l'activity principale qui lance un son. Bémol on ne peux pas accèder directement à l'activity mais il faut faire comme dans les exemples et ajouter une classe Java static qui va charger la lib C et qui contient les déclarations de fonctions native et des appels C->Java.

Un peu de doc sur jni :

http://developer.android.com/guide/practices/design/jni.html

http://m.heid.free.fr/expander/jni/jni.html

http://www.up-comp.com/french/ressources/normes/services/java/callobjectmethod/index.html

http://java.sun.com/docs/books/jni/html/jniTOC.html

Autre problème la lecture des fichiers tels les sauvegardes, les images se fait aussi depuis le code Java si on veut supporter les Android < 2.3.

Astuces : lorsque l'on veut enregistrer un fichier de sauvegarde on veux qu'il soit dans un répertoire Android particulier. C'est un répertoire protégé du système. Je donne le code :

```
FileOutputStream fos = mAct.openFileOutput(_filename.substring(5),
Context.MODE_PRIVATE);
fos.write(_bt);
fos.close();
```

Le Context permet de spécifier que c'est dans ce dossier. Voir la doc pour plus d'infos. mAct est mon activity dont j'ai passé au préalable la référence

Il faudra donc pour la lecture soit utiliser notre activity pour accéder au fichier de sauvegarde soit utiliser ce que l'on appel le AssetManager.

Les assets :

On a un dossier assets/ dans le projet. Il sert à mettre les fichiers utiles pour l'application. Les fichiers s'utilise différemment de ceux présent dans res/. Ils sont plus prévus pour justement les appli native.

Pour aller lire le contenu des assets il faut utiliser le AssetManager et récupérer les bytes du fichier et renvoyer au code C. Puis faire une conversion pour que ce soit compréhensible par le C.

Ou pour jouer un audio utiliser les AssetFileDescriptor et utiliser la fonction MediaPlayer ::setDataSource (puisque l'on gère en Java).

http://developer.android.com/reference/android/content/res/AssetFileDescriptor.html

http://developer.android.com/reference/android/content/res/AssetManager.html

Le debug pas à pas du code natif depuis eclipse : Beurk

Alors là je déconseille pour m'y être frotté. La configuration du debug pas à pas du code natif est super compliqué et très très très lent.

Voici la marche à suivre : http://www.eclipse.org/sequoyah/documentation/native_debug.php

Peut-être qu'avec la dernière version de Sequoyah les choses sont meilleures (v2.0.1)

http://www.eclipse.org/sequoyah/index.php#components

Cycle de vie :

Une application Android utilise une classe dérivant de Activity qui gère le multitâche. Il faut donc gérer le cycle de vie de l'activité si l'on lance des thread, si l'on utilise des données à sauvegarder…

Attention le context opengl n'est pas sauvegarder lorsqu'une activity prend le focus. Il faut alors recharger toutes les textures avant de les réutiliser.

Il faut lire ces pages pour connaître les bases:
http://developer.android.com/guide/topics/fundamentals.html

Ensuite pour l'activité cette page :
http://developer.android.com/reference/android/app/Activity.html