Exploration du Module Material (2ème partie) et jeu avec les fichiers

Merci à Diamond Editions pour son aimable autorisation pour la mise en ligne de cet article, initialement publié dans Linux Magazine N°78

Olivier Saraja -olivier.saraja@linuxgraphic.org

A ce jour, la version 2.40 de Blender se dessine plus précisément chaque jour, mais la documentation de l'API Python accuse du retard et n'est pour l'instant conforme qu'à la version 2.37 de Blender. Dès Blender 2.38 pourtant, de nouvelles fonctions et méthodes étaient disponibles, sans qu'elles ne soient présentées dans l'article « Exploration des modules Material et Texture de l'API Python de Blender » de GNU/Linux Magazine #75 (en ligne sur http://www.linuxgraphic.org) fidèle à la version « courante » de la documentation.

Nous réparerons aujourd'hui cette lacune du précédent article, en recensant la plupart des nouvelles méthodes et en montrant un exemple concret d'usage. Nous ne reviendrons pas sur les bases acquises lors de ce premier article. Nous approfondirons en revanche notre apprentissage des possibilités de *scripting*, grâce à un modeste exercice où il s'agira d'écrire (puis de lire) des données dans un fichier texte créé au-travers de l'API Python de Blender.

1. Nouvelles Méthodes liées à l'onglet Mirror Transp

Dans cette partie, il s'agira de compléter la liste des méthodes permettant d'utiliser les options de *Ray Transparency* de Blender. Ces options sont facilement identifiables dans l'onglet ***Mirror Transp*** du menu ***Shading*** (**[F5]**), ***Material Buttons***.

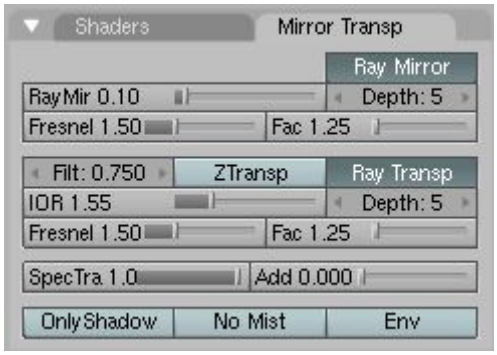

Figure 01: l'onglet Mirror Transp

Pour rappel, il vous faut au minimum importer dans Blender le module Material. Cela se traduit par les deux premières lignes de code de votre script au moins égales aux suivantes:

```
01:    import Blender
02:    from Blender import Material
```

1.1 setMode()

L'usage de cette méthode permet d'activer certaines options relatives aux *shaders*, en particulier les propriétés de réflexion ou de transparence des matériaux. En particulier, la prise en compte de la transparence par le *raytracer* intégré au moteur de rendu se fait par la transmission du paramètre

'RayTransp' à la méthode **setMode()**. Par exemple:

```
[material.data].setMode('Traceable', 'Shadow', 'Radio', 'RayMirr', 'RayTransp')
```

permet d'activer (en plus des options généralement activées par défaut) les reflets (**'RayMirr'**) et la transparence (**'RayTransp'**).

1.2 setFilter()

Il s'agit là d'une option particulièrement intéressante pour la simulation de matériaux de type verre coloré. En effet, la transparence est traditionnellement gérée par un paramètre **Alpha**, qui définit la proportion selon laquelle la couleur de l'objet transparent se mélange avec les couleurs de l'arrière-plan. Le problème est que le verre ne teinte alors l'arrière-plan que proportionnellement à son opacité, ce qui revient à dire que plus **Alpha** est faible, moins la couleur du verre est visible.

C'est là que l'option **Filter** (disponible depuis Blender v2.37) entre en jeu pour corriger cela, et apporter à vos matériaux de verre un réalisme accru. Plus la valeur **Filter** sera élevée, plus votre « verre » sera « teinté dans la masse ». Voici quelques exemples de valeurs de **Filter** pour illustrer son fonctionnement:

Figure 02: des valeurs de Filter respectivement égales à 0.10, 0.50 et 1.00

La méthode permettant de régler ce paramètre est tout simplement de la forme **setFilter([valeur])** , et prend comme argument une valeur comprise entre **0.00** et **1.00**. La méthode **getFilter()** permet de récupérer la valeur ***Filter*** d'un matériau. Par exemple:

```
mat.setFilter(0.75)
```

1.3 setIOR()

Cette méthode permet de déterminer l'indice de réfraction du matériau, c'est-à-dire la déformation de la trajectoire d'un rayon lumineux au-travers de l'objet transparent. La valeur par défaut est 1.00, c'est à dire qu'aucune distorsion du trajet des rayons lumineux n'intervient: en regardant au-travers de l'objet, vous voyez exactement ce qu'il y a derrière, sans effet de déformation. Une valeur conventionnelle pour l'eau est 1.33 et pour le verre, 1.51.

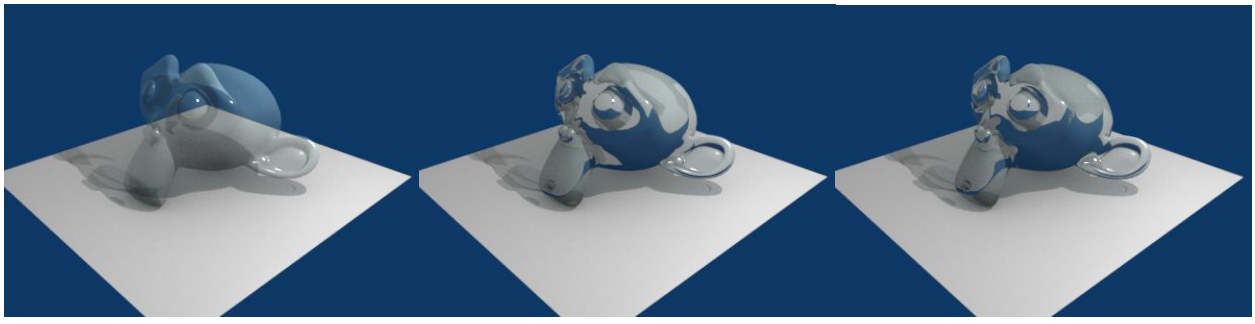

Figure 03: avec des valeurs IOR de 1.00, 1.33 et 1.51, nous avons ici les équivalents virtuels du plastique, de l'eau et du verre

La méthode permettant de régler ce paramètre est de la forme **setIOR()**, et accepte des arguments variant

entre **1.00** et **3.00**. La méthode **getIOR([valeur])** permet de récupérer la valeur ***IOR*** du matériau. Par exemple:

```
mat.setIOR(1.51)
```

1.4 setTransDepth()

L'un des principaux inconvénient du *raytracing*, c'est que de nombreux rayons sont lancés. Par exemple, pour les reflets, un rayon peut rebondir indéfiniment d'une surface réfléchissante à l'autre, entraînant des temps de calcul infinis. Un mécanisme itératif permet de mettre fin aux calculs après un certain nombre de rebonds, définis à l'avance au-travers du paramètre **Depth** de la section ***RayMirr***. Le principe est le même pour ***RayTransp***: il vous faut déterminer le nombre maximum de surfaces transparentes traversées par un rayon donné. ***RayTransp*** dispose donc de son propre paramètre **Depth**. S'il est trop faible, vous pourrez observer dans vos objets transparents (en fonction de la complexité de leur forme ou leur nombre) des tâches noires; il vous suffit alors de l'augmenter graduellement jusqu'à la disparition totale des tâches noirs.

La méthode **setTransDepth([valeur])** vous permet donc de déterminer la « profondeur itérative » des rayons traversant des objets transparents, et accepte comme argument un entier compris entre **0** et **10**. La méthode **getTransDepth()** permet pour sa part de récupérer la valeur **Depth** d'un matériau. Par exemple:

```
mat.setTransDepth(10)
```

1.5 setFresnelTrans()

L'effet Fresnel est un phénomène qui modifie les propriétés de réflexion ou de transparence de l'objet en fonction de l'angle formé entre le « vecteur » de la caméra et la normale à la surface de l'objet affecté. Par exemple, avec une valeur de **Fresnel** en réflexion élevée, une boule théoriquement totalement réfléchissante, ne sera réfléchissante que sur ses « bords ». Dans le même ordre d'idée, avec une valeur de **Fresnel** en transparence élevée, une boule totalement transparente sera plus opaque sur les « bords » et plus transparente « au centre ». Ce phénomène est plus facilement observable sur les images qui suivent lorsque l'indice de réfraction du matériau est proche de **1.00**. L'**Alpha** est en revanche égal à **0.20**, c'est à dire qu'il présente une légère opacité pour les besoins de la démonstration, et l'on voit clairement que l'effet Fresnel intensifie la transparence de l'objet au « centre » de celui-ci.

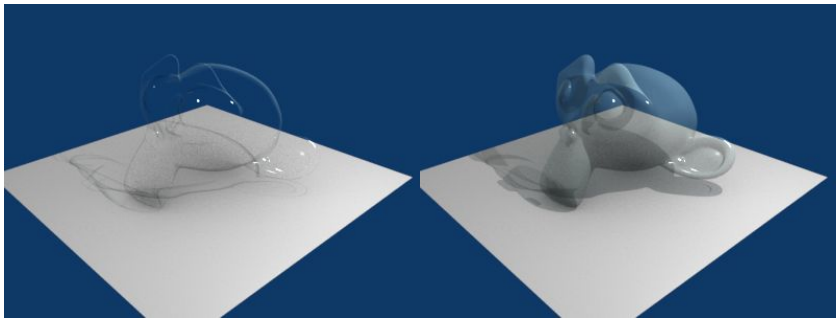

Figure 04: avec une valeur de Fresnel de 5.0 (à gauche), l'objet à l'air plus transparent qu'avec une valeur de 0.0 (à droite) mais conserve une opacité cohérente sur les « bords »

La méthode **setFresnelTrans([valeur])** permet donc de déterminer la force de l'effet Fresnel sur la valeur de transparence de votre objet, et accepte des valeurs comprises entre **0.0** et **5.0**. Bien évidemment, la méthode **getFresnelTrans()** permet de récupérer la valeur ***Fresnel*** du matériau transparent. Par exemple:

```
mat.setFresnelTrans(3.50)
```

1.6 setFresnelTransFac()

Le Facteur de Fresnel détermine tout simplement l'intensité de la transition entre les surfaces opaques et transparentes pilotées par l'effet Fresnel. Une valeur élevée indique une transition très brutale, et donc une intensification de la transparence de l'objet sauf sur les bords les plus extrêmes.

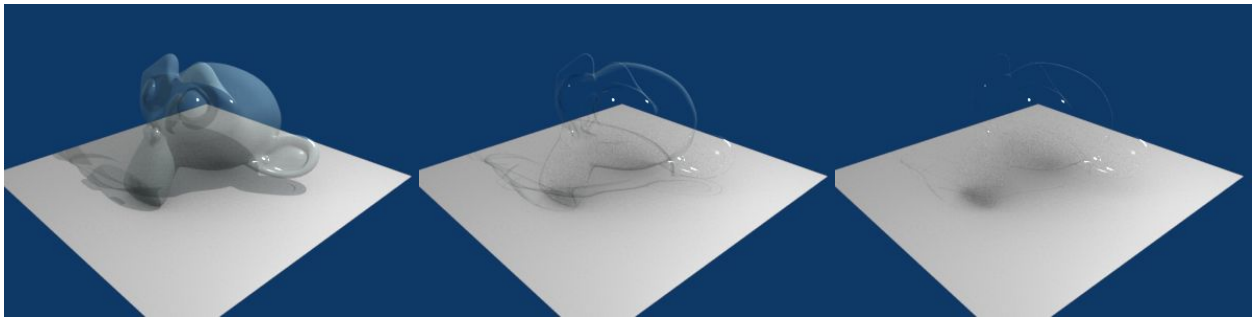

Figure 05: Facteur 1.00 (à gauche), Facteur 1.25 (au centre) et Facteur 2.0 (à droite), la transparence s'accentue, même au-delà du paramètre Alpha

La méthode **`setFresnelTransFac([valeur])`** permet donc de déterminer le facteur de l'effet Fresnel , et accepte des valeurs comprises entre **`1.0`** et **`5.0`**. Bien évidemment, la méthode **`getFresnelTransFac()`** permet de récupérer la valeur ***Fresnel*** du matériau transparent. Par exemple:

```
mat.setFresnelTransFac(2)
```

2. Nouvelles méthodes liées aux shaders

De nouvelles méthodes ont également été implémentées pour exercer un certain contrôle sur les nouveaux shaders de Blender. Diffuse shaders: Minnaert, Oren-Nayar, Toon. Specular shaders: WardIso, Blinn, Toon.

2.1 setDiffuseShader()

Cette méthode permet de déterminer le type de *Diffuse Shader* du matériau. Il admet des valeurs entières comprises entre 0 et 3, chaque valeur correspondant à un *shader* différent:

```
0: Lambert
1: Oren-Nayar
2: Toon
3: Minnaert
```

Par exemple, si vous souhaitez effectuer un rendu toonesque, c'est-à-dire obtenir un *Diffuse Shader* **`Toon`**, vous utiliserez par exemple:

```
mat.setDiffuseShader(2)
```

Bien sûr, la méthode **`getDiffuseShader()`** vous permet au contraire de connaître le type de Diffuse Shader de votre matériau.

2.2 setSpecShader()

Dans la même logique, il vous est possible de déterminer le type de *Specular Shader* du matériau grâce à la méthode **`setSpecShader()`**, qui admettent des valeurs entières comprises entre 0 et 4:

```
0: CookTorr
1: Phong
2: Blinn
3: Toon
4: WardIso
```

Pour en revenir à notre exemple de rendu toonesque, vous utiliseriez donc:

```
mat.setSpecShader(3)
```

et **`getSpecShader()`** pour connaître le type de *Specular Shader* de votre matériau.

2.3 Les méthodes liées aux Diffuse Shaders

Certains *Diffuse Shaders* ont des paramètres qui leur sont propres. Le tableau qui suit récapitule les méthodes disponibles, en précisant le *Diffuse Shader* concerné, et les intervalles de valeurs numériques possibles. Par soucis d'exhaustivité, le tableau contient également les méthodes relatives au *shader* par défaut (Lambert) et qui sont incidemment partagée par tous les *Diffuse Shaders*.

Méthode de type set...	Diffuse Shader concerné	Nom du bouton dans Blender	Intervalles de valeurs possibles	Méthode de type get...
`SetDiffuseDarkness([val])`	Minnaert	Dark	de 0.0 à 2.0	**`getDiffuseDarkness()`**
`setRoughness([val])`	Oren-Nayar	Rough	de 0.0 à 3.1	**`getRoughness()`**
`setDiffuseSmooth([val])`	Toon	Smooth	de 0.0 à 1.0	**`getDiffuseSmooth()`**
`setDiffuseSize([val])`	Toon	Size	de 0.0 à 3.14	**`getDiffuseSize()`**
`setRef([val])`	Lambert, Oren-Nayar, Toon, Minnaert	Ref	de 0.0 à 1.0	**`getRef()`**

2.4 Les méthodes liées aux Specular Shaders

De la même façon que nous avons recensé dans le tableau précédent tous les *Diffuse Shaders* disponibles, vous trouverez ci-dessous les méthodes relatives à tous les *Specular Shaders*.

Méthode de type set...	Diffuse Shader concerné	Nom du bouton dans Blender	Intervalles de valeurs possibles	Méthode de type get...
`setRms([val])`	WardIso	rms	de 0.0 à 0.4	**`getRms()`**
`setRefracIndex([val])`	Blinn	Refr	de 0.0 à 3.1	**`getRefracIndex()`**
`setSpecSmooth([val])`	Toon	Smooth	de 0.0 à 1.0	**`getSpecSmooth()`**
`setSpecSize([val])`	Toon	Size	de 0.0 à 1.53	**`getSpecSize()`**
`setSpec([val])`	CookTorr, Phong, Blinn, Toon, WardIso	Ref	de 0.0 à 2.0	**`getSpec()`**
`SetHardness([val])`	CookTorr, Phong, Blinn	Hard	de 1 à 511	**`getHardness()`**

Par exemple:

```
matToon.setRGBCol([1.0,0.0,0.0])
matToon.setRef(0.8)
```

```
matToon.setDiffuseSmooth(0.25)
matToon.setDiffuseSize(1.0)
matToon.setSpec(0.35)
matToon.setSpecSmooth(0.15)
matToon.setSpecSize(0.05)
matToon.setDiffuseShader(2)
matToon.setDiffuseShader(3)
```

2.5 setTranslucency()

Ce paramètre détermine le niveau de translucidité du matériau. La translucidité est un phénomène qui intervient surtout sur des matériaux très fin comme une membrane, ou un abat-jour, ou une robe, ou encore certaine matière pouvant véhiculer une lumière interne, comme la cire ou la porcelaine: l'objet laisse passer une partie de la lumière, de sorte que même une face non exposée à la lumière s'en retrouve légèrement éclairée.

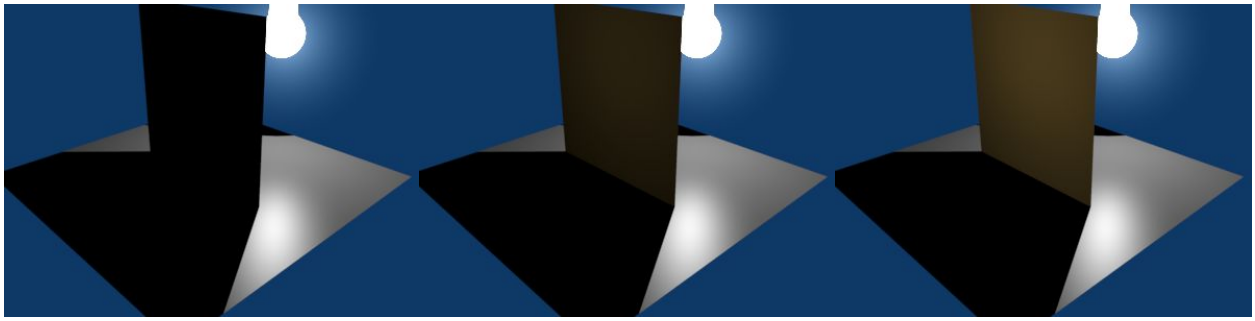

Figure 06: l'influence de la translucidité sur l'illumination d'un objet: de gauche à droite, des valeurs de Translucency égales à 0.00, 0.20 et 0.35

La méthode **`setTranslucency([valeur])`** permet donc de déterminer la translucidité du matériau, et accepte des valeurs comprises entre **`0.0`** et **`1.0`**. Bien évidemment, la méthode **`getTranslucency()`** permet de récupérer la valeur ***Translucency*** du matériau. Par exemple:

```
mat.setTranslucency(0.35)
```

3. Manipulations de base sur les fichiers: le module Window

Nous nous contenterons aujourd'hui d'actions élémentaires, mais ô combien incontournables, la lecture et l'écriture dans des fichiers externes. La plupart du temps, ce sera pour vous une façon d'obtenir un état de votre script (si vous avez la flemme de regarder dans la console), ou de sauvegarder et rappeler ultérieurement des paramètres propres à votre script. Ce qui nous intéressera aujourd'hui sera toutefois la création d'un importeur/exporteur basique de matériaux écrit en Python.

3.1 La fonction FileSelector()

Il s'agit d'une fonction du Module Window de l'API Blender. Cela implique bien évidemment que les deux premières lignes de votre script, pour lequel vous souhaitez lire ou écrire dans un fichier externe, devront au minimum être:

```
01:    import Blender
02:    from Blender import Window
```

Cette fonction a pour objectif d'ouvrir dans la fenêtre courante le sélecteur de fichiers de Blender, et d'effectuer une action prédéfinie. Par exemple:

```
Window.FileSelector(sauveMateriau, « Sauvegarder le Matériau »)
```

va appeler le sélecteur de fichiers de Blender, vous proposer un bouton d'action ***Sauvegarder le Matériau***, et

appeler la fonction (normalement prédéfinie) **sauveMateriau()**.

Figure 07: exemple de la fonction fileSector(...) en action dans Blender, invoquée par Python

3.2 Ecrire dans un fichier

Supposons que nous avons récupéré le premier matériau du premier objet sélectionné de notre scène à l'aide des lignes suivantes:

```
01:    import Blender
02:    from Blender import Material, Object, Window
03:    obj=Object.GetSelected()[0]
04:    mat=Material.Get()[0]
```

Nous pouvons extraire du matériau n'importe quelle propriété dont nous avons découvert la méthode grâce à une ligne de type **mat.get...()** et la stoquer dans une variable. Par exemple, récupérons le nom du matériau:

```
05:    Name = mat.getName()
```

Maintenant, supposons le code qui suit:

```
06:    def sauveNom(fileName):
07:           global Name
08:           dataFile = open(fileName,"w")
09:           dataFile.write((Name)+"\n")
10:           dataFile.close()
```

En ligne **06**, nous déclarons une nouvelle fonction, qui utilise la variable **fileName** comme argument. La variable à sauvegarder est **Name**, définie en ligne **05**. Mais n'ayant pas été définie au sein de la fonction, pour en faire usage à l'intérieur de celle-ci, il faut la déclarer comme étant globale. En ligne **08**, nous ouvrons (**open()**) en mémoire le fichier symbolisé par la variable **fileName**, et le déclarons ouvert pour écriture (« **w** »).

En ligne **09**, nous écrivons effectivement (**write()**) la variable **Name** dans le fichier, et nous y ajoutons un saut de ligne pour bien séparer la donnée sauvegardée des données qui suivront éventuellement (**+« \n »**). Enfin, en ligne **10**, après enregistré dans le fichier toutes les variables qui nous intéressaient, nous n'avons plus qu'à clore le fichier **fileName**. Notre fonction de sauvegarde dans un fichier est désormais complète.

Nous allons maintenant l'appeler via le **FileSelector**, déjà introduit un peu plus tôt. Pour ce faire, considérons le bout de code suivant:

```
11:    Window.FileSelector(sauveNom, "Sauvegarder le Nom")
```

Le premier argument entre parenthèses permet d'appeler la fonction **sauveNom**, définie entre les lignes **06** à **10**. Le second permet de nommer le bouton de validation de l'opération (voir Figure 08).

Figure 08: écriture du fichier de sauvegarde

3.3 Lire à partir d'un fichier

Pour la lecture d'un fichier, la philosophie est globalement la même. En ligne **03**, nous définissons la variable Name, mais nous la laissons vierge (« »). La récupération de l'objet et du matériau à modifier se font très classiquement au-travers des lignes **04** et **05**.

```
01:    import Blender
02:    from Blender import Object, Material, Window
03:    Name = ""
04:    obj = Object.GetSelected()[0]
05:    mat = Material.Get()[0]
```

Maintenant la partie qui nous intéresse le plus ici, la fonction qui permettra la lecture du fichier. Nous commençons en ligne **07** par ouvrir le fichier (**open()**) en lecture (*read*, « **r** »). Puis nous lisons une à une les lignes du fichier (**realines()**) et nous stoquons le résultat de la lecture dans une liste (**dataImported**) en ligne **08**. Comme précédemment, nous pouvons fermer le fichier (**close()**) en ligne **09** puis, en ligne **10**, assigner à la variable **Name** la chaîne de caractères (**str()**) de la première ligne (**dataImported[0]**). En ligne **11**, nous retournons simplement la variable **Name** nouvellement renseignée.

```
06:    def chargeNom(fileName):
07:            dataFile = open(fileName, "r")
08:            dataImported = dataFile.readlines()
09:            dataFile.close()
10:            Name = str(dataImported[0])
11:            return Name
```

En ligne 12**,** nous définissons une nouvelle fonction, dont l'objectif sera d'attribuer le nom retourner par la fonction **chargeNom()** au matériau dans la ligne **13**. Cette fonction sera très classiquement appelée par la fonction **FileSelector()**.

```
12:    def callback(fileName):
13:            mat.setName(chargeNom(fileName))
14:    Window.FileSelector(callback, "Charger le Nom")
```

Vous noterez que cette solution n'est utilisable que pour l'import d'une ligne seulement, l'import de plusieurs lignes à la fois demandant un code assez différent, mais nous nous en tiendront là pour aujourd'hui, l'objectif de lecture dans un fichier depuis l'API Python ayant été atteint.

3.4 Exemple d'un exporteur primitif de matériau

Le script qui suit se propose d'extraire d'un objet toutes les informations relatives à son premier matériau (chaque objet pouvant avoir jusqu'à 16 matériaux associés). Il reste encore un peu de travail pour importer, en plus, chacun des huit canaux de texture (l'approche est la même) et les stoquer dans un même fichier pour obtenir la sauvegarde complète, grâce à Python, d'un matériau. L'intérêt? Tout simplement se constituer une bibliothèque de matériaux ne nécessitant pas de conserver des fichiers blend plus ou moins obscurs et d'importer, grâce à la fonction ***Append*** (**[Maj]+[F1]**) les matériaux qui y sont cachés.

```
01:    import Blender
02:    from Blender import Material, Object, Window
03:
04:    # Selection de l'objet selectionne:
05:    obj=Object.GetSelected()[0]
06:
07:    # Recuperation de son premier materiau:
08:    mat=Material.Get()[0]
09:
10:    # Recuperation de ses proprietes:
11:    Name = mat.getName()
12:    Mode = mat.getMode()
13:    DiffuseShader = mat.getDiffuseShader()
14:    SpecShader = mat.getSpecShader()
15:    Alpha = mat.getAlpha()
```

```
    RGBCol = mat.getRGBCol()
    Ref = mat.getRef()
    Hardness = mat.getHardness()
    RayMirr = mat.getRayMirr()
    MirrDepth = mat.getMirrDepth()
    FresnelMirr = mat.getFresnelMirr()
    FresnelMirrFac = mat.getFresnelMirrFac()
    IOR = mat.getIOR()
    TransDepth = mat.getTransDepth()
    FresnelTrans = mat.getFresnelTrans()
    FresnelTransFac = mat.getFresnelTransFac()
    Filter = mat.getFilter()
    Translucency = mat.getTranslucency()

    # Sortie console des valeurs importees:
    print "Objet: ",obj
    print "Materiau: ",mat
    print "Nom materiau: ",Name
    print "modes: ",Mode
    print "shader diffus: ",DiffuseShader
    print "shader speculaire: ",SpecShader
    print "alpha: ",Alpha
    print "Couleur: ",RGBCol
    print "Ref: ",Ref
    print "Hard: ",Hardness
    print "RayMirr: ",RayMirr
    print "MirrDepth: ",MirrDepth
    print "FresnelMirr: ",FresnelMirr
    print "FresnelMirrFac: ",FresnelMirrFac
    print "IOR: ",IOR
    print "TransDepth: ",TransDepth
    print "FresnelTrans: ",FresnelTrans
    print "FresnelTransFac: ",FresnelTransFac
    print "Filter: ",Filter
    print "Translucency: ",Translucency

    # Fonction d'ecriture des donnees dans un fichier:
    def sauveMateriau(fileName):
            global obj,mat,Name,Mode,DiffuseShader,SpecShader,Apha,RGBCol,Ref
            global Hardness, RayMirr, MirrDepth, FresnelMirr, FresnelMirrFac
            global IOR, TransDepth,FresnelTrans, FresnelTransMax, Filter,
Translucency
            dataFile = open(fileName,"w")
            dataFile.write((Name)+"\n")
            dataFile.write(str(DiffuseShader)+"\n")
            dataFile.write(str(SpecShader)+"\n")
            dataFile.write(str(Alpha)+"\n")
            dataFile.write(str(RGBCol)+"\n")
            dataFile.write(str(Ref)+"\n")
            dataFile.write(str(Hardness)+"\n")
            dataFile.write(str(RayMirr)+"\n")
            dataFile.write(str(MirrDepth)+"\n")
            dataFile.write(str(FresnelMirr)+"\n")
            dataFile.write(str(FresnelMirrFac)+"\n")
            dataFile.write(str(IOR)+"\n")
            dataFile.write(str(TransDepth)+"\n")
            dataFile.write(str(FresnelTrans)+"\n")
            dataFile.write(str(FresnelTransFac)+"\n")
            dataFile.write(str(Filter)+"\n")
            dataFile.write(str(Translucency)+"\n")
            dataFile.close()

    # Appel du selecteur de fichier pour la sauvegarde des donnees:
    Window.FileSelector(sauveMateriau, "Sauvegarder le Materiau")
```

Voici un exemple de fichier texte produit par cet exporteur:

```
Material
0
0
0.10000000149
```

```
[1.0, 0.0, 0.0]
1.0
250
0.10000000149
5
1.5
1.25
1.54999995232
5
1.5
1.25
0.75
0.75
```

Il ne vous reste maintenant plus qu'à écrire le script Python qui permettra d'importer tous ces paramètres, dans le bon ordre, dans Python, et vous aurez un utilitaire complet d'import/export de matériaux. Pour peu que vous y ajoutiez une jolie interface graphique, vous seriez au summum de l'utilité!

4. Conclusion

Cet article a été l'occasion de combler quelques lacunes (dues à une documentation pas très à jour du Gnu/Linux Magazine numéro 75, mais aussi de commencer à jouer avec les fichiers externes. En effet, depuis le début de cette série, nous avons bien sûr exploré les divers modules, fonctions et méthodes de l'API Python de Blender, mais également vu comment construire des interfaces graphiques, comment extraire des informations de scènes existantes, comment lier dynamiquement des scripts plutôt que de les lancer au-travers de l'interface graphique et, aujourd'hui, comment travailler avec des fichiers simples. Petit à petit, notre arsenal de compétences s'agrandit, et avec celui-ci, la possibilité pour nous autres codeurs dilettantes, de produire des scripts de plus en plus intéressants.

5. 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**