Initiation à la programmation COFFEE. (partie 1.)

Par Tengaal (tengaal@libertysurf.fr) 18/03/2002.

Présentation

Le langage COFFEE est un langage propre à Cinéma 4D, permettant de créer des scripts personnels pour simplifier des tâches, créer des outils particuliers, animer des objets… La documentation sur toutes les fonctions COFFEE et leur syntaxe se trouve sur le site www.plugincafe.com dans la rubrique « Developpers Centers » puis sur le lien « Download SDK », on y trouve les différents SDK pour programmer des plugins avec un compilateur extérieur à Cinema 4D, et la documentation COFFEE pour la programmation des expressions et plugins dans Cinema 4D.

Les bases.

Pour commencer, on va parler de la structure d'un programme COFFEE, pour cela, lancez Cinema 4D, créez un objet Neutre puis appliquez lui une expression COFFEE. Si vous ouvrez ce tag, une fenêtre s'ouvre et vous avez le texte suivant :

```
main(doc,op)
{

}
```

Comme en C++, le programme est constitué de une ou plusieurs fonctions, la fonction « main » est la fonction principale executée en priorité, elle constitue le noyaux de votre programme.

Dans le cas du COFFEE, « main » reçoit deux paramètres donnés par C4D, « doc » représente l'objet qui donne accès au projet (ou document) et « op » représente l'objet auquel est associée l'expression COFFEE que vous êtes en train de créer. (On peut bien sûr changer le nom de ces paramètres.)

Enfin, les accolades « {« et « } » sont utilisées pour définir le début et la fin d'une suite d'instructions, chaque instruction ou commande se termine par un point-virgule « ; ».

Voilà pour l'approche, maintenant nous allons parler de l'utilisation des variables.
Pour information, on peut placer des commentaires dans un programme COFFEE à condition d'utiliser « // » devant la ligne de commentaire ou bien « /* (plusieurs lignes de commentaires) */ » quand on a plusieurs lignes successives de commentaires encadrées avec « /* » et « */ ».

Voici un exemple de commentaires dans une expression COFFEE:

```
/* Ce script COFFEE est un exemple
   d'initiation pour la programmation
   d'expressions et de plugins dans
   Cinema 4D.
*/

main(doc,op)
{
   // on place le programme ici.
}
```

Les variables en COFFEE

Deux types de variables sont utilisables en COFFEE, les constantes et les variables.

Les constantes sont des valeurs fixées définitivement et non modifiables par le programme, pour créer une valeur de ce type, on utilise la commande « const var »

Les variables sont des valeurs qui peuvent être changées par le programme, suite à une opération mathématique, ou bien pour stocker une valeur temporairement. Une variable se définie avec la commande « var ».

Exemple :

```
const var MaValeur=1234 ;          // la constante MaValeur vaut 1234
var MaVariable ;                   // la variable MaVariable est créée.

main(doc,op)
{
  MaVariable=10 * MaValeur – 50 ;  //maintenant MaVariable vaut 12290.
}
```

Dans le cas des variables, on peut les créer sans leur donner de valeur initiale ou bien comme pour une constante, on peut leur donner une valeur au moment de leur création.

Exemple : var MaVariable=120 ; // on créé MaVariable puis on l'initialise à 120.

Bien sûr, les variables et les constantes ne servent pas seulement à stocker des valeurs numériques, elles peuvent représenter des objets, des vecteurs (position, rotation et taille), des propriétés d'objets…

NB : Les variables définies hors des fonctions comme dans l'exemple précédent, seront accessibles par toutes les fonctions, par contre on peut utiliser des variables locales utilisables uniquement à l'intérieur d'une fonction en créant ces variables dans la fonction.

Exemple :

```
const var MaConstante=20 ;     // Création d'une constante
var VariablePublic ;           // Création d'une variable publique


MaFonction()                    // Définition d'une fonction personnelle sans paramètres
{
 var Variable ;                // Cette variable n'est utilisable que dans MaFonction()
 VariableLocale=10*VariablePublic ;
 return VariableLocale ;       // On renvoit le résultat à la commande qui a appelée MaFonction
}


main(doc,op)
{
 var Variable ;                          // Cette variable n'est utilisable que dans main(doc,op)
 VariablePublique=2*MaConstante ;        // On modifie VariablePublique
 VariableLocale=MaFonction() ;           // VariableLocale contient le résultat de MaFonction()
}
```

Dans cet exemple, Cinema 4D va commencer par créer MaConstante et VariablePublique puis va aller directement à **main(doc,op)**. Là, on modifie VariablePublique puis on appelle **MaFonction()** qui utilise elle aussi VariablePublique pour calculer sa valeur locale VariableLocale renvoyée comme résultat avec la commande « return ».
Voilà pour la création et l'utilisation des variables en COFFEE, passons maintenant à la création des fonctions personnelles.
Les différents types de variables seront exposées et expliquées dans une prochaine partie.

Création de fonctions personnelles

Comme on l'a vu précédemment, Cinema 4D exécute toujours la fonction **main(doc.op)** d'un programme COFFEE, et c'est donc à l'intérieur de ce noyau qu'on peut appeler des fonctions personnelles.

Ces fonctions doivent être placées après la définition des variables publiques et avant la fonction **main(doc,op)** qui par conséquent, doit être la dernière fonction du programme COFFEE. Voici un exemple schématique de la structure d'un programme COFFEE :

Définitions des constantes et variables publiques (« const var » et « var ».)

Définition des fonctions perso.

```
Fonction1(paramètres_reçus)
{
  // ici le programme de la fonction1.
  return réponse ;
}

Fonction2(paramètres_reçus)
{
  // ici le programme de la fonction2.
  return réponse ;
}

(etc…)
```

Fonction principale

```
main(doc,op)
{
   appel1=Fonction1(paramètres_transmis);
   appel2=Fonction2(paramètres_transmis) ;
}
```

Structure et définition d'une fonction :

```
NomFonction(p1,p2,p3,…,pn)    // p1,p2,p3…pn sont des variables utilisable en locale
                              // elles contiennent les valeurs transmises par l'appelant.
{
  // opérations utilisants les paramètres reçus p1,p2,p3 … pn
  return réponse ;
}
```

Appel d'une fonction :

```
Variable=NomFonction(v1,v2,v3,…vn) ;   // v1,v2,v3,…vn sont envoyés comme paramètres.
```

La réponse renvoyée par la fonction personnelle peut être une variable ou bien une valeur booléenne TRUE ou FALSE pour dire à la commande d'appel si tout s'est bien passé (return TRUE) ou si il y a eut un problème (return FALSE). Pour l'appel, on peut donc avoir

```
Variable=Fonction(paramètres) ;  // Variable vaut donc soit TRUE ou FALSE.
                                 // on peut ensuite tester Variable.
```

ou bien on peut directement tester le résultat de la fonction :

```
if (Fonction(paramètres)==TRUE)
{
  // les commandes placées ici sont exécutées si la Fonction répond TRUE
}
```

Nous allons d'ailleurs voir maintenant comment gérer des conditions et tester des valeurs.

Conditions et tests de valeurs

Dans un programme COFFEE, on peut avoir besoin d'exécuter des commandes en fonction de la valeur d'une variable, comme par exemple dans le cas ou on appelle une fonction et que celle-ci renvoit une réponse TRUE ou FALSE, en fonction de la réponse, les commandes qui suivent l'appel de cette fonctions peuvent être adaptées.

L'instruction IF: If (variable==condition) instruction ;

Explication : l'instruction if va exécuter instruction si (variable==condition) est vrai. (TRUE)

Exemple :
```
var MaVariable=5 ;
var reponse ;
If (MaVariable==5) reponse=Fonction(MaVariable) ;
// instructions suivantes
```

Explication : On crée MaVariable initialisée à 5 et reponse sans valeur ni type.
MaVariable==5 est vrai donc on appelle Fonction avec le paramètre 5.
Si MaVariable est différente de 5 alors on va directement aux instructions suivantes.

NB : Si on a plusieurs instructions à exécuter lors d'une condition, il suffit des les placer entre accolades :
```
if (variable==condition)
{
  instruction1 ;    // ce bloc d'instruction sera exécuté si variable==condition.
  instruction2 ;
  instruction3 ;
  …
}
```

L'instruction IF – ELSE IF :
```
if (condition1) {instruction1 ;}
else if (condition2) {instruction2 :}
```

Explication : else if permet d'enchaîner le test si condition1 n'est pas vrai, et teste une autre condition.

L'instruction IF – ELSE :
```
if (condition) {instruction1 ;}
else {instruction2 ;}
```

Explication : else permet d'exécuter instruction2 si la condition n'est pas vraie.

Les opérateurs de comparaison pour tester une condition:
== teste l'égalité de 2 valeurs.
!= teste l'inégalité de 2 valeurs.
<= teste si une valeur est supérieure ou égale à une autre.
< teste si une valeur est strictement supérieure à une autre.

L'instruction SWITCH :
```
switch (MaValeur)
{
  case(v1) : instruction1 ;  // si MaValeur=v1
             break ;
  case(v2) : instruction2 ;  // si MaValeur=v2
             break ;
        …

  default :  instructionN ;  // sinon
             break ;
}
```
Explication : switch permet d'associer une action pour différentes valeurs possible de MaValeur.

Voilà pour cette première partie.