Projet Image pour Multimédia

Partie animation

Par Karim MOULAY

Licence 3 - Année 2008/09

Sommaire

Présentation :

Présentation général :

Parmi les parties vues en cours dans le cadre de ce module, j'ai choisi la partie animation en ***POV-Ray***. Pour présenter le travail, j'utilise le montage vidéo. Il y a donc quelques petits films dans lesquels il y a une série de séquences d'animations.

Pour chaque séquence, il y a une petite explication de ce qu'il va ce passé et comment le faire. Les séquences ce succèdent pour expliquer comment réaliser la scène final. C'est une sorte d'évolution.

Les vidéos principales :

Les vidéos que nous allons voir sont :

- l'animation d'un robot
- L'animation d'une sphère
- L'animation de texture
- Exemple de ce que des professionnels de Povray savent faire.

Les scènes :

Présentation général :

Le but de mon projet est de présenter les animations que nous pouvons faire avec ***Pov-Ray*** grâce à ce que nous avons vu en cour. L'intérêt est de voir comment faire des animations grâce à la simple manipulation de la variable "clock". Toute fois ; j'ai essayé aussi d'exploiter les différentes autres parties du cours, à savoir la vidéo, les transformations, la texture et le SMIL.

Après différents changements dans la manière de réaliser les choses ; j'ai gardé la version que voici. L'animation que j'ai prévue à la base a été décomposée en 4 autres animations.

Vous trouverez donc :

- Une animation qui concerne le robot en 3D (réaliser en TP).
- Une animation des textures avec un cylindre.
- Une animation que j'ai appelé super NOVA. Elle représente l'explosion d'une sphère.
- Et en dernier la transformation d'une sphère en diamant et la présentation de ce dernier.
- Une dernière petite vidéo avec un petit "best of" de ce qui ce fait, qui est possible de faire…

Toutes ses animations sont sous format vidéo et sont présentées dans une fenêtre SMIL.

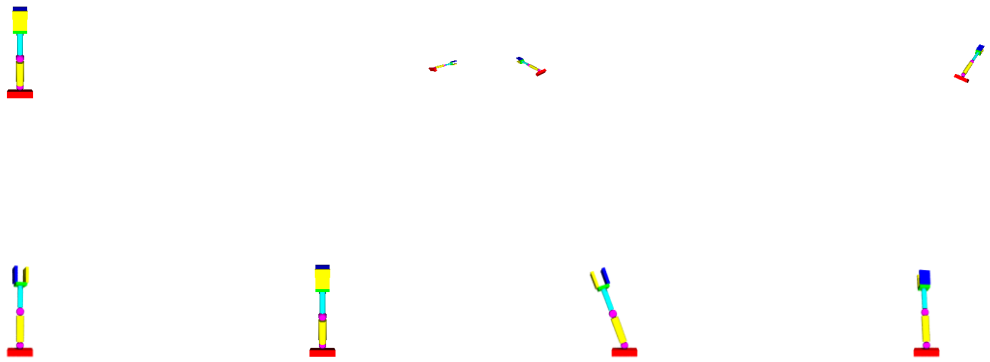
Résultat en images :

Animation robot :

Il s'agit du déplacement du robot vu en TP. Comme l'on peut le voir dans le montage que j'ai fait ; on commence par expliquer comment animer un objet. Le plus simple est de voir l'allongement d'un cylindre qui représente un axe, pour voir ce que c'est qu'une animation à une dimension :

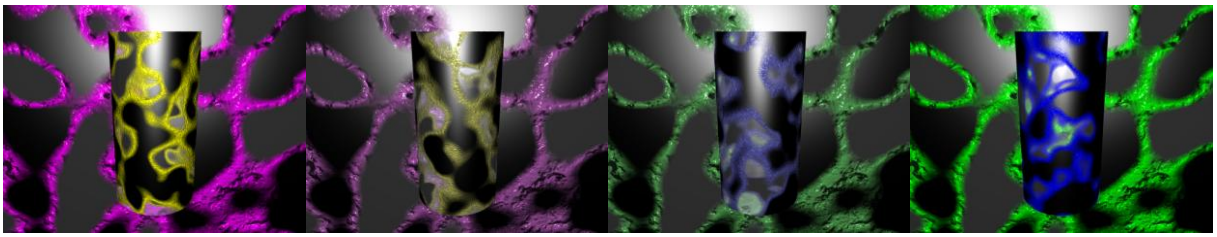
Par ma suite, une fois que l'on sait que l'on peut faire une animation à une dimension ; on jeu simultanément sur deux dimensions, et voici un aperçu de ce que ça donne :

Une fois atteint ce niveau, il suffit à présent de placer l'objet robot à la place du petit cône pour avoir l'animation souhaiter (dans mon cas, le robot). Bien sur dans cette partie, je ne parle pas du coté technique du travail, on verra ça dans le prochain chapitre :



Animation texture :

Je n'ai pas fait de tuto pour cette scène, mais on peut dire qu'elle complète l'animation du robot. Car il s'agit d'une animation faite sur le robot, mais en plus de simples se sont des variations de coordonnées que nous allons voir plus tard comment elles sont faites.

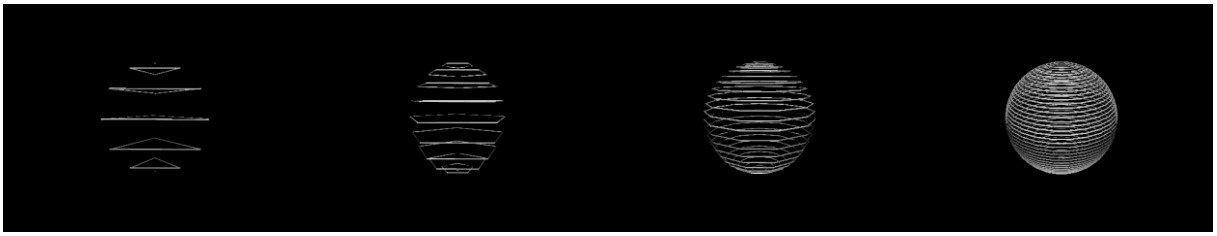


Animation de sphère :

Cette vidéo regroupe deux scènes, la dite super NOVA et la transformation de la sphère en diamant. Pour commencer, j'explique comment créer une sphère par facette. Et pour cela, je commence par ce qui est simple à faire car vue en TP, un cylindre par facette.

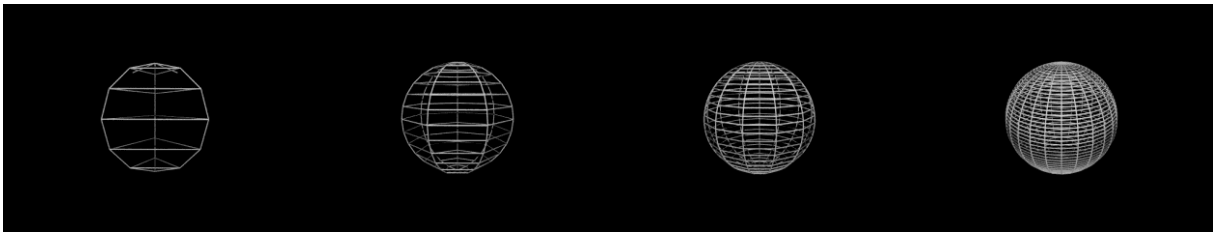
Par la suite, si l'on agit comme pour le robot sur deux axes simultanément, on peut dessiner une sphère. Et tous comme pour le cylindre, on fait évoluer la forme.

Dessin de la sphère par augmentation des parallèles



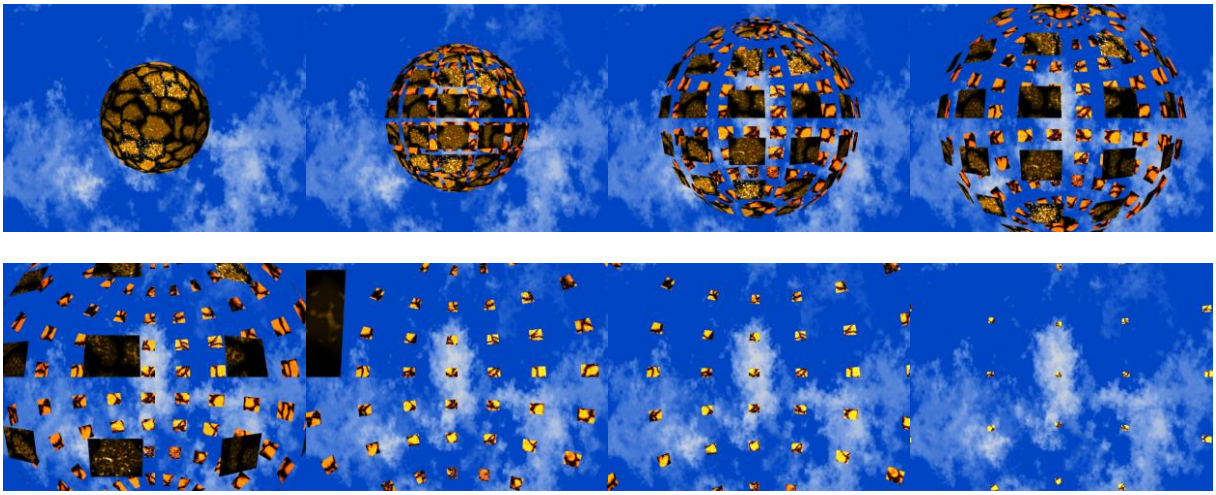
Dessin de la sphère par augmentation des méridiens

Dessin de la sphère par augmentation des parallèles et des méridiens



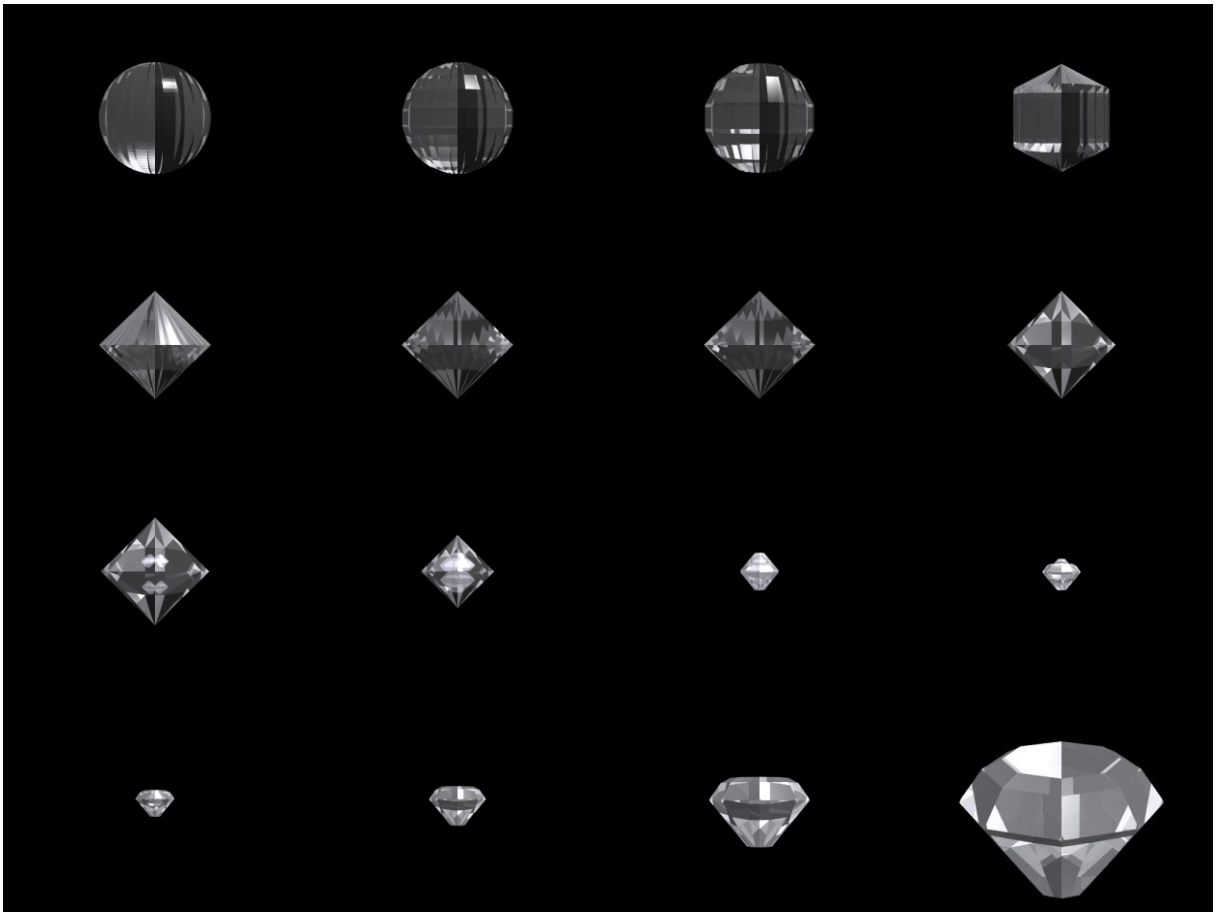
Super NOVA

Après avoir créé la sphère en facette, nous pouvons commencer à manipuler les facettes pour simuler l'explosion d'une étoile.



Transformation d'une sphère en diamant

C'est la combinaison de ce que l'on a vu avant, avec quelques artifices caché pour faire l'ensemble. Voici un petit aperçu ; mais il faut voir la vidéo.



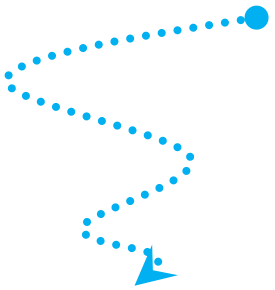
Présentation technique :

Il a était demander que dans le rapport il n'y a pas de code source. Toute fois, il m'a semblé indispensable de donner quelques fonctions utilisé pour expliquer les choses. Je n'y ai ajouté que le strict minimum (d'après mes critères).

La principal variable dont tous dépends est la variable "**clock**" Tous dépend d'elle.

i. Présentation de l'animation du robot :

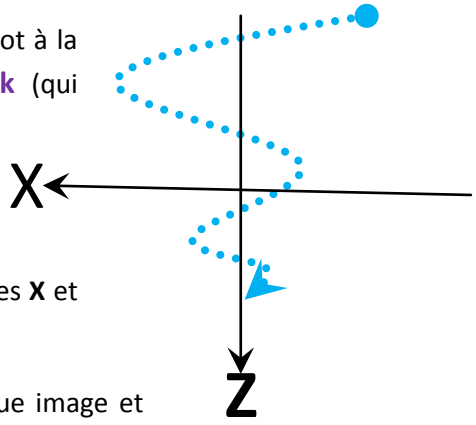
Pour cette animation ; il m'a fallut exploiter quelques fonctions mathématiques de base. L'animation est un simple déplacement du robot selon une trajectoire dont voici une représentation de la courbe utilisé:



Cette courbe est une courbe sinusoïdale avec le rayon qui diminue avec la progression de l'objet.

Voici la fonction ***Pov-Ray*** qui a permis cela:

translate < **clock** ***cos(mod(clock,24)*(pi/12))**, 0, **clock** >



Donc comme vous le voyez; cette fonction positionne le robot à la position souhaitée dans la scène. Grace à la variable **clock** (qui remplace le **X** de la fonction mathématique), le robot avance en sur l'axe des Z d'in pas régulier (**X**) et reste fixe (constante) sur l'axe des Y. Nous avons alors à faire à une simple fonction avec des variations sur **2D** et non pas sur **3D**. Ce qui donne l'objet qui ce déplace latéralement sur l'axe des **X** et qui progresse sur l'axe des **Z**.

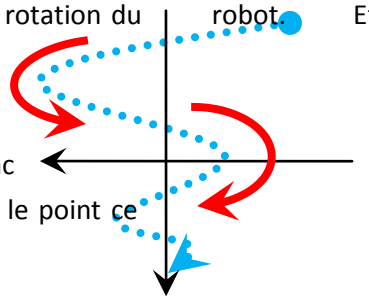
Pour créer l'animation, la variable **clock** change pour chaque image et prend une valeur différente (incrémenter ou décrémenter selon le min et max) pour changer la position de l'objet. Dans la fonction que j'ai écrit; je considère que la période est égale à 24 top. Ceci explique donc la formule de la translation sur **X**.

Vous avez aussi le socle du robot qui ce tourne pour qu'il soit toujours dans la trajectoire. Pour ce faire; voilà ce qu'il faut faire.

rotate < **0, 15*clock*(-1+div(1+cos(clock*pi/12),1)*2), 0**>

Oui, la formule peux vous paraitre un peut trop complexe. Mais… Je détail un peut plus ; vous avez **15*clock**; cette partie sert juste à faire tourner le robot selon son niveau sur la trajectoire. Je multiplie par **15** car j'ai décidé de décomposer la période de la trajectoire en **24** temps. Donc **15** est seulement **360°/24**. Mais l'autre partie; à quoi sert-elle? Voici un schéma de l'animation:

Vous voyez les flèches rouges qui servent à connaitre le sens de rotation du robot. Et comme vous le voyez; le sens de rotation change par rapport à votre position sur l'axe des Z.



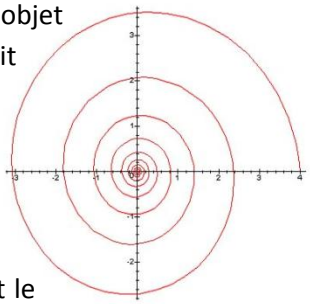
Pour avoir une image plus fluide et plus réaliste; il faut donc multiplier par ±1. **e=cos(clock*pi/12)** varie de **-1** à **1**; il est positif si le point ce trouve à gauche de **Z** et vice versa.

e'=div(e,1); cette opération permet d'avoir la partie entière de la division de **e/1**. Cette opération donne comme résultat la partie entière de **e** (ce n'est pas la bonne formule; mais elle me permet tous de même d'avoir la partie entière de **e)**. Le résultat obtenue et 0 ou 1. **(e'*2)-1** est la formule finale. Après cette opération, on obtient un résultat qui est **-1** ou **1**. Donc nous avons l'angle qui change de signe selon ça position par rapport à **Z**.

Au finale; vous obtenez une animation plus ou moins simple. Mais surtout que pour la créer les notions vues en cours suffisent largement (avec quelques ajustements) et pour la créer, il suffit tous simplement de programmer 2 lignes (ou 1 si vous avez besoin d'un effet plus basic).

L'animation finale du robot consiste en sa rotation sur lui-même avec un effet spirale (escargot).

Pour cela; il suffit d'incliner le robot d'un angle **α** et de faire tourner l'objet sur lui-même jusqu'à ce qu'il s'arrête de lui-même lorsqu'il ce serait redressé (façon de parler bien sur). Pour la rotation, rien de compliqué, il faut faire tourner la rotule du robot sur elle-même comme dans le cas précédant avec cette angle 15*clock.



Pour le redressement, c'est une diminution du rayon que fait l'inclinaison tu corps de la pièce. Comme l'inclinaison est un angle que fait le corps du robot ; c'est en réalité une diminue progressivement de par exemple 30° grâce à clock seulement.

Résultat, le robot qui est lancé sur un élan, se redresse tout doucement et progressivement.

Présentation de la texture animée :

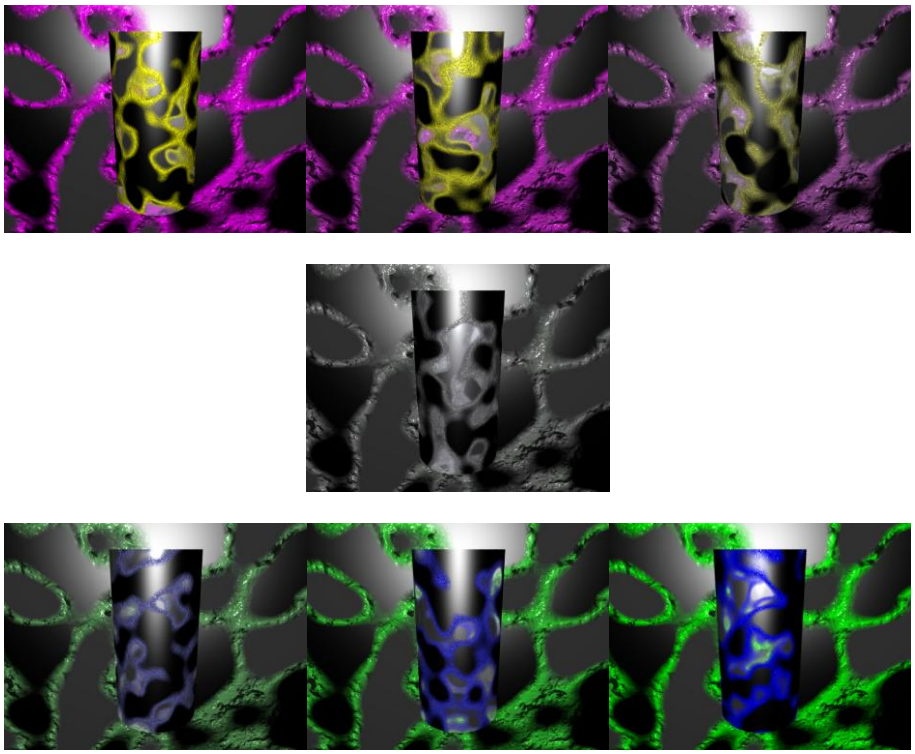
La texture animée ne change pas des autres animations. Elle est faite avec le minimum de codes. Pour la réaliser; vous voyez que les deux textures ressemblent beaucoup aux color_map vue en cours, le tous dans un bump. Seulement, ici les couleurs sont plus complexes que celles obtenues avec un color_map.

Du fait que j'avais besoin de rajouter une couleur transparente dans ma texture, et que ceci c'est avéré impossible avec cette méthode; j'ai recherché une solution sur la documentation de Povray. Et là, j'ai trouvé un effet très intéressant; texture_map. S'utilise comme color_map; mais avec la possibilité de rajouter des textures au lieu de couleurs. Donc plus de soucis.

Pour les couleurs, qui varient, ce n'est qu'un simple changement de variable. Rien de plus. Comme vous le savez la couleur est composée d'un mélange de 3 primitives; **RGB**. Il me suffit donc juste de les faire varier. Par exemple pour passer du Jaune au Bleu; il faut mettre dans la formule pigment color < 1-clock, 1-clock, clock> avec clock qui varie entre 0 et 1. Résultat, au début on commence avec <1, 1, 0> (jaune) et on finit avec <0, 0, 1> (bleu).

Idem pour le font avec ces paramètres de début <1, 0, 1> et de fin <0, 1, 0> grâce à cette fonction < 1-clock, clock, 1-clock>.

Pout les déformations de la texture, il s'agit de variations de **l'échelle** (scale) + **translation** (tanslate) + **rotation** (rotate).



Présentation des sphères

Pour les animations qui vont suivre; elles utilisent toutes un élément fondamental que j'ai créé avec la sphère en facettes. Je vous épargne le code source (sachant qu'il n'est pas compliquer à comprendre ou à réaliser pour quelqu'un qui a suivi ce module). Je me dois tous de même au minimum vous expliquer les paramètres de la fonction à laquelle je fais appel.

Sphere(r, n_p, n_m, d)

r: Rayon de la sphère.

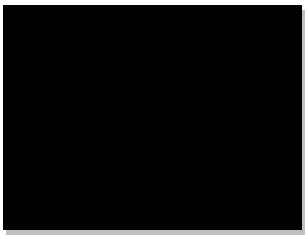
n_p: Nombre de parallèle souhaiter.

n_m: Nombre de méridien.

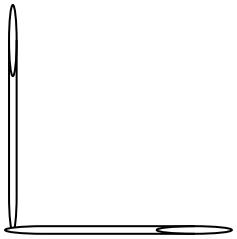
d: Déplacement. Cette option permet de déplacer les facettes pour les éloigner de leur centre de gravité (0 pour la position par défaut). Vous en verrez l'intérêt dans l'animation que j'ai appelé **SuperNova**.

La sphère – Au commencement.

A la différence; cette sphère, n'ai pas faite de polygones comme les autres. Elle est faite de cotes.



Facet

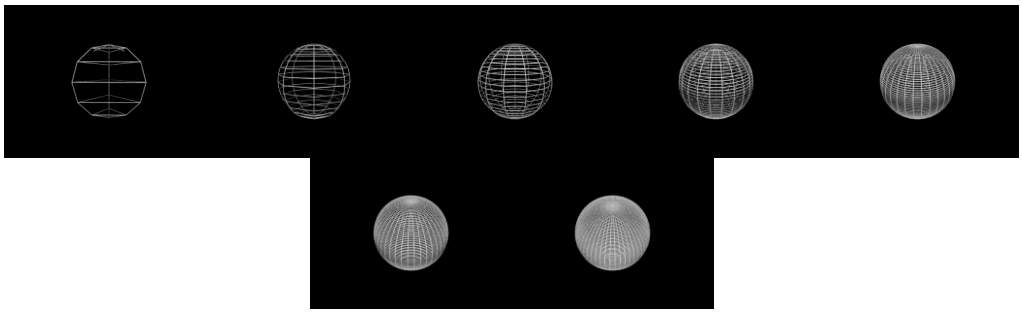


Côte

Pour le reste; il s'agit d'une augmentation simultanée des méridiens et des parallèles:

Sphere(1, clock, clock, 0)

Pour la caméra, elle aussi est en mouvement. Elle ce déplace sur l'axe des Y.



SuperNova

Pour cette animation, j'ai, comme je vous l'ai déjà dit, utilisé le déplacement de facettes. Elle n'est donc pas compliquée à réaliser grâce à la fonction que j'ai écrit:

Sphere (1, 15, 15, clock)

Pour les textures; c'est une exploitation de ce que nous avons vu en cour. Si vous regardez le code source; vous aurez peut être du mal à trouver l'appel de la sphère. Vous aurez aussi remarqué que lors de l'explosion de la sphère, à l'intérieur elle est plus rouge. Ceci est dû à la **light_source** qui ce trouve dans la sphère et qui est de couleur rouge, du moins pour l'apparence. Mais en réalité, c'est la sphère qui se trouve dans la **light_source**. Oui, cette méthode permet d'avoir la lumière qui ne sort pas de la sphère. Donc pas d'ombre en dehors (Looks_like).

Pour l'environnement; c'est une sphère avec une texture ciel au lieu du traditionnel plan.

AnimeSphere

Toute l'animation est conçue grâce à 2 objets ; la sphère en facette et le diamant (objet que j'ai réalisé lors de l'un de mes anciens projets PovRay). Pour le code source du diamant, il se base sur le code source des cubes en facette.

La scène est un ensemble de morceaux de petites animations qui ce suivent les unes derrière les autres et qui ce lancent successivement.

Première partie:

C'est une animation qui se fait sur la diminution des parallèles de la sphère. Voici comment elle se fait:

Sphere(1, clock, 15, 0)

Deuxième partie:

C'est une animation de qui se fait sur la diminution des méridiens de la sphère. Voici comment elle se fait:

Sphere (1, 1,clock, 0)

Troisième partie:

C'est ici qu'entre le deuxième objet (le diamant). Vous avez simultanément le rayon de la sphère qui diminue, et l'accroissement des 2 diamants superposés:

Sphere(1-clock, 1, 1, 0)

Et les deux diamants, c'est le changement d'échelle:

scale clock*Quatrième partie:*

L'angle de vue de la caméra diminue:

angle 70

Pour le reste, se sont des rotations de la caméra sur différents axes.

Conclusion :

Voici donc comment ce présente mon projet. Pour la démonstration; bien que ce n'ai pas encore fin, ce sera organiser et présenter dans un projet SMIL.

Sinon le projet était intéressant à réaliser. Il y a beaucoup d'autre choses que j'aurai voulu y intégré, mais cela me paraît suffisant pour un début.

Il est aussi dommage que le rendu ne soit pas fais en temps réel (comme pour SVG), mais j'ai choisie tous de même ***Pov-Ray*** pour des raisons de meilleure qualité de rendu (du moins par rapport aux connaissances que j'ai acquis dans le domaine).

Mon but secondaire dans ce projet est de démonter qu'il est possible de créer des animations et animer des objets sans avoir besoin de taper des centaines de lignes de codes.

Exemple , toute l'animation du robot, comme vous le voyez ne demande pas beaucoup de connaissance en ***Pov-Ray***. Il aura été possible de faire des effets beaucoup plus spectaculaires avec soit d'autres logiciels plus appropriés, ou encore de faire plus de codes et de faire des choses encore mieux, mais, le problème est que le rendu demanderait beaucoup plus de temps à la compilation. De plus mon but est de montrer qu'une animation qui peut paraître complexe à réaliser, peut ce faire en quelques lignes de codes (3 lignes dans mon cas).

De plus, même un débutant en ***Pov-Ray*** comme nous le sommes à notre niveau peut comprendre et/ou réaliser le programme.

Concernant la super NOVA ; il était prévu aussi que les facettes fassent des rotations aléatoire lors de l'explosion. Le problème est que le code source de la sphère n'était pas prévu pour réaliser cette manipulation. Il y avait des solutions pour y remédier; mais encore une fois, une de mes priorités était d'optimiser le plus le code pour qu'il y ait le minimum de calcul à faire pour le rendu des animations.

J'espère vous avoir convaincu (ou arriverait à vous convaincre) de la simplicité de la création des animations de base.

En attente du jour de la présentation, j'espère que la lecture de ce rapport ne vous a pas était pénible.

Rendez vous à la pratique.

Karim MOULAY