Blender: exploration du Module Lamp

Merci à Diamond Editions pour son aimable autorisation pour la mise en ligne de cet article, initialement publié dans Linux Magazine N°80

Olivier Saraja - olivier.saraja@linuxgraphic.org

La version 2.40 de Blender est enfin sortie, avec son lot d'améliorations et de nouvelles fonctionnalités. Mais si tous les utilisateurs de Blender conviennent que le logiciel s'est considérablement enrichi en fonctionnalités étonnantes et utiles, les codeurs de scripts Python rongent leur frein, tant les bouleversements apportés (en particulier dans le module Armature) remettent en cause un grand nombre de scripts déjà existants. Certains laissent toutefois penser que la prochaine version de Blender sera moins riche en nouvelles fonctionnalités, mais restaurera un certain confort de travail pour les développeurs de scripts.

Figure 01: le « splash screen » de Blender v2.40

En attendant de savoir ce qu'il en sera réellement, il convient de reconnaître que le retard accumulé dans l'écriture de la documentation de l'API Python de Blender a été presque totalement résorbé; le brillant résultat de cette nouvelle mouture de la documentation peut être consulté ici: **http://www.blender.org/documentation/240PythonDoc/index.html**.

Pour l'heure, nous nous contenterons de poursuivre notre bonhomme de chemin, et de continuer à découvrir petit à petit toute la richesse de l'API Python de Blender. A ce titre, nous nous intéresserons cette fois-ci au module ***Lamp***, simple à maîtriser mais riche en possibilités.

1. Un peu de théorie: méthodes d'illumination

En règle générale, il faut considérer que l'éclairage à partir d'une source lumineuse unique est une très mauvaise chose: les ombres ont une dureté surnaturelle, et la plupart du temps, l'éclairage diffus dû au rebond des rayons lumineux sur les surfaces n'est pas pris en compte de manière conventionnelle. Malgré cette outrageante imperfection il s'agit bien de la méthode d'éclairage de la scène par défaut de Blender; il faut comprendre qu'il ne s'agit là que d'un point de départ, et que ce ne doit surtout pas être la règle!

Ces dernières années, la plupart des didacticiels ayant trait aux images de synthèse ou à la photographie suggéraient la même chose au sujet de l'éclairage: l'usage de trois sources de lumière, méthode bien connue des photographes (et pendant de longues années, des cinéastes). Le principe en est simple: une première lumière, très intense, éclaire le sujet sur l'un des côtés, tandis qu'une deuxième, beaucoup plus faible éclaire

le côté opposé. Une troisième, enfin, est placée derrière le sujet afin de mettre en valeur la silhouette du sujet. Bien sûr, la caméra fait face au sujet. En adjoignant des couleurs différentes à chacune de ces trois lampes, il est alors assez aisé de montrer le sujet à son avantage, surtout en absence d'environnement solide, et ainsi d'éviter les rendus plats et fades; par exemple, la lumière placée en arrière plan donne des résultats très dramatiques avec une couleur froide, comme un bleu profond, lorsque la lumière latérale faible est chaude (rouge, par exemple) et l'autre lumière opposée d'une couleur suggérant l'ambiance de la scène.

Mais si les détails du sujet ressortent assez bien, cette technique d'illumination est malheureusement tellement classique qu'elle en est devenu immédiatement reconnaissable, et elle stéréotype nombre de rendus, tous logiciels et moteurs de rendu confondus. C'est d'ailleurs pour cette raison que les industries photographiques et cinématographiques ont elles-mêmes abandonné ce standard d'illumination **[1]**, pour ne la conserver que dans des cas spécifiques.

Heureusement, des techniques de rendu plus modernes ont permis de mettre à portée de toutes les compétences et de toutes les bourses des solutions accentuant le réalisme de l'éclairage. Parmi celles-ci, l'une des plus anciennes est l'usage de la *radiosité*. Le principe en est relativement simple: les sources lumineuses émettent une certaine quantité d'énergie, qui sera réfléchie par les objets de la scène; en fonction de la longueur d'onde de la couleur de l'objet touché, une part d'énergie lumineuse sera donc absorbée pour éclairer l'objet, et une autre sera réfléchie selon un angle « de rebond » jusqu'à toucher un autre objet. Bien sûr, la lumière perd de son énergie à mesure qu'elle rebondit, mais par rebonds successifs, la lumière a tendance à « remplir » la scène toute entière. Cette méthode a ceci d'intéressant que le rayon lumineux qui rebondit se teinte légèrement de la couleur de l'objet heurté, ce qui a tendance à procurer aux scènes 3D une ambiance lumineuse souvent chaleureuse, chère aux architectes qui font des rendus d'intérieur. Ensuite, elle atténue fortement les ombres, jusqu'à les rendre si diffuses qu'elles sont parfois à peine distinguables.

Il existe encore d'autres techniques, dites d'illumination globale, basées par exemple sur la méthode de *Monte-Carlo*, ou encore sur le *lancé de photons*, qui ont chacune leurs avantages et inconvénients sur la méthode de *radiosité* conventionnelle, mais Blender ne faisant nativement pas usage de ces méthodes, nous ne chercherons pas à les présenter. Sachez toutefois que des moteurs de rendu comme **Yafray [2]** ou **Art of Illusion [3]** les prennent en charge, et qu'il existe sur le marché quantités de moteurs spécialisés dans le rendu lumineux plus ou moins réaliste: **Radiance [4]**, **Lightflow [5]**, ou **Maxwell Render [6]** n'en sont vraiment que quelques uns parmi des dizaines, chacun mettant en avant tel ou tel autre algorithme d'illumination.

Bien sûr, Blender est très en retard sur ces technologies (Lightflow n'est pourtant plus très jeune), mais il n'en reste pas moins qu'un usage intelligent des différents types de lampes existants permet souvent de simuler des éclairages suffisamment proches de la réalité pour que l'on se laisse tromper. Et si la tâche de l'éclairage est souvent ingrate, il pourra se révéler utile de l'automatiser au moyen de scripts Python. Cet article a donc pour objet de vous donner les bases de l'utilisation du module **Lamp** de l'API Python de Blender.

2. Les lampes de Blender

Mais juste avant d'aborder en profondeur le module **Lamp** de l'API, étudions très brièvement les sources lumineuses disponibles. Blender nous propose en effet différents types de lampes; chacune vient avec des paramètres différents, qui permettent de répondre aux besoins qui ont présidé la nécessité de mise en place d'une telle lampe dans Blender. En effet, chaque lampe illumine la scène d'une façon qui lui est particulière. Comprendre cette façon revient à apprendre comment se servir efficacement de la lampe en question.

2.1 La lampe

`Lamp` est la source lumineuse la plus classique: la lumière irradie à part de la lampe, dans toutes les directions à la fois. Elle est idéale pour simuler des lumières « ponctuelles » comme une ampoule ou une bougie uniques.

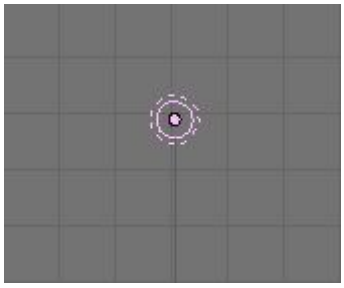

Figure xx: la source de type Lamp

2.2 L'aire lumineuse

Contrairement à la source de type **`Lamp`**, la lumière émane cette fois-ci d'une surface, d'où son nom: **`Area`**. Celle-ci peut-être carrée ou rectangulaire (il est possible de spécifier séparément ses longueur et largeur) et l'impression d'éclairage par une surface est produite par un échantillonnage d'une lampe simple le long des deux directions de l'aire. Ce type de source lumineuse est idéal pour simuler l'illumination produite par une fenêtre, un écran cathodique, une plaque de néons, et ainsi d'obtenir des ombres douces.

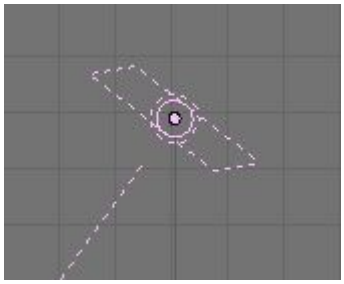

Figure xx: la source de type Area

2.3 Le spot

Egalement un grand classique, le **`Spot`** était historiquement la seule source lumineuse de Blender à pouvoir projeter des ombres, avant l'avènement du *ray tracing* au sein du moteur interne. Il reste toutefois un moyen d'éclairage très utilisé, malgré la complexité relative de sa mise en oeuvre. Le spot émet un cône de lumière depuis l'origine de la source lumineuse, et dans une direction particulière. Son usage produit souvent des effets très théâtraux.

2.4 Le soleil

Le type de lampe **`Sun`** est le plus simple, et est défini par sa seule orientation: quelle que soit la position de cette lampe dans votre scène, celle-ci sera éclairée rigoureusement de la même façon, avec des rayons orientés parallèlement à l'orientation propre de la lampe, et une énergie strictement constante, insensible à la distance entre la lampe et la scène. La position de la lampe ne compte donc rigoureusement pas, au contraire de son orientation.

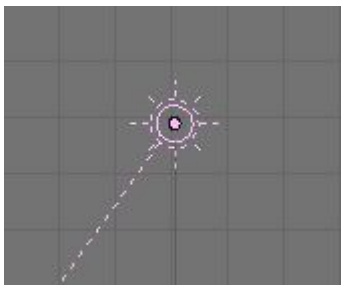

Figure xx: la source de type Sun

2.5 L'hémi

Ce type de lampe n'est utilisé que pour simuler l'éclairage très doux et très diffus d'un ciel couvert. A l'instar de la lampe de type **`Sun`**, la position de la lampe de type **`Hemi`** n'a pas d'importance, au contraire de son orientation. Ses rayons lumineux sont projetés par un vaste hémisphère.

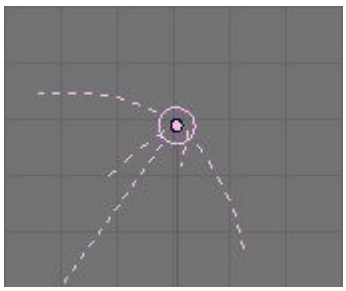

Figure xx: la source de type Hemi

2.6 Blender et les ombres

A noter que la plupart de ces lampes peuvent générer des ombres, à quelques exceptions près, soit en utilisant soit la méthode du *raytracing* (option ***Ray Shadows***), soit la méthode du *Shadow buffering* (option ***Buffer Shadows***).

Avec la première méthode, des rayons lumineux sont lancés depuis la lampe, selon la méthode qui lui est propre. Si un rayon rencontre un objet, il illumine celui-ci et obscurcit les éléments de la scène qu'il occulte. L'avantage de cette méthode est que les ombres sont toujours précises et nettes, au prix toutefois d'un temps de rendu accru, et qu'il n'y a pas de réglage à effectuer à part activer l'option appropriée. Toutes les lampes disposent donc d'un bouton ***Ray Shadow*** dans l'onglet ***Shadow and Spot***, à l'exception de la lampe de type **`Hemi`**, qui ne projette jamais d'ombre.

La seconde méthode relève du « bricolage intelligent ». Un rendu fictif est effectué depuis la lampe comme s'il s'agissait d'une caméra; les silhouettes des objets sont capturés et enregistrés dans un *buffer* en fonction de la distance les séparant de la lampe de référence. Les silhouettes ainsi capturées sont alors projetées sur les objets situés à l'autre extrémité du *buffer*.

Par exemple, l'image qui suit présente le *buffer* de la scène par défaut, « vue » depuis la lampe par défaut.

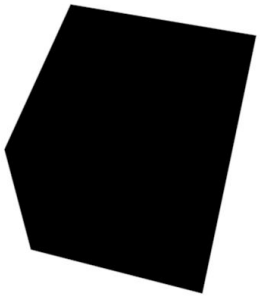

Figure xx: exemple de Shadow buffer de la scène par défaut, rendu depuis la lampe par défaut

C'est cette image qui aurait été « projetée » sur le sol fictif que la scène aurait pu contenir, sous le cube. On devine instantanément que la résolution de rendu a une importance cruciale: avec une trop faible résolution, les bords de l'ombre vont inévitablement pixeliser, présentant un crénelage évident. L'usage du *Shadow buffer* se complique alors car, parmi les paramètres le caractérisant, se trouveront par exemple la taille du *buffer* (***Shadow Buffer Size***) et la profondeur de l'anti-crénelage (***Samples***).

L'avantage de cette méthode est qu'elle est rapide à calculer; en revanche, sa qualité peut être faible si elle est mal paramétrée, et les réglages qui lui sont liés peuvent devenir complexe. A noter que seul le **`Spot`** offre le bouton ***Buffer Shadows*** permettant d'activer cette option. Avant l'apparition du *raytracing* dans le moteur de rendu de Blender, le **`Spot`** avec cette option active était la seule façon de produire des ombres.

Enfin, veuillez noter, dans tous les cas, que pour que votre rendu baigne dans les ombres, il vous faut activer le bouton ***Shadow*** dans le panneau ***Render*** des ***Render buttons*** du menu ***Scene***.

3. Le Module Lamp

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

```
01:    import Blender
02:    from Blender import Lamp
03:    ...
```

Mais pour peu que vous souhaiteriez activer l'éclairage par *raytracing* ou le rendu avec ombres, d'autres lignes sont nécessaires pour modifier le contexte de rendu. Il faut alors commencer par importer le sous-module **`Render`** depuis le module **`Scene`**, après avoir importé celui-ci même depuis **`Blender`**. Cela se traduit alors par les lignes suivantes:

```
01:    import Blender
02:    from Blender import Lamp, Scene
03:    from Blender.Scene import Render
```

Pour travailler avec le module **`Scene`**, il faut récupérer la scène courante et la stocker dans une variable, arbitrairement nommée **`scn`**. Pour cela, faisons usage de la fonction **`GetCurrent()`** du module **`Scene`**:

```
04:    scn = Scene.GetCurrent()
```

Quant au contexte de rendu de cette scène, il est possible de le récupérer grâce à la méthode **`getRenderingContext()`** et de la stocker dans une variable nommée arbitrairement **`context`**:

```
05:    context = scn.getRenderingContext()
```

L'activation de l'option ***Ray*** et de l'option ***Shadow*** se fait très simplement par l'usage des méthodes **`enableRaytracing()`** et **`enableShadow()`**, où l'état des options est défini par une valeur **`1`** (actif) ou **`0`**

(inactif):

```
06:    context.enableRayTracing(1)
07:    context.enableShadow(1)
```

Bien sûr, vous pouvez choisir de n'activer que l'une ou l'autre de ces deux options; si vous n'en voulez aucune, les lignes **03** à **07** sont totalement optionnelles, de même que l'import du module **Scene** en ligne **02**.

Figure xx: les options Shadow et Ray à activer, dans Blender

La création d'une lampe dans Blender pourrait se faire en quatre lignes élémentaires.

```
08:    l = Lamp.New('Spot','MonSpot')
...
101:   ob = Object.New('Lamp')
102:   ob.link(l)
103:   scn.link(ob)
```

En ligne **08**, nous créons grâce à la fonction **New()** du module **Lamp** un bloc de données ***Lamp*** de type **Spot** et portant le nom (dans Blender) **LA:MonSpot** (afin de le différencier de la lampe nommée **LA:Spot** de la scène par défaut), et nous le stockons dans une variable sobrement nommée **l**. Les différents type de ***Lamp*** qu'il est possible de créer grâce à la fonction **New()** sont respectivement: **'Lamp'**, **'Sun'**, **'Spot'**, **'Hemi'** et **'Area'** (il y a également le type **'Photon'** exploitable au-travers du moteur de rendu Yafray, que nous laisseront de côté dans cette exploration).

En ligne **101**, nous créons enfin un objet de type ***Lamp*** grâce à la fonction **New()** du module **Object**. En ligne **102**, nous lions le bloc de données ***Lamp*** nommé **l** à l'objet nommé **ob** grâce à la fonction **link()**. Et enfin, en ligne **103** pour que l'objet nouvellement créé apparaisse dans la scène, nous le lions (toujours grâce à la fonction **link()**) à la scène **scn**. Un coup de **[Alt]+[P]** dans la fenêtre d'édition, et la vue 3D doit s'actualiser pour présenter un nouveau ***Spot***, situé à l'origine, orienté dans la direction des z négatifs et portant le nom **LA:MonSpot**.

Il ne nous reste plus qu'à explorer tous les paramètres que nous pouvons définir au-travers du module **Lamp**.

3.1 Paramètres communs à toutes les lampes

Nous allons commencer par préciser les paramètres qui sont communs à toutes les lampes. On les retrouve dans le panneau ***Lamp*** des ***Lamps buttons***, du menu ***Shading*** de Blender.

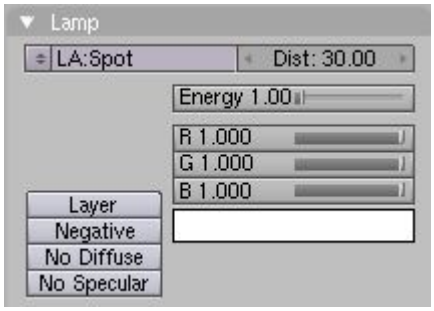

Figure xx: dans l'onglet Lamp, les paramètres communs à toutes les sources lumineuses

La méthode setDist():

Cette méthode permet de paramétrer la valeur ***Dist***, qui correspond à la distance (en unités de Blender) à partir de laquelle l'intensité lumineuse est moitié moindre qu'à l'origine. Elle admet des valeurs allant de **`0.01`** à **`5000.00`**. Par exemple:

```
l.setDist(30.00)
```

La méthode setEnergy():

Cette méthode permet de régler le paramètre ***Energy***, qui détermine l'énergie à l'origine de la source lumineuse. Elle admet des valeurs allant de **`0.00`** à **`10.00`**. Par exemple:

```
l.setEnergy(1.00)
```

La variable col = [R, G, B]:

La variable **`col`** est un vecteur dont les trois composantes sont respectivement **`R`**, **`G`** et **`B`**. Par exemple:

```
l.col = [0.5, 0.5, 0.5]
```

produit une lumière de couleur « gris moyen » et:

```
l.col = [1.0, 1.0, 1.0]
```

une lumière de couleur blanche. Chaque composant peut varier de **`0.0`** à **`1.0`**. Alternativement, il est possible de spécifier indépendamment les valeurs de chaque composante:

```
l.R = 1.00
l.G = 0.33
l.B = 0.80
```

La méthode setMode():

Cette méthode permet d'activer diverses options des panneaux ***Lamp*** et ***Shadow and Spot***. Lorsque vous l'employez, seule les modes spécifiés sont actifs, ceux passés sous silence sont inactifs. Il en résulte que la ligne:

```
l.setMode()
```

désactive tous les modes de la lampe. Le panneau Lamp propose les options suivantes, communes à toutes les lampes.

- ***Layer***: cette option permet à la source lumineuse de n'éclairer que les objets placés sur le même calque qu'elle. Par exemple: **`l.setMode('Layer')`**.
- ***Negative***: permet à la source lumineuse d'émettre de la lumière négative. Par exemple, **`l.setMode('Negative')`**.
- ***No Diffuse***: les matériaux illuminés par cette source ne tiendront pas compte de leur paramètre ***Ref***, comme si sa valeur était égale à **`0.000`**. Par exemple: **`l.setMode ('NoDiffuse')`**.
- ***No Specular***: les matériaux illuminés par cette source ne tiendront pas compte de leur paramètre ***Spec***, comme si sa valeur était égale à **`0.000`**. Par exemple: **`l.setMode ('NoSpecular')`**.

Enfin, à l'exception de la source de type ***Hemi***, toutes les lampes partagent les modes suivants, issus du panneau ***Shadow and Spot*** de Blender:

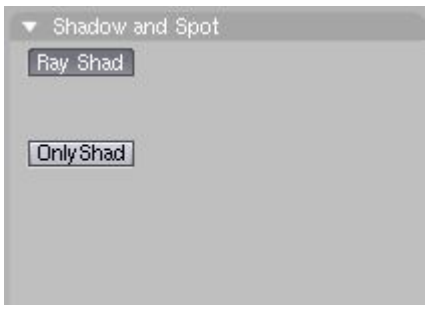

Figure xx: dans l'onglet Shadow and Spot, les paramètres communs à toutes les sources lumineuses (à l'exception de Hemi)

- ***Ray Shadow***: si cette option est activée, l'ombre projetée par les objets éclairés est calculée par la méthode du *Ray tracing* (plus longue à calculer mais plus précise) plutôt que par celle du *scanline*. Par exemple: `l.setMode('RayShadow')`.
- ***Only Shadow***: si cette option est activée, la lampe ne produit que des ombres, sans éclairer la surface des objets se trouvant dans sa zone d'effet. Par exemple: `l.setMode('OnlyShadow')`.

3.2 La lampe

Ce type de lampe propose deux Modes supplémentaires, ainsi que deux méthodes supplémentaires:

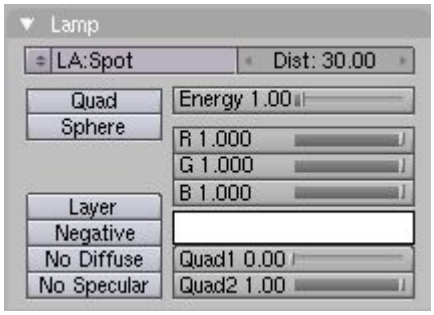

Figure xx: le panneau Lamp de la source lumineuse de type Lamp

La méthode setMode() propre aux lampes:

- ***Quad***: lorsque ce mode est actif, l'atténuation lumineuse est alors proportionnelle à l'inverse du carré de la distance à la source lumineuse. Sinon, elle est simplement linéaire. Ce mode produit des éclairages plus réalistes et se déclare grâce à la méthode **`setMode()`**. Par exemple: `l.setMode('Quad')`.
- ***Sphere***: lorsque ce mode est actif, l'énergie décroît progressivement (linéairement ou non, selon que le mode ***Quad*** soit également actif ou non) pour atteindre la valeur ***Energy*** `0.000` lorsque la distance ***Dist*** est atteinte (au lieu d'être simplement divisée par deux à cette même valeur). Ce mode se déclare grâce à la méthode **`setMode()`**. Par exemple: `l.setMode('Sphere')`.

Les méthodes setQuad1() et setQuad2():

Ces deux méthodes agissent conjointement et permettent de contrôler le taux d'atténuation de la lumière; si la variable passée à **`setQuad1()`** est égale à `1.000` et **`setQuad2()`** à `0.000`, l'atténuation est parfaitement linéaire. Si la variable passée à **`setQuad1()`** est égale à `0.000` et **`setQuad2()`** à `1.000`, l'atténuation est parfaitement proportionnelle à l'inverse du carré de la distance. Jouer sur les deux valeurs permet donc de pondérer les deux effets, mais leur usage est, de par leur nature très mathématique, réservée à ceux qui connaissent leur fonctionnement théorique. Elles admettent des valeurs pouvant aller de **`0.00`** à **`1.00`**. Par

exemple:

```
l.setQuad1(0.00)
l.setQuad2(1.00)
```

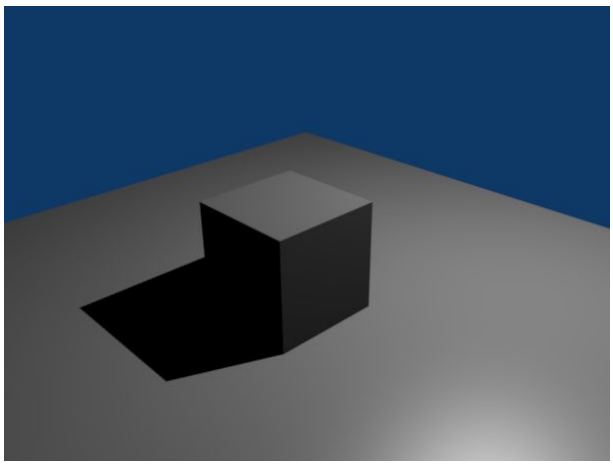

Figure xx: exemple d'éclairage utilisant une lampe

3.3 L'aire lumineuse

Les nouvelles méthodes disponibles pour ce type de lampe permettent de déterminer la taille de l'aire ainsi que son échantillonnage dans les directions X et Y. La possibilité d'activer ou de paramétrer des options disponibles dans le panneau ***Shadow and Spot*** reste encore à implémenter dans l'API, qui ne supporte donc que partiellement, à ce jour, les aires lumineuses, selon le contenu de la documentation officielle.

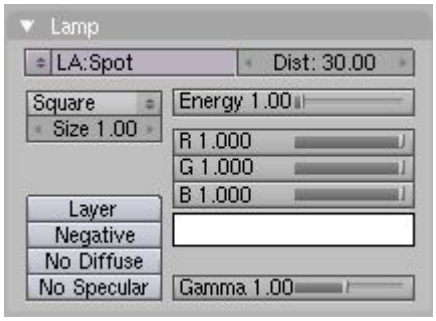

Figure xx: le panneau Lamp de la source lumineuse de type Area

Les méthodes setAreaSizeX() et setAreaSizeY():

La variable de la méthode **`setAreaSizeX()`** détermine la taille (en unités de Blender) dans les deux directions locales X et Y si la lampe **`Area`** est du type Square, sauf si elle est du type Rect, auquel cas **`setAreaSizeY()`** détermine la tille dans la direction locale Y. A noter que ces valeurs n'affectent pas l'énergie lumineuse de la surface.

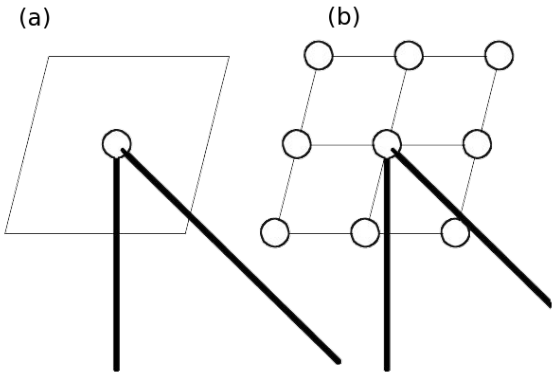

Figure xx: tiré de la documentation, un schéma illustrant les variables RaySamples X et Y

Le panneau ***Shadow and Spot*** de la lampe de type **`Area`** s'enrichit pour sa part de quelques paramètres supplémentaires.

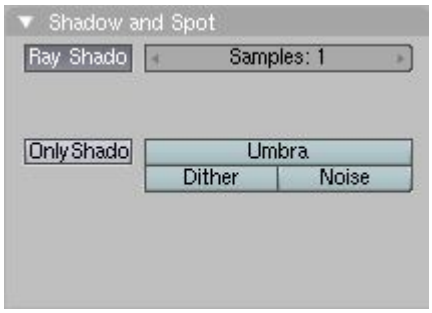

Figure xx: le panneau Shadow and Spot de la source lumineuse de type Area

Les méthodes setRaySamplesX() et setRaySamplesY:

La variable de ma méthode **`setRaySamplesX()`** détermine le nombre de lampes élémentaires placées sur chacun des deux côtés de la lampe **`Area`** si celle-ci est du type Square, sauf si elle est du type Rect, auquel cas **`setRaySamplesY()`** détermine le nombre de lampes élémentaires dans la direction locale Y de la lampe. Par exemple:

```
l.setRaySamplesX(2)
l.setRaySampleY(3)
```

indiquent que l'aire lumineuse est simulée avec (deux fois trois) six lampes au total. Bien sûr, plus le nombre de lampes simulées est élevé, plus les temps de rendu seront augmentés, mais plus les ombres obtenues seront douces. La valeur maximale qu'il est possible de passer à chacune des deux méthodes est égale à **16**.

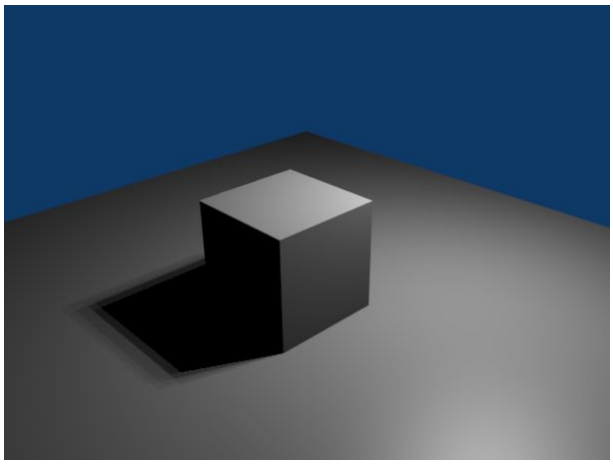

Figure xx: exemple d'éclairage utilisant une aire lumineuse

3.4 Le spot

La lampe de type **`Spot`** est la plus complexe mise à disposition des scripteurs. Si elle reste très abordable lorsque le mode *Ray Shadow* est utilisé, elle se complique considérablement avec le mode *Buffer Shadow* qui propose de nombreuses nouvelles méthodes, correspondant aux paramètres apparaissant dans le panneau ***Shadow and Spot*** de Blender. En effet, il est possible d'y définir la forme du cône de lumière, de régler la qualité et la profondeur des ombres, ainsi que de spécifier la densité du halo lumineux.

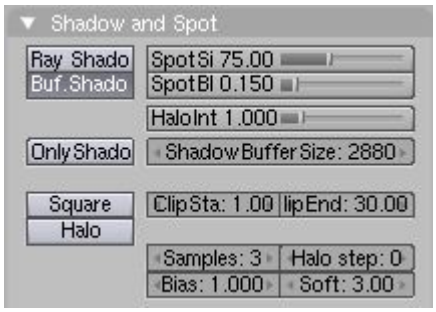

Figure xx: le panneau Shadow and Spot de la source lumineuse de type Spot

La méthode setMode() propre aux spots:

- ***Buffer Shadow***: si cette option est activée, l'ombre projetée par les objets éclairés est calculée par la méthode du *Shadow Buffering* propre au moteur de rendu de type *scanline*. L'ombre est bien plus rapide à calculer que par la méthode du *Ray tracing*, mais est beaucoup moins précise et nécessite des réglages parfois complexes pour donner de bons résultats. Par exemple: **`l.setMode('Shadows')`**.
- ***Halo*:** en activant cette option, le cône de lumière peut-être échantillonner pour simuler un effet de lumière volumétrique (un halo de lumière *visible*, comme les rayons du soleil traversant la voûte feuillue d'un sous-bois, par exemple). Par exemple: **`l.setMode ('Halo')`**.

Les paramètres qui suivent permettent de définir le cône de lumière. Ils sont valables aussi bien pour le mode *Ray Shadow* que le mode *Buffer Shadow*.

Les méthodes setSpotSize() et setSpotBlend():

La valeur passée à la méthode **`setSpotSize()`** représente l'angle d'ouverture du spot. Quand à celle passée à la méthode **`setSpotBlend()`**, elle détermine le gradient qui assure la transition entre la zone éclairée par le spot et la zone d'ombre au-delà du cône de lumière; une faible valeur indique une transition très brutale, et donc une frontière très dure. Au contraire, une valeur élevée indique une zone de pénombre importante. Par exemple:

```
l.setSpotSize(75.00)
l.setSpotBlend(0.150)
```

Les valeurs de **`setSpotSize()`** peuvent varier de **`1.00`** à **`180.00`**, tandis que pour **`setSpotBlend()`**, elles varient de **`0.000`** à **`1.000`**.

Les méthodes qui suivent ne sont en revanche valables que pour le mode *Buffer Shadow*.

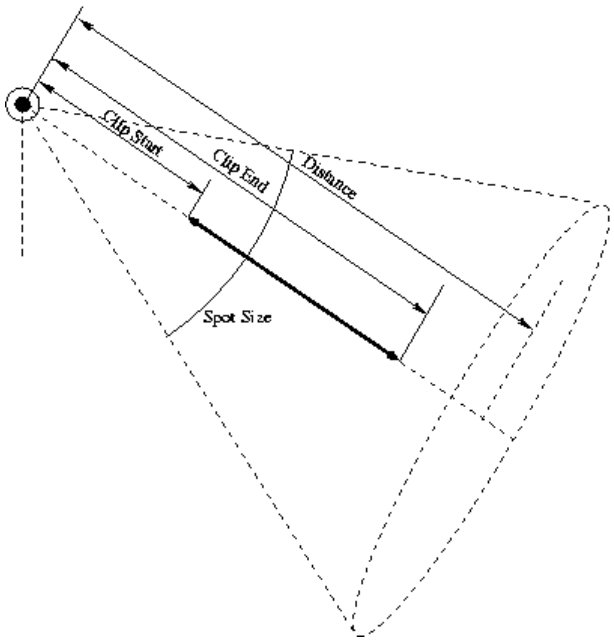

Figure xx: tiré de la documentation Blender, un schéma expliquant la géométrie du Spot

La méthode setBufferSize():

Cette méthode détermine plus ou moins de la résolution de l'ombre; plus cette valeur est élevée, plus l'ombre (et surtout ses frontières) sera précise et clairement dessinée, mais nécessitera des temps de rendu supérieurs; la valeur passée à la méthode peut varier de **`512`** à **`10240`**, et doit idéalement être un multiple de 64. Par exemple: **`l.setBufferSize(2880)`**.

Les méthodes setClipStart() et setClipEnd():

Les variables respectivement passées à ces deux méthodes définissent la fourchette de distances entre lesquelles le **`Spot`** prendra en compte la présence d'objets pour le calcul des ombres éventuelles; pour obtenir des ombres de la meilleur qualité possible, il faut traditionnellement essayer de régler la valeur passée à **`setClipStart()`** un tout petit peu avant le premier objet illuminé par le **`Spot`**, et celle passée à **`setClipEnd()`** un tout petit peu après le dernier objet illuminé, mais ce type de réglage est beaucoup moins aisé sous Python que dans l'interface graphique de Blender. Par exemple:

```
l.setClipStart(0.1)
l.setClipEnd(100.0)
```

La méthode setSamples():

La valeur (comprise entre **`1`** et **`16`**) passée à cette méthode peut être interprétée comme la qualité d'anti-crénelage du bord des ombres; plus la valeur est élevée, plus lisse est l'ombre, mais au détriment du temps de rendu. On utilisera une valeur élevée avec une faible valeur pour **`setBufferSize()`**. Par exemple:

```
l.setBufferSize(1024)
l.setSamples(3)
```

La méthode setSoftness():

L'usage de cette méthode permet d'adoucir les ombres; elle accepte une variable comprise entre **`1.00`** et **`100.00`**; plus elle est élevée, plus la transition entre zone ombrée et zone éclairée est progressive.

Les paramètres qui suivent enfin permettent de créer des effets de lumière volumétrique. A l'exception de **`HaloInt`**, ils ne sont disponibles qu'avec la méthode de *Shadow Buffering*.

La méthode setHaloInt():

Cette méthode définit l'intensité du Halo lumineux; il n'est pris en compte que si le mode **`Halo`** est actif, et accepte une valeur comprise entre **`0.000`** et **`5.000`**. Par exemple:

```
l.setMode('Halo')
l.setHaloInt(2.25)
```

La méthode setHaloStep():

La valeur passée à cette méthode (comprise entre **`0`** et **`12`**) représente le « pas » d'échantillonnage de l'effet de lumière volumétrique; pour que ce paramètre soit pris en compte, il faut que le mode **`Halo`** soit actif. Par exemple:

```
l.setMode('Halo')
l.setHaloInt(2.25)
l.setHaloStep(3)
```

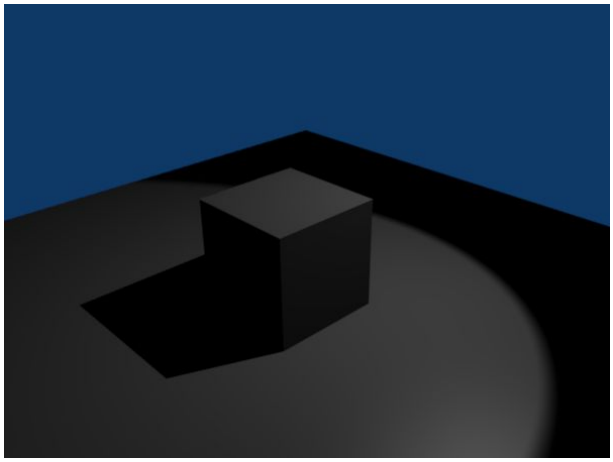

Figure xx: exemple d'éclairage utilisant un spot

3.5 Le soleil et l'hémi

Ces types de lampes sont les plus simples implémentés dans Blender, et ne proposent aucun paramètre nouveau. La lampe de type Hémi est encore plus simple dans la mesure où aucune option d'ombrage n'est disponible pour elle.

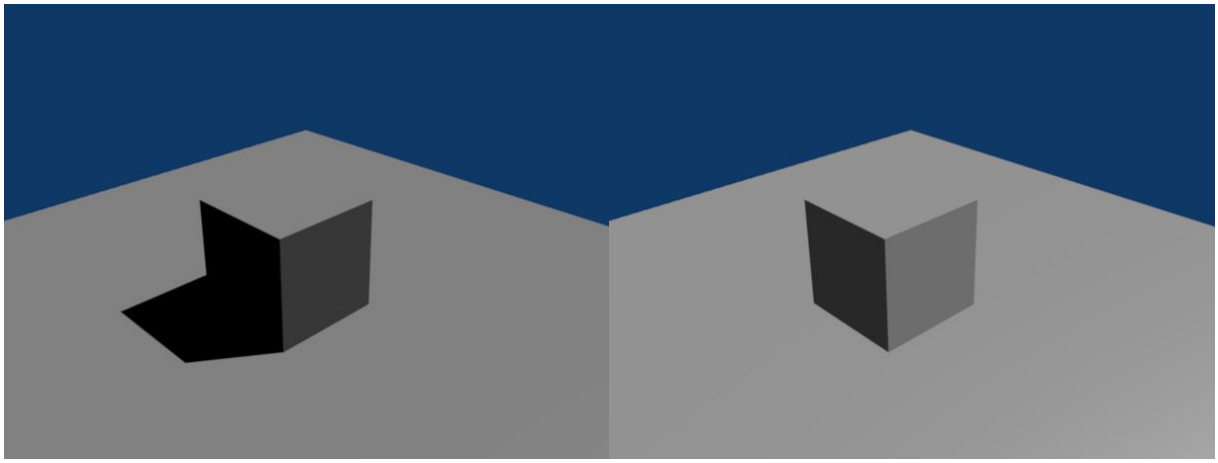

Figure xx: exemple d'éclairage utilisant respectivement un soleil et un hémi

4. Exemples de scripts à étudier

Vous trouverez ci-après la description de quelques scripts utiles, qui proposent surtout des usages originaux

du module Lamp. Les décortiquez et chercher à comprendre leur fonctionnement aidera à progresser dans votre étude du scripting python.

4.1 Création d'une rampe de spots

Vous trouverez dans le didacticiel Blender de **JM Soler** (**http://jmsoler.free.fr/didacticiel/blender/tutor/cpl_b223new.htm#rampedespots**) un script qui, appliqué à une demi-sphère, propose de simuler l'illumination globale par une méthode très connue et très prisée avant l'avènement de l'occlusion ambiante dans le moteur de rendu de Blender: la « duplication » de spots sur les points de contrôle d'une hémi-sphère. Le script en question permet donc de positionner sur chaque point de contrôle du maillage de l'objet sélectionné une lampe de type `Spot`, orienté conformément aux normales des faces adjacentes à un point donné.

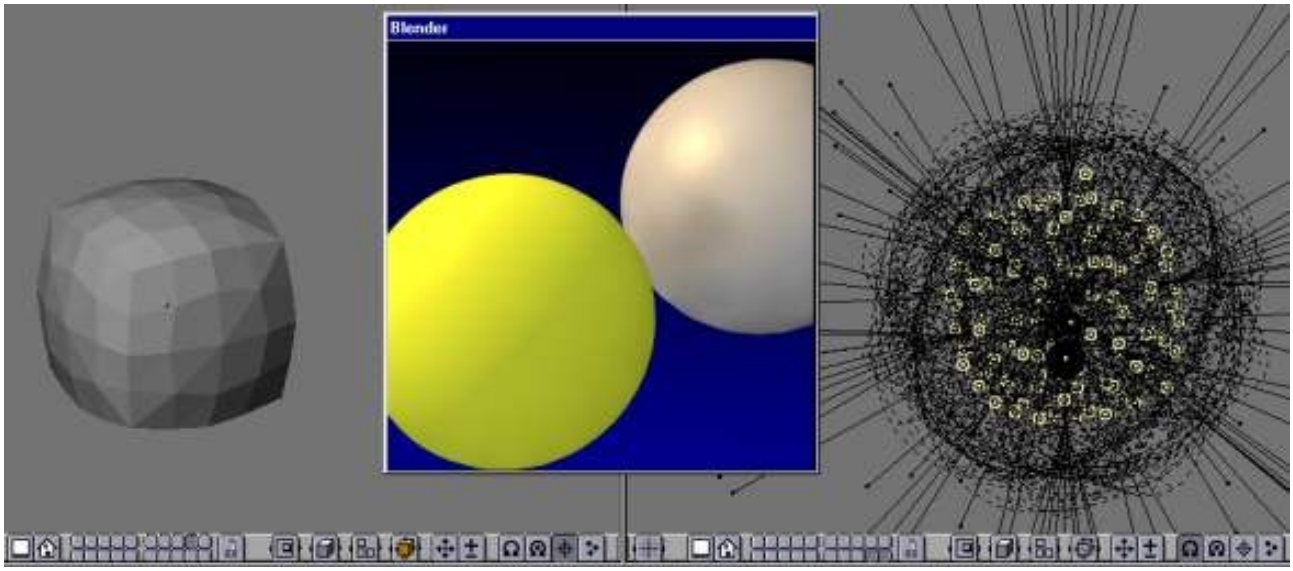

Figure xx: création d'une rampe de spots à partir d'un maillage quelconque

4.2 Générateur d'éclairs

Ce script assez simple mais astucieux de **timmeh** permet de mettre en oeuvre les notions acquises lors des précédents articles, en particulier l'article sur le module IPO. Il permet de simuler l'éclairage erratique produit par une succession d'éclairs dont la distance, la puissance, et la hauteur peuvent être différentes; il permet également de gérer des répétitions d'éclairs, ainsi que des répliques des éclairs principaux. Le script est disponible à l'adresse suivante: **http://www.elysiun.com/forum/viewtopic.php?t=44856**.

4.3 Convertisseur de température Kelvin en couleur RGB

Un script très intéressant de **Mikejedw** pour vous aider à illuminer vos scènes de façon plus réaliste. La couleur de température **[7]** est une valeur en Kelvin qui indique la teinte d'une source de lumière spécifique. Par exemple, une lampe halogène aura une couleur de température comprise entre 3000 et 3200 K, et le soleil à son zénith une couleur de température d'environ 5800K. La première aura une couleur d'un bleu clair tirant vers le jaune, tandis que la seconde aura une couleur également bleu clair, mais à peu près pur **[8]**. Grâce à ce script, vous pourrez donc déterminer la couleur de température d'une lampe directement en spécifiant sa valeur en Kelvin, ou choisir dans une liste de réglages pré-déterminés. Le script est disponible à l'adresse suivante: **http://www.elysiun.com/forum/viewtopic.php?t=34303**.

5. Conclusion

Nous venons de voir qu'un module sans prétention comme **`Lamp`** permet de mettre en oeuvre des scripts plutôt intéressants et originaux, sans être particulièrement difficile à maîtriser. Seul le type de lampe **`Spot`** propose des options qui peuvent paraître difficile à maîtriser en dehors de Blender même, mais une bonne connaissance de cette lampe dans Blender vous sera sans aucun doute d'une aide certaine au-travers de Python.

Le dernier mot concernera l'excellente initiative de la Fondation Blender qui ont compilé diverses version de Blender, notamment pour tenir compte de la disponibilité, pour les distributions récentes comme vieillissantes, de Python 2.3 ou 2.4, au choix.

6. Notes

[1] Un article plein de parti pris sur les techniques d'illumination en 3D: **http://www.itchy-animation.co.uk/light.htm**

[2] Yafray, le moteur de rendu compagnon de Blender: **http://yafray.org/**

[3] Art of Illusion, suite de modélisation 3D au moteur de rendu performant: **http://www.artofillusion.org/**

[4] Radiance, l'une des références libres: **http://radsite.lbl.gov/radiance/**

[5] Lightflow, un ancien qui ne bouge plus trop: **http://www.lightflowtech.com/**

[6] Maxwell Render, le ténor du moment, qui vise le photoréalisme, mais commercial et non Libre:
http://www.maxwellrender.com/

[7] La couleur de température selon Wikipédia: **http://fr.wikipedia.org/wiki/Temp%C3%A9rature_de_couleur**

[8] Un extrait [en] très concis sur les couleurs de température de 3drender:
http://www.3drender.com/glossary/colortemp.htm

7. Liens

La page de Blender (version actuelle: v2.40): **www.blender.org**

La documentation officielle de python pour Blender v2.40:
http://www.blender.org/documentation/240PythonDoc/index.html

La documentation de python: **http://docs.python.org/download.html**