Exploration du module Ipo 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°79

Olivier Saraja- olivier.saraja@linuxgraphic.org

Lors de notre exploration du module Camera (en ligne sur http://www.linuxgraphic.org), nous nous sommes essayé à la création d'un script permettant d'animer le facteur de zoom de la caméra. L'usage des courbes IPO nous auraient alors considérablement facilité la tâche.

Mais tout ce que nous avons été en mesure de faire était de lier une Ipo existante à la caméra à animer. L'objet de cet article sera donc de compléter notre connaissance de l'API Python de Blender afin que nous soyons en mesure de travailler avec les courbes *Ipo* avec plus d'aisance. Mais avant d'aller plus loin, nous pouvons commencer par nous demander ce qu'est une courbe *Ipo*.

1. Les courbes Ipo

Contrairement à ce que beaucoup de gens pensent, *Ipo* n'est pas un acronyme quelconque,mais une abréviation. En fait, une *Ipo Curve* signifie courbe d'interpolation. Cela veut dire que nous pouvons spécifier quelques points clés, et que Blender va se charger de faire passer une courbe au-travers de ces points. Jusque là, peut d'intérêt, mais lorsque l'on comprend que l'on peut associer une courbe *Ipo* à virtuellement n'importe quelle variable de Blender, on réalise soudain qu'il est possible de contrôler la valeur de la variable en fonction du temps. En effet, la courbe *Ipo* est tracée sur un graphe présentant en ordonnée la valeur de la variable, et en abscisse les frames (l'unité de temps interne à Blender).

1.1 Création d'une Ipo curve élémentaire

Vous pouvez associer une *Ipo curve* à un objet. En fait, vous pouvez associer une même *Ipo curve* à plusieurs objets, ou plusieurs courbes à un même objet, mais nous sortons là du cadre de notre étude, tout à fait élémentaire. Pour comprendre comment cela marche, lancez Blender, ou ouvrez une nouvelle session en utilisant la combinaison de touches **`[Ctrl]`**+**`[X]`**. En cliquant avec le **`Bouton Central`** de la souris sur le bord séparant la ***vue 3D*** de la ***Fenêtre des Boutons***, coupez la ***vue 3D*** en deux en choisissant ***Split Area*** dans le menu flottant qui apparaît. Tout à fait à gauche de la barre de menu de l'une des deux fenêtres, vous trouverez une icône représentant un repère sur une grille. En cliquant dessus vous affichez la liste des types de fenêtre disponibles. Choisissez ***Ipo Curve Editor***. Dans la ***vue 3D***, sélectionnez le cube central à l'aide du **`Bouton Droit`** de la souris s'il ne l'est pas déjà, et reportez à nouveau votre attention dans l'***Ipo Curve Editor***.

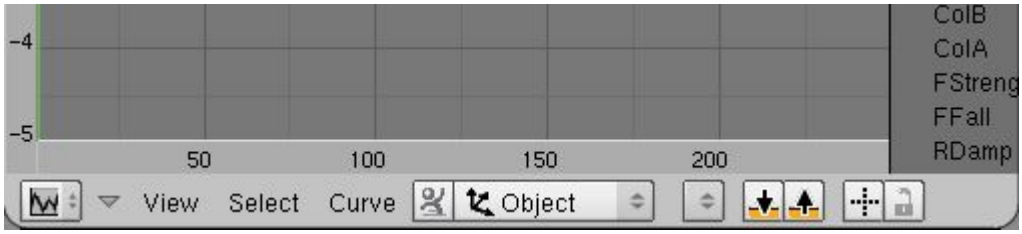

Figure 01: la barre de menu de l'IPO Curve Editor

Dans la barre de menu de celle-ci se trouve en particulier un bouton déroulant définissant le type d'*Ipo* qu'il est possible de créer. Par défaut, c'est une *Ipo* de type ***Object*** qui est affichée, comme nous pouvons le remarquer sur la Figure 01. Mais nous pouvons le modifier, pour créer une *Ipo* ***Material***, ***World***, ***Texture***, ***Shape***, ***Constraint***, et ***Sequence***. Si vous observez la partie droite de l'éditeur, vous noterez la liste exhaustive des paramètres pour lesquels il est possible d'établir une courbe *Ipo*, et vous noterez que cette liste varie en fonction du type d'*Ipo* spécifié.

Juste à droite du sélecteur de type d'*Ipo*, vous trouverez le très classique bouton ascenseur qui permet d'afficher la liste des *Ipo* déjà créées. Si vous cliquez dessus alors qu'aucune *Ipo* n'existe encore, Blender vous propose d'en créer une grâce au menu flottant qui apparaît: ***Add New***. Un nouveau champ apparaît alors, précédé des deux lettres **`IP:`** (indiquant un bloc de données de type *Ipo*), qui vous permet d'associer un nom à la courbe *Ipo* nouvellement créée (et si le nom proposé par défaut par Blender ne vous convient pas, vous pouvez bien sûr le changer).

Si vous cliquez avec le **`Bouton Gauche`** de la souris n'importe où dans le graphe, une barre verticale verte apparaît pour symboliser la position du curseur de la souris dans le temps, exprimé en frames. Cliquez sur l'une des variables dans la colonne de droite de l'éditeur. Par exemple, sur ***RotZ***. Maintenant, n'importe où dans le graphe, cliquez avec le **`Bouton Gauche`** de la souris, tout en maintenant la touche **`[Ctrl]`** pressée: des points apparaissent sous votre curseur. Positionnez maintenant votre curseur de souris dans la ***vue 3D*** et utilisez la combinaison de touches **`[Alt]+[A]`** pour lancez la simulation d'animation: vous devriez voir votre cube tourner sur lui-même conformément à la valeur imposée à ***RotZ*** dans l'éditeur.

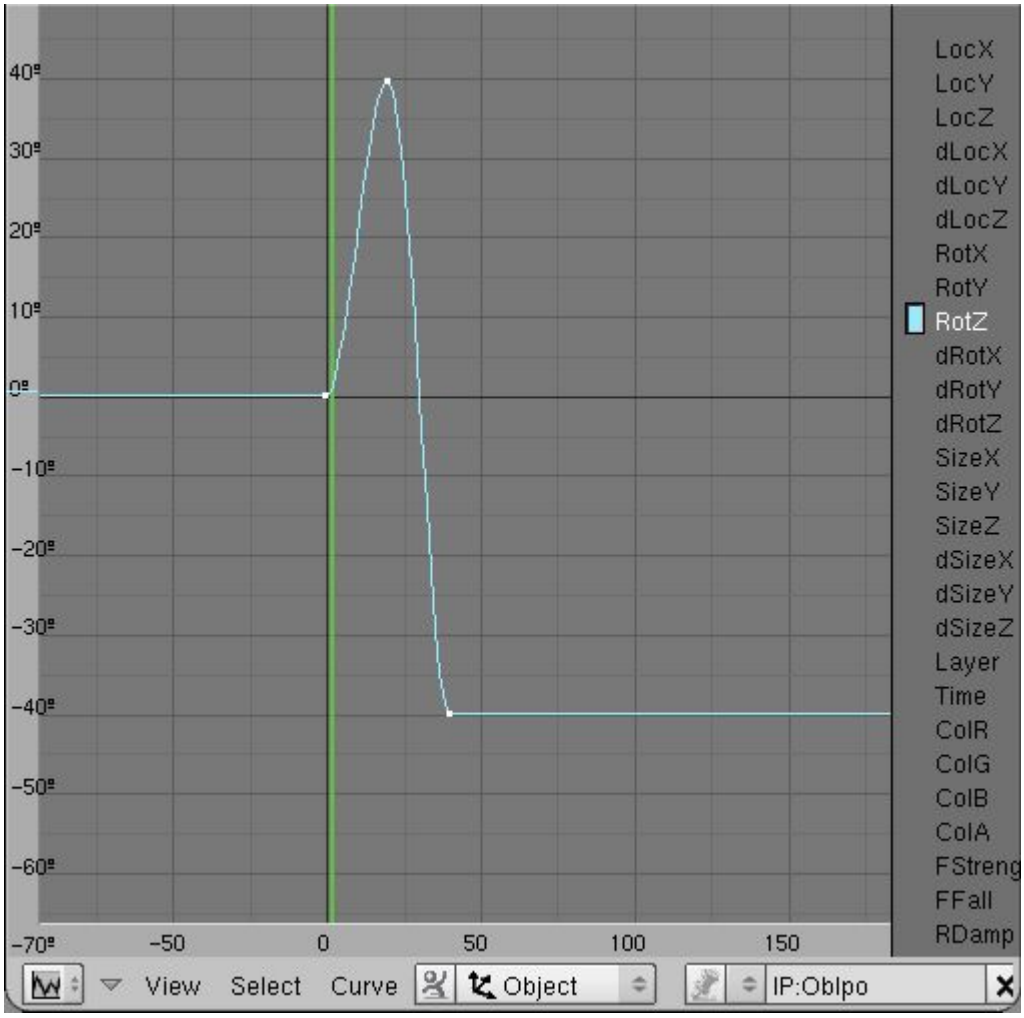

Figure 02: utilisez [Ctrl]+[Bouton Gauche] de la souris pour « poser » des points sur la courbe

1.2 Quelques options incontournables

Vous pouvez avoir plusieurs courbes *Ipo* associées au même objet, mais une seule par paramètre. Sélectionnez l'une de vos courbes (si vous en avez créées plusieurs) et dans le menu ***Curve***, et choisissez ***Interpolation Mode***: vous avez le choix entre ***Constant***, ***Linear*** et ***Bezier***. Il s'agit ici de spécifier à Blender s'il doit rejoindre les points créés par des plateaux horizontaux (***Constant***), des segments de droite (***Linear***) ou une courbe de Bezier (***Bezier***), qui est l'option par défaut. Généralement, vous choisissez la première option pour un mouvement très saccadé,la deuxième pour un mouvement mécanique parfaitement asservi, et la troisième pour un mouvement souple, ou légèrement amorti.

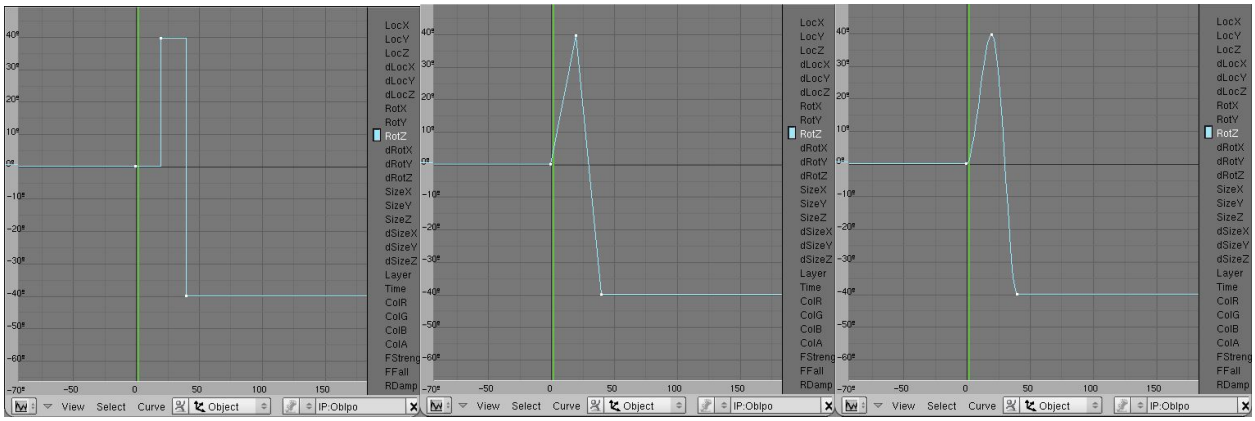

Figure 03: les différents modes d'interpolation: Constant, Linear et Bezier

Toujours dans le menu ***Curve***, intéressez-vous désormais à ***Extend Mode***: vous pouvez choisir entre ***Constant***, ***Extrapolation***, ***Cyclic*** et ***Cyclic Extrapolation***. La première option maintient la dernière valeur de la portion d'*Ipo* constante dans le temps, ce qui en quelque sorte, permet de « geler » cette valeur. La deuxième, ***Extrapolation***, permet de « projeter » la courbe de sorte à ce que la variation de la valeur soit maintenue au-delà du dernier point (voir Figure 04).

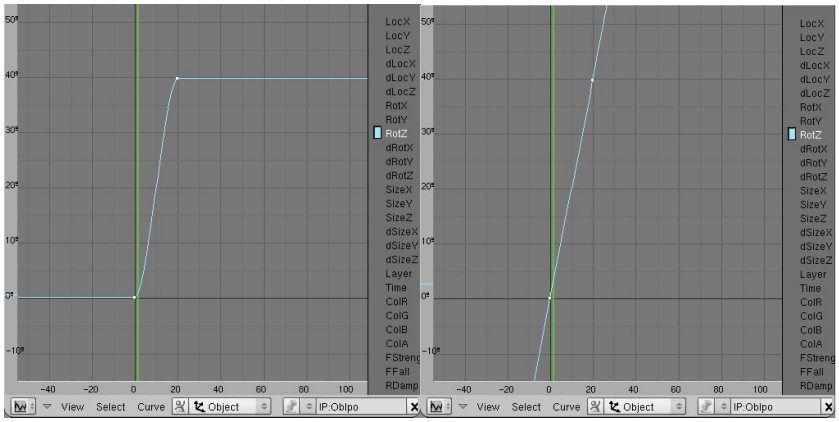

Figure 04: les modes d'extension Constant et Extrapolation

La troisième option, ***Cyclic***, permet de répéter indéfiniment la portion de courbe déjà décrite. Bien sûr, si l'ordonnée du dernier point ne coïncide pas avec l'ordonnée du premier, la courbe est alors discontinue, mais cela ne posera pas de problème particulier à Blender. Idem avec l'option ***Cyclic Extrapolation***, qui projette à l'infini la tendance de la portion de courbe (voir Figure 05).

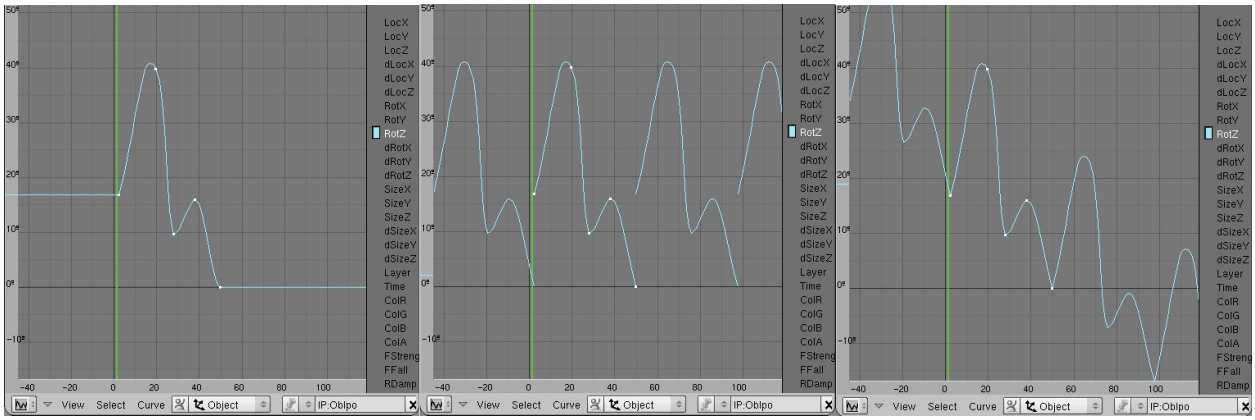

Figure 05: les modes d'extension Constant, Cyclic et Cyclic Interpolation

2. Le module IPO

Après cette introduction aux courbes *Ipo* directement au-travers de Blender, nous devrions nous sentir à l'aise pour tenter d'en créer une première au-travers de l'API Python de Blender. Lancez une session de Blender, ne touchez rien à la scène par défaut (en particulier, laissez le cube sélectionné), et partagez l'espace de travail de façon à avoir une fenêtre pour le Text Editor, une fenêtre pour l'Ipo Curve Editor et une dernière pour la vue 3D.

2.1 Création d'un bloc Ipo

Mais comme chaque module avant son usage, il convient d'importer le module *Ipo*, généralement dans la deuxième ligne de votre script:

```
01:    import Blender
02:    from Blender import Ipo
```

Pour le reste, nous allons travailler sur la base de la scène par défaut qui s'affiche normalement lorsque vous lancez une nouvelle session de Blender. Cette scène comporte un cube, une lampe et une caméra, et dans ce cadre, nous allons créer notre première courbe *Ipo*. L'objectif sera de faire tourner le cube de la scène autour de lui-même, de façon continue et infinie. Pour y parvenir, il nous faut faire appel au module **`Object`** qui, seul, permet de manipuler les paramètres de transformation (position, rotation, échelle) du cube. Nous allons donc modifier la deuxième ligne de notre script:

```
02:    from Blender import Ipo, Object
```

Maintenant, nous allons créer un bloc *Ipo* grâce à la fonction **`New()`**, et l'assigner à une variable. Cela se fait de la façon suivante: **`[variable.ipo] = Ipo.New('[arg1]','[arg2]')`**. Le premier argument permet de définir le type de bloc *Ipo* souhaité, parmi **`Object`**, **`Camera`**, **`World`**, **`Material`**, **`Texture`**, **`Lamp`**, **`Action`**, **`Constraint`**, **`Sequence`**, **`Curve`** et **`Key`**. Le second argument est une simple chaîne de caractères permettant de nommer cette courbe. Enfin, la variable à laquelle est assignée la courbe peut porter n'importe quel nom:

```
03:    ipo = Ipo.New('Object','CubeIpo')
```

2.2 Attribution du bloc Ipo à un objet

La suite part du principe que le cube est sélectionné dans Blender au moment où vous utilisez la combinaison de touches **`[Alt]+[P]`** pour lancer le script. Nous allons donc dans un premier temps utiliser la fonction **`GetSelected()`** du module **`Object`** pour mettre dans une liste tous les objets sélectionnés:

```
04:    object_list = Object.GetSelected()
```

Puis, dans un second temps, extraire le premier objet de la liste (portant l'indice **`[0]`**) et lui affecter un nom de

variable particulier:

```
05:    cube = object_list[0]
```

Nous pouvons désormais travailler librement avec la variable **cube**, qui fait référence à l'objet du même nom, dans la scène de départ par défaut. Utilisons maintenant la méthode **setIpo()** afin d'assigner le bloc **ipo** à la variable **cube**.

```
06:    cube.setIpo(ipo)
```

2.3 Création de la courbe Ipo

Une fois ceci fait, nous choisissons quel canal du bloc *Ipo* affectera le cube. Nous souhaitons faire tourner le cube sur lui-même, nous choisissons donc le canal **RotZ**, et utilisons la méthode **addCurve()** pour ajouter au bloc **ipo** déjà existant la courbe correspondante. Nous l'affectons ensuite à une variable arbitrairement nommée **rtz**:

```
07:    rtz = ipo.addCurve('RotZ')
```

La ligne suivante utilise la méthode **setInterpolation()** qui nous permet de définir le type d'interpolation entre deux points de la courbe **rtz**, au choix parmi **'Constant'**, **'Bezier'**, ou **'Linear'**. Dans la mesure où nous souhaitons une rotation parfaitement continue, nous choisirons la dernière option, **'Linear'**:

```
08:    rtz.setInterpolation('Linear')
```

Nous souhaitons également que le mouvement soit infini, il faudrait donc l'extrapoler indéfiniment. C'est toute l'utilité de la méthode **setExtrapolation()** qui permet de déterminer le comportement de la courbe **rtz** au-delà des bornes de définition, avec pour options **'Constant'**, **'Extrapolation'**, **'Cyclic'** ou **'Cyclic_extrapolation'**. Dans notre cas, nous choisirons **'Extrapolation'**:

```
09:    rtz.setExtrapolation('Extrapolation')
```

Nous avons maintenant défini le bloc *Ipo*, le canal affecté, le type de courbe. Mais la courbe en elle-même n'est pas encore définie! Il faut lui spécifier des points de passage; dans notre exercice, deux points seront suffisants. Le premier représentera la position angulaire (0°) de notre cube à la frame 1, et il sera défini grâce à la méthode **addBezier()**. La première coordonnée indique la frame d'appartenance du point, la seconde la valeur du canal pour cette frame. Attention, pour les rotations, une unité de valeur vaut 10°. Par conséquent, 1 vaut 10°, 4,5 vaut 45°, 9 vaut 90° et ainsi de suite.

```
10:    rtz.addBezier((1,0))
```

Nous souhaitons qu'en une seconde, le cube fasse un quart de tour sur lui-même. Une seconde étant égale à 25 frames (ceci est paramétrable dans le panneau ***Format*** des boutons de rendu (***Render buttons***) du menu ***Scene*** (**[F10]**)), nous spécifierons notre deuxième point à la frame n°26, et lui donnerons une valeur égale à 9 (équivalent à 90°, soit un quart de tour).

```
11:    rtz.addBezier((26,9))
```

Il ne nous reste plus qu'à tester notre script, en utilisant la désormais classique combinaison de touches **[Alt]+[P]** avec le curseur de la souris dans la fenêtre ***Text Editor***. La fenêtre ***Ipo Curve Editor*** se met immédiatement à jour, une courbe bleue apparaissant pour le canal ***RotZ***. En outre, le champ **IP:** dans la barre de menu de cette dernière fenêtre indique bien **CubeIpo**, confirmant le fonctionnement apparent de notre script.

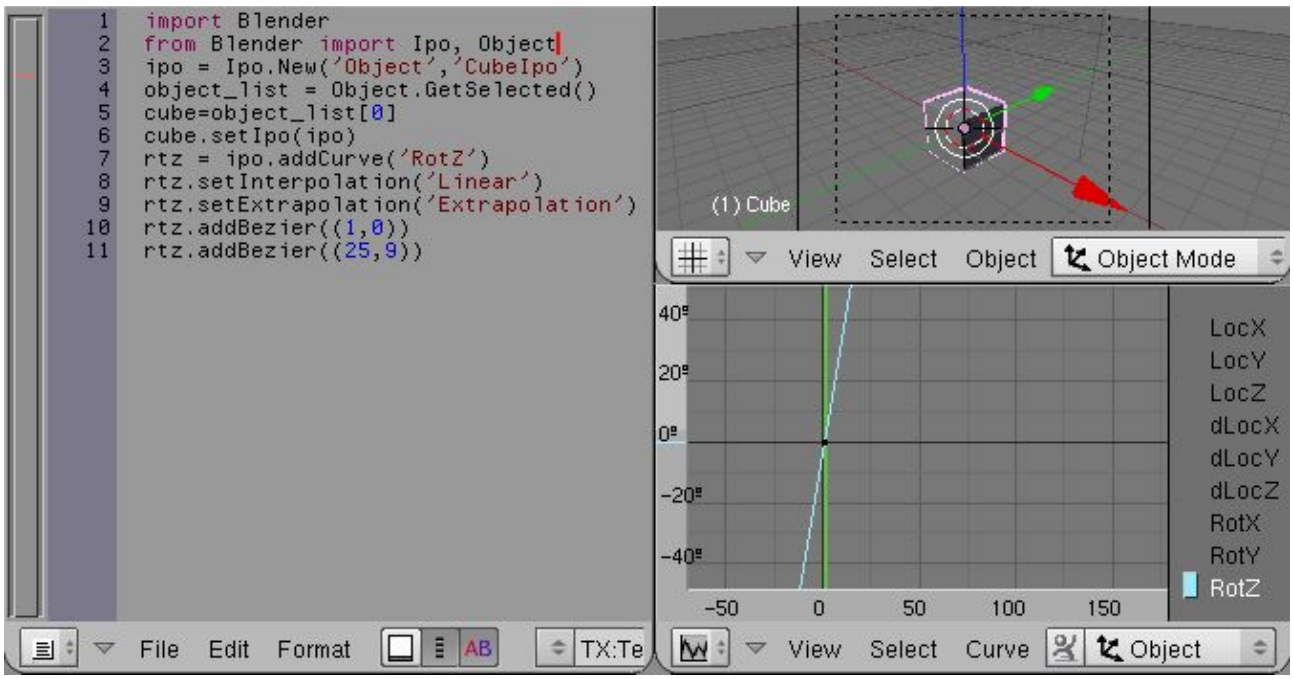

Figure 06: notre petit script en action!

Mais il n'y a qu'une façon certaine de vérifier que tout se passe comme souhaité: la prévisualisation de l'animation! Déplacez le curseur de la souris au-dessus d'une ***vue 3D*** et appuyez sur la touche **[0]** pour afficher la vue depuis la caméra. La combinaison de touches **[Alt]+[A]** permet de jouer l'animation en temps réel dans la ***vue 3D*** dans laquelle se trouve le curseur de la souris. Et là, nous constatons que tout fonctionne!

2.4 Résumé de notre code

```
01:    import Blender
02:    from Blender import Ipo
03:    ipo = Ipo.New('Object','CubeIpo')
04:    object_list = Object.GetSelected()
05:    cube = object_list[0]
06:    cube.setIpo(ipo)
07:    rtz = ipo.addCurve('RotZ')
08:    rtz.setInterpolation('Linear')
09:    rtz.setExtrapolation('Extrapolation')
10:    rtz.addBezier((1,0))
11:    rtz.addBezier((26,9))
```

3. Autres paramètres

Vous l'avez sans doute compris, les informations liées aux *Ipo* sont hiérarchisées sur plusieurs niveaux, chaque niveau correspondant à une Classe Python. Tout en haut de l'échelle se trouve la classe ***Ipo***, qui permet de créer des objets *Ipo*; elle permet en particulier de spécifier le type de l'*Ipo* (**`Object`**, **`Camera`**...) et d'activer autant de courbes que de canaux existants pour un type donné (par exemple, **`LocX`**, **`RotZ`**... pour un type **`Object`**). Mais ces courbes sont pour l'instant vierges, elles ne seront spécifiées que grâce à la classe ***IpoCurve***, immédiatement inférieure dans l'échelle hiérarchique. Cette classe permet de déterminer l'allure des courbes en définissant les modes d'**`Interpolation`** et d'**`Extrapolation`** des courbes *Ipo*, mais aussi d'ajouter ou supprimer des noeuds à ces courbes. En effet, les courbes sont constituées de noeuds, et la classe la plus basse dans l'échelle hiérarchique, la classe ***BezTriple***, permet d'agir sur les coordonnées de ces noeuds.

Dans la deuxième partie de cet article, lors de l'écriture de notre script d'exemple, nous nous sommes servi de quelques fonctions et méthodes issues des deux classes supérieures. Par exemple, la fonction `New()` et la méthode `addCurve()` font partie de la classe ***Ipo***, tandis que les méthodes `setIntepolation()`, `setExtrapolation()` et `addBezier()` font partie de la classe ***IpoCurve***. N'ayant pas opéré d'action au niveau des noeuds (à part leur création au niveau de la classe ***IpoCurve***), la classe ***BezTriple*** n'a pas été mise à contribution.

Nous allons maintenant explorer les principales méthodes, en particulier celles passées sous silence. Nous ne chercherons pas à être exhaustif, seulement à présenter les plus courantes d'entre elles, ce qui nous limitera essentiellement aux classes Ipo et IpoCurve.

3.1 Classe Ipo

Les méthodes `addcurve([nom])` et `delCurve([nom])`: la première méthode permet d'ajouter une nouvelle courbe et de la nommer, tandis que la seconde permet de désigner une courbe dont le nom est connu, et de la supprimer. Par exemple:

```
12:    ipo.delCurve('RotZ')
```

permet d'effacer la courbe créée en ligne 07 de notre script.

La méthode `EvaluateCurveOn([position courbe],[frame])`: cette méthode permet de choisir l'une des courbes en fonction de sa position dans l'*Ipo* (un peu comme s'il s'agissait de son indice), et pour une valeur de temporelle `[frame]` donnée, de demander à Blender d'évaluer la valeur de la courbe ce moment. Par exemple:

```
12:    angle = ipo.EvaluateCurveOn(0,25)
13:    print angle
```

permet d'afficher la valeur de la courbe numéro 0 (la première créée) à la frame 25. Pour résultat de la commande `print`, la console nous renvoie la valeur 9.0, soit 90°.

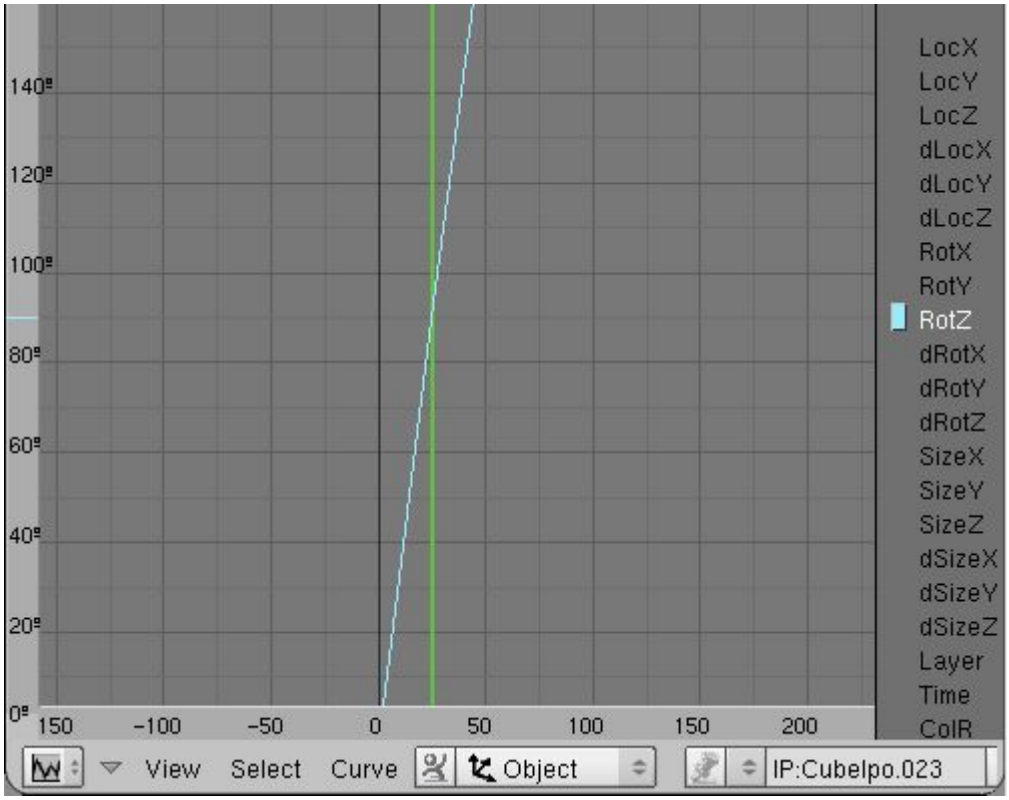

Figure 07: pour la frame n°50, RotZ a effectivement une valeur de 90°

Les méthodes `getBlocktype()` et `setBlocktype([type])`: la première méthode permet de lire le type d'*Ipo* d'une courbe donnée, tandis que la seconde permet de modifier celui-ci! Attention, l'usage de cette dernière méthode n'est recommandée que si vous savez rigoureusement ce que vous faîtes, car les résultats peuvent vite devenir imprévisibles. De plus, `getBlocktype()` vous retournera une valeur numérique entière, au lieu d'un nom lisible.

Les méthodes `getCurve([nom])` et `getCurves()`: la première méthode permet de sélectionner une courbe dont vous connaissez le nom, tandis que `getCurves()` permet de sélectionner toutes les courbes disponibles, généralement pour les placer dans une liste.

Les méthodes `getName()` et `setName([nom])`: la première méthode permet de lire le nom de l'*Ipo* (celui que l'on lit dans le bouton `IP:` de la barre de menu de l'***Ipo Curve Editor***), tandis que la seconde permet de changer celui-ci. Par exemple:

```
12:    iponame = ipo.getName()
13:    print iponame
```

retournera la chaîne de caractères `CubeIpo` (spécifié en ligne 03 de notre script) dans la console.

La méthode `getNcurves()`: cette méthode permet de lire le nombre de courbes associées au bloc *Ipo*.

3.2 Classe IpoCurve

Les méthodes `addBezier([coordonnées])` et `delBezier([numéro])`: la première méthode permet d'ajouter un noeud à la courbe de Bezier, sous forme de couple `([frame],[valeur])`. La seconde permet de supprimer un point en particulier: il suffit d'indiquer comme paramètre le numéro du point, en commençant par 0. Par exemple:

```
12:    rtz.delBezier(1)
13:    rtz.addBezier((50,9))
```

permet d'effacer le point numéro 1 soit le deuxième créé, dans le sens croissant des abscisses) et d'en créer un autre aux coordonnées frames = 50 et valeur = 9. Il en résulte une pente de la droite *Ipo* `RotZ` moitié moindre que précédemment.

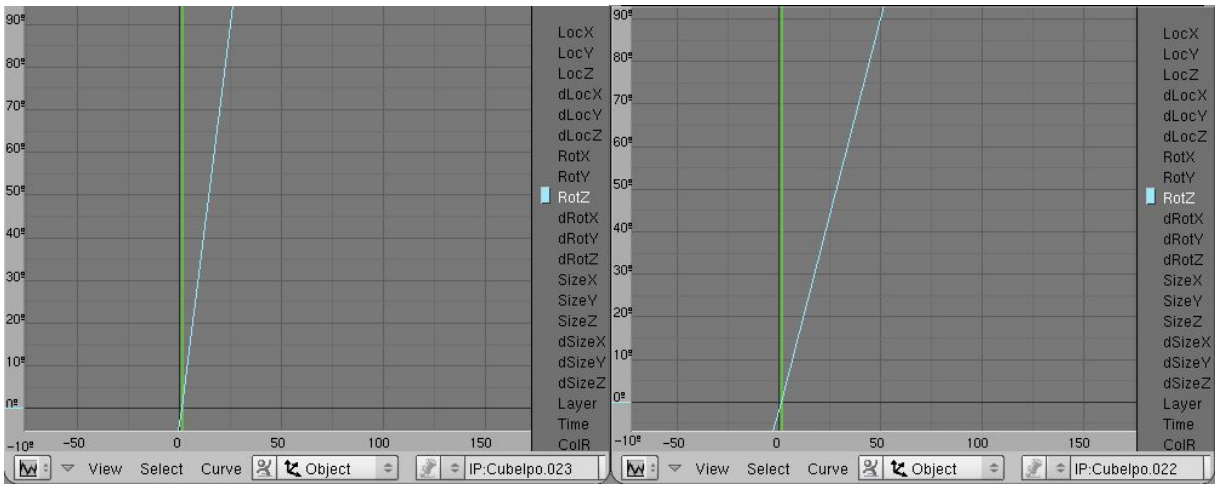

Figure 08: la deuxième version présente une pente deux fois plus faible; la rotation du cube est donc deux fois plus lente!

Les méthodes `getExtrapolation()` et `setExtrapolation([mode])`: la première méthode permet de lire le mode d'extrapolation d'une courbe, tandis que la seconde permet de spécifier (ou changer) le mode d'extrapolation de la courbe, avec un argument parmi les suivants: `Constant`, `Extrapolation`, `Cyclic` ou `Cyclic_extrapolation`.

Les méthodes `getInterpolation()` et `setInterpolation([mode])`: la première méthode permet de lire le mode d'interpolation d'une courbe, tandis que la seconde permet de spécifier (ou changer) le mode

d'interpolation de la courbe, avec l'un des arguments parmi les suivants: `Constant`, `Bezier`, ou `Linear`.

La méthode `getName()`: elle permet de lire le nom du canal auquel la courbe est associée. Par exemple, `LocX`, `RotZ`, `Energ` et autres.

3.3 Classe BezTriple

Vous me pardonnerez d'être discret sur les méthodes de la classe ***BezTriple***, la manipulation des noeuds et poignées étant tout sauf intuitive sous forme de code. En effet, les noeuds des courbes de Bezier sont constitués d'un point central et de deux poignées; les vecteurs formés entre le point et chacune des poignées constituent les tangentes à la courbe de Bezier, ce qui permet de piloter précisément la forme de la courbe au niveau du noeud. Dans Blender, vous avez à votre disposition une interface graphique et des outils d'édition de courbes très puissants. Il vous est ainsi facile de choisir un noeud Bezier, et d'éditer individuellement les poignées et le point central. Par exemple, sur la Figure 09, en mode Edition, les poignées du dernier noeud ont été déplacées de sorte que la tangente à la courbe ne soit pas horizontale, comme c'est le cas pour les autres noeuds.

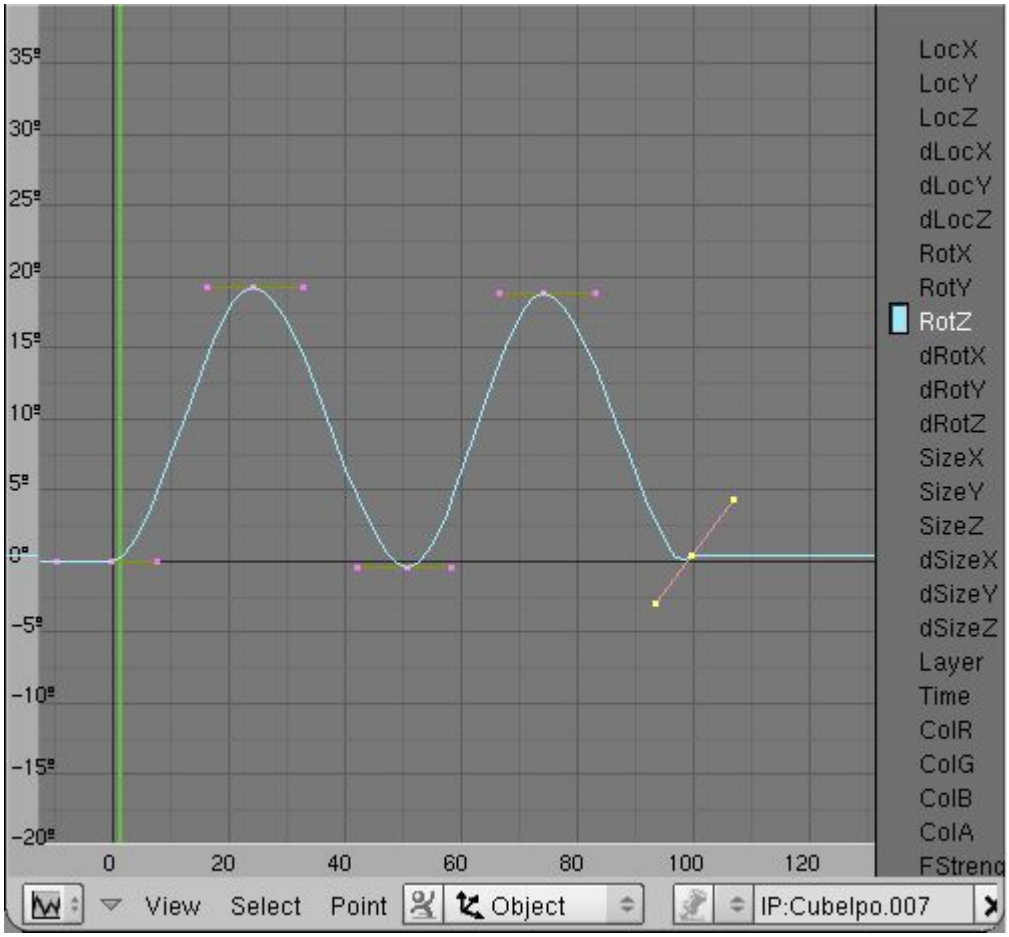

Figure 09: les poignées du dernier noeud ont été réorientées

Mais la complexité ne s'arrête pas là, car Blender propose différents type de poignées: ***Aligned***, ***Free*** et ***Vector***:

- **Aligned:** si vous déplacez la seconde poignée, Blender déplace en temps réelle la première poignée, de sorte que le point central et les deux poignées soit toujours placés sur un même segment. C'est le cas de figure présenté sur l'illustration précédente.
- **Free:** vous pouvez déplacer indépendamment chacune des deux poignées, chaque segment (formé à partir du point central) sera une tangente à laquelle la courbe se conformera.

- **Vector:** avec le point central immobile, la première poignée sera toujours orientée vers les précédent point central sur la courbe, et la seconde poignée toujours orientée vers le prochain point central de la courbe.

Les méthodes `getPoints()` et `setPoints([coordonnées])`: la première permet de relever les coordonnées du noeud (le point central), tandis que la seconde permet d'en spécifier les nouvelles coordonnées.

La méthode `getTriple()`: cette méthode permet de relever les coordonnées x, y et z de chacune des poignées et du point central.

Il n'est pour l'instant donc pas prévu de donner un accès en écriture aux coordonnées des poignées, seulement de les relever au-travers de la méthode `getTriple()`.

3.4 Méthodes connexes

Il ne s'agit pas à proprement parler de méthodes issues du module ***Ipo***, mais du module ***Object***. Elles sont en effet indispensables pour lier une *Ipo* à un objet donné, et nous avons jugé opportun d'en parler ici.

Les méthodes `getIpo()` et `setIpo([nom])`: la première méthode retourne l'*Ipo* associée à l'objet, tandis que la seconde assigne à l'objet l'*Ipo* identifiée par la variable `[nom]`.

La méthode `clearIpo()`: cette méthode permet de débarrasser un objet donné de toute *Ipo* qui lui serait rattachée. Par exemple:

```
12:    cube.clearIpo()
```

détache de l'objet `cube` l'*Ipo* `CubeIpo` créée en ligne 03 de notre script. Si l'Ipo n'est plus liée à l'objet, elle existe toutefois toujours par blender, et peut être réutilisée, par exemple pour un autre objet.

Le script suivant fait appel à un mélange de ces méthodes. Il a pour effet de récupérer tous les objets de la scène, d'assigner la courbe *Ipo* en première instance à l'objet `cube`, puis de la détacher de celui-ci pour la lier finalement à l'objet `lamp`.

```
01:    import Blender
02:    from Blender import Ipo, Object
03:    ipo = Ipo.New('Object','CubeIpo')
04:    object_list = Object.Get()
05:    camera=object_list[0]
06:    cube=object_list[1]
07:    lamp=object_list[2]
08:    cube.setIpo(ipo)
09:    rtz = ipo.addCurve('RotZ')
10:    rtz.setInterpolation('Linear')
11:    rtz.setExtrapolation('Extrapolation')
12:    rtz.addBezier((1,0))
13:    rtz.addBezier((25,9))
14:    cube.clearIpo()
15:    lamp.setIpo(ipo)
```

4. Conclusions

Nous venons de voir que la définition de courbes *Ipo* au travers de l'API Python de Blender ne présente pas de difficultés majeures, la principale subtilité concernant la hiérarchie implicite entre les **Ipo** et les ***IpoCurves***, qui déterminent donc l'ordre de définition des données et fonctions dans nos scripts. Pour le reste, le fonctionnement du module ***Ipo*** est un reflet presque parfait du bloc de données *Ipo* dans Blender, et ne devrait pas poser de problème insurmontable: si vous savez mettre en place une courbe *Ipo* dans Blender, vous savez désormais également la mettre en oeuvre avec Python.

A l'instant de l'écriture de ces lignes, une version *Release Candidate* de Blender v2.40 vient d'être mise en ligne par la *Blender Foundation*. Lorsque vous les lirez, il est probable que la version finale soit enfin disponible en téléchargement. Toutefois, il risque de s'écouler un certain temps avant que la documentation officielle de l'API Python de Blender soit mise à jour. Vous trouverez malgré tout de nombreuses informations juteuses quant aux nouveautés relatives à Python sur la page suivante: **http://www.blender.org/documentation/237PythonDoc/index.html**. Inutile de préciser qu'il s'agit d'une lecture fortement conseillée!

5. Liens

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

La documentation officielle de python pour Blender v2.37a:
http://www.blender.org/documentation/237PythonDoc/index.html

La documentation officieuse de python pour Blender v2.40(RC):
http://members.iinet.net.au/~cpbarton/ideasman/BPY_API/

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