Exploration du module Camera de l'API Python Blender

Merci à Diamond Editions pour son aimable autorisation pour la mise en ligne de cet article, initialement publié dans Linux Magazine N°77

Olivier Saraja - olivier.saraja@linuxgraphic.org

Une fois de plus, nous poursuivons notre découverte de l'API Python de Blender, et dans cet article un peu plus court que d'habitude, nous aborderons la gestion des objets de type Caméra.

Grâce aux éléments de l'API Python que nous allons découvrir, nous verrons comment créer un script interagissant dans les deux sens avec Blender: récupération d'objets et de paramètres issus d'une scène existante vers l'API Pyhton, puis après traitement particulier, modification dans Blender des objets en question en vue d'une animation. En effet, la définition d'une caméra via l'API Python de Blender étant des plus simples (il n'y a que peu de paramètres, au final, réellement disponibles), nous allons profiter de cet article pour apprendre quelques astuces de programmation qui nous permettront de faire, dans le futur, quelques scripts un peu plus intéressants que notre moyenne actuelle, et ce, quels que soient les modules mis en oeuvre.

1. Le module Camera

A l'image de la Caméra à l'intérieur de Blender même, la Caméra de l'API Python ne dispose pas d'un grand nombre de paramètres, ainsi qu'en témoigne la figure 01. Pour une fois, il s'agit plus d'un état de fait de Blender, que de l'état d'avancement de l'API, souvent incomplète. L'étude des différents paramètres de l'API s'en trouvera donc d'autant facilitée.

Figure 01: les paramètres par défaut de Blender pour la Caméra

1.1 Création de votre première caméra

Comme chaque module avant son usage, le module **Camera** doit impérativement être importé, généralement dans la deuxième ligne de votre script:

```
01:    import Blender
02:    from Blender import Camera
```

Malheureusement, en l'état, cela ne vous servira pas à grand chose. Dans **Premiers Pas avec Python** (dans *GNU/Linux Magazine* numéro 73 ou en ligne sur **http://www.linuxgraphic.org**), nous avons vu que la procédure pour créer une caméra via Python dans Blender consistait à, plus ou moins dans l'ordre: (1)

importer la scène courante, (2) créer le bloc de données python pour la caméra, (3) créer l'objet qui deviendra la caméra, (4) lier le bloc de données python à l'objet caméra, (5) lier l'objet caméra à la scène et, enfin, déclarer la caméra nouvellement créée comme étant la caméra active. Il devient alors évident qu'il vous faudra également importer les modules **Scene** et **Object** pour faire bon usage du module Camera. La première ligne de votre script sera donc:

```
02:    from Blender import Camera, Object, Scene
```

Nous allons maintenant importer en mémoire la scène courante de Blender, dans laquelle l'objet devra être, ultérieurement, inséré: **`[nom.data] = Scene.getCurrent()`** où **`[nom.data]`** est le nom du bloc de données sous Python relatif à la scène courante.

```
03:    cur = Scene.getCurrent()
```

Une fois ceci fait, nous pouvons commençons la création de la caméra, tout d'abord sous forme de donnée sous Python: **`[nom.data] = Camera.New('[Type]','[CA:Blender]')`**. **`[nom.data]`** correspond au nom du bloc de données Python, tandis que **`[CA:Blender]`** correspond au nom que prendra le bloc de données de la Caméra à l'intérieur de Blender. Enfin, **`[Type]`** peut au choix être soit une caméra de type orthogonale (**`'ortho'`**) ou perspective (**`'persp'`**)

```
04:    c = Camera.New('ortho','Camera-Vue01')
```

Mais notre caméra n'a pour l'instant aucune substance dans Blender, il s'agit juste d'un jeu de données inutilisé. Nous allons donc créer un nouvel objet (pour l'instant vide) dans Blender: **`[nom.data] = Object.New('Camera','[OB:Nom]')`**. A nouveau, **`[nom.data]`** correspond au nom du bloc de données Python, tandis que **`Camera`** précise le type d'objet à créer (d'autres valeurs sont bien sûr possibes en fonction du type d'objet à créer et que **`[OB:Nom]`** indique le nom que portera l'objet dans Blender.

```
05:    ob = Object.New('Camera','Vue01')
```

> **Astuce:**
>
> Si vous souhaitez faire des rendus « techniques » (sous-entendu, en mode '**`ortho`**' plutôt que '**`persp`**'), vous pouvez très bien créer quatre caméra que vous souhaiteriez appelez, par exemple: **`VueDeDessus`**, **`VueDeFace`**, **`VueDeCote`**, **`Perspective`**.

Mais comme souligné précédemment, il ne s'agit pour l'instant que d'un objet vide; pour lui donner corps, nous allons lier l'entité '**`c`**' à l'objet '**`ob`**':

```
06:    ob.link(c)
```

Enfin, il ne nous reste plus qu'à lier l'objet '**`ob`**' pleinement déterminé à la scène '**`cur`**' courante:

```
07:    cur.link(ob)
```

Nous pourrions penser en avoir terminer, mais ce n'est pas tout à fait le cas. Vous pouvez avoir dans vos scènes plusieurs caméras, seule le point de vue de la toute première à avoir été créée sera prise en compte au moment du rendu. Dans Blender, pour activer une autre caméra, il vous suffit de sélectionner celle de votre choix, et de taper [Ctrl]+[0] du pavé numérique. Au-travers de l'API Python, l'idée est presque identique: pour la scène '**`cur`**' courante, vous définissez la caméra active '**`ob`**'.

```
08:    cur.setCurrentCamera(ob)
```

La combinaison de touches **`[alt]+[p]`** lorsque le curseur de la souris est dans le fenêtre texte du script aura donc pour effet de créer une caméra, regardant vers les z négatifs, et localisée à l'origine de la scène.

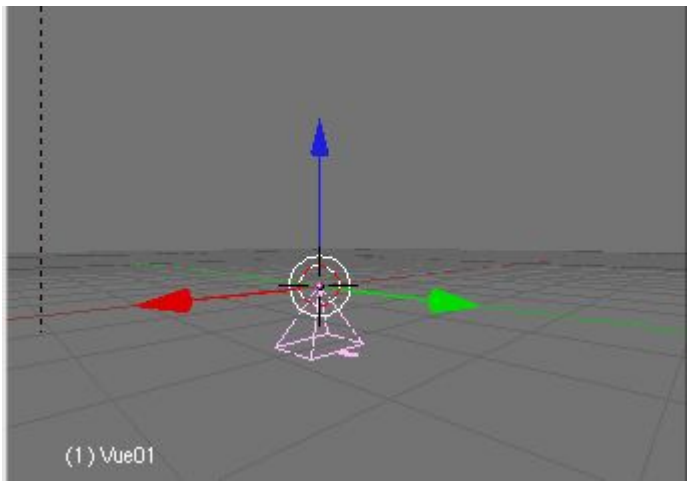

Figure 02: la création d'un caméra « par défaut »

1.2 Autres paramètres de la caméra

Comme la figure 01 l'indique clairement, il n'y a pas un très grand nombre de paramètres disponibles pour ce module. Il sera donc assez facile d'en faire le tour. La méthode qui suit permet de nommer les blocs de données relatifs à la caméra.

La méthode setName:

Par défaut, le bloc de données à l'intérieur de Blender pour la première caméra porte le nom **`CA:Camera`**. Si vous créez plusieurs caméras, la deuxième s'appellera **`Camera.001`**, la troisième **`Camera.002`** et ainsi de suite. Telle sera en fait la politique de nommage si vous n'avez pas spécifié, lors de la fonction **`Camera.New('[type]'),'[CA:nom]')`** de nom pour le bloc de données de la caméra dans Blender. La commande **`nom.data].setName('[CA:nom]')`** vous permet donc de nommer, *a posteriori*, le bloc de données dans Blender. Par exemple, les deux lignes:

```
04:    c = Camera.New('ortho')
09:    c.setName('Camera-Vue01')
```

sont équivalentes à:

```
04:    c = Camera.New('ortho','Camera-Vue01')
```

Les méthodes qui suivent permettent de déterminer le fonctionnement et les réglages propres à la caméra.

La méthode setType:

Nous avons déjà vu, lors de la création de l'objet Camera, qu'il était possible de déterminer son type grâce à la commande **`Camera.New('[type]', '[Nom]')`**. Mais il est possible de procéder à la création en ne passant aucun argument à la foncion **`Camera.New()`** et de définir *a posteriori*, comme précédemment, le type de caméra. Il en résulte que les trois lignes:

```
04:    c = Camera.New()
09:    c.setName('Camera-Vue01')
10:    c.setType('ortho')
```

sont équivalentes à:

```
04:    c = Camera.New('ortho','Camera-Vue01')
```

La méthode setLens:

Ce paramètre permet de définir le choix de l'objectif virtuel de votre caméra, symbolisé par le bouton **Lens** dans l'interface de Blender. Par défaut, la valeur est de 35 mm. En utilisant l'API Python, il vous est possible de spécifier une nouvelle valeur grâce à la commande **`[nom.data].setLens([valeur])`**. Par exemple:

```
04:    c = Camera.New('persp','Camera-Vue01')
10:    c.setLens(55)
```

Cette valeur peut varier de 1.0 à 250.0, mais il est surtout important de noter qu'elle est exclusivement utilisable avec une caméra de type '**persp**'.

La méthode setScale:

De même que la méthode **setLens** n'est utilisable qu'avec une caméra de type '**persp**', **setScale** ne fonctionne qu'avec une caméra de type '**ortho**'. En mode de projection orthogonale, il n'y a pas de ligne de fuite ni d'effet de perspective, aussi les distances ne comptent elles pas. Ce type de projection est surtout utilisé en dessin technique, et obéit au même règle. En définissant une caméra de type '**ortho**', vous déterminez la vue d'observation de votre objet ou de votre scène. Mais vous souhaiterez certainement que l'objet « remplisse » au maximum la vue, ou vous apprécierez la possibilité de zoomer sur un détail de votre objet. Pour ce faire, vous utiliserez à votre avantage la méthode **setScale**: **[nom.data].setScale ([valeur])**. Par exemple:

```
04:    c = Camera.New('ortho','VueDeFace')
10:    c.setScale(2.5)
```

Cette valeur peut varier de 0.01 à 1000.00, et est exclusivement utilisable avec une caméra de type '**ortho**'.

Les méthodes setClipSta et setClipEnd:

Dans Blender, le moteur de rendu n'est pas capable d'effectuer le tracé d'objets situés à l'infini. Il est donc nécessaire de définir une « distance », en unités de Blender, depuis la caméra et au-delà de laquelle les objets de la scène ne seront pas pris en compte et ne figureront pas sur le rendu (ou figureront de façon tronquée s'ils sont à la frontière). Cette distance a été baptisée Clip End, et c'est elle que l'on retrouve sous le nom énigmatique de **ClipEnd** dans les boutons de la Figure 01. Par défaut, elle est égale à 100.0, mais elle peut varier de 1.0 à 5000.0. On la définit via l'API Python de la façon suivante: **[nom.data].setClipEnd ('[valeur]')**. Dans le même ordre d'idée, il existe une distance minimale, en-dessous de laquelle les objets ne seront pas rendus, symbolisée par le bouton **ClipSta** (pour Clip Start, bien évidemment): **[nom.data].setClipStart('[valeur]')**. Par défaut, elle est égale à 0.1 mais elle peut varier de 0.0 à 100.0. Bien sûr, spécifier une valeur de **ClipEnd** inférieure à **ClipSta** ne va pas vous avancer à grand chose, mais Blender ne vous en empêchera pas!

```
11:    c.setClipStart(2.5)
12:    c.setClipEnd(5.0)
```

Les méthodes qui suivent influent sur l'affichage de l'objet Caméra dans Blender, sans avoir pour autant d'effet particulier au moment du rendu.

La méthode setMode:

Cette méthode permet d'activer (ou désactiver) l'affichage des limites de la caméra (**ClipSta** et **ClipEnd**), ou les distances d'effet du brouillard (**Mist**). A cette fin, cette méthode admet jusqu'à deux variables, qui sont des chaînes de caractères: **'showLimits'** et **'showMist'**. Bien sûr, les apostrophes sont essentielles, comme lors de tout usage de chaînes de caractères comme argument d'une méthode. Si un argument n'est pas passé, les limites ne sont pas affichées, de sorte que l'usage de la commande **[nom.data].setMode()** sans aucun argument peut servir à désactiver les deux modes d'affichage. La syntaxe complète de cette méthode est donc quelque chose comme: **[nom.data].setMode('[argument1]','[argument2]')**. Par exemple:

```
13:    c.setMode('showLimits')
```

permet d'afficher les limites **ClipSta** et **ClipEnd** de la caméra, mais pas la distance d'effet de **Mist**.

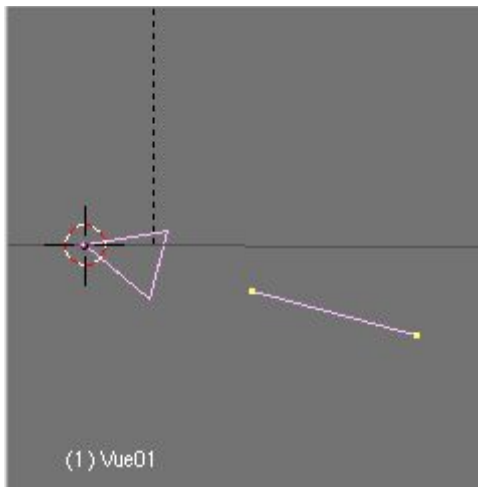

Figure 03: illustration des méthodes setClipStart, setClipEnd et setMode('showLimits')

La méthode setDrawSize:

Cette méthode permet de déterminer la taille de l'objet caméra dans la scène. Il s'agit ici de modifier la taille de l'élément tracé dans une fenêtre 3D, de sorte à améliorer sa visibilité ou celle de la scène, et en aucun cas d'affecter le rendu de l'image. Dans Blender, cela se règle par l'usage du bouton **DrawSize**. Via l'API Python, il est également possible de déterminer la taille de l'objet grâce à une commande du type: **`[nom.data].setDrawSize([valeur])`**, **`[valeur]`** pouvant varier de 0.1 à 10.0. La valeur par défaut de Blender est 0.5. Par exemple, pour avoir une caméra deux fois plus grosse dans la vue 3D:

```
14:    c.setDrawSize(1.0)
```

La méthode setIpo:

Cette méthode permet de lier une courbe **Ipo** à une caméra, en utilisant une commande de type: **`[nom.data].setIpo('[IP:nom]')`** où **`[nom.data]`** est, comme d'habitude, le nom du bloc de données correspondant à la caméra, et **`[IP:nom]`** le nom de la courbe Ipo à lier à la caméra. Supposons l'existence, dans notre scène, d'une courbe Ipo nommée **`CaIpo`**, telle que sur la figure qui suit.

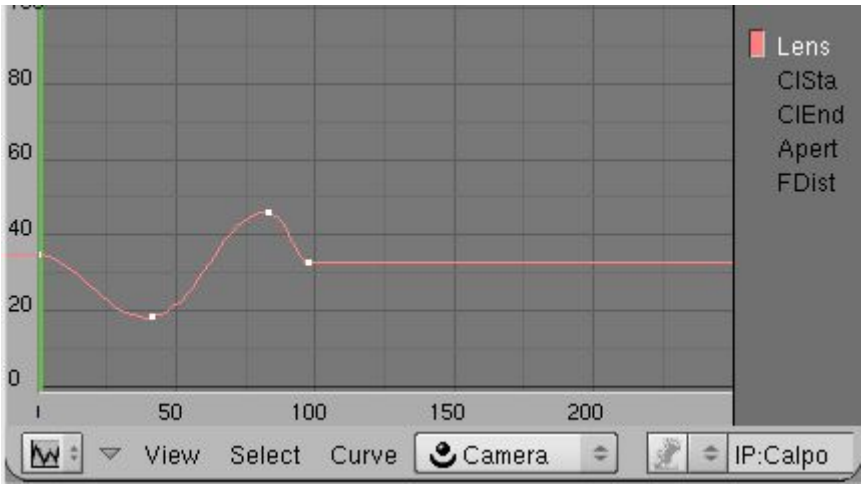

Figure 04: une courbe Ipo de type Camera

Nous allons d'abord l'importer dans l'API Python, au moyen de la ligne suivante:

```
15:    curv = Blender.Ipo.Get('CaIpo')
```

Nous verrons toutefois dans un prochain article comment créer une courbe IPO de toute pièce via l'API Python. Nous nous contenterons aujourd'hui de lier cette courbe Ipo existante à notre caméra, au moyen de la commande suivante: **`[nom.data].setIpo([IP:nom.courbeIpo])`** où **`[IP:nom.courbeIpo]`** est tout simplement le nom de la courbe à importer. Par exemple:

```
16:    c.setIpo(curv)
```

2. Quelques ateliers pratiques

Jusqu'à présent, dans cette série d'articles, nous avons cherché à explorer l'API Python Blender avec comme prétexte de créer des objets, caméras, lumière, et autres matériaux en faisant exclusivement usage de code en python. Mais ce faisant, nous nous sommes privés de la richesse de l'API en matière d'interactivité. En effet, celle-ci nous permet aussi bien de créer, que d'importer des objets pré-existants dans le contexte de l'API et de là, les retravailler avant de les « re-injecter » dans Blender. Ainsi, partout où nos avons vu jusqu'à présent des méthodes commençant par **`set[paramètre]`**, sachez qu'il existe également des méthodes commençant par **`get[paramètre]`**, parmi bien d'autres possibilités. Avant de poursuivre, jetons-y un oeil attentif.

2.1 Jouer avec les méthodes de type get

Prenons pour exemple la scène par défaut de Blender: lancez le logiciel, ou réinitialisez la session grâce à la combinaison de touches **`[alt]+[x]`**. Séparez l'écran en deux et dans l'une des deux fenêtres, appelez l'éditeur de texte: [**`Maj]+[F11]`** et créez un nouveau fichier texte: **`[alt]+[n]`**. Saisissez alors le code suivant:

```
01:    import Blender
02:    from Blender import Camera
03:    camdata = Camera.Get()
04:    print camdata
```

Nous ne commenterons pas les deux premières lignes, désormais évidentes pour tout lecteur régulier. La troisième ligne est beaucoup plus intéressante. En premier niveau, disons que cette ligne recherche (**`Get`**) toutes les caméras (**`Camera`**) portant n'importe quel nom **`()`** et place le résultat dans la variable **`camdata`**. La quatrième ligne, tout simplement, affiche le contenu de **`camdata`** dans la console:

```
[[Camera "Camera"]]
```

En deuxième niveau, les choses sont beaucoup plus intéressantes: si votre scène comporte plusieurs caméras, **`camdata`** devient alors une liste. Donc, si vous insérez dans la scène de base quelques caméras supplémentaires (qui prendront automatiquement les noms **`CA:`** et **`OB: Camera.001`**, **`Camera.002`**, etc.) et que vous exécutez à nouveau le script (**`[alt]+[p]`**), la console affiche alors:

```
[[Camera "Camera"], [Camera "Camera.001"], [Camera "Camera.002"]]
```

si vous avez inséré jusqu'à deux caméras supplémentaires et en gardant les noms par défaut. Mais comment faire pour interagir, à tour de rôle, avec chacune des caméras existantes? Très simplement, jetons un oeil au code suivant pour nous en convaincre, avec trois caméras dans notre scène (la caméra par défaut, plus deux caméras positionnées au hasard dans la scène):

```
01:    import Blender
02:    from Blender import Camera, Object
03:    camdata = Camera.Get()
04:    print camdata
05:    for cam in camdata:
06:            camname = cam.getName()
07:            print camname
08:            camobj = Object.Get(camname)
09:            camobj.setLocation(0,0,0)
10:    Blender.Redraw()
```

En ligne 02, nous importons le module **`Object`** en plus du module **`Camera`**, afin de pouvoir intervenir, plus loin, sur la localisation physique des caméras. En ligne 03, nous rapatrions (**`Get`**) la liste de toutes les caméras (**`Camera`**) et nous les plaçons dans une variable (**`camdata`**) qui devient donc une variable de type liste, puisque notre scène contient plusieurs caméras. Pour nous assurer que cette opération s'est bien déroulée, nous pouvons, facultativement, en ligne 04, afficher dans la console les noms de caméras

trouvées. En ligne 05 commencent les choses intéressantes; comme nous l'avons déjà vu dans l'article d'introduction de la série (*Blender: Premiers pas avec Python*, dans Linux Magazine N°73), nous mettons en place une boucle pour agir sur chaque objet **cam** de la liste **camdata**.

La ligne 06 permet de définir une variable **camname** dans laquelle nous enregistrons le nom (**getName**) de la camera (**cam**) courante (à titre de vérification, optionnellement, dans la ligne 07, nous affichons ce nom dans la console). En ligne 08, nous sélectionnons (**Get**) un objet (**Object**) en fonction de son nom (**camname**) et nous stockons le résultat dans une variable **camobj**. Enfin, en ligne 09, nous déplaçons (**setLocation()**) cet objet (**camobj**) au centre de la scène. Arrivés au bout de la boucle, nous passons à la caméra (**cam**) suivante, jusqu'à arriver à la dernière caméra de la liste **camdata**.

2.2 Un petit script: le zoom paramétré

L'objectif du script suivant sera tout simplement de permettre à une caméra donnée, de zoomer d'un certain facteur, sur un laps de temps imparti. Soyons clair: il est sans doute plus aisé d'obtenir le même résultat grâce aux courbes **Ipo**, mais le but étant d'apprendre à faire des choses amusantes, nous retiendrons bien évidemment la solution python.

Nous noterons toutefois un avantage de Python par rapport à **Ipo** dans ce cadre là: changer la vitesse de zoom, ou la *frame* de démarrage du zoom s'avère être une formalité sous Python, alors qu'elle nécessiterait plusieurs manipulations dans la fenêtre **Ipo** avant que la courbe d'animation soit conforme à nos souhaits.

Voici le code que nous allons décortiquer. Il est sans prétention et largement perfectible.

```
01:    import Blender
02:    from Blender import Camera, Object
03:
04:    #=========================================
05:    zoom_fac = 2.0            # Facteur de zoom
06:    zoom_dur = 48             # Delai de zoom
07:    frame_sta = 25            # 1ere frame de zoom
08:    lens_sta = 35.0           # Valeur lens de depart
09:    #=========================================
10:
11:    lens_fin = lens_sta*zoom_fac
12:    zoom_inc = (lens_fin-lens_sta)/zoom_dur
13:
14:    frame = Blender.Get('curframe')
15:
16:    if frame < frame_sta:
17:           lens = lens_sta
18:    if (frame >= frame_sta) & (frame<(frame_sta+zoom_dur)):
19:           lens = lens_sta+(frame-frame_sta)*zoom_inc
20:    if frame >= (frame_sta+zoom_dur):
21:           lens = lens_sta*zoom_fac
22:
23:    camdata = Camera.Get()[0]
24:    camdata.setLens(lens)
25:    Blender.Redraw()
```

Nous partons du principe que le paramètre **Lens** de notre caméra est au départ de 35.0. Nous la consignons dans la variable **lens_sta**. Nous souhaitons zoomer d'un facteur deux (**zoom_fac = 2.0**) et que le temps imparti au zoom (**zoom_dur** pour durée) soit de 48 *frames*. Enfin, nous décidons que la caméra commencera à zoomer à partir de la *frame* numéro 25 (**frame_sta**).

Les lignes 11 et 12 permettent de formaliser les calculs nécessaires à l'obtention du résultat souhaité: **lens_fin** sera le produit de la valeur initiale de l'objectif (**lens_sta**) et du facteur de zoom (**zoom_fac**). Nous déterminons ensuite l'incrément (**zoom_inc**) à ajouter à la valeur de l'objectif pour que le zoom soit très progressif d'une *frame* à la suivante; cette valeur est tout simplement égale à la différence entre la valeur d'objectif finale (**lens_fin**) et initiale (**lens_sta**), divisé par le nombre de *frames* (**zoom_dur**) sur lequel la

transition doit s'opérer.

> **Attention:** Nous n'avons déclaré aucune de ces variables comme des entiers ou des réels, Python se chargeant d'interpréter la nature de tous ces chiffres, selon qu'ils soient décimaux ou non. Quelle importance? Ouvrez une console, lancez une session python et exécutez les calculs suivants:
>
> ```
> [olivier@icare ~]$ python
> Python 2.4 (#2, Feb 13 2005, 22:08:03)
> [GCC 3.4.3 (Mandrakelinux 10.1 3.4.3-3mdk)] on linux2
> Type "help", "copyright", "credits" or "license" for more information.
> >>> (70-35)/48
> 0
> >>> (70.0-35)/48
> 0.72916666666666663
> ```
>
> Vous voyez que le résultat est radicalement différent!
>
> Dans notre code, **`zoom_dur`** et **`frame_sta`** représentent un nombre de frames, il est normal qu'ils soient des entiers. En revanche, **`zoom_fac`** et **`lens_sta`** peuvent théoriquement accepter des valeurs décimales. Mais si vous ne les écrivez pas sous forme décimale (par ex: 2.0 et 35.0 à la place de 2 et 35) ils seront interprétés comme des entiers, et Python arrondira systématiquement les résultats de calcul à l'entier inférieur. Si nous exécutons manuellement les calculs, voici ce que nous obtenons:
>
> **`lens_fin = 35 * 2 = 70`**
>
> **`zoom_inc = (70 – 35) / 48 = 0.729`**
>
> mais Python arrondissant les calculs sur entiers à l'entier immédiatement inférieur, **`zoom_inc = 0`**! Et dans la mesure il s'agit de l'augmentation de la valeur **`lens`** d'une frame à l'autre, il en résulte que la caméra ne zoome plus!

La ligne 14 est particulièrement importante pour notre script: c'est celle qui permet de récupérer le numéro de la *frame* courante, ce qui sera utile pour déterminer la valeur instantanée du **Lens** de la caméra! Pour ce faire, nous utilisons la fonction **`Get()`** du module **`Blender`**. Cette fonction permet de récupérer l'état d'un grand nombre de variables, comme la frame courante de l'animation (**`'curframe'`**), la version de Blender (**`'version'`**, utile pour certains scripts, seulement compatibles avec certaines versions de Blender) ou tout un tas de chemins par défaut pour les scripts, les textures, les fichiers (**`'homedir'`**, **`'scriptsdir'`**, **`'renderdir'`**...), etc.

Entre les lignes 16 et 21, nous définissons trois comportements de la caméra en fonction du temps: du début de l'animation jusqu'à la *frame* de début du zoom, de la *frame* de début du zoom jusqu'à la fin du zoom, et au-delà de la dernière *frame* du zoom. Sur toute la première phase, la caméra conservera sa valeur d'objectif d'origine: **`lens = lens_sta`**. Après avoir zoomé, sur la troisième phase, la valeur de l'objectif sera à sa valeur de zoom: **`lens = lens_sta*zoom_fac`**. Et entre les deux, logiquement, la valeur de l'objectif sera égale à la valeur d'origine plus le nombre de *frames* écoulées depuis le début de cette phase, multiplié par l'incrément de zoom: **`lens = lens_sta+(frame-frame_sta)*zoom_inc`**.

Il ne nous reste plus qu'à récupérer la caméra courante grâce à la fonction **`Get()`** du module **`Camera`**, qui génère donc une liste arbitrairement nommée **`camdata`**. Comme nous nous intéressons à la première (et seule!) caméra de la liste, nous le précisons par l'indice **`[0]`**, ajoutée à la ligne habituelle: **`camdata = Camera.Get()[0]`**.

Nous pouvons maintenant appliquer la valeur **`lens`** calculée à la caméra désormais accessible par l'API Python sous le nom de **`camdata`**, grâce à la méthode **`setLens([valeur])`** du module **`Camera`**, et enfin ordonner que soit mises à jour les fenêtres 3D de Blender grâce à la fonction **`Redraw()`** du module **`Blender`**.

Enfin terminé? Pas tout à fait! Si vous utilisez la combinaison de touches **`[alt]+[p]`** pour exécuter le script, la valeur **Lens** de la caméra active sera modifiée conformément aux instructions du script, mais lors de la

prévisualisation de l'animation (**`[alt] + [a]`**) directement dans la fenêtre 3D, il ne va rien se passer.

Il existe en effet une façon de mettre en pratique les scripts que nous n'avons encore jamais évoquée: les liens de scripts, ou *Scriptlinks* en anglais! Tout se passe en fait dans Blender, au niveau du menu **Script**, dans l'onglet **Scriptlinks**.

Figure 05: à gauche, le menu Scripts lorsque la caméra de la scène est sélectionnée, au milieu, le même menu lorsque le bouton Camera Script Links est coché, et enfin, à droite, le même menu avec les options de gestion du script

Par défaut, l'option **Enable Script Links** est activée, avec trois icônes qui permettent de lier des scripts à plusieurs familles, en fonction de l'objet sélectionné dans la fenêtre 3D. Par exemple, dans la Figure 05 (à gauche), avec la caméra sélectionnée, on a droit à la possibilité de lier des scripts de type **Object**, **World** ou **Camera**. Sélectionnons la troisième icône, et constatons (Figure 05, au milieu) qu'il est désormais possible de lier un Script à la caméra en choisissant celui-ci au-travers du bouton **New**. Blender vous offre automatiquement le choix parmi la liste des fichiers ouverts ou composés dans la fenêtre d'édition de texte (accessible par **`[maj] + [f11]`**), et ensuite vous permet de déterminer quel événement déclenchera une action du script:

FrameChanged: le script sera ré-évalué à chaque changement de frame, dans le cadre d'une animation. C'est le cas qui nous intéresse pour notre script de zoom paramétré.

Redraw: le script sera ré-évalué à chaque fois que les fenêtres de Blender seront rafraîchies.

Render: le script sera ré-évalué à chaque rendu.

Pour notre part, lorsque nous activerons **New**, nous choisirons **`Text`** puisqu'il s'agit du nom par défaut de tout script en cours d'écriture, à moins que vous ayez eu recours à l'option **Save as...** du menu **File** de la fenêtre d'édition de texte. Par défaut, Blender nous propose **FrameChanged** comme événement déclencheur du script, ce qui nous convient tout à fait.

Une fois cette préparation faite, retournons dans la vue 3D de notre scène, et avec le curseur de la souris dans la fenêtre en question, lançons la prévisualisation de l'animation grâce à la combinaison de touches **`[maj] + [a]`**. Pendant les 25 premières *frames* (jusqu'à **`frame_sta`**), il semble ne rien se passer, la valeur de l'objectif en restant à sa valeur d'origine (**`lens_sta`**)(voir Figure 06, à gauche) ; les 48 suivantes (la durée **`zoom_dur`**) la caméra zoome progressivement le sujet (voir Figure 06, au centre). Ensuite, et ce, indéfiniment, elle conserve sa nouvelle valeur d'objectif (**`lens_fin`**) après avoir atteint sa valeur maxi (voir Figure 06, à droite).

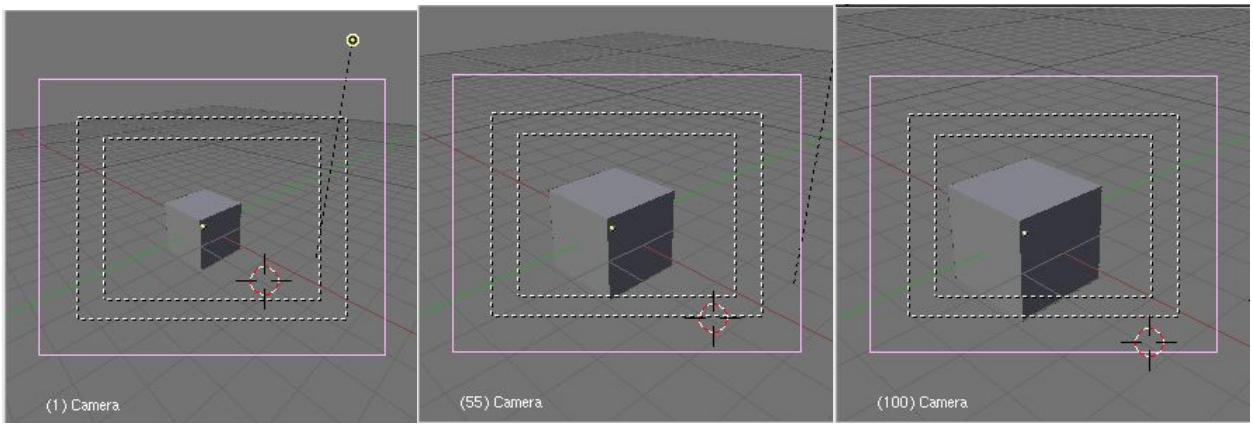

Figure 06: prévisualisation de l'animation, notre script de zoom paramétré fonctionne à la perfection!

3. Quelques scripts intéressants à étudier

Les scripts qui suivent sont plus ou moins puissants ou utiles, mais se montrent tous intéressants à étudier. L'auteur de l'article n'a été impliqué ni de près ni de loin dans leur réalisation, il les présente ici à fin pédagogique. Il en existe plusieurs autres, non moins intéressants, mais une recherche à l'aide de n'importe quel moteur sur l'internet devrait vous y conduire sans trop de difficultés.

3.1 Set Active Camera

Un script sans prétention par « panzi », et que l'on trouve sur les forums Python d'Elysiun à l'adresse suivante: **http://www.elysiun.com/forum/viewtopic.php?t=49615**. Il permet simplement de récupérer l'objet sélectionné, de vérifier qu'il s'agisse bien d'une caméra, et enfin de la déclarer comme étant la caméra active de la scène. Plusieurs petites choses intéressantes dans ce script. Tout d'abord, la structure:

```
try:
      ...
except:
      ...
```

qui permet de retourner un message d'erreur à l'utilisateur s'il a oublié de sélectionner un objet avant de lancer le script, en faisant appel à la fonction **`PupMenu`** du module **`Draw`**. L'autre point intéressant concerne la possibilité d'enchaîner (et parfois même d'imbriquer) des fonctions et méthodes les unes derrière les autres:

```
Scene.GetCurrent().setCurrentCamera(cam)
```

Cette ligne permet de récupérer la scène courante, et dans celle-ci, déterminer **`cam`** comme étant la caméra active.

3.2 Blender Camera to Face Aligner

Ce script est un petit peu plus ambitieux que la moyenne. Il permet d'aligner la caméra de façon à ce qu'elle observe le plus « perpendiculairement » possible un groupe de faces donné, à condition de les avoir sélectionné en mode **Sélection de Faces** (touche **`[f]`**). Il est possible de trouver le script à l'adresse suivante: **http://www.alienhelpdesk.com/index.php?id=20**. L'objectif initial était de pouvoir effectuer le rendu de certaines faces, et de récupérer le résultat du rendu (au-travers d'un logiciel de retouche d'images comme *The Gimp*) dans une carte UV. En clair, cela permettrait de récupérer, au prix de quelques « grosses » étapes supplémentaires, le résultat du *vertex painting*, de l'*ambient occlusion* ou même des textures procédurales directement dans une texture image « mappable » sur l'objet. Ce script est particulièrement intéressant à étudier de par la présentation très claire de l'interface graphique (voir *Création*

d'une interface graphique sous Blender pour vos scripts Python, dans GNU/Linux Magazine N°74, ou en ligne sur **http://www.linuxgraphic.org**). A noter l'existence d'un script très similaire, livré en standard avec Blender: **Texture Baker** (**`Scripts > UV > Texture Baker`**).

3.3 Camera jitter

Ce script est devenu un classique, bien avant qu'il n'apparaisse, sous son ultime forme, dans la documentation officielle de l'API Python, comme un exemple d'usage du module **`Noise`**. Il est disponible à l'adresse suivante: **http://www.blender.org/modules/documentation/234PythonDoc/Noise-module.html**. Utilisé en guise de **Scriptlink** d'une caméra, avec **FrameChanged** comme événement déclencheur, ce script permet permet de faire tanguer la caméra dans des directions aléatoires. En modifiant un peu les paramètres, il est possible de simuler le point de vue d'un homme saoul, ou un cameraman en train de filmer un tremblement de terre.

3.4 Changement de caméra à la volée

Le spécialiste français du *scripting* Python pour Blender (en l'occurence JM Soler) a produit un intéressant script très ludique, permettant de simuler un effet très « *Matrix – the movie* » ou « *L'armée de Terre recrute* » au sein d'une animation. Le script en lui-même se trouve à l'adresse suivante: **http://jmsoler.free.fr/didacticiel/blender/tutor/cpl_changerdecamera.htm**, et permet donc de passer rapidement d'une caméra à l'autre au cours d'une animation, et lorsque les caméras en question sont arrangées en cercle autour du sujet, l'effet peut être particulièrement saisissant, et coûte beaucoup moins cher qu'au cinéma!

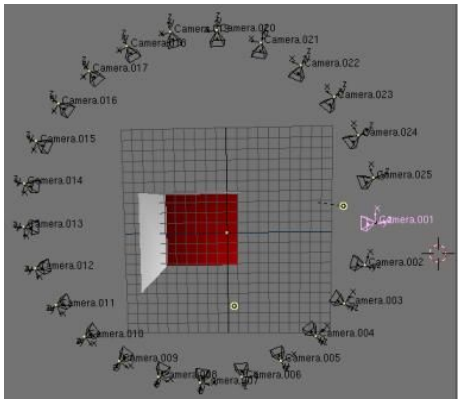

Figure 07: prêt à entrer dans la matrice, Néo?

Au-delà de cet usage créatif de Python, cette page propose surtout des explications sur le moyen d'associer une interface graphique à un script lié (Blender V2.27 et inférieur). En effet, à chaque fois qu'un script est exécuté, les variables déclarées au sein de Python sont systématiquement remises à zéro, et il est donc théoriquement impossible de conserver les paramètres fixées par la GUI à chaque changement de *frame*. Sur cette page, on apprend comment passer les paramètres de la GUI dans le module Blender et à travailler à partir de celui-ci.

4. Conclusions

L'exploration du module Camera de l'API Python n'est pas une longue affaire, étant donnée la simplicité de ce module. Cet article a donc été l'occasion de s'intéresser un peu plus à la programmation de scripts modestes qu'à l'apprentissage de chaque paramètre et chaque possibilité d'un module donné de l'API. L'usage des fonctions **`Get()`** et des méthodes à base de **`get...()`** est une façon très commode de

communiquer avec une scène Blender existante, tandis que les méthodes à base de **`set...()`** permettent la plupart du temps de modifier celle-ci.

Nous avons également eu l'occasion de découvrir une nouvelle façon d'utiliser les scripts python: en plus du traditionnel **`[alt] + [p]`** dans une fenêtre d'édition de texte contenant le code en question, nous avons vu comment lier un script à un objet et contrôler sa ré-évaluation automatique, par exemple à chaque changement de *frame*.

Il faut, enfin, minimiser la portée de ce qui a été réalisé aujourd'hui comme script: le « zoom paramétré ». En effet, n'importe quelle courbe **Ipo** bien configurée aurait tout aussi bien fait l'affaire, la seule différence étant que le script offre un peu plus de versatilité, et permet de changer aisément et rapidement plusieurs paramètres essentiels, comme la *frame* de départ de l'animation des paramètres de la caméra, leur durée et leur variation propre de valeur. Mais nous aurons prochainement l'occasion d'établir des scripts un peu plus élaborés...

Liens

La page de Blender (version actuelle: v2.37a): **www.blender.org**

La documentation officielle de python pour Blender:
http://www.blender.org/documentation/237PythonDoc/index.html

La documentation de python: **http://www.python.org/doc/2.3.5/lib/lib.html**