Atelier Povray – 31 novembre 2004

Introduction (Wikipédia)

POV-Ray

POV-Ray (*P*ersistance *o*f *V*ision *Ray*-tracer), ou *POV* est un logiciel gratuit, en constante évolution, de *lancer de rayons*, « *Ray tracing* » en anglais, (technique de synthèse d'image en 3D), disponible sur une grande variété de plate-formes (Windows, MacOS, Linux, etc). Il est basé à l'origine sur DKBTrace, et dans une moindre proportion sur Polyray. Il n'est pas un logiciel libre, mais ses sources sont tout de même disponibles selon les conditions de la licence POV-Ray.

POV ne dispose pas d'une interface graphique comme la plupart des logiciels de synthèse actuels, mais utilise des scripts de description des scènes, dans lesquels tous les objets, les lumières, ... doivent être entrés. Cela permet d'avoir des formes de bases (sphères, boîtes, tores, etc) mais aussi des volumes ou des surfaces basés sur des fonctions mathématiques, les isosurfaces (par exemple : function {x*x - **F**/y*y + z*z} dessine une sorte de puits de gravité, avec **F** qui représente sa force). Il est aussi possible d'importer des objets d'autres logiciels (comme 3D Studio Max, Poser, etc) qui seront rendus dans POV comme un assemblage de triangles, mais il est très difficile d'exporter des objets POV vers d'autres formats.

Lancer de rayon

Le lancer de rayon (ray tracing en anglais) est une technique de rendu en synthèse d'image simulant le parcours inverse de la lumière de la scène vers l'œil.

Cette technique simple reproduit les phénomènes physiques que sont la réflexion et la réfraction. Une mise en œuvre naïve du lancer de rayon ne peut rendre compte d'autres phénomènes optiques tels que les caustiques (taches lumineuses créées à l'aide d'une lentille convergente par exemple) et la dispersion lumineuse (la radiosité s'attaque à ce problème).

En revanche, contrairement à d'autres algorithmes de synthèse d'image, elle permet de définir mathématiquement les objets à représenter et non pas seulement par une multitude de facettes.

Le lancer de rayon consiste, pour chaque pixel de l'image générée, à lancer un rayon depuis le point de vue (la caméra) dans la scène 3D. Le premier point d'impact du rayon sur un objet définit l'objet concerné par le pixel correpondant.

Des rayons sont ensuite lancés depuis le point d'impact en direction de chaque source de lumière pour déterminer sa luminosité (est-il éclairé ou à l'ombre d'autres objets ?). Cette luminosité combinée avec la couleur de l'objet ainsi que d'autres informations éventuelles (réflexions, transparence, etc.) déterminent la couleur finale du pixel.

Cette technique permet la génération d'images très réalistes mais peut requérir un temps de calcul colossal en fonction de la complexité de la scène 3D. Jusqu'en 2003, la puissance des ordinateurs ne permettait pas le calcul d'images en temps réel. Depuis, sous certaines conditions, de nombreuses optimisations de l'algorithme permettent un rendu en temps intéractif (quelques images par seconde).

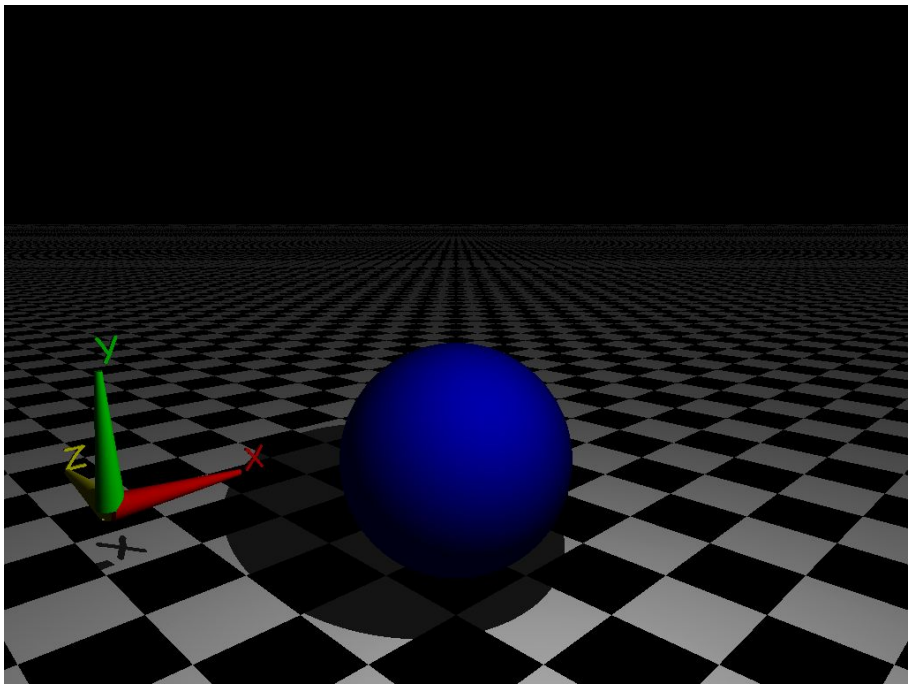
Une petite introduction

Pov-RAY est un langage de script. L'image est générée à partir d'un code source.

Les bases

Chaque scène doit comporter une caméra (bien qu'il y en ai une définie par défaut sinon), et au moins une lumière.

Voici un exemple d'une petite scène.



Il est imporant de noter que nous nous trouvons dans un repère de style « main gauche » : le pouce est l'axe des X, l'index l'axe Y, et le majeur Z. Contrairement à la plupart des logiciels de 3D, Y corresponds à la hauteur (en général c'est Z).

Essayez de reproduire la scène précédente (sans le trièdre). La syntaxe des élements est la suivante :

```
camera{
	location <x, y, z> // position de la caméra, c'est le point de vue, l'œil de la scène
	look_at <x, y, z> // endroit que fixe la caméra, là ou elle regarde
}
light_source{
	<x, y, z> // position de la source lumineuse
	color rgb <r, v, b> // couleur de la source lumineuse
}
sphere{
	<x,y,z> //centre
	r // rayon de la sphère
}
plan{
```

```
      <x, y, z>, // vecteur de la normale au plan, c'est à dire le vecteur
permendiculaire au plan, et définissant ou se situe le haut. Attention, il y a une
vigule après ce vecteur...

      d // décalage du plan par rapport à l'origine du repère, sur l'axe du vecteur
normal

}
```

Les valeurs des coordonnées `<x, y, z>` sont des réels quelconques (`0.1, 7.2, 50, 104.1, ...`), tandis que les valeurs de couleurs sont des nombres réels entre 0 et 1. Par exemple, pour obtenir du rouge, on utilisera <1,0,0>, du bleu <0,0,1>, ou un gris <0.5, 0.5, 0.5>. Si vous préférez ou avez l'habitude de travailler avec des valeurs RVB entre 0 et 255, vous pouvez utiliser la division : `<255,0,0>/255` pour du rouge par exempe.

Si vous essayez avec ces informations là, vous allez obtenir une scène toute noire. En effet, vos objet n'ont pas de couleur ! Afin de donner une couleur, il faut ajouter une déclaration “pigment” à l'objet. Cette déclaration permet d'utiliser toutes sortes de couleurs.

Par exemple, pour une couleur unie :

```
object{

      ...

      pigment{

            color rgb <r, v, b>

      }

}
```

ou alors pour le damier

```
object{

      ...

      pigment{

            checker

            color rgb <r1, v1, b1>

            color rgb <r2, v2, b2>

      }

}
```

Grâce à ces informations, vous devez pouvoir sans problèmes reproduire la scène ci dessus :

```
camera{

      location <0.5,0.5,-3>

      look_at <0,0,0>

}


light_source{

      <1,2,-2>

      color rgb <1, 1, 1>

}
```

```
sphere{
	<0,0,0>
	1
	pigment{
		color rgb <0,0,1>
	}
}


plane{
	<0,1,0>,
	-1.5
	pigment{
		checker
		color rgb <0,0,0>
		color rgb <1,1,1>
	}
}
```

Lancement de POV-Ray

Afin d'effectuer le rendu de la scène, il vous faut taper le code source dans un éditeur de texte. `Kate` possède un mode de coloration syntaxique pour les fichiers POV-Ray.

Enregistez votre fichier avec l'extension `.pov` et vous pouvez alors lancer le rendu de votre scène :

```
povray +P +H800 +W600 fichier.pov
```

La commande povray va vous générer un fichier `.png` dans le même répertoire. Povray prend comme arguments des sortes de booléens, qui seront activés (+) ou désactivés (-). Par exemple, `+P` permet de mettre en pause Povray lorsqu'il a terminé la construction de la scène, afin de la visualiser sans ouvrir le fichier `PNG`. Par contre, certains arguments, comme `+H800 +W600` (optionel, permet de définir la taille de l'image de sortie – Height Weigth) peuvent prendre indifférement un « + » ou un « - ».

Les différents objets simples

La sphere

Nous l'avons vue ci dessus. Pour obtenir une forme élipsoïde, il faudra utiliser les transformations de mise à l'échelle (voir plus loin).

La boîte

On donne les coordonnées de deux points diagonalement opposés.

```
box{
	<x1, y1, z1> //premier coin
```

```
      <x2, y2, z2> //deuxième coin
}
```

Les faces seront parallèles aux plans de coordonnée. Pour incliner ce pavé, il faudra utiliser les transformations.

Le cylindre

On donne les centres et le rayon des disques des extrémités. On peut supprimer les disques des extrémités grâce au mot-clé open.

```
cylinder{
      <x1, y1, z1> //centre du premier disque
      <x2, y2, z2> //centre du deuxième disque
      r //rayon du cylindre
      open //optionnel
}
```

Le cône

Il s'agit en fait de tronc de cône. On donne les centres et les rayons des disques des extrémités. On peut aussi utiliser le mot open.

```
cone{
      <x1, y1, z1>     //centre du premier disque
      R1                    //son rayon
      <x2, y2, z2>     //centre du deuxième disque
      R2                   //son rayon
}
```

Le tore

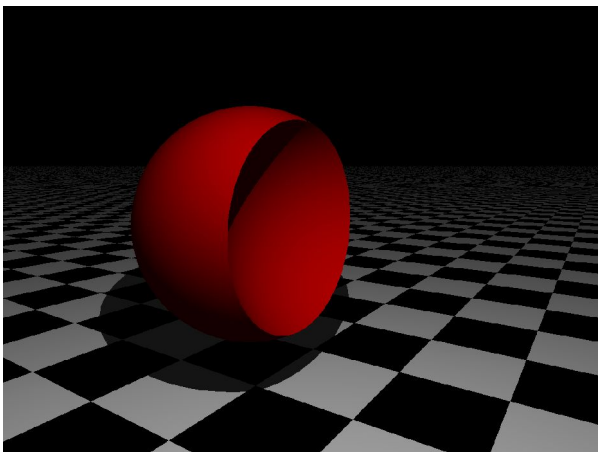
On donne la valeur de son rayon extérieur et intérieur.

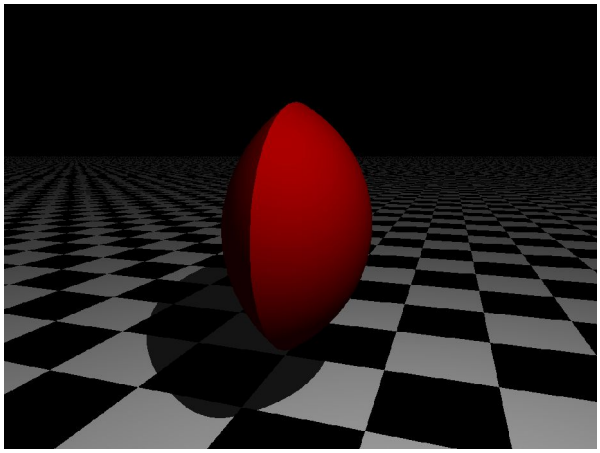
```
torus{
R, //rayon du grand cercle
r // rayon du petit cerle
}
```

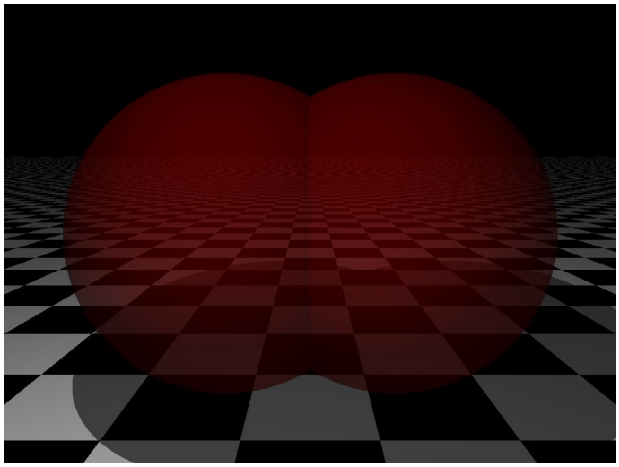
Les opérations sur les objets

Il est possible de faire intéragir les objets les uns avec les autres, de manière à les soustraire, en faire l'intersection, etc..

- “`difference`” : rend le premier objet privé des suivants.
- “`union`” : réunit tous les objets donnés.
- “`intersection`” : rend ce qui est commun à tous les objets.
- “`merge`” : fusionne les objets. Comme une union, mais dans le cas d'objets transparents, les “coutures” intérieures disparaissent







L'union a pour seul interet de pouvoir manipuler plusieurs objets ensemble lors des transformations.

```
union{
	sphere{...}
	sphere{...}
	translate <xt, yt, zt>
}
```

Les transformations

Rotations

La rotation permet de faire tourner un objet selon un ou plisieurs axes. L'angle de rotation s'exprime

en degrés (entre 0 et 360). La rotation s'effectue toujours par rapport à l'origine du repère ! De plus, l'ordre d'enchainement des rotations à son importance !

```
object{
	...
	rotate <angle_x, angle_y, angle_z>
}
```

Translations

```
object{
	...
	translate <xt, yt, zt>
}
```

Mise à l'échelle

Attention également avec la mise à l'échelle, elle va multiplier les coordonnées de l'objet par rapport aux facteurs donnés !

```
object{
	...
	scale <facteur_x, facteur_y, facteur_z>
}
```

Pour aller plus loin

Le site http://povray.free.fr fourni une documentation assez complète présentant les différentes commandes disponibles.

Par ailleurs, vous pouvez jeter un œil sur `kpovmodeler`, disponible avec KDE, qui permet de modéliser des scènes POV-Ray de manière un peu plus conviviale.

Mise en pratique

Bibliographie

http://povray.free.fr

http://www.povray.org

http://www.oyonale.com/ldc/english/wetbird.htm

http://fr.wikipedia.org