PREMIER PAS EN IMAGE DE SYNTHÈSE

Note :
La version électronique de ce document est disponible sur le site web du Stéréo-Club Français.
www.stereo-club.fr (rubrique *Doc en ligne*)
Elle inclut en annexe tous les fichiers sources qui permettent de générer les images de cette présentation.

Introduction

Dans cet article, pas d'appareil photo, ni de pellicule, de crayons, de papier,… nous allons générer des images stéréo à partir de rien (*from scratch* diront les anglicistes, *ex nihilo* répondront les latinistes). Enfin rien, façon de parler car nous allons quand même avoir besoin d'un ordinateur pour générer des images de synthèse, c'est-à-dire des images calculées mathématiquement.

Les calculs sont généralement très lourds car, une image numérique étant une matrice de points élémentaires, il faut calculer la couleur de chaque point de cette image.

Il existe un grand nombre de méthodes pour générer des images de synthèse sur ordinateur mais toutes reposent en fait sur les principes suivants :

- Modélisation de la scène, c'est à dire décrire tous les objets que l'on veut dessiner ainsi que leurs positions respectives. Par exemple, une table est constituée d'un plateau rectangulaire et de quatre pieds cylindriques aux quatre coins, sur la table on place un verre cylindrique transparent.
- Éclairage de la scène, il faut au moins une source lumineuse sinon noir absolu garanti !
- Définition du point de vue, c'est à dire là où se trouve l'observateur (on parle plus souvent de *camera virtuelle*) et dans quelle direction il porte son regard.

Le Lancer de Rayon

Une des méthodes les plus courantes est celle dite du « lancé de rayons » (*ray-tracing* en anglais). Elle simule la propagation de la lumière dans un environnement en « lançant » des rayons lumineux pour déterminer comment ils vont interagir avec les objets de la scène. Pour simplifier, on peut dire que l'on remonte le trajet des rayons lumineux depuis l'observateur vers la scène puis jusqu'aux sources lumineuses et chaque fois que l'on va rencontrer un objet de la scène, on va calculer, selon sa texture, sa surface, les rayons dérivés par les règles classiques de l'optique (réflexion, diffusion, réfraction,…). La couleur des objets rencontrés ainsi que la couleur des sources lumineuses va donner sa couleur au point situé à l'origine de la remonté (donc à l'extrémité du rayon). On répète l'opération pour tous les points de l'image à générer : pour chaque point on détermine ainsi sa couleur et peu à peu l'image se construit.

On imagine donc la quantité phénoménale de calculs à effectuer : sur une scène complexe composée de milliers d'objets, les calculs pour un seul point vont être complexes et il faut répéter cela plus d'un million de fois pour une image tenant sur un écran classique de 17 pouces. On comprend donc que plus rapide sera l'ordinateur, meilleur ce sera !

Les images générées peuvent être très réalistes puisque des phénomènes optiques comme les réflexions et les transparences sont restitués intrinsèquement. Mais du fait de son caractère très gourmant en temps de calcul, le lancé de rayon est souvent délaissé pour des méthodes moins fidèles certes mais plus rapides (pour les jeux vidéo avec représentation en 3D).

Les outils

L'outil de base se nomme ici POV-Ray. C'est un logiciel gratuit pouvant fonctionner sur une multitude d'ordinateurs différents. Existant depuis plus de 15 ans, il dispose de quantités d'exemples, outils, tutoriaux et objets tout faits prêts à être réutilisés. La version PC dispose en plus d'un éditeur efficace permettant de définir la mise en scène des objets à représenter.

POV-Ray est aussi le seul logiciel d'image de synthèse à avoir été utilisé dans l'espace ! En 2002, le Sud-africain Mark Shuttleworth a généré l'image *Atteindre les étoiles* dans la Station Spatiale Internationale.

Les sites web montrant des images réalisées avec POV-Ray abondent. On peut citer le site de l'IRTC qui organise tous les mois un concours d'images de synthèse et celui (en français) de Gilles Tran aux intrigantes images et aux commentaires poético-surréalistes (quel dommage qu'il ne connaisse pas la stéréo !).

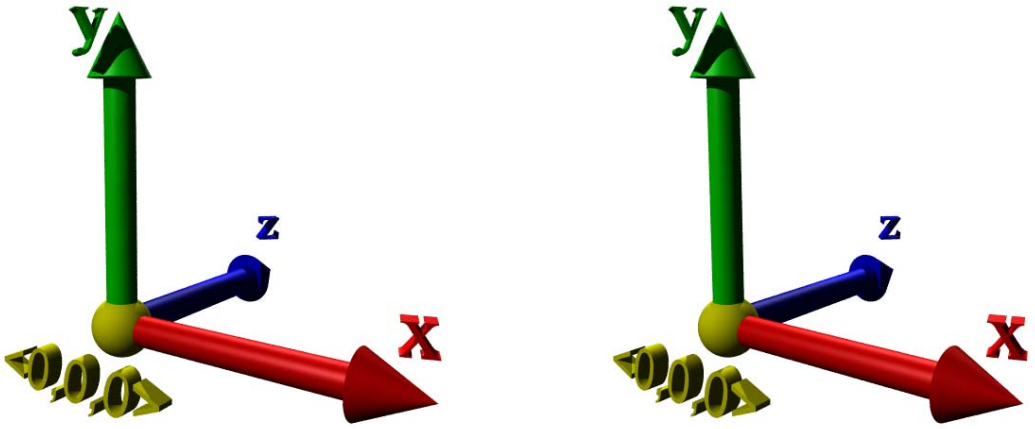
Créons notre première image

Comme nous l'avons vu au début, pour générer une image de synthèse nous devons définir trois choses : les objets, l'éclairage et le point de vue.

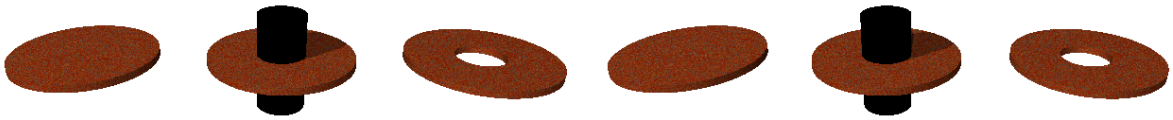
Objets composant la scène

POV-Ray utilise le système de coordonnées cartésiennes x, y et z pour situer un objet dans l'espace. Dans POV-Ray l'espace de travail est infini et les dimensions sont arbitraires : on choisira par exemple des millimètres pour représenter une pièce de monnaie, des centimètre pour du mobilier et des mètres pour des bâtiments.

Traditionnellement, la hauteur est déterminée par la coordonnée *y* et le repère est orienté comme suit :



POV-Ray utilise un langage de description pour décrire les objets composant une scène. On part de formes élémentaires de base (sphère, cylindre, cône, plan, cube,…) qu'on peut assembler pour construire des objets plus complexes. On peut également réaliser des opérations telles que intersection, différence, fusion,… Par exemple pour créer une rondelle, on fera la différence entre un cylindre très aplati et un autre plus petit pour faire un trou au milieu.



Pour modéliser un objet plus complexe, comment doit-on s'y prendre ?

> *[…] de diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre.*
> Descartes.

Il faut donc décomposer l'objet en autant de composants élémentaires, facilement identifiables puis de recomposer l'objet complet à partir de ses sous-parties. Prenons l'exemple d'une simple table en bois : elle est composée d'un plateau (un parallélépipède aplati) et de quatre pieds (quatre cylindres allongés).

Éclairage

Sans lumière la scène sera entièrement noire ! Il faut donc ajouter une ou plusieurs sources lumineuses pour éclairer nos objets. Dans POV-Ray la lumière la plus simple est un point qui illumine dans toutes les directions. On va donc la placer assez loin et en hauteur pour qu'elle ne soit pas directement visible et donner des ombres qui se projettent au sol.

Point de vue

Le point de vue est la position de la caméra virtuelle qui va prendre la scène en photo. POV-Ray propose de nombreux paramètres pour les caméras mais les deux paramètres fondamentaux sont la position de la caméra *location* et le point visé par elle *look_at*. Par défaut, la caméra est bien verticale (pas besoin de niveau à bulle !).

Les fichiers *.pov*

Un fichier *.pov* contient la description d'une scène. C'est un fichier au format texte et on utilisera l'éditeur de POV-Ray pour le modifier. Nous choisissons ici les centimètres comme unité pour la suite de la présentation.

Après avoir installé POV-Ray sur l'ordinateur, partons d'un fichier vide que nous nommerons *Etape_02.pov* et définissons la caméra pour notre scène :

```
camera {
    location <150, 180, -220>
    look_at  <0,    90,    0>
}
```

La caméra est à hauteur d'homme : 1,80 m vers le haut en *y*, un peu à droite (1,5 m sur l'axe *x*) et en arrière (2,2 m en arrière sur l'axe *z*) et elle regarde vers le point zéro mais un peu au dessus (90 cm au dessus exactement).

L'éclairage maintenant (Q*ue la lumière soit !*) :

```
light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}
```

Une lumière blanche R = G = B = 1 (les trois composantes Rouge, Vert et Bleu sont au même niveau) située à 5 m de haut, 2 m à gauche et 6 m en arrière.

Si nous lançons le calcul de cette scène (on parle aussi de « rendu » de l'image), nous obtiendrons une image entièrement noire bien qu'une source lumineuse soit présente dans notre scène. Cela s'explique par le fait qu'il n'y a aucun objet pour arrêter les rayons lumineux (les renvoyer vers nous plus exactement).

C'est un peu comme une scène de théâtre vide… C'est même pire car il n'y a même pas de plancher ! Pour bien se repérer, je trouve pratique de définir un dallage au niveau du sol (donc y = 0) pour bien positionner les objets dans l'espace. Le dallage régulier permet aussi de mesurer visuellement les distances. On pourra toujours le changer ou le cacher au besoin plus tard.

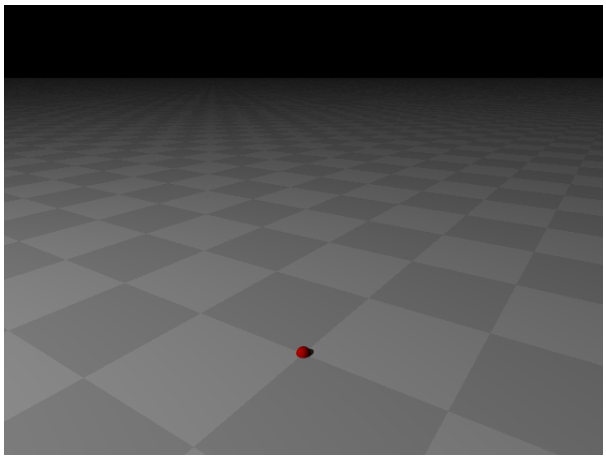
```
plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
```

Le plan, infini, est perpendiculaire à l'axe *y* donc horizontal, il est à la position zéro sur cet axe et on lui applique une texture spéciale de type *échiquier* avec des cases alternativement blanches et gris clair. Avec le paramètre d'échelle *scale*, on agrandit le dallage 100 fois, chaque dalle fait donc 1 mètre de côté.

Pour mieux se repérer, on va placer quelque chose au point 0,0,0 : une petite sphère rouge :

```
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }
```

La sphère fait 5 cm de rayon et est de couleur rouge (seule la composante R est non nulle).
Voilà l'image générée avec notre « plancher », c'est un peu sinistre mais quand même mieux que le noir intégral !



Menuiserie

Construisons maintenant notre table en bois. Quand on conçoit des objets, il est conseillé de définir d'abord un modèle de l'objet. C'est ce modèle qui est ensuite utilisé pour placer les objets dans la scène. De cette manière, si notre table est utilisée dans notre scène pour faire une rangée de tables, une correction apportée au modèle seul affectera toutes les tables bâtie sur ce modèle.

Pour utiliser des textures prédéfinies (bois, métal, verre,…), nous allons inclure le fichier standard de définition des textures (à placer au tout début du fichier) :

```
#include "textures.inc"
```

Nous allons définir (ou *déclarer*) une table de 1 x 1,5 m et 90 cm de haut. En suivant les conseils de Descartes, nous définissons deux modèles d'objets élémentaires : un plateau et un pied de table.

```
#declare plateau = box { <-75,0,-50> <75,2.5,50>
                     texture { Tan_Wood scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5
                     texture { Tan_Wood scale 0.5 rotate x*90 } }
```

Il est aussi conseillé de construire tous les modèles des objets autour du point zéro. Cela facilite la manipulation des objets, en particulier les rotations. Le plateau s'étend donc de 75 cm de chaque côté du point zéro en largeur, de 50 cm de chaque côté en profondeur et fait 2,5 cm d'épaisseur.

Les pieds sont de simples cylindres de 90 cm de haut et 5 cm de diamètre. On utilise une texture de bois assez foncé pour le plateau et les pieds. La texture des pieds est tournée de 90° pour qu'ils semblent taillés dans le fil du bois. Elle est également plus fine (paramètre d'échelle *scale* = ½) pour représenter du bois plus dur. Le plateau lui a une texture agrandie x2 pour un bois plus grossier.

Assemblons maintenant la table :

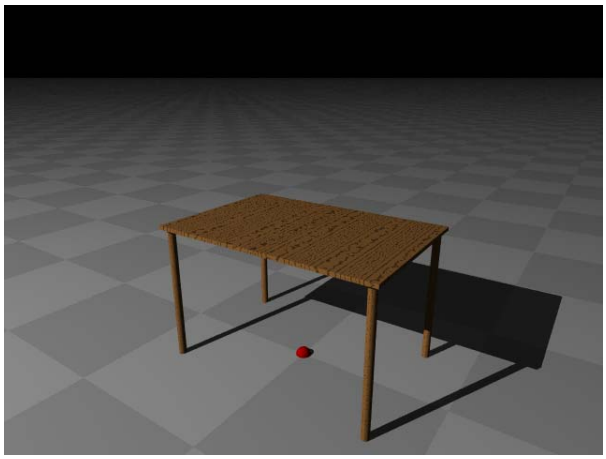
```
#declare table =
    union {
        object { plateau  translate <  0,90,  0> }
        object { pied     translate <-70, 0,-45> }
        object { pied     translate < 70, 0,-45> }
```

```
        object { pied     translate <-70, 0, 45> }
        object { pied     translate < 70, 0, 45> }
    }
```

On a soulevé le plateau de 90 cm pour glisser dessous quatre pieds placés à chacun des coins. Notez qu'il faut placer les éléments constitutifs dans des objets génériques *object*. Un modèle de table est maintenant défini, on va maintenant l'utiliser pour placer un exemplaire de la table dans la scène :

```
object { table }
```

Voici le résultat de la nouvelle scène :



Comme nous n'avons pas spécifié de translation elle se trouve à l'endroit défini dans le modèle, donc juste au-dessus du point zéro.

Et le relief ?

Comme c'est l'ordinateur qui fait tout le travail de calcul, il est tentant de lui demander de générer une nouvelle image de la même scène avec un point de vue légèrement décalé pour constituer des couples d'images stéréo.

Au début, je déterminais "à la main" les deux positions de la caméra pour les deux vues gauche et droite. C'est un peu fastidieux si on veut avoir une translation correcte, mesurable et respectant le parallélisme des axes de visée. J'ai donc fait quelques recherches sur la génération d'images stéréo avec le logiciel de ray-tracing POV-Ray. Voici ce que j'ai trouvé de plus intéressant :

1) Une attache stéréo virtuelle

Avec POV-Ray on peut représenter les phénomènes optiques comme les réflexions, Glenn McCarter a donc conçu une attache stéréo virtuelle construite en plaçant quatre miroirs devant la camera ! Il fallait y penser même s'il semble que l'on ai les mêmes problèmes qu'avec une vraie attache : déformations trapézoïdales, images étroites par exemple.

2) Calcul automatique des positions Gauche et Droite

Joerg Schrammel a écrit un fichier à inclure qui sert à calculer automatiquement les positions gauche et droite. C'est effectivement plus malin et plus pratique que de calculer les deux positions manuellement. Il faut quand même corriger un peu le calcul pour translater aussi le point visé par la caméra si souhaite éviter la convergence des axes optiques.

3) Une version stéréoscopique de POV-Ray

POV-Ray est distribué dans le mode "open source", c'est-à-dire que le code source est disponible. On peut le modifier pour générer des nouvelles versions « non officielles » de POV-Ray.

Hermann Voßeler a donc travaillé sur une version modifiée appelée *StereoPOV* qui calcule **simultanément** les deux points de vue et produit directement une image comportant les deux vues du couple stéréo côte à côte ! D'après son auteur, on gagne entre 25 et 60 % de temps de calcul car certaines opérations peuvent être partagées entre les deux vues.

Des nouvelles commandes permettent de contrôler la camera stéréoscopique. Je conseille d'intervenir sur les deux paramètres suivants :

- *stereo_base* La valeur de la base stéréo, exprimée dans l'unité de mesure de la scène.
- *window_distance* La distance de la fenêtre stéréo dans la même unité.

C'est assez magique : il suffit d'activer le mode stéréo, de définir les deux paramètres ci-dessus et de lancer le rendu de l'image. On obtient, en une seule opération, les deux vues du couple avec la fenêtre stéréo impeccablement positionnée là où on le voulait. Aucun problème d'alignement, même pas besoin d'utiliser un logiciel de montage stéréo comme Stéréo-Vue, SPM ou AnaBuilder !

Notre table en relief

On utilisera maintenant StereoPOV à la place de la version standard de POV-Ray. Il convient bien sûr, de télécharger et d'installer StereoPOV sur son ordinateur.

Pour obtenir notre table en relief, démarrer StereoPOV, recharger notre fichier *.pov* contenant la scène et y apporter quelques petites modifications. StereoPOV fonctionne aussi en mode non stéréo qui est plus rapide et donc plus commode dans les phases de mise au point de la scène. Pour passer facilement d'un mode à l'autre et contrôler la caméra stéréo, on va définir trois constantes en début du fichier :

```
#declare STEREO       = 1;
#declare STEREOBASE   = 7;
#declare STEREOWINDOW = 200;
```

La première constante contrôle le mode stéréo (1) ou mono (0) et les deux autres définissent une camera stéréo très standard : 7 cm de base et une fenêtre stéréo positionnée à 2 mètres de distance.

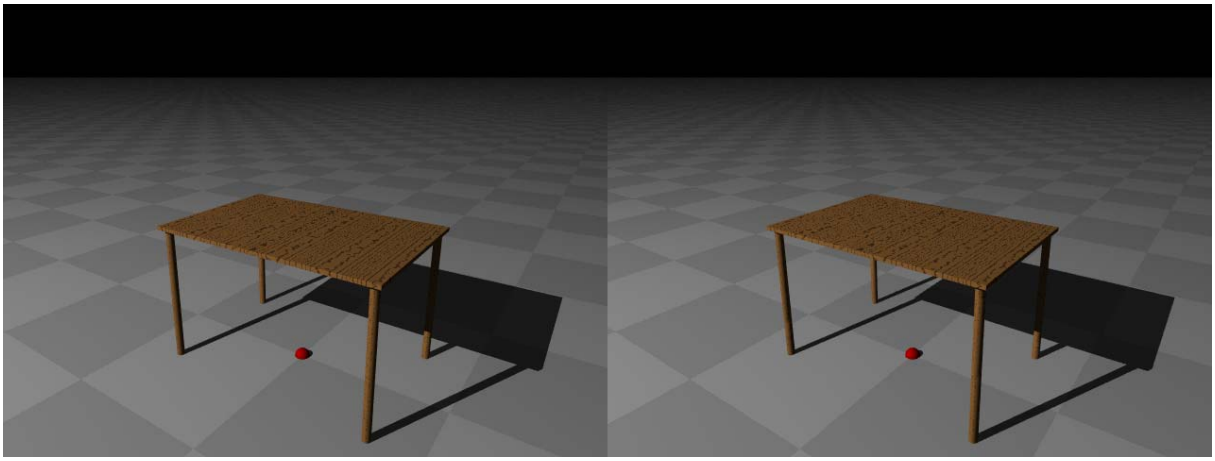
Pour utiliser une version « non officielle » de POV-Ray, il est obligatoire d'ajouter au fichier *pov* la ligne suivante :

```
#version unofficial stereopov 0.2;
```

Dans la définition de la caméra, on ajoute les lignes suivantes (en gras) qui vont activer le mode stéréo :

```
camera {
    location <150, 180, -220>   look_at <0,90,0>
#if (STEREO)
   stereo_base STEREOBASE
   window_distance STEREOWINDOW
   cross_eyed
#end
}
```

On peut omettre la ligne *cross_eyed* (vision croisée) si on désire générer un couple en vision parallèle. On relance le calcul et hop ! on obtient un couple stéréo et notre table est en relief !



Quelques objets complémentaires

Le ciel

Pour meubler un peu notre scène, nous allons créer quelques objets en plus. En premier, remplacer le sinistre fond noir par un ciel plus expressif.

Inclure les définitions des couleurs standard (ligne à ajouter au début du fichier) :

```
#include "colors.inc"
```

Utiliser la primitive *sky_sphere* (sphère du ciel). Par exemple, pour une ambiance de coucher de soleil (extrait d'un fichier d'exemple fourni avec POV-Ray) :

```
sky_sphere {
    pigment {
        wrinkles
        turbulence 0.3
        omega 0.707
        octaves 5
        color_map {
            [0.0 color DustyRose * 2.5]
            [0.2 color Orange ]
            [0.8 color SlateBlue * 0.25]
            [1.0 color SkyBlue]
        }
        scale <0.5, 0.1, 1000>
    }
}
```

Des verres sur la table

Il est très facile de modéliser un verre ordinaire : on part d'un cylindre avec une texture *verre* prise dans les textures standard et on l'évide en faisant une différence avec un cylindre plus petit. Attention, il faut lui laisser un fond ! C'est pour cela que le deuxième cylindre (celui qui creuse le verre) commence à 1 cm de hauteur pour laisser un fond de 1 cm d'épaisseur.

```
#declare verre =
    difference {
        cylinder { <0,0,0> <0,15,0> 3 }
```

```
        cylinder { <0,1,0> <0,16,0> 2.8 }
        texture { NBglass }
    }
```

Notez que le cylindre servant à creuser le verre est plus long que le cylindre de verre lui-même : il s'arrête à 16 de hauteur au lieu de 15. Il faut toujours faire dépasser un peu les objets servant d'emporte-pièce pour éviter la création d'une surface infiniment mince qui risquerait de perturber les calculs.

Un verre vide, c'est un peu triste, nous allons mettre quelque chose dedans : du liquide et une paille.

```
#declare paille =
    difference {
        cylinder { <0,0,0>  <0,20,0> 0.30 }
        cylinder { <0,-1,0> <0,21,0> 0.28 }
        texture { Candy_Cane }
    }
```

La paille est un cylindre évidé sur lequel on applique la texture *sucre d'orge* rouge et blanche.

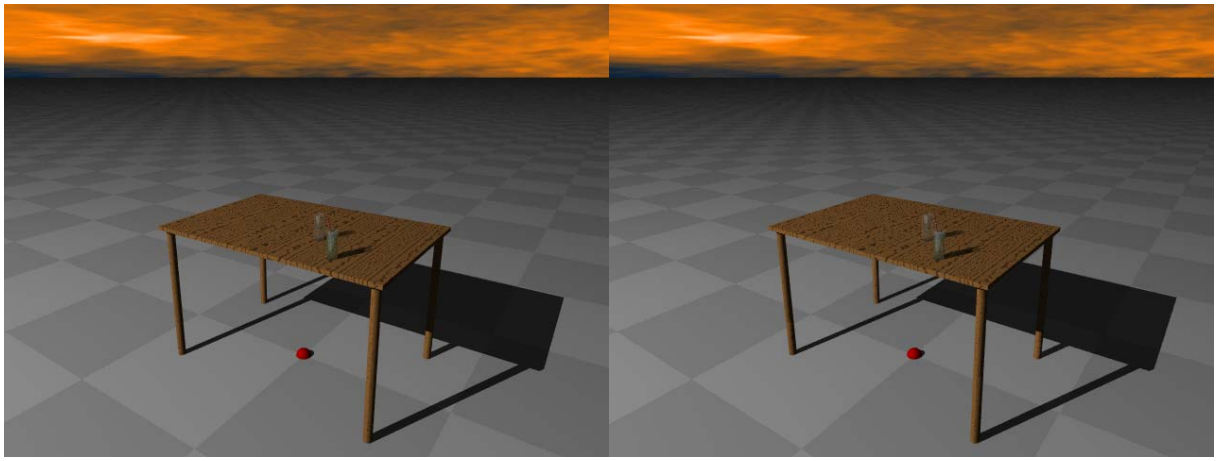
Le verre rempli est construit à partir du verre standard et de la paille. Notez la légère inclinaison de la paille de 10° et son décalage pour qu'elle semble reposer contre le bord interne du verre. Pour le liquide, on place juste dans le verre un autre cylindre qui a une texture transparente, par exemple la texture *bouteille de vin*.

```
#declare verre_rempli =
union {
    object { verre }
    object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
    object { paille rotate z*10 translate <1,1,0> }
}
```

Voilà ! Il ne reste plus qu'à placer les objets sur la table à différentes positions :

```
object { verre        translate <20,92.5,-15> }
object { verre_rempli translate <40,92.5,-35> }
```

et à recalculer la scène.



Un gros plan

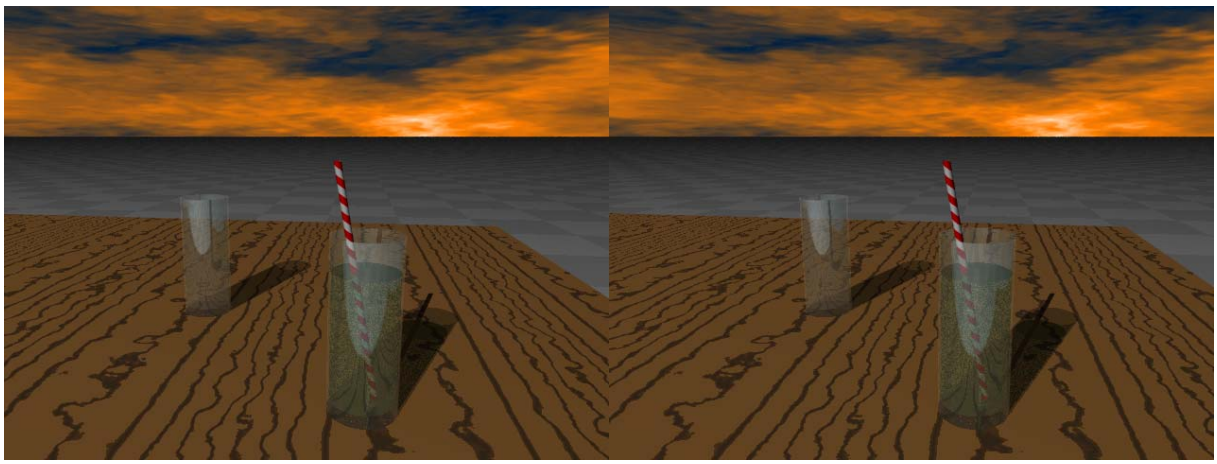
Pour faire un gros plan sur les verres, il suffit de changer la position de la caméra pour s'approcher de la table et changer aussi souvent le point visé :

```
    location <40, 115, -70>   look_at <30,100,0>
```

La base stéréo n'est plus adaptée et doit être modifiée :

```
#declare STEREOBASE   = 2;
#declare STEREOWINDOW = 30;
```

avec une base de 2 cm, c'est d'une caméra macro-stéréoscopique dont on dispose maintenant !



Un titre

POV-Ray peut aussi manipuler et afficher du texte. Ajoutons donc un titre à notre scène.

Définition d'une couleur pour le texte :

```
#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;
```

Deux lignes de texte :

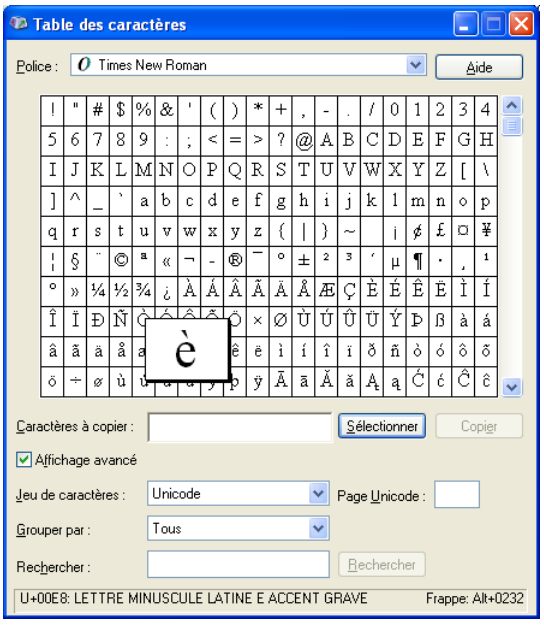
```
text {
   ttf "times.ttf" "L'image de synthese en relief, c'est facile avec" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
}

text {
    ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
    pigment { zBrightGold }
    finish { reflection .25 specular 1 }
    scale <3, 5, 10>
    rotate <0, -5, 0>
    translate <28, 114.5, -50>
}
```

Le texte est défini par la police à utiliser, ici du *Times* et par le texte à écrire. La suite de la commande permet de préciser la couleur, la finition (aspect brillant de la surface par exemple), la taille sur les trois axes (on notera que la deuxième ligne a bien plus de « profondeur » : x10 au lieu de x3 pour la première) et la position dans l'espace (rotation et translation).

Notes :

1) **Polices** : Il faut spécifier le nom du fichier contenant la police (fichiers *TrueType .ttf*). Pour obtenir ce nom, ouvrir le *Panneau de configuration* de Windows et ouvrir l'élément *Polices*. Le mode d'affichage *Détails* montre alors les noms des fichiers.

2) **Caractères accentués** : On voit que l'accent grave a été omis sur le mot *Synthèse*. Logiciel américain d'origine, POV-Ray ne comprend pas d'emblée nos caractères accentués. Il faut en premier lui indiquer que l'on souhaite utiliser un jeu de caractères étendu.
   Ajouter au fichier *pov* la ligne suivante (au début du fichier) :
   ```
   global_settings { charset utf8 }
   ```
   Remplacer le **e** par le code **\u00E8** :
   ```
   ttf "times.ttf" "L'image de synth\u00E8se en relief,...
   ```
   Pour trouver le code du caractère accentué, utiliser l'utilitaire *Table de caractères* de Windows : sélectionner le caractère désiré et son code est affiché en bas à gauche.



Titrage de photos stéréo existantes

Les fonctions de tracé de texte peuvent aussi nous servir à réaliser des titres qui seront placés sur des photos stéréo classiques. Enregistrer notre scène dans un nouveau fichier et supprimer tous les objets à l'exception des deux lignes de texte.

Pour faciliter l'insertion du titre dans l'image, on demandera à POV-Ray de générer l'image sur un fond transparent. Pour cela, ouvrir le fichier de configuration de POV-Ray (POV-Ray et StereoPOV utilisent le même fichier de configuration) par la commande *Edit master POVRAY.INI* du menu *Tools* et ajouter la ligne suivante à la fin du fichier :

```
Output_Alpha=on
```

Lancer le rendu de l'image. Ensuite ouvrir dans un logiciel de traitement d'image l'image à titrer et le titre qui vient d'être généré. Copier le titre et le coller par-dessus l'image.

Réalisation de fantôgrammes

Les fantôgrammes (ou phantograms en anglais[1]) sont à la mode en ce moment ! Ces anaglyphes à la perspective forcée qui semblent sortir de la feuille où ils sont imprimés sont effectivement très spectaculaires !

En prise de vue réelle, ils demandent une grande précision à toutes les étapes de la réalisation. L'image de synthèse ici nous facilite la tâche : il est facile dans le monde virtuel d'avoir un angle d'exactement 45° ou des éléments rigoureusement parallèles.
Le principe des fantôgrammes est de réaliser la prise de vue à 45° au-dessus de l'objet à reproduire, de corriger les perspectives avec un logiciel de traitement d'image, d'imprimer l'anaglyphe et de l'observer sous le même angle de 45°[2].

Nous allons donc déplacer notre caméra virtuelle pour quelle soit positionnée à 45° au dessus du verre avec la paille (nous repartons de la scène en gros plan en la simplifiant, un seul verre et sans les titres). Celui-ci est positionné dans l'espace en <40, 92.5, -35> c'est là que notre caméra regardera (paramètre *look_at*). Pour obtenir un angle de 45° depuis ce point, il suffit de s'éloigner (ici sur l'axe *z*) et de monter (axe *y*) de la même distance. En s'éloignant de 25 unités, notre caméra devient :

```
location <0+40, 25+92.5, -25-35>
look_at  <40,92.5,-35>
```

Notez l'utilisation d'expressions mathématiques pour rendre la position plus compréhensible. Les valeurs de la position du verre (40, 92.5, -35) se retrouvent maintenant trois fois dans notre scène et la valeur d'éloignement (25) deux fois. Une meilleure manière de procéder est d'utiliser des définitions de constantes.

Avant la définition de la caméra ajouter les deux lignes suivantes :

```
#declare Position_Verre = <40,92.5,-35>;
#declare Distance_Fanto = 25;
```

La première ligne définit la position en trois dimensions du verre et la seconde l'éloignement de la caméra par rapport au sujet. La définition de la caméra devient :

```
location Position_Verre+<0,Distance_Fanto,-Distance_Fanto>
look_at  Position_Verre
```

Notez que l'on peut ajouter des coordonnées entre elles pour calculer la position de la caméra.
On utilise maintenant la constante pour positionner le verre :

```
object { verre_rempli  scale y*0.5  translate Position_Verre }
```

De cette manière, si on souhaite par la suite modifier l'éloignement ou la position du verre, il suffira de changer la définition des deux constantes et les nouvelles valeurs seront utilisées pour placer le verre et calculer les paramètres de la caméra.

Mise en place d'un cadre

Pour corriger les perspectives dans les images produites on utilise, en prise de vue réelle, un cadre qui donne des angles droits et la dimension de la scène. Ici on va simuler ce cadre en en

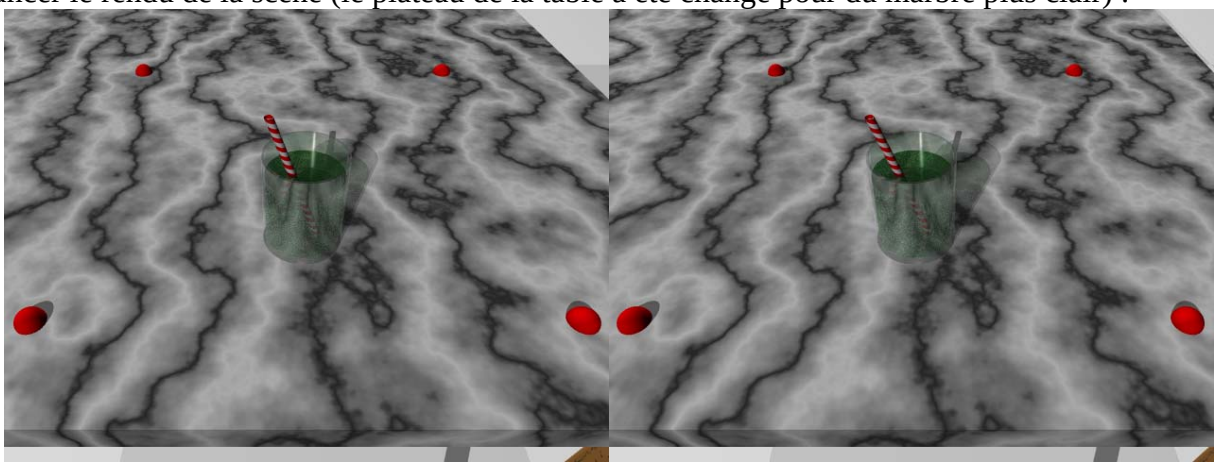
[1] Les fantôgrammes sont aussi connus sous les noms de « pop-up anaglyph », « anaglyphe surgissant », « anaglyphe en jaillissement », « morphoglyphe », « phantaglyph® »,…
[2] Les fantôgrammes, pour leur majorité, sont créés avec un angle de 45° mais tous les angles de 0 à 90° sont – en théorie – possibles. De plus quand on parle de fantôgramme à 30°, on ne sait pas s'il s'agit de 30° depuis la verticale ou depuis l'horizontal, 45° évite cette ambiguïté !

matérialisant les quatre coins. Nous pourrions également utiliser pour cela le pavage régulier du sol mais comme le dessus de la table le masque partiellement, il est plus simple de définir notre propre cadre.

Le cadre est simplement constitué de quatre petites sphères (à peine visible en haute résolution) placées en carré autour du verre. Ici encore la définition de constantes facilite la définition d'objets ayant des caractéristiques communes (hauteur, taille,…).

```
#declare pCadre =  18;     // Taille du cadre
#declare pDia   =   1;     // Diamètres des billes
#declare xC     =  40;     // Position en x,y et z du centre du cadre
#declare yC     =  92.5;
#declare zC     = -25;
sphere { < pCadre+xC, yC, pCadre+zC>, pDia pigment {color <1,0,0>} }
sphere { <-pCadre+xC, yC, pCadre+zC>, pDia pigment {color <1,0,0>} }
sphere { <-pCadre+xC, yC,-pCadre+zC>, pDia pigment {color <1,0,0>} }
sphere { < pCadre+xC, yC,-pCadre+zC>, pDia pigment {color <1,0,0>} }
```

Lancer le rendu de la scène (le plateau de la table a été changé pour du marbre plus clair) :



Les billes rouges du cadre ont été grossies pour être plus visibles, en pratique elles doivent être le plus petit possible pour que le pointage de leur centre soit en suite le plus précis possible.

Quelques conseils :

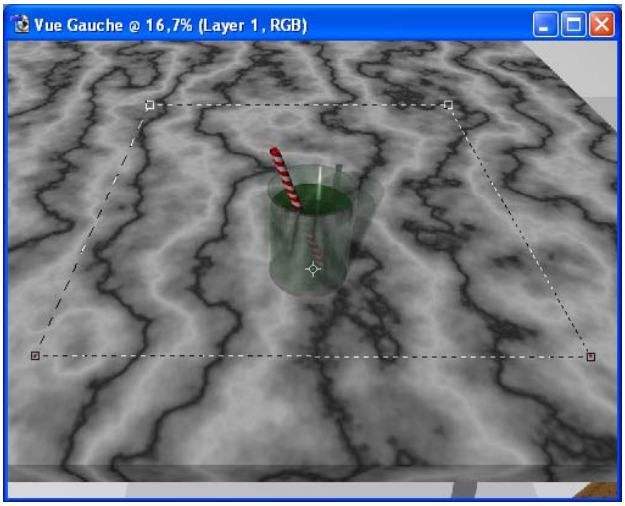
1) Il n'est pas obligatoire de cadrer finement l'image. Il vaut mieux générer une image de plus grande taille et découper dedans la zone utile.
2) Les objets plutôt plats marchent mieux pour construire des fantôgrammes. Le verre haut a été transformé en verre à whisky en lui appliquant un facteur 0,5 sur l'axe vertical.
3) Des auteurs conseillent d'utiliser une base stéréo standard (6,5 cm) quelque soit la position de prise de vue de manière à bien restituer la taille réelle des objets. D'autres par contre préconise d'augmenter la base par l'inverse du sinus de l'angle (donc d'augmenter de 40% pour un angle de 45°).

Correction des perspectives

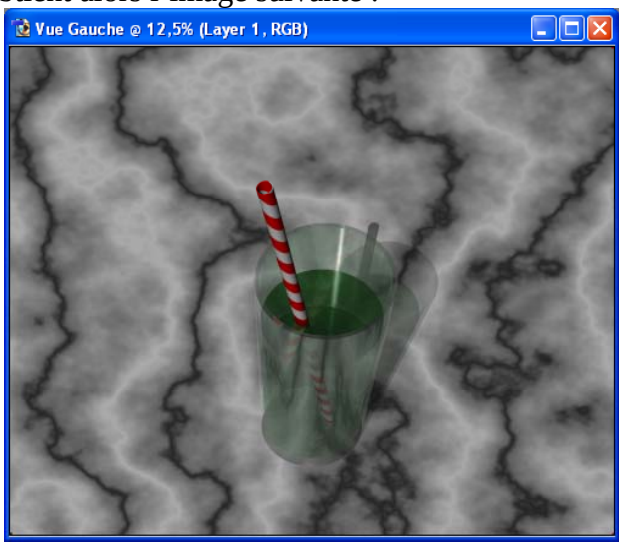
Nous avons vu que la correction des perspectives est le « truc » des fantôgrammes, c'est ça qui va les faire sortir du support et les faire apparaître sous leur aspect d'origine. La transformation mathématique est assez complexe mais, heureusement, elle peut être réalisée facilement avec des outils standard de traitement d'image.

Utilisation de Photoshop

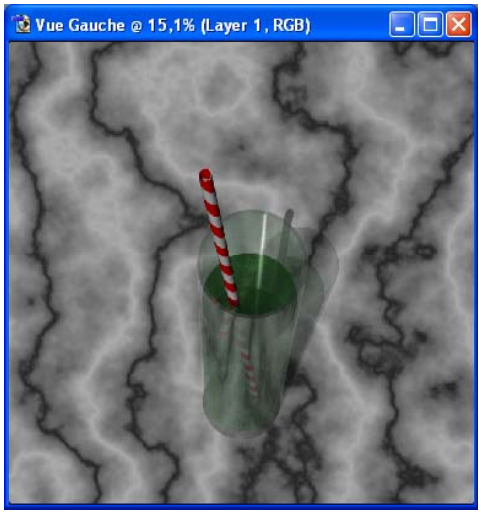
L'outil de découpe de Photoshop a une particularité très intéressante : au lieu de tracer un rectangle de sélection avec la souris on peut déplacer chacun des coins indépendamment des autres. Les deux vues du couple stéréo sont à traiter de la même manière. Ouvrir une vue et, avec l'outil de *Découpe* (ou *Crop Tool*) sélectionner notre cadre virtuel. Il faut cocher l'option *Perspective* pour pouvoir déplacer chaque coin indépendamment. Utiliser un zoom important pour positionner finement chaque coin sur la bille rouge correspondante :



Valider l'opération. On obtient alors l'image suivante :



Photoshop a éliminé les parties de l'image en dehors du cadre et a rétabli les angles droits, il faut maintenant rétablir les proportions de l'image. A l'origine notre cadre est un carré. Nous allons donc redimensionner l'image pour quelle soit carrée :

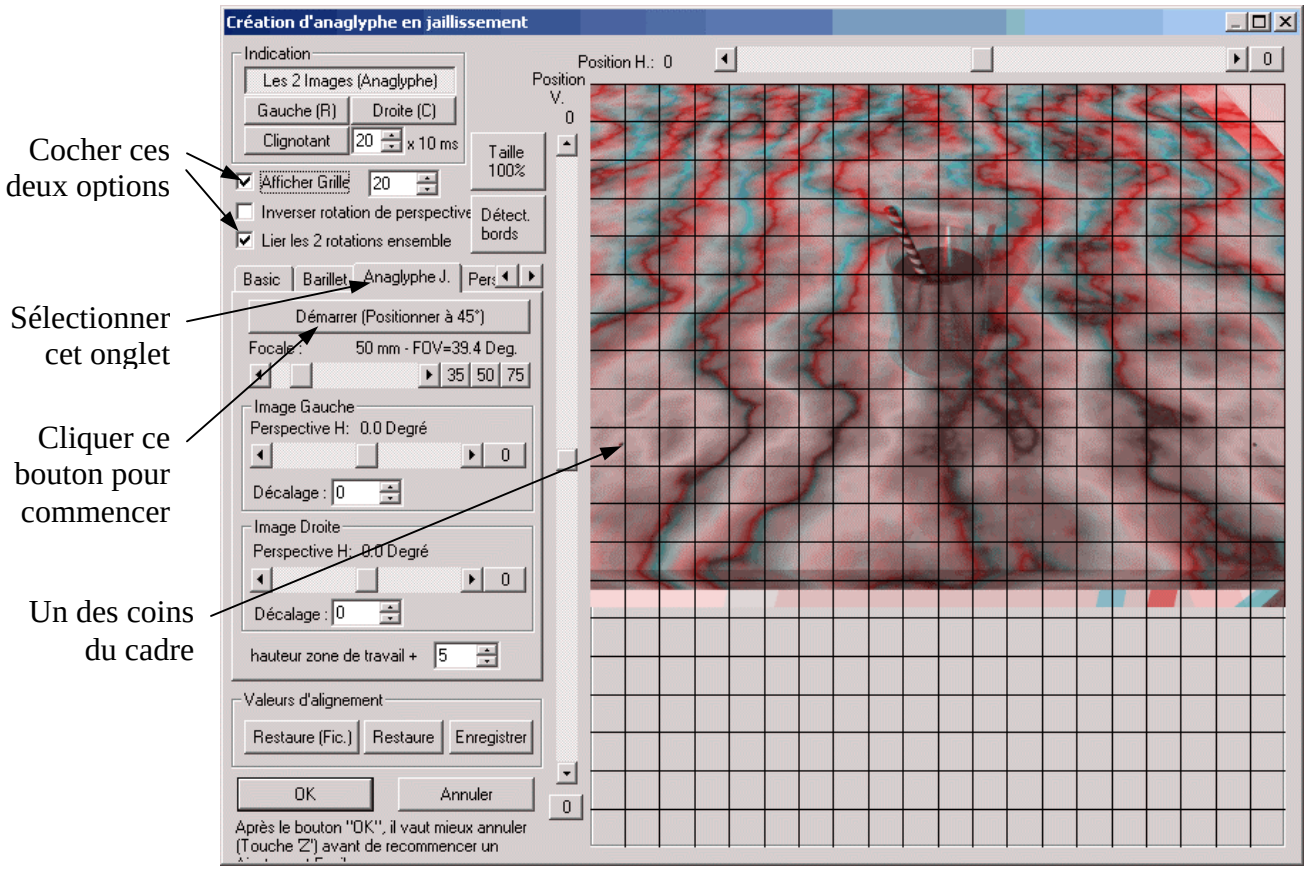


Enregistrer l'image et répéter les mêmes opérations pour la vue droite du couple. Générer ensuite un anaglyphe et l'imprimer sur une feuille de papier.

Utilisation de StereoPhoto Maker

StereoPhoto Maker de Masuji Suto possède un module pour le traitement des fantôgrammes.

Ouvrir l'image comportant les deux vues du couple avec la commande *Ouvrir une image stéréo* du menu *Fichier*. Vérifier que l'image est bien affichée en mode parallèle, si ce n'est pas le cas, utiliser la commande *Échanger Gauche/Droite* du menu *Affichage* (ou presser la touche X) pour décroiser le couple. Ensuite utiliser la commande *Créer un Anaglyphe en jaillissement* du menu *Ajustement*, la fenêtre du module fantôgramme s'ouvre alors :



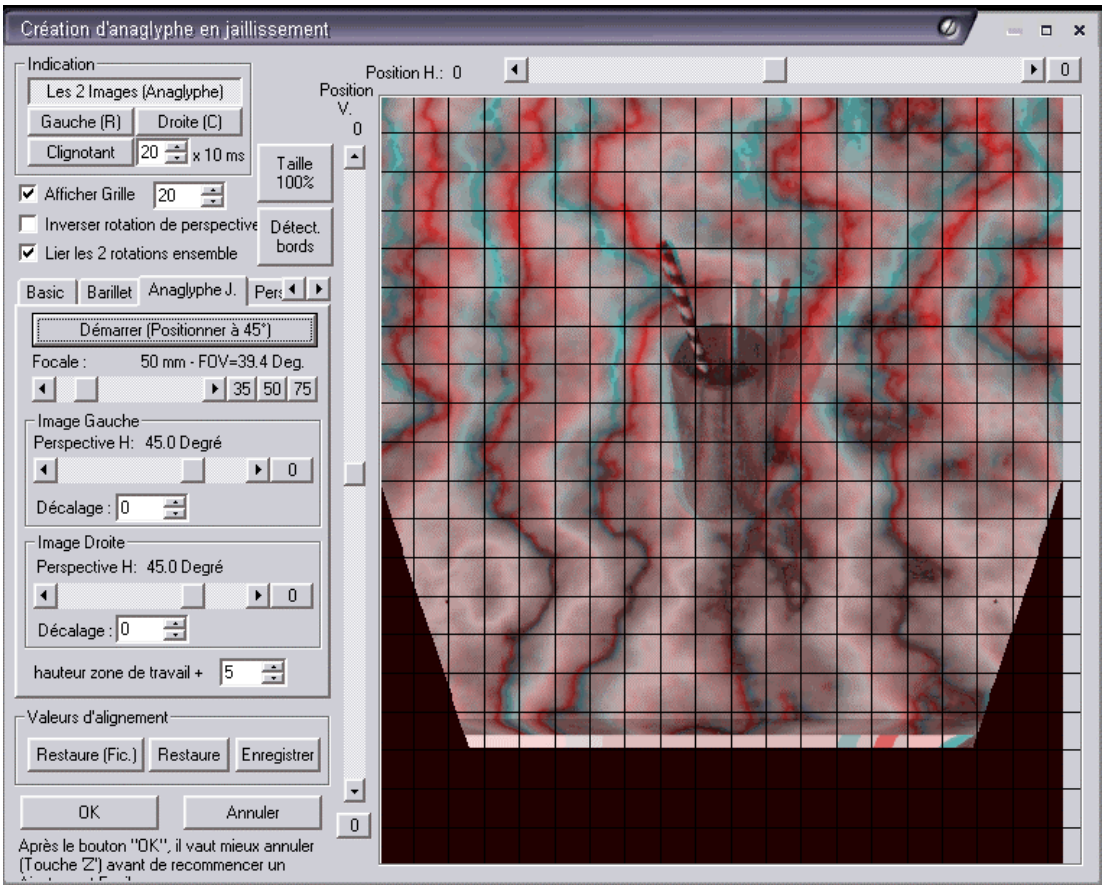
Les deux vues du couple sont affichées en mode anaglyphe N&B donc superposés.
Le but est le suivant : superposer les coins du cadre entre les vues gauche et droite le plus précisément possible et que ce double cadre doit être un carré. On voit aussi que les motifs du marbre de la table ne se superposent pas, quand les coins seront bien alignés et les angles bien redressés, les motifs se superposeront parfaitement (et seront donc exactement dans le plan de la fenêtre stéréoscopique ou de la feuille de papier sur laquelle le fantôgramme sera imprimé).

Pour nous aider, cochons l'option *Afficher Grille* pour afficher un quadrillage sur les images : il nous facilitera les alignements des points du cadre. Augmenter au besoin le pas de la grille pour plus de précision.

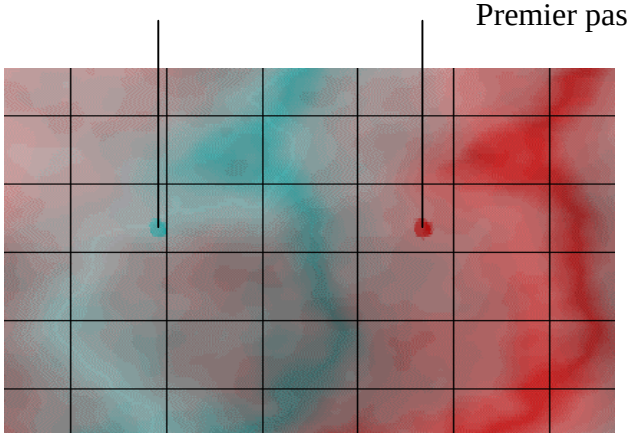
Cocher également l'option *Lier les 2 rotation ensemble* car, si les images ont été bien réalisées (!), les corrections de perspective à appliquer seront les mêmes sur l'image gauche et sur l'image droite.

Sélectionner l'onglet *Anaglyphe J.* car c'est ici que les corrections seront effectuées pour la plupart des images à traiter.

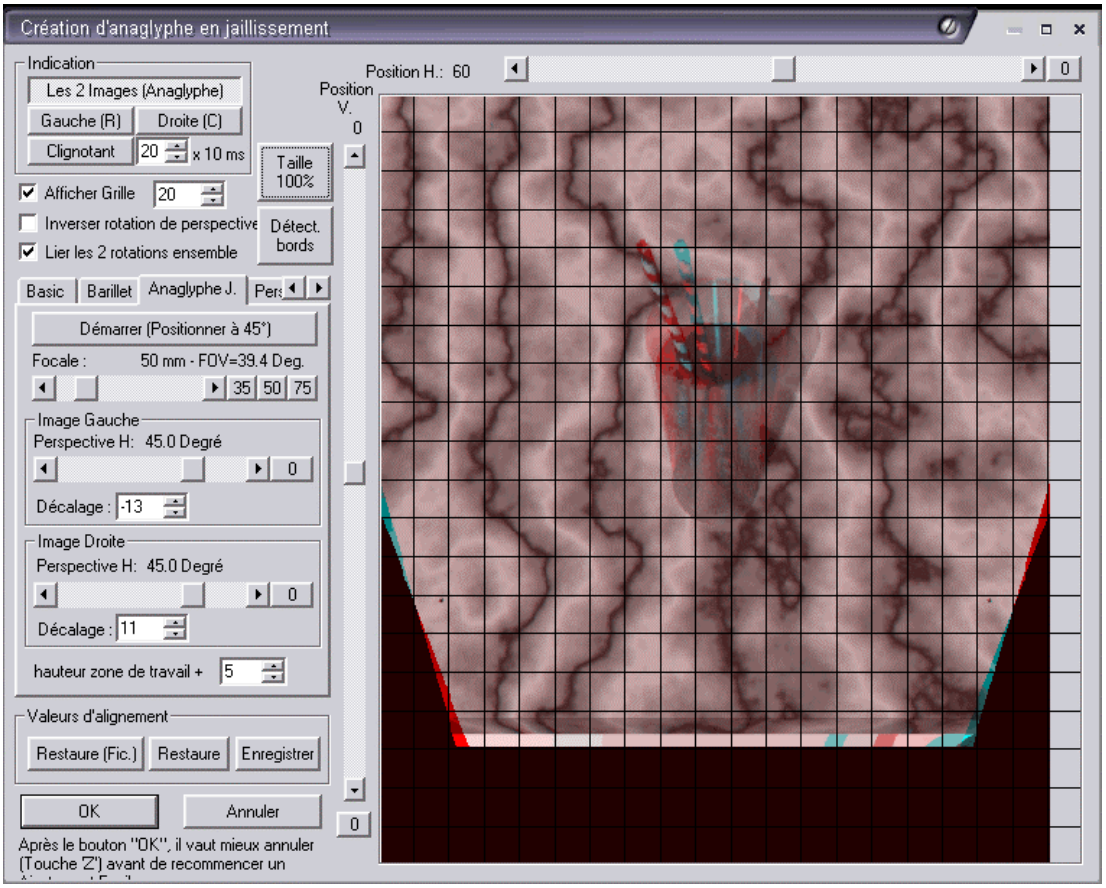
Commencer le travail par cliquer le bouton *Démarrer (Positionner à 45°)* : l'image dans la zone de travail devient ceci :



Au besoin aligner les deux coins inférieurs du cadre avec les curseurs *Position H* et *Position V.* Puis continuer à modifier la perspective (généralement l'augmenter) pour amener les coins supérieurs à la verticale de ceux du bas.
On constate que, si les points inférieurs sont superposés, les coins supérieurs ne le sont pas : les images rouge et cyan d'un point sont séparées (utilise le bouton *Taille 100%* pour travailler plus précisément).

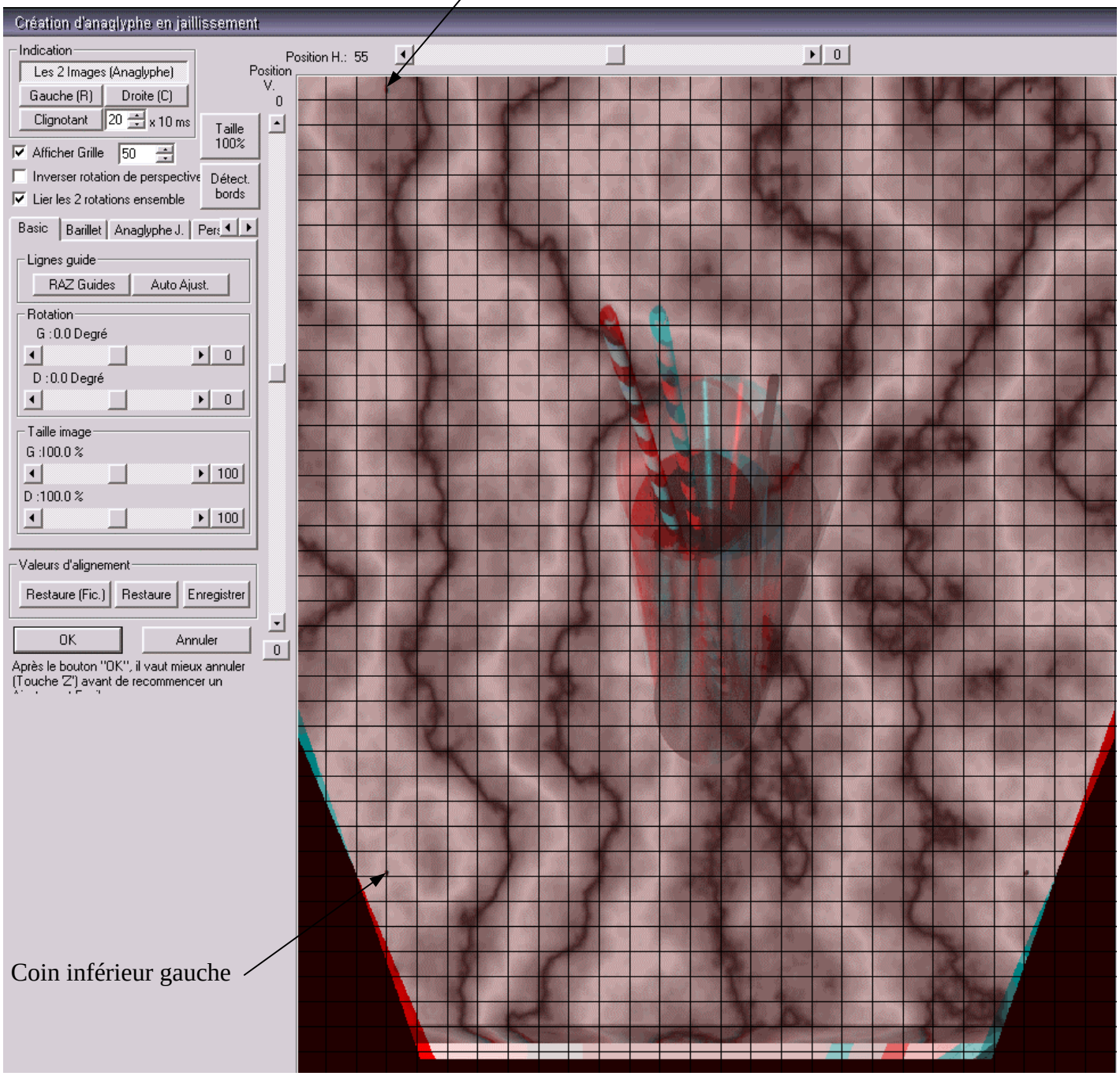


On a corrigé ici seulement la perspective due au fait que l'appareil photo, surélevé, pointait vers le bas. Il faut aussi corriger une différence de perspective horizontale : l'objectif de gauche regarde un peu plus à gauche que celui de droite et a donc une vision différente de son frère jumeau. Pour cela utiliser les deux champs *Décalage* pour amener les coins supérieurs en coïncidence : répartir le décalage global sur les deux images (en général négatif pour l'image gauche et positif pour l'image droite). Attention : vérifier de temps en temps l'alignement de coins du bas qui vont s'écarter un peu dans l'opération : utiliser le curseur *Position H* pour les superposer à nouveau avant de continuer à affiner le décalage. Note : si les coins du haut disparaissent hors de la zone de travail, augmenter la *hauteur* (de) *la zone de travail*. Voici le résultat :

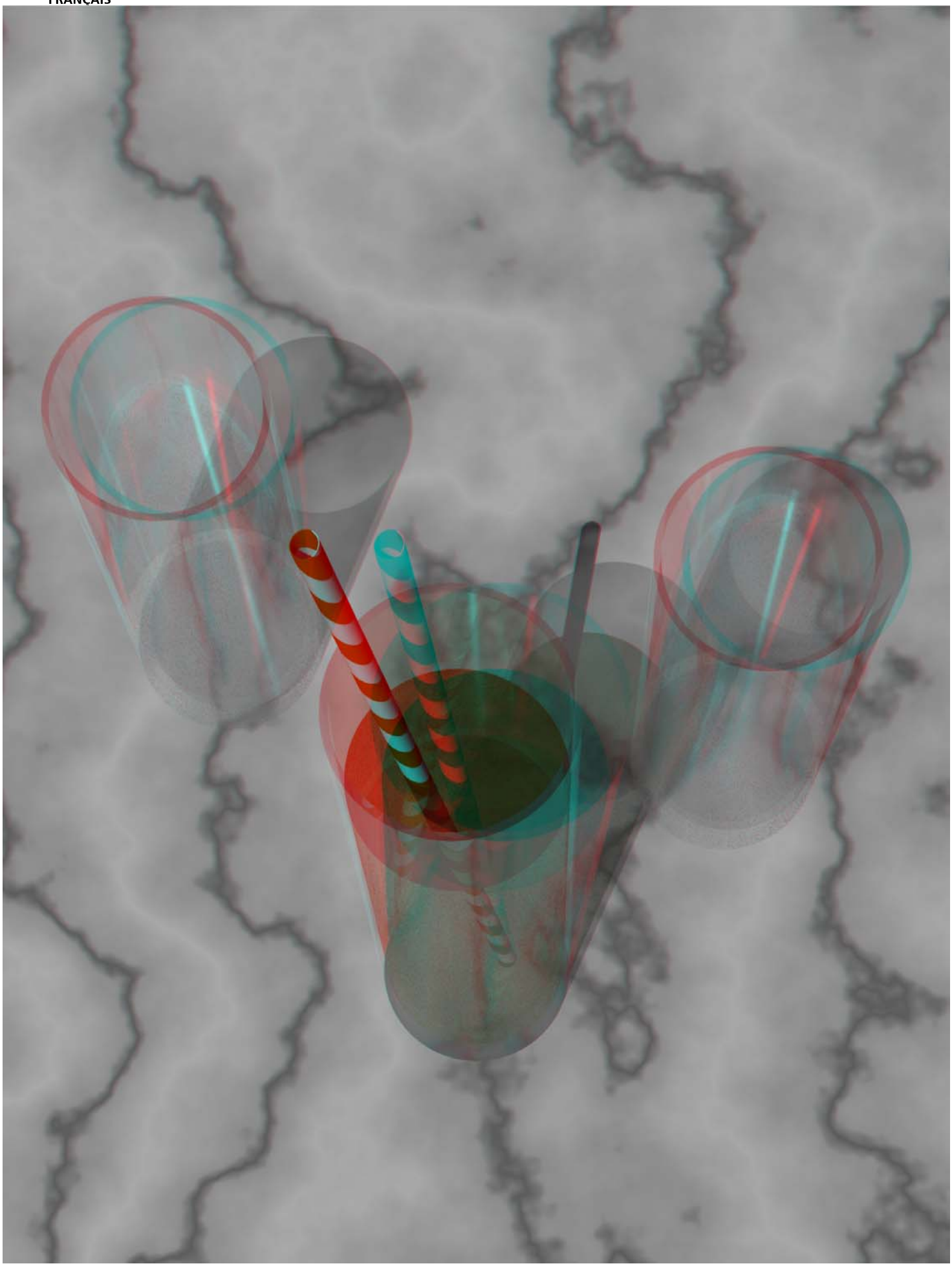


Pour terminer, utiliser la grille pour aligner verticalement les coins hauts et bas du cadre : modifier avec précision la correction de perspective. Pour des images photographiques (c'est à dire pas des images de synthèse) il sera sans doute nécessaire de corriger d'autres paramètre : on fera par exemple un *Ajustement automatique* et jouer sur les *Rotations* de l'onglet *Basic*.

Voici le résultat final :



Cliquer le bouton *OK* pour valider les corrections et retourner à la fenêtre principale de StereoPhotoMaker. Recadrer l'image pour éliminer les coins du cadre et enregistrer l'image stéréo (en mode côte à côte ou en anaglyphe).



Animation

POV-Ray propose des outils simples pour faire des animations. Nous avons vu précédemment comment faire des images fixes, une animation sera simplement une suite d'images fixes présentées au spectateur en succession rapide (en général 25 images par secondes). Pour obtenir un film qui n'endormira pas le spectateur, il faut bien sûr qu'il se passe quelque chose donc que quelque chose change d'une image à l'autre.

En général dans une animation on fera varier :
- la position des objets : des éléments vont se déplacer dans la scène ;
- la forme des objets : pousse d'une plante, déformation sur un choc par exemple ;
- la position de la caméra pour changer de point de vue : travellings, panoramiques, caméra « subjective », effet de zoom ;

Mais on peut aussi :
- faire varier les paramètres stéréoscopiques : adapter progressivement la base si la caméra se rapproche ou s'éloigne d'un objet ;
- déplacer la source lumineuse pour simuler le déplacement du soleil, en changer la teinte ou l'intensité ;
- faire apparaître ou disparaître des objets, changer leur couleur, leurs texture, leur transparence ;
- faire tourner la « sphère du ciel » pour simuler le déplacement des nuages ;

Nous avons vu que tous les éléments d'une scène POV-Ray sont paramétrés : les cordonnées, les tailles, les couleurs,… sont représentées par des nombres. Pour créer une animation, il suffira donc de faire varier une ou plusieurs de ces valeurs en fonction du temps qui passe.
Pour cela, POV-Ray propose une variable prédéfinie appelée *clock* (horloge) qui va varier en fonction du temps. Là encore, POV-Ray n'impose pas d'unité : on pourra faire varier cette horloge par exemple de 0 à 100% ou bien décider qu'elle représente une durée en secondes depuis le début de l'animation.

Une animation simple

Pour commencer nous allons simplement représenter un verre qui se vide en faisant varier le niveau du liquide à l'intérieur. Le début de notre animation sera comme la scène de l'étape précédente puis nous verrons le verre se vider progressivement jusqu'à être complètement vide.

Reprenons la définition de notre verre rempli :

```
#declare verre_rempli =
union {
    object { verre }
    object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
    object { paille rotate z*10 translate <1,1,0> }
}
```

Le liquide est représenté par le deuxième *object* : un simple cylindre emboîté dans le verre et doté d'une texture transparente. Le cylindre est vertical et sa base se situe à 1 unité de hauteur (correspondant à l'épaisseur du fond du verre) et le sommet à 12 unités. Pour simuler la baisse du liquide il suffira de raccourcir le cylindre en amenant, progressivement, le sommet de 12 à 1. Nous utiliserons une horloge en % variant de 0 à 1.

Le verre paramétré pour l'animation devient alors :

```
#declare verre_rempli =
union {
  object { verre }
  object { cylinder {<0,1,0> <0,12-11*clock,0> 2.8 } texture {NBwinebottle} }
  object { paille rotate z*10 translate <1,1,0> }
}
```

Pour la première image de l'animation, *clock* vaut zéro et le sommet du cylindre est donc à 12-11*0 = 12, ce qui correspond à la hauteur initiale de liquide dans le verre. Pour la dernière image de l'animation, *clock* vaut 1, le sommet du cylindre est à 12-11*1 = 1, le verre est donc vide !

Pour générer l'animation, le fichier *.pov* va être joué de manière répétitive pour différentes valeurs d'horloge de manière à générer toutes les images de l'animation. On définit comment va se dérouler la séquence dans un fichier spécifique de contrôle de l'animation. Ce fichier contient au minimum les informations suivantes :

- le nom du fichier *.pov* à utiliser,
- les valeurs initiale et finale de l'horloge ;
- les numéros de la première et de la dernière image de la séquence.

Par exemple, pour notre verre nous aurons (avec aussi des commentaires commençant par un point-virgule) :

```
; Nom du fichier POV à utiliser
Input_File_Name=Etape_14.pov

; Numéro de la première image et numéro de la dernière image
Initial_Frame =  1
Final_Frame   = 25

; Valeurs initiale et finale de l'horloge
Initial_Clock = 0
Final_Clock   = 1
```

L'animation va faire varier l'horloge de 0 à 1 et générer 25 images entre ces deux bornes. On remarque qu'il est possible de ne générer qu'une partie de l'animation en changeant les valeurs : si on a déjà généré les douze premières images, on peut relancer le calcul pour la suite en spécifiant 13 comme image de début et 0.5 comme valeur d'horloge initiale. Dans le même ordre d'idée, pour des animations très lourdes on peut découper le travail entre plusieurs ordinateurs, chacun travaillant sur une petite tranche de temps et générant les images correspondantes.

On enregistre le fichier de contrôle de l'animation avec une extension *.ini* (pour s'y retrouver on lui donne souvent le même nom que le fichier *.pov* correspondant). Le lancement du « rendu » ne se fait plus sur le fichier *.pov* (qui ne génère qu'une seule image) mais sur le fichier *.ini* qui va générer toute la séquence.

Lançons donc le calcul et patientons le temps que toutes les images soient créées (les images générées se nomment *<nom du fichier pov><compteur>.png*). Surprise ! Seulement 24 images sont générées, la dernière correspondant au verre vide est celle qui manque. Consultons la fenêtre de trace, il y a effectivement une erreur :

```
Rendering frame 25 of 25

File: Etape_13-A.pov  Line: 104
    object { verre }
    object { cylinder { <0,1,0> <0,12-11*clock,0> 2.8 } <----ERROR
```

```
Parse Error: Degenerate cylinder, base point = apex point.
Returned from renderer with error status
```

Pour le verre vide nous avons placé dans le verre un cylindre de liquide de hauteur nulle ce qui n'est pas autorisé par POV-Ray. Pour contourner ce problème, nous allons utiliser une instruction conditionnelle pour éviter de générer ce cylindre quand l'horloge vaut 1.

La nouvelle version du verre devient :

```
#declare verre_rempli =
union {
  object { verre }
#if (clock < 1)
  object { cylinder {<0,1,0> <0,12-11*clock,0> 2.8 } texture {NBwinebottle} }
#end
  object { paille rotate z*10 translate <1,1,0> }
}
```

Cette nouvelle version fonctionne mais n'est pas très satisfaisante : on mélange en effet les choses, définition du verre et spécificités de l'animation. Le code en devient moins facile à comprendre. Une meilleure méthode est de définir un verre qui aura un paramètre *hauteurLiquide*. Dans le contexte de la définition du verre, on vérifie que cette hauteur est bien positive (et inférieure à la hauteur du verre pendant qu'on y est !) et on place le cylindre de liquide si c'est le cas. Du côté de l'animation, l'horloge fera varier la valeur du paramètre *hauteurLiquide*. Le code gagne en clarté car on devine tout de suite que l'animation va faire jouer le niveau du liquide dans le verre.

```
#declare hauteurLiquide = 11 - 11*clock;

#declare verre_rempli =
union {
  object { verre }
#if ((hauteurLiquide > 0) * (hauteurLiquide <= 14))
  object { cylinder {<0,1,0> <0, 1+hauteurLiquide,0> 2.8 } texture {NBwinebottle}}
#end
  object { paille rotate z*10 translate <1,1,0> }
}
```

Dans cette animation, l'eau disparaît sans raison apparente, pour rendre la scène plus crédible et amusante, nous allons transférer le liquide d'un verre dans un autre. Nous avons donc besoin de deux objets *verre_rempli* dans notre scène dont les niveaux seront gérés indépendamment. Avec la définition actuelle utilisant la commande *#declare*, ce n'est pas possible car le niveau est spécifié par la variable *hauteurLiquide* et tous les verres que l'on créera en utilisant ce modèle auront la même quantité de liquide. Nous allons donc changer la définition du modèle de verre pour pouvoir paramétrer le niveau de liquide verre par verre.

Pour cela nous utilisons la commande *#macro* à la place de la commande *#declare* et nous indiquons un paramètre *hauteurLiquide* :

```
#macro verre_rempli( hauteurLiquide )
union {
    object { verre }
#if ((hauteurLiquide > 0) * (hauteurLiquide <= 14))
    object { cylinder { <0,1,0> <0,1+hauteurLiquide,0> 2.8 } texture {
NBwinebottle } }
#end
    //object { paille rotate z*10 translate <1,1,0> }
}
#end
```

Nous pouvons maintenant créer nos deux verres et contrôler leurs niveaux respectifs indépendamment (le verre 1 se vide et le verre 2 se remplit) :

```
#declare hauteurLiquide1 = 11 - 11*clock;
#declare hauteurLiquide2 = 11*clock;

object { verre_rempli(hauteurLiquide1)        translate <42,92.5,-35> }
object { verre_rempli(hauteurLiquide2)        translate <20,92.5,-28> }
```

Pour relier les deux verres entre eux nous allons créer une grande paille doublement coudée en réutilisant les éléments déjà créés. Trois pailles droites pour les trois sections et deux coudes créés en prenant un quart de tore. Les cinq éléments sont assemblés dans un nouvel objet *pailleCoudee*.

Vue éclatée

Détail d'un coude

Assemblage final

1) Une nouvelle paille droite sans texture

```
#declare paille2 =
    difference {
        cylinder { <0,0,0>  <0,20,0> 0.30 }
        cylinder { <0,-1,0> <0,21,0> 0.28 }
    }
```

2) Définition d'un coude avec un quart de tore : le tore (évidé par un autre tore légèrement plus petit) est découpé (commande *clipped_by*) par deux plans. Le coude est alors tourné de 90° pour être vertical.

```
#declare paille2Coude =
object {
  difference {
    torus { 1, 0.30 }
    torus { 1, 0.28 }
  }
  clipped_by { plane { x, 0 } }
  clipped_by { plane { z, 0 } }
  rotate x*90
}
```

3) Assemblage de la grande paille doublement coudée : trois pailles de base (2 verticales et une horizontale) reliée par deux coudes (le deuxième est tourné de 180° pour faire face au premier). Les coordonnées de chaque élément doivent être ajustées pour que les éléments se raccordent sans interstice visible. Notez aussi que la texture « sucre d'orge » est appliquée à l'objet entier et non à chacun de ses constituant pour obtenir une texture d'un seul tenant.

```
#declare pailleCoudee =
union {
  object { paille2 }
  object { paille2Coude translate <1, 20, 0> }
```

```
  object { paille2 rotate z*90 translate <21,21,0> }
  object { paille2Coude rotate y*180 translate <21, 20, 0> }
  object { paille2 translate <22, 0, 0> }
  texture { Candy_Cane }
}
```

Il ne reste plus qu'à bien positionner tous les éléments de la scène pour que chacune des deux extrémités de la paille trempe bien dans le verre correspondant.

On peut choisir un mouvement plus lent du liquide en demandant la génération de 72 images qui donneront 3 secondes de vidéo. Lors de l'assemblage de la vidéo (par exemple avec VirtualDUB) on peut ajouter, à la suite, la même série de 72 images mais dans l'ordre inverse. En jouant la vidéo en boucle, on aura le spectacle d'un mouvement perpétuel de transvasement entre les vases communicants !

Une application aux réseaux lenticulaires

La photo stéréoscopique classique nécessite deux images correspondant aux points de vue de chacun des deux yeux. Par contre dans la réalisation d'images utilisant des réseaux lenticulaires, on utilise généralement plus que 2 images : 6, 10 voire 20 ou 30 ! En prise de vue réelle, les difficultés vont évidemment croissant avec le nombre d'images à obtenir. En image de synthèse par contre l'ordinateur travaille pour nous !

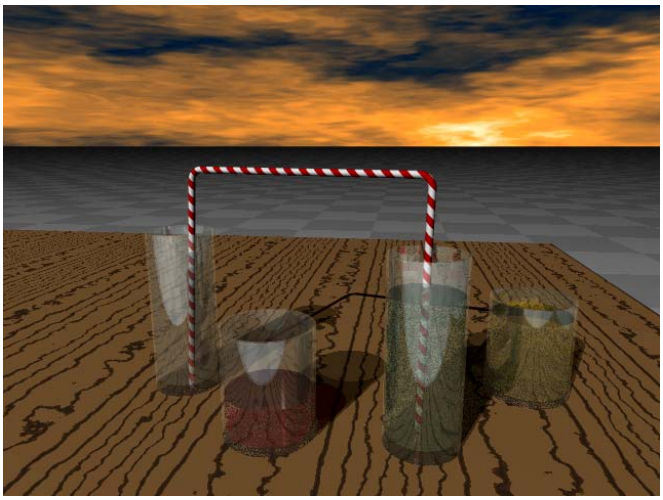
Voyons comment nous pouvons utiliser le concept de l'animation pour automatiser la production d'une série d'image : la variable horloge *clock* ne sera plus utilisée pour déplacer ou modifier des objets dans la scène (on veut au contraire que rien ne bouge entre chaque « cliché ») mais pour déplacer la caméra elle même comme on le fait dans le monde réel lors de la prise de vue stéréo en deux temps. L'ordinateur facilite les choses ici en permettant de calculer un décalage précis entre les images tout en maintenant un cadrage rigoureux, deux éléments nécessaires à l'élaboration de clichés lenticulaires.

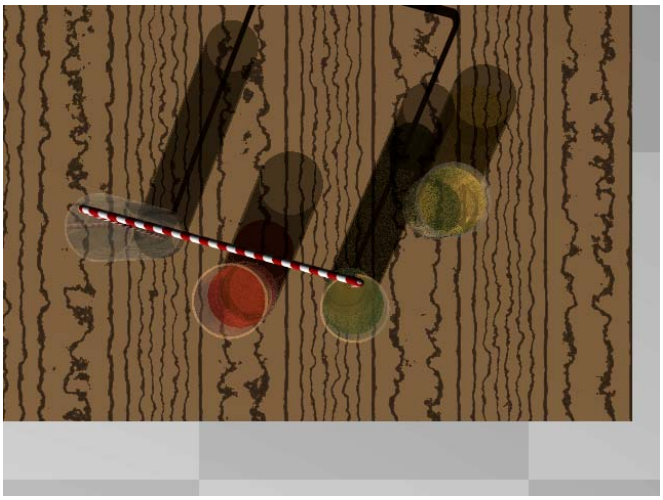
Avant de mettre en place la scène, quelques astuces pour se faciliter le travail avec POV-Ray :

1) <u>Voir la scène sous un nouvel angle.</u>
   POV-Ray, nous l'avons vu, n'est un outil graphique qu'en bout de chaîne, la définition des scènes se fait de manière textuelle ce qui peut rebuter certains débutants. Il existe quantité d'outils autour de POV-Ray : des modeleurs pour faciliter la construction de certaines pièces ou des interfaces graphiques aidant à la construction des scènes (le rendu final étant toujours effectué dans POV-Ray). On peut aussi simplement, déplacer la caméra pour mieux contrôler la disposition des éléments constitutifs de la scène. Nous n'avons que quelques verres sur la table mais – même avec le relief ! – il peut être un peu délicat de bien les positionner : pour par exemple s'assurer que la paille doublement coudée trempe bien dans les deux verres. Une approche « mathématique » serait de calculer rigoureusement la position de chaque objet, le résultat sera forcément conforme !
   On se contente souvent de composer sa scène par ajustements successifs de la position des objets. La vue « par dessus » facilite souvent le travail. Dans la définition de la caméra, on peut définir plusieurs couples position/point visé et n'en avoir un – au choix – qui ne soit pas en commentaire :

```
camera {
    //location < 40,  115, -70>     look_at  < 30,  100,   0>
    location < 40,  150,   -30>   look_at  < 40,  100,   -30>
    ...
```

Ici, la position d'origine de la caméra est simplement mise en commentaire // devant la ligne pour la conserver et une caméra verticale est créée sur la deuxième ligne (les coordonnées de *location* et de *look_at* ne diffèrent que par la valeur en *y*).


Vue normale

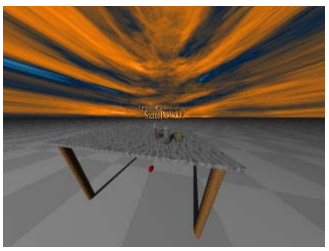

Vue de dessus

2) Changer de focale

Il est possible de faire un gros plan ou un plan plus large d'un objet sans pour autant changer la caméra ou les objets de place. Le mot clé *angle* placé dans la définition de la caméra permet de contrôle l'angle de champ horizontal de la caméra, par exemple :

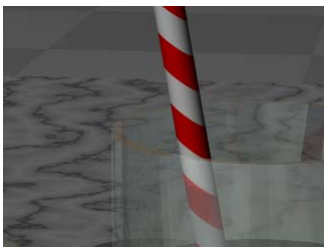
```
camera {
    ...
    angle 45
    ...
```

On indique simplement la valeur en degré de cet angle : petite valeur pour un effet télé-objectif et grande valeur pour un grand-angle.


angle = 150°

angle = 90°

angle = 45°


angle = 10°

Note : Dans le cas d'images stéréoscopiques, il sera probablement nécessaire de corriger la base et la position de la fenêtre stéréoscopique.

3) Transgresser les lois de l'optique.

POV-Ray permet de définir des objets qui vont défier les lois de la physique : on peut par exemple définir des sources lumineuses qui ne projettent pas d'ombre ! Cela peut être utile pour déboucher une zone d'ombre comme par un coup de flash sans pour autant que les ombres projetées par les objets deviennent multiples (paramètre *shadowless* dans la définition d'une source lumineuse).

Les objets de leur côté peuvent, tels des vampires, ne pas se refléter dans les surfaces réfléchissantes (paramètre *no_reflection*) ou bien ne pas projeter d'ombres (paramètre *no*_shadow) : les ombres générées par un titre par exemple ne sont pas forcément souhaitables. Autre possibilité avec le paramètre *no_image*, l'objet ne sera pas visible mais pourra projeter une ombre ! J'ai utilisé cette possibilité dans une scène où la photo d'un personnage réel

devait être intégré dans l'image finale : l'incrustation manquait de crédibilité car le personnage ne projetait pas d'ombre. Un mannequin modélisé grossièrement avec quelques cylindres prend la place du personnage, avec le paramètre *no_image*, il n'est pas visible mais projette néanmoins une ombre qui semblera être celle du personnage incrusté dans la scène.

4) Gestion du temps

Quand on lance directement le rendu sur un fichier *.pov* destiné à faire une animation, la variable horloge *clock* a toujours la valeur 0. Il peut être souhaitable, dans la phase de mise au point de la scène, d'afficher des images à un point donné de l'animation. Générer toutes les images jusqu'à ce point, serait une perte de temps considérable. Le plus pratique est de définir une variable intermédiaire, par exemple nommée *clk* et qui sera utilisée dans la scène à la place de la variable standard *clock*.

Cette variable est définie comme suit :

```
#declare clk = 0.0+clock;
```

Pour l'animation *clk* vaudra *clock* et tout se déroulera normalement. Pour calculer une image aux deux tiers de l'animation par exemple, on remplacera le *0.0* par *0.66*. Il ne faudra bien sûr pas oublier de remettre la définition avec *0.0* avant de lancer le calcul complet de l'animation.

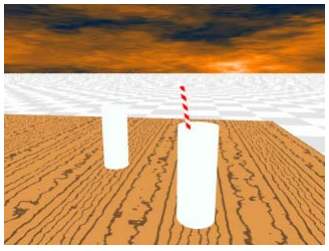
Une autre possibilité intéressante est d'afficher des informations dans la fenêtre de trace. La fenêtre de trace est accessible par l'onglet *Messages* ou bien en même temps que la fenêtre d'édition avec la commande *Show Message Window* (ou presser les touches *Ctrl+M*). Cette fenêtre affiche de nombreuses informations sur le calcul de l'image mais il est aussi possible d'y afficher ses propres messages avec la commande *#debug*. Lors de la mise au point d'une scène complexe avec de nombreux objets et des mouvements compliqués, on peut alors afficher les valeurs de paramètres clés. Ici, nous nous contenterons d'afficher la valeur de la variable *clock* (ou plutôt de la variable intermédiare *clk*) :

```
#debug concat("Horloge = ", str(clk,5,5), "\n")          // Affichage dans la
fenêtre des messages de la valeur courante de l'horloge
```

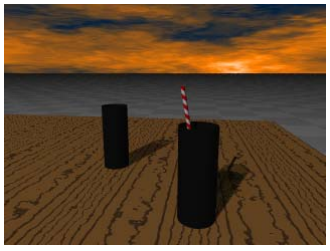
5) Accélération de l'affichage

Plus la scène devient complexe, plus le calcul de l'image devient long même avec un ordinateur rapide. La mise au point de la scène peut alors devenir laborieuse. Pour accélérer les choses, on peut faire plusieurs choses.

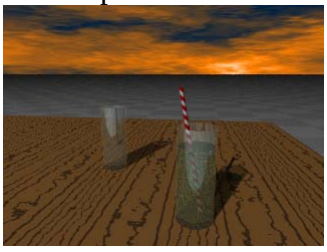
- Ne pas hésiter à réduire la taille des images générées : la liste déroulante des résolutions propose de nombreux choix. Une imagette de 160x120 pixel est souvent suffisante pour centrer un texte dans l'image par exemple.
- Sélectionner une résolution sans anti-crénelage (marquées *No AA*). L'anti-crénelage améliore beaucoup la qualité des images mais est très gourmant en temps de calcul.
- Forcer une qualité d'image dégradée. POV-Ray propose une échelle de qualité de 0 à 11 (la qualité par défaut est 9). En diminuant la qualité, les ombres, les transparences disparaissent, les couleurs se simplifient… Taper la qualité désirée dans la zone de saisie située à la droite de la liste déroulante des résolutions. Par exemple :



+q0
13 s



+q5
17 s



+q11
40 s

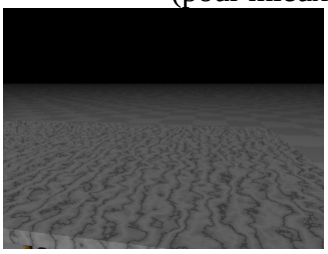
- Désactiver la stéréoscopie. Je sais c'est un peu sacrilège mais l'image se construira au moins deux fois plus vite ! Fermer la fenêtre qui affiche l'image entrain de se construire accélère également le rendu. S'il faut 485 secondes pour générer l'étape 14 en stéréo avec affichage de l'image simultané, il n'en faut plus que 78 pour générer la même image sans affichage et 39 s en monoscopie toujours sans affichage.
- Une fois une partie de la scène au point, il n'est plus nécessaire de passer du temps à l'afficher lors du réglage d'une autre partie. J'utilise des sortes de variables de contrôle qui vont décider si une partie doit être tracée ou non. Les définitions de ces variables sont regroupées au début du fichier. Découpons notre scène en plusieurs blocs cohérents : la table, les verres, le ciel et le titre. Définissons des variables de contrôle pour chacun des blocs :

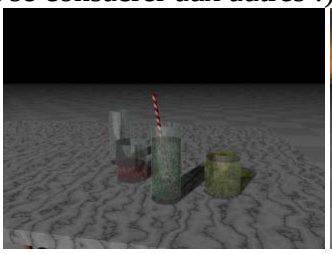
```
#declare TRACE_TABLE    = 1;
#declare TRACE_VERRES   = 1;
#declare TRACE_TITRE    = 1;
#declare TRACE_CIEL     = 1;
```

Maintenant encadrons les blocs avec des instructions conditionnelles, par exemple pour le ciel :

```
#if (TRACE_CIEL)
sky_sphere {
    pigment {
        wrinkles
[...]
}
#end
```

En mettant certaines variables à 0 on peut supprimer l'affichage des objets correspondant (pour mieux se consacrer aux autres !).


Seulement la table
12 s

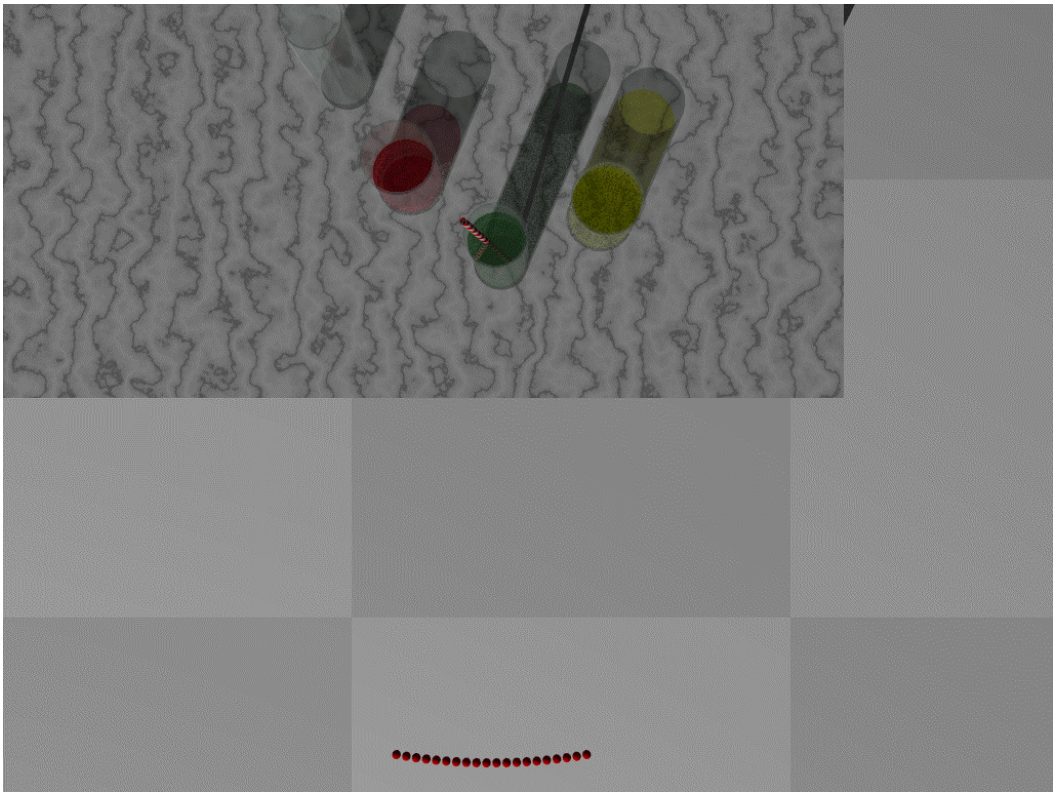

Les verres en plus
56 s

Titre et ciel
26 s

La scène compète
73 s

Revenons maintenant à notre image lenticulaire. Notre caméra (monoscopique cette fois) va se déplacer en arc de cercle en restant pointé sur notre sujet (les verres sur la table). On peut bien sûr expérimenter des déplacements purement rectilignes, avec ou sans convergence. POV-Ray propose toutes les fonctions mathématiques classiques pour faciliter les calculs.

Dans la vue de dessus suivante, les positions successives de la caméra ont été représentées par des points rouges : 20 positions réparties sur un angle de 20°.



Voici comment nous allons définir notre caméra mobile.

1) Définissons des variables pour les positions initiales issue de la scène de l'étape précédente :

```
#declare camPos_org = < 40,  115,  -80>;  // Paramètres d'origine de la caméra
#declare camLA_org  = < 30,  100,    0>;  // Point visé par la caméra
#declare verrePos   = < 37,   92.5,-35>;  // Position du verre avec la paille
```

2) La fonction *vlength* permet de calculer la distance entre deux points. Cela nous permet de calculer la distance entre la caméra et le centre de la scène, ici le verre avec la paille. Ce point restera fixe, la caméra tournera autour de lui tout en restant centré sur le point visé initial. L'utilisation d'une fonction de calcul permettra, si besoin, de changer ces paramètres et tout se recalculera automatiquement. La commande *#debug* permet d'afficher la valeur dans la fenêtre des messages.

```
#declare distCam = vlength(camPos_org-verrePos);   // Calcul de la distance caméra -
sujet
#debug concat("Distance nominale au sujet = ", str(distCam,7,2), "\n")
```

3) Comme la caméra va effectuer un arc de cercle « à plat », c'est à dire avec un axe de rotation vertical, nous calculerons plutôt une distance projetée sur la surface de la table.

```
#declare distCam = vlength(<camPos_org.x,0,camPos_org.z>-<verrePos.x,0,verrePos.z>);
#debug concat("Distance au sujet projetee sur la table = ", str(distCam,7,2), "\n")
```

4) La caméra va décrire un arc de cercle autour du sujet (tout en restant à la même distance). La position initiale correspond au milieu de l'arc de cercle. Nous définissons une variable *camRotation* qui spécifie la taille en degré de l'arc du cercle.

```
#declare camRotation = 20;          // La caméra va tourner de cet angle durant
l'animation
```

5) Calcul de l'angle de la caméra, sa position sur l'arc de cercle en fait, en fonction de l'horloge d'animation. Les angles sont ici en degrés.

```
#declare camAngle = 270 + clk*camRotation - camRotation/2;
#debug concat("Angle la camera = ", str(camAngle,4,2),  "\n")
```

6) Un peu de trigonométrie pour calculer la position en *x*, *y* et *z* de la caméra. Les fonctions sinus et cosinus utilisent des angles en radians mais POV-Ray fournit la constante prédéfinie *pi* pour faciliter la conversion. Notez aussi qu'on peut simplement additionner des coordonnées. Pour le point visé, il reste fixe, nous recopions simplement la valeur définie au début.

```
#declare camPos = verrePos + < distCam * cos(camAngle / 180 * pi), camPos_org.y-
verrePos.y, distCam * sin(camAngle / 180 * pi) >;
#declare camLA  = camLA_org;
#debug concat("Position de la camera = <", str(camPos.x,4,2), ", ", str(camPos.y,4,2),
", ", str(camPos.z,4,2), ">\n")
```

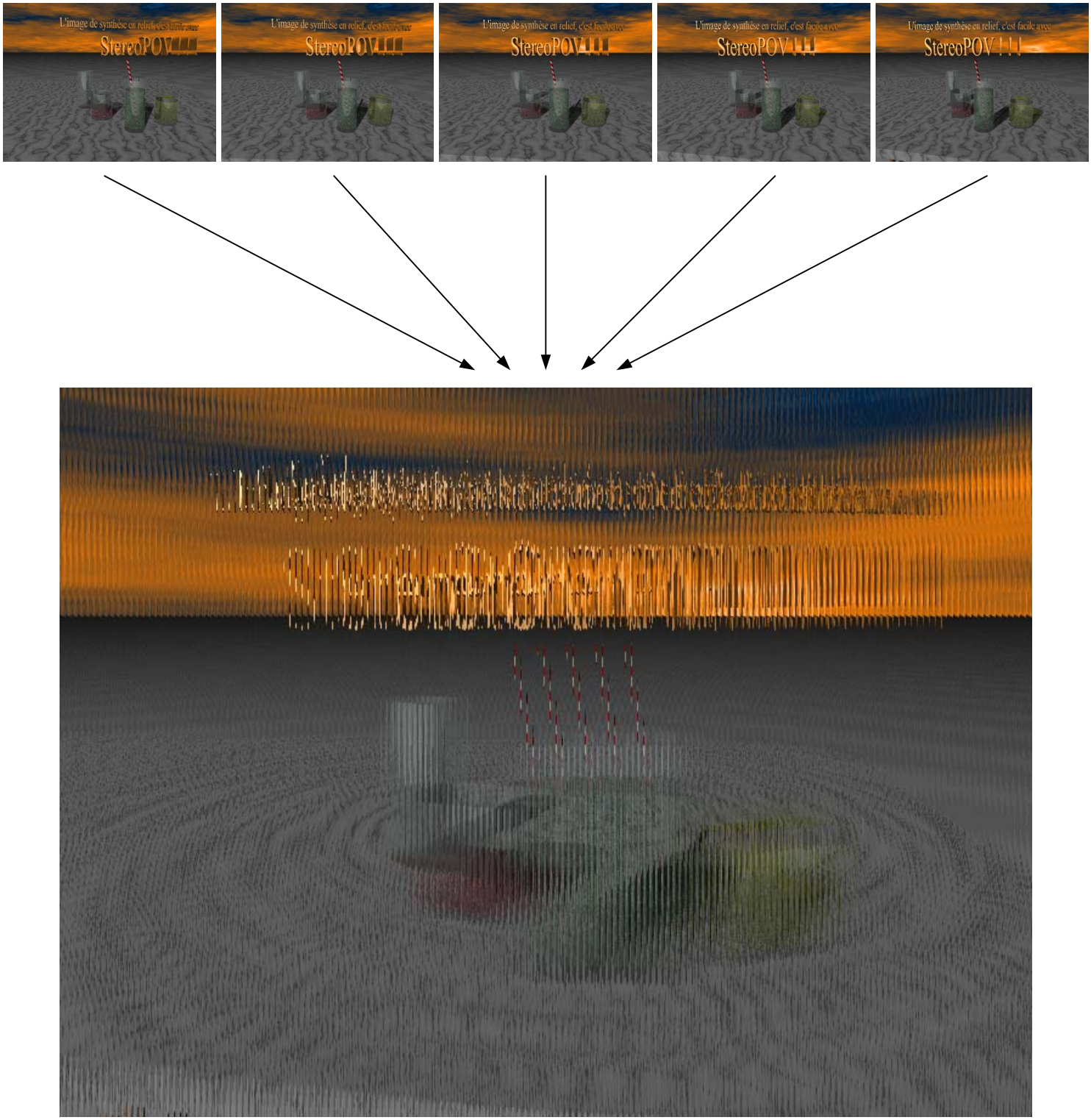
7) Ouf ! Nous pouvons maintenant définir la caméra elle-même en utilisant la position qui vient d'être calculée.

```
camera {
    location camPos
    look_at  camLA
}
```

Voici la trace générée pour une valeur d'horloge = 0.5 :

```
Horloge = 0.50000
Distance nominale au sujet =   50.40
Distance nominale au sujet projetee sur la table =   45.10
Angle la camera = 280.00
Position de la camera = <44.83, 115.00, -79.41>
```

Voici un exemple d'image entrelacée générée avec 5 images.



La taille de l'image doit être calculée précisément en fonction de la taille et du pas du réseau qui sera utilisé. La même précision doit également être appliquée au moment de l'impression.

Conclusion

Seule une infime partie des fonctionnalités de POV-Ray a été dévoilée ici. Les possibilités sont illimitées ! De nombreuses personnes fournissent leurs fichiers *.pov* avec leurs images et ce sont souvent des mines d'informations et de trucs que l'on peut réutiliser dans ses propres créations.

Liens Internet utiles

- Site officiel POV-Ray
  http://www.povray.org
- Site de StereoPOV
  http://stereopov.ichthyostega.de
- De bons didacticiels sur POV-Ray en français
  http://povray.free.fr
  http://www.f-lohmueller.de/pov_tut/pov__fra.htm
- IRTC : le concours mensuel d'images de synthèses (beaucoup d'images sont présentées avec les fichiers *.pov* directement utilisables)
  http://www.irtc.org
- Oyonale, le site de Gilles Tran et son étrange *Livre des Commencements*, j'adore !
  http://www.oyonale.com/oy_fr.htm
- POV-Ray dans l'espace !
  http://www.oyonale.com/iss/francais/makingof.htm
- Fichier de Joerg Schrammel (calcul des positions G & D de la caméra)
  http://www.schrammel.org/stereo-pov.php
- L'attache stéréo virtuelle de Glenn McCarter
  http://www.geocities.com/SoHo/Gallery/2006/include.htm

Concernant les fantôgrammes :

- Le site de Steve Hugues sur les fantôgrammes avec un tutoriel vraiment très complet :
  http://www.shughes.org/phantograms
  La propriété (et l'antériorité) intellectuelle concernant les fantôgrammes fait débat, voir par exemple la description des brevets :
  http://www.shughes.org/phantograms/patents.htm
- Comment créer des fantôgrammes par Shab Levy
  http://www.gravitram.com/Images%205/HOW%20TO%20CREATE%20PHANTOGRAMS%20for%20screen.pdf
- Terry Wilson réalise de jolis fantôgrammes visibles sur son site Web :
  http://terryfic3d.com/Phantograms_sku13.html
- Barry Rothstein s'est rendu célèbre avec son livre *Phantograms from Nature, Western USA* contenant de spectaculaires fantôgrammes
  http://www.3ddigitalphoto.com/

- Une rétrospective sur les fantôgrammes dans StereoViews (Sept 2004)
  http://www.cascade3d.org/newsletters/CSC200409.pdf
- Owen Western, l'« inventeur » des fantôgrammes fait désormais des « Phantaglyph » !
  http://www.3d-onthelevel.com/

Fichiers d'exemple

Les fichiers d'exemple suivant le déroulement de la présentation sont disponibles dans la suite du document.

Pour chaque fichier :

1) Copier le texte affiché dans le cadre,
2) Dans StereoPOV, créer un document vide (cliquer le bouton New, en haut à gauche),
3) Coller le texte dans le nouveau document,
4) Enregistrer le nouveau document en utilisant le nom de fichier correspondant (*Etape_01.pov, Etape_02.pov,…*).

- Fichier Etape_01.pov
  C'est un fichier vide, la scène générée est bien sûre toute noire !

```
```

- Fichier Etape_02.pov
  Une caméra et une source lumineuse → toujours une image noire…

```
camera {
   location <150, 180, -220>
   look_at  <0,    90,    0>
}

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}
```

- Fichier Etape_03.pov
  Ajout d'un dallage faisant office de sol.

```
camera {
   location <150, 180, -220>
   look_at  <0,    90,    0>
}

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}


plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
```

- Fichier Etape_04.pov
  Ajout de commentaires dans le fichier et d'une bille rouge pour figurer la coordonnée origine 0,0,0.

```
// ------------------------------------------------------------------------------------------
//                          Stéréo-Club Français - Séance Technique du 21 mars 2007
//                           Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 4 : Première image visible ! Caméra bien positionnée + source lumineuse + objet --> scène
//       Note : le centimètre est choisi ici comme unité de mesure.
// ------------------------------------------------------------------------------------------

// ------------------------------ Déclaration de la caméra et du point de vue ---------------------------

camera {
```

```
   location <150, 180, -220>
   look_at  <0,    90,    0>
}

// ------------------------------- Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}

// ------------------------------------- Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }
```

- Fichier Etape_05.pov
  Menuiserie : construction de la table en bois.

```
// ------------------------------------------------------------------------------------------------------
//                          Stéréo-Club Français - Séance Technique du 21 mars 2007
//                           Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 5 : La table
//      - inclure les textures standard de POV-Ray
//      - définir le plateau de la table (géométrie + texture)
//      - définir un modèle de pied (géométrie + texture)
//      - assembler la table : 1 plateau + 4 pieds --> modèle de table
//      - placer une instance du modèle de table dans la scène
// ------------------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray

// ------------------------------- Déclaration de la caméra et du point de vue ---------------------------

camera {
   location <150, 180, -220>
   look_at  <0,    90,    0>
}

// ------------------------------- Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00> }

// ------------------------------------- Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }

// ---------------------------------------- La Table ----------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }
```

```
object { table }
```

- Fichier Etape_06.pov
  La même table mais en relief !

```
// ----------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 6 : La table en relief
//      - déclarer les paramètres de la vision stéréoscopie : base et position de la fenêtre stéréo
//      - définir la "version" cible du logiciel pour pouvoir utiliser les fonctionnalités spécifique
//        de la version stéréoscopique de POV-Ray (StereoPOV).
//      - inclure les paramètres stéréo dans la déclaration de la caméra
//      - choisir de générer une image en vision croisée (mettre "cross_eyed") ou parallèle (par défaut)
// ----------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 7;      // Base stéréoscopique
#declare STEREOWINDOW = 200;    // Position de la fenêtre stéréoscopique (distance depuis la caméra)


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel

// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   location <150, 180, -220>
   look_at  <0,    90,    0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}

// ------------------------------- Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00> }

// ---------------------------------------- Le sol et l'origine -------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ------------------------------------------ La Table ----------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
```

```
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }
```

- Fichier Etape_07.pov
  Ajouter un ciel de soleil couchant pour égayer la scène.

```
// ----------------------------------------------------------------------------------------------
//                          Stéréo-Club Français - Séance Technique du 21 mars 2007
//                           Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 7 : Ajouter un ciel
//      - inclure les couleurs standard de POV-Ray
//      - ajouter un objet complexe prédéfini "sphère céleste"
// ----------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"           // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 7;      // Base stéréoscopique
#declare STEREOWINDOW = 200;    // Position de la fenêtre stéréoscopique (distance depuis la caméra)


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel

// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   location <150, 180, -220>
   look_at  <0,    90,    0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}

// ------------------------------- Déclaration de la (ou des) source lumineuse ----------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00> }

// ------------------------------------- Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ----------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
```

```
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }

// ---------------------------------------- Le ciel ----------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}
```

- Fichier Etape_08.pov
  Poser des verres sur la table.

```
// ------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 8 : Poser des verres sur la table
//      - définir un verre (utiliser une texture prédéfinie transparente)
//      - définir une paille (avec une autre texture prédéfinie)
//      - construire un verre rempli
//      - ne pas oublier de poser des instances de ces objets sur la table
// ------------------------------------------------------------------------------------------------

#include "textures.inc"        // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"          // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO      = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 7;      // Base stéréoscopique
#declare STEREOWINDOW = 200;    // Position de la fenêtre stéréoscopique (distance depuis la caméra)


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel

// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   location <150, 180, -220>
   look_at  <0,    90,    0>
```

```
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}

// ------------------------------ Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00> }

// ------------------------------------- Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }

// ---------------------------------------- Le ciel -------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, la paille-----------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =
```

```
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }

#declare verre_rempli =
union {
   object { verre }                                          // Le verre rempli est construit à partir du verre de
base
   object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }       // Un autre cylindre pour
représenter le liquide contenu dans le verre
   object { paille rotate z*10 translate <1,1,0> }                  // Et on ajoute une paille pour faire joli !
}

object { verre       translate <20,92.5,-15> }        // Placer les objets sur la table
object { verre_rempli translate <40,92.5,-35> }
```

- Fichier Etape_09.pov
  Vue rapprochée sur les verres.

```
// ----------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 9 : Vue rapprochée sur les verres
//      - raprocher la caméra du sujet (changer aussi le point de vue)
//      - diminuer la base en conséquence et rapprocher aussi la fenêtre stéréo
// ----------------------------------------------------------------------------------------------------

#include "textures.inc"        // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"          // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel

// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   location < 40,  115, -70>
   look_at  < 30,  100,   0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}

// ------------------------------- Déclaration de la (ou des) source lumineuse ----------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00> }

// ---------------------------------- Le sol et l'origine -------------------------------------------
```

```
plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ------------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }

// ---------------------------------------- Le ciel -------------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, la paille----------------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }

#declare verre_rempli =
union {
   object { verre }                                          // Le verre rempli est construit à partir du verre de
base
   object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }       // Un autre cylindre pour
représenter le liquide contenu dans le verre
```

```
   object { paille rotate z*10 translate <1,1,0> }                    // Et on ajoute une paille pour faire joli !
}

object { verre        translate <20,92.5,-15> }        // Placer les objets sur la table
object { verre_rempli translate <40,92.5,-35> }
```

- Fichier Etape_10.pov
  Ajout d'un titre.

```
// ----------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 10 : Ajout d'un titre
//      - ajouter un objet "text" pour chaque ligne de texte
//      - donner police, couleur, texture aux objets texte
//      - dimensionner, orienter, positionner les objets texte à la bonne place
// ----------------------------------------------------------------------------------------------------

#include "textures.inc"        // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"          // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel

// ------------------------------ Déclaration de la caméra et du point de vue ----------------------------

camera {
   location < 40,  115, -70>
   look_at  < 30,  100,   0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}

// ------------------------------- Déclaration de la (ou des) source lumineuse ----------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}

// ------------------------------------- Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
```

```
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }

// ---------------------------------------- Le ciel ----------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, la paille-------------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }

#declare verre_rempli =
union {
   object { verre }
   object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
   object { paille rotate z*10 translate <1,1,0> }
}

object { verre        translate <20,92.5,-15> }
object { verre_rempli translate <40,92.5,-35> }


// ---------------------------------- Le titre ----------------------------------------------------

#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;

text {
   ttf "times.ttf" "L'image de synthese en relief, c'est facile avec" 0.3, 0
```

```
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
}

text {
   ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <3, 5, 10>
   rotate <0, -5, 0>
   translate <28, 114.5, -50>
}
```

- Fichier Etape_11.pov
  Corrections à apporter pour afficher des caractères accentués.

```
// ----------------------------------------------------------------------------------------------
//                          Stéréo-Club Français - Séance Technique du 21 mars 2007
//                           Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 11  : Ajout d'un titre avec des caractères accentués
//      - ajouter la directive pour spécifier le jeux de caractère à utiliser
//      - coder les caractères accentués à utiliser
// ----------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"           // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel
global_settings { charset utf8 }        // Nécessaire pour pouvoir utiliser les caractères accentués


// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   location < 40,  115, -70>
   look_at  < 30,  100,   0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}


// ------------------------------- Déclaration de la (ou des) source lumineuse ----------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}
```

```
// ------------------------------------ Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ---------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }


// ---------------------------------------- Le ciel ----------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, la paille----------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }

#declare verre_rempli =
union {
   object { verre }
```

```
   object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
   object { paille rotate z*10 translate <1,1,0> }
}

object { verre        translate <20,92.5,-15> }
object { verre_rempli translate <40,92.5,-35> }


// ------------------------------------ Le titre -----------------------------------------------------------

#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;

text {
   ttf "times.ttf" "L'image de synth\u00E8se en relief, c'est facile avec" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
}

text {
   ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <3, 5, 10>
   rotate <0, -5, 0>
   translate <28, 114.5, -50>
}
```

- Fichier Etape_12.pov
  Titrage de photos stéréo existantes.

```
// ------------------------------------------------------------------------------------------------------
//                          Stéréo-Club Français - Séance Technique du 21 mars 2007
//                           Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 12  : Titrage de photos existantes
//      - à partir de l'étape 11, on supprime tous les objets sauf les lignes de titre
//      - ajouter la directive Output_Alpha=on dans le fichier Povray.ini pour obtenir un fond transparent
// ------------------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"           // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 1;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel
global_settings { charset utf8 }        // Nécessaire pour pouvoir utiliser les caractères accentués


// ------------------------------- Déclaration de la caméra et du point de vue ---------------------------

camera {
```

```
   location < 40,  115, -70>
   look_at  < 30,  100,   0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}


// ------------------------------- Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}


// ------------------------------------ Le titre ------------------------------------------------------

#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;

text {
   ttf "times.ttf" "L'image de synth\u00E8se en relief, c'est facile avec" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
}

text {
   ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <3, 5, 10>
   rotate <0, -5, 0>
   translate <28, 114.5, -50>
}
```

- Fichier Etape_13.pov
  Réalisation d'un fantôgramme.

```
// ------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 13 : Génération d'un fantôgramme
//      - placer la caméra précisément à 45 par rapport à la table et changer le point de vue pour que
//        la vue soit bien centrée sur le verre avec la paille. On définit une constante pour la position
//        du verre et la position de la caméra est ensuite déduite de cette position.
//      - ajout d'une deuxième lumière "sans ombre" pour adoucir la scène et déboucher les contre-jour
//        sans pour autant avoir une deuxième ombre.
//      - la texture bois utilisée est trop sombre et complexe pour le fantôgramme. On la remplace par une
//        texture "marbre" mais celle-ci est trop contrastée pour réaliser un bon anaglyphe. On récupère
//        la texture du fichier standard pour la modifier (éclaircir ses couleurs) dans notre fichier.
// ------------------------------------------------------------------------------------------------

#include "textures.inc"        // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"          // Utilisation du fichier définissant les couleurs standard de POV-Ray


global_settings { charset utf8 }        // Nécessaire pour pouvoir utiliser les caractères accentués
```

```
// Paramètres stéréoscopiques
#declare STEREO        =  1;
#declare STEREOBASE    =  4;
#declare STEREOWINDOW  = 28;


#if (STEREO)
#version unofficial stereopov 0.21;     // Obligatoire pour utiliser la version stéréo du logiciel
#end

#declare Position_Verre = <40,92.5,-35>;
#declare Distance_Fanto = 25;

// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------
camera {

   location Position_Verre+<0, Distance_Fanto, -Distance_Fanto>
   direction <0,0,1>
   look_at Position_Verre
   ////location <40,92.5+50,-35>

#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}


// ------------------------------- Déclaration des sources lumineuses ----------------------------------
light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}
light_source { <-200, 500, 600> color rgb <1.00, 1.00, 1.00> shadowless }


// -------------------------- Déclaration d'une nouvelle texture plus claire -----------------------------
#declare White_Marble_Map2 =
color_map {
   [0.0 rgb <0.9, 0.9, 0.9>]
   [0.8 rgb <0.75, 0.75, 0.75>]
   [1.0 rgb <0.5, 0.5, 0.5>]
}

// White marble with black veins.
#declare White_Marble2 =
pigment {
   marble
   turbulence 1
   color_map { White_Marble_Map2 }
}


// ---------------------------------------- La Table ------------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>   texture { White_Marble2 scale 10 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5 texture { Tan_Wood scale 0.5 rotate x*90 } }


#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
```

```
      object { pied    translate <-70, 0,-45> }
      object { pied    translate < 70, 0,-45> }
      object { pied    translate <-70, 0, 45> }
      object { pied    translate < 70, 0, 45> }
   }
object { table }

// ------------------------------------ Les verres, la paille-------------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =

   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }


#declare verre_rempli =
union {
   object { verre }
   object { cylinder { <0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
   object { paille rotate z*10 translate <1,1,0> }
}

#declare verre_rempli2 =
union {
   object { verre }
   object { cylinder { <0,1,0> <0,6,0> 2.8 } texture { Ruby_Glass } }
   scale <1.25,0.33,1.25>
}

#declare Yellow_Glass=
texture {
   NBglass
   pigment { rgbf <0.98, 0.98, 0.1, 0.9> }
}
#declare M_Yellow_Glass = material {texture {Yellow_Glass} interior {Glass_Interior}}

#declare verre_rempli3 =
union {
   object { verre }
   object { cylinder { <0,1,0> <0,12.5,0> 2.8 } material { M_Yellow_Glass } }
   scale <1.25,0.33,1.25>
}

object { verre_rempli2       translate <32,92.5,-29> }
object { verre_rempli3       translate <48,92.5,-33> }
object { verre_rempli scale <1,0.5,1>  translate Position_Verre - z*5 }



// ------------------------------------ Le sol et l'origine ------------------------------------------
// Pas vraiment utile dans cette scène...
plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
```

```
sphere {       // Sphere at 0,0
     <0,0,0>, 5
     pigment {color <1,0,0>}
//     finish {phong 0.3 phong_size 10}
}

// ---------------------------------- Cadre pour le fantogramme ----------------------------------------

#declare pCadre =  33;         // Largeur du cadre (c'est un carré)
#declare pDia   =   0.5;       // Diamètre des billes matérialisant les coins du cadre
#declare xC     =  40;         // Position du centre du cadre en x, y et z
#declare yC     =  92.5;
#declare zC     = -30;
#declare ANGLE  =   0;         // Eventuellement, rotation du cadre autour de l'axe vertical

object {
  union {
    sphere { < pCadre/2,0, pCadre/2>, pDia pigment {color <1,0,0>} no_shadow }
    sphere { <-pCadre/2,0, pCadre/2>, pDia pigment {color <1,0,0>} no_shadow }
    sphere { <-pCadre/2,0,-pCadre/2>, pDia pigment {color <1,0,0>} no_shadow }
    sphere { < pCadre/2,0,-pCadre/2>, pDia pigment {color <1,0,0>} no_shadow }
  }
  scale        <1,1,1>        // Eventuellement pour un cadre non carré, changer le x ou le z
  rotate       -y*ANGLE
  translate     <xC,yC,zC>
}
```

- Fichier Etape_14.pov
  Réalisation d'une animation simple

```
// ------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 14 : Réalisation d'une animation simple
//      - utilisation de la variable prédéfinie clock
//      - définition d'une variable régissant la hauteur de liquide dans le verre, contrôle des valeurs
//        admissibles
//      - faire varier la hauteur du liquide en fonction de la variable horloge.
// ------------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"           // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 0;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel
global_settings { charset utf8 }        // Nécessaire pour pouvoir utiliser les caractères accentués


// ------------------------------ Déclaration de la caméra et du point de vue ---------------------------

camera {
   location < 40,  115, -70>
```

```
   look_at  < 30,  100,   0>
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}


// ------------------------------ Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}


// ------------------------------------ Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ---------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }


// ---------------------------------------- Le ciel ----------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, la paille----------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
```

```
      texture { NBglass }
   }

#declare paille =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }


#declare hauteurLiquide = 11 - 11*clock;

#declare verre_rempli =
union {
   object { verre }
#if ((hauteurLiquide > 0) * (hauteurLiquide <= 14))
   object { cylinder { <0,1,0> <0,1+hauteurLiquide,0> 2.8 } texture { NBwinebottle } }
#end
   object { paille rotate z*10 translate <1,1,0> }
}

object { verre        translate <20,92.5,-15> }
object { verre_rempli translate <40,92.5,-35> }
```

Fichier Etape_14.ini : fichier de contrôle de l'animation

```
; ------------------------------------------------------------------------------------------------
;                        Stéréo-Club Français - Séance Technique du 21 mars 2007
;                         Premiers pas en images de synthèse - Pierre Meindre
;
; Etape 14  : Réalisation d'une animation simple
;      - horloge variant de 0 à 1 (pourcentage d'avancement de l'animation)
;      - nombre d'images à générer : animation plus ou moins fluide, plus ou moins rapide
; ------------------------------------------------------------------------------------------------

;Antialias=Off

;Antialias_Threshold=0.2
;Antialias_Depth=3
Jitter=off

; Nom du fichier POV à utiliser
Input_File_Name=Etape_14.pov

; Numéro de la première image et numéro de la dernière image
Initial_Frame =  1
Final_Frame   = 25

; Valeurs initiale et finale de l'horloge
Initial_Clock = 0
Final_Clock   = 1
```

- Fichier Etape_14B.pov
  Réalisation d'une animation simple, Vases communicants

```
// ------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
```

```
//                      Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 14B : Réalisation d'une animation simple, Vases communicants
//      - définition d'une grande paille doublement coudée reliant deux verres.
//      - définition d'un modèle paramétrable de verre pour avoir deux verres remplis différemment et
indépendamment.
//      - faire varier la hauteur du liquide dans les deux verres en fonction de la variable horloge.
// ---------------------------------------------------------------------------------------------

#include "textures.inc"         // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"           // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 0;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel
global_settings { charset utf8 }        // Nécessaire pour pouvoir utiliser les caractères accentués


// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   location < 40,  115, -70>
   look_at  < 30,  100,   0>
   //location < 40,  150,   -30>   look_at  < 40,  100,   -30>          // Vue de dessus pour faciliter le placement
des objets sur la table

#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}


// ------------------------------- Déclaration de la (ou des) source lumineuse ----------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}


// ------------------------------------- Le sol et l'origine -------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// ---------------------------------------- La Table ---------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>        texture { Tan_Wood  scale 2 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5      texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
```

```
      object { pied     translate < 70, 0, 45> }
   }


object { table }


// ---------------------------------------- Le ciel ------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, les pailles ------------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }


// Nouvelle paille de base (sans texture)
#declare paille2 =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
   }

// Définition d'un coude avec un quart de tore
#declare paille2Coude =
object {
  torus { 1, 0.30 }
  clipped_by { plane { x, 0 } }
  clipped_by { plane { z, 0 } }
  rotate x*90
}

// Assemblage de la grande paille doublement coudée
#declare pailleCoudee =
union {
  object { paille2 }
```

```
  object { paille2Coude translate <1, 20, 0> }
  object { paille2 rotate z*90 translate <21,21,0> }
  object { paille2Coude rotate y*180 translate <21, 20, 0> }
  object { paille2 translate <22, 0, 0> }
  texture { Candy_Cane }
}


#macro verre_rempli( hauteurLiquide )
union {
   object { verre }
#if ((hauteurLiquide > 0) * (hauteurLiquide <= 14))
   object { cylinder { <0,1,0> <0,1+hauteurLiquide,0> 2.8 } texture { NBwinebottle } }
#end
   //object { paille rotate z*10 translate <1,1,0> }
}
#end

#declare verre_rempli2 =
union {
   object { verre }
   object { cylinder { <0,1,0> <0,6,0> 2.8 } texture { Ruby_Glass } }
   scale <1.25,0.6,1.25>
}

#declare Yellow_Glass=
texture {
   NBglass
   pigment { rgbf <0.98, 0.98, 0.1, 0.9> }
}
#declare M_Yellow_Glass = material {texture {Yellow_Glass} interior {Glass_Interior}}

#declare verre_rempli3 =
union {
   object { verre }
   object { cylinder { <0,1,0> <0,12.5,0> 2.8 } material { M_Yellow_Glass } }
   scale <1.25,0.6,1.25>
}

#declare hauteurLiquide1 = 11 - 11*clock;
#declare hauteurLiquide2 = 11*clock;


object { verre_rempli(hauteurLiquide1)       translate <42,92.5,-35> }
object { verre_rempli(hauteurLiquide2)       translate <20,92.5,-28> }

object { verre_rempli2      translate <30,92.5,-35> }
object { verre_rempli3      translate <52,92.5,-25> }

object { pailleCoudee   rotate x*5 rotate y*15    translate <20.5,92, -28.5> }


//object { verre       translate <20,92.5,-15> }
//object { verre_rempli translate <40,92.5,-35> }


// ---------------------------------- Le titre ------------------------------------------------------
/*
#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;

text {
   ttf "times.ttf" "L'image de synth\u00E8se en relief, c'est facile avec" 0.3, 0
```

```
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
}

text {
   ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <3, 5, 10>
   rotate <0, -5, 0>
   translate <28, 114.5, -50>
}
*/
```

Fichier Etape_14B.ini : fichier de contrôle de l'animation

```
; ------------------------------------------------------------------------------------------------------
;                        Stéréo-Club Français - Séance Technique du 21 mars 2007
;                         Premiers pas en images de synthèse - Pierre Meindre
;
; Etape 14B  : Réalisation d'une animation simple
;      - horloge variant de 0 à 1 (pourcentage d'avancement de l'animation)
;      - nombre d'images à générer : animation plus ou moins fluide, plus ou moins rapide
; ------------------------------------------------------------------------------------------------------

;Antialias=Off

;Antialias_Threshold=0.2
;Antialias_Depth=3
Jitter=off

; Nom du fichier POV à utiliser
Input_File_Name=Etape_14B.pov

; Numéro de la première image et numéro de la dernière image
Initial_Frame =  1
Final_Frame   = 73

; Valeurs initiale et finale de l'horloge
Initial_Clock = 0
Final_Clock   = 1

; 800x600 : 2h20
```

- Fichier Etape_15.pov
  Animation et lenticulaire (1) - Vue de dessus

```
// ------------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 15  : Animation et lenticulaire (1) - Vue de dessus
//      - Quelques modifications : utilisation du plateau en marbre et des verres définis à l'étape 13
//      - Ajout de "no_shadow" aux objets text pour éviter leur ombre sur la surface de la table
// ------------------------------------------------------------------------------------------------------
```

```
#include "textures.inc"        // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"          // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 0;      // Permet de tracer en mono (mettre la valeur 0), plus rapide ou en stéréo
(mettre la valeur 1)
#declare STEREOBASE   = 2;      // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;     // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence


#version unofficial stereopov 0.2;      // Obligatoire pour utiliser la version stéréo du logiciel
global_settings { charset utf8 }        // Nécessaire pour pouvoir utiliser les caractères accentués


// ------------------------------- Déclaration de la caméra et du point de vue ----------------------------

camera {
   //location < 40,  115, -70>     look_at  < 30,  100,   0>          // Position d'origine de la caméra (mise en
commentaire)
   location < 40,  150,   -30>   look_at  < 40,  100,   -30>          // Vue de dessus pour faciliter le placement
des objets sur la table
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}


// ------------------------------- Déclaration de la (ou des) source lumineuse ----------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00> }


// ------------------------------------- Le sol et l'origine ------------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


// -------------------------- Déclaration d'une nouvelle texture plus claire -----------------------------
#declare White_Marble_Map2 =
color_map {
   [0.0 rgb <0.9, 0.9, 0.9>]
   [0.8 rgb <0.75, 0.75, 0.75>]
   [1.0 rgb <0.5, 0.5, 0.5>]
}

// White marble with black veins.
#declare White_Marble2 =
pigment {
   marble
   turbulence 1
   color_map { White_Marble_Map2 }
}

// ---------------------------------------- La Table ------------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>       texture { White_Marble2  scale 4 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5     texture { Tan_Wood scale 0.5 rotate x*90 } }
```

```
#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }


// ---------------------------------------- Le ciel ----------------------------------------------------

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
         [1.0 color SkyBlue]
      }
      scale <0.5, 0.1, 1000>
   }
}

// ------------------------------------ Les verres, la paille----------------------------------------

#declare verre =
   difference {
      cylinder { <0,0,0> <0,15,0> 3 }
      cylinder { <0,1,0> <0,16,0> 2.8 }
      texture { NBglass }
   }

#declare paille =
   difference {
      cylinder { <0,0,0>  <0,20,0> 0.30 }
      cylinder { <0,-1,0> <0,21,0> 0.28 }
      texture { Candy_Cane }
   }


#declare verre_rempli =
union {
   object { verre }
   object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
   object { paille rotate z*10 translate <1,1,0> }
}

#declare verre_rempli2 =
union {
   object { verre }
   object { cylinder { <0,1,0> <0,6,0> 2.8 } texture { Ruby_Glass } }
   scale <1.25,0.6,1.25>
}
```

```
#declare Yellow_Glass=
texture {
   NBglass
   pigment { rgbf <0.98, 0.98, 0.1, 0.9> }
}
#declare M_Yellow_Glass = material {texture {Yellow_Glass} interior {Glass_Interior}}

#declare verre_rempli3 =
union {
   object { verre }

   object { cylinder { <0,1,0> <0,12.5,0> 2.8 } material { M_Yellow_Glass } }
   scale <1.25,0.6,1.25>
}

object { verre              translate <20,92.5,-15> }
object { verre_rempli       translate <37,92.5,-35> }
object { verre_rempli2      translate <27,92.5,-26> }
object { verre_rempli3      translate <48,92.5,-30> }


// ------------------------------------ Le titre ------------------------------------------------------

#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;


text {
   ttf "times.ttf" "L'image de synth\u00E8se en relief, c'est facile avec" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
   no_shadow
}

text {
   ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <3, 5, 10>
   rotate <0, -5, 0>
   translate <28, 114.5, -50>
   no_shadow
}
```

- Fichier Etape_16.pov
  Animation et lenticulaire (2)

```
// ---------------------------------------------------------------------------------------------------
//                        Stéréo-Club Français - Séance Technique du 21 mars 2007
//                         Premiers pas en images de synthèse - Pierre Meindre
//
// Etape 16  : Animation et lenticulaire (2)
//      - Utilisation de traces d'exécution
//      - Utilisation de l'horloge d'animation (clock)
//      - Paramétrage de la position de la caméra mobile
//      - Création d'un script d'animation (Etape_15.ini)

// Messages : Fenêtre "Messages", Alt+M, fenêtres splitable
```

```
// ------------------------------------------------------------------------------------------

#include "textures.inc"        // Utilisation du fichier définissant les textures standard de POV-Ray
#include "colors.inc"          // Utilisation du fichier définissant les couleurs standard de POV-Ray

// Paramètres stéréoscopiques
#declare STEREO       = 0;     // Chaque image pour le lenticulaire est monoscopique
#declare STEREOBASE   = 2;     // Base quasi macro-stéréoscopique (2 cm)
#declare STEREOWINDOW = 30;    // Position de la fenêtre stéréoscopique (distance depuis la caméra)
ajustée en conséquence

#declare TRACE_TABLE    = 1;
#declare TRACE_VERRES   = 1;
#declare TRACE_TITRE    = 1;
#declare TRACE_CIEL     = 1;


#version unofficial stereopov 0.21;      // Obligatoire pour utiliser la version stéréo du logiciel
global_settings { charset utf8 }         // Nécessaire pour pouvoir utiliser les caractères accentués


// ------------------------------------ Utilisation de l'horloge d'animation ------------------------------
#declare clk = 0.00+clock;
#debug concat("Horloge = ", str(clk,5,5), "\n")         // Affichage dans la fenêtre des messages de la valeur
courante de l'horloge


// -------------------------------- Déclaration de la caméra et du point de vue ---------------------------

#declare camPos_org = < 40,  115,  -80>;         // Paramètres d'origine de la caméra
#declare camLA_org  = < 30,  100,    0>;
#declare verrePos   = < 37,   92.5,-35>;         // Position du verre avec la paille


#declare distCam = vlength(camPos_org-verrePos);       // Calcul de la distance caméra - sujet
#debug concat("Distance nominale au sujet = ", str(distCam,7,2), "\n")
#declare distCam = vlength(<camPos_org.x,0,camPos_org.z>-<verrePos.x,0,verrePos.z>);       // Calcul de
la distance caméra - sujet projetée sur le plan de la table
#debug concat("Distance nominale au sujet projetee sur la table = ", str(distCam,7,2), "\n")


// La caméra va décrire un arc de cercle autour du sujet (tout en restant à la même distance)
#declare camRotation = 20;               // La caméra va tourner de cet angle durant l'animation
#declare camAngle = 270 + clk*camRotation - camRotation/2;
#debug concat("Angle la camera = ", str(camAngle,4,2),  "\n")
#declare camPos = verrePos + < distCam * cos(camAngle / 180 * pi), camPos_org.y-verrePos.y, distCam *
sin(camAngle / 180 * pi) >;
#declare camLA  = camLA_org;
#debug concat("Position de la camera = <", str(camPos.x,4,2), ", ", str(camPos.y,4,2), ", ",
str(camPos.z,4,2), ">\n")

camera {
   location camPos     look_at  camLA          // Position calculée de la caméra mobile

   //location < 40,  150,   -30>   look_at  < 40,  100,   -30>          // Vue de dessus pour faciliter le placement
des objets sr la table
#if (STEREO)
  stereo_base STEREOBASE
  window_distance STEREOWINDOW
  cross_eyed
#end
}
```

```
// ------------------------------ Déclaration de la (ou des) source lumineuse ---------------------------

light_source { <-200, 500, -600> color rgb <1.00, 1.00, 1.00>}


// ---------------------------------------- Le sol et l'origine -----------------------------------------

plane { y, 0 pigment {checker color rgb 1.0, color rgb 0.9 scale 100} }
sphere {   <0,0,0>, 5   pigment {color <1,0,0>}   }


#if (TRACE_TABLE)
// ------------------------- Déclaration d'une nouvelle texture plus claire -----------------------------
#declare White_Marble_Map2 =
color_map {
   [0.0 rgb <0.9, 0.9, 0.9>]
   [0.8 rgb <0.75, 0.75, 0.75>]
   [1.0 rgb <0.5, 0.5, 0.5>]
}

// White marble with black veins.
#declare White_Marble2 =
pigment {
   marble
   turbulence 1
   color_map { White_Marble_Map2 }
}

// ---------------------------------------- La Table ----------------------------------------------------

#declare plateau = box { <-75,0,-50> <75,2.5,50>       texture { White_Marble2  scale 4 } }
#declare pied    = cylinder { <0,0,0> <0,90,0> 2.5     texture { Tan_Wood scale 0.5 rotate x*90 } }

#declare table =
   union {
      object { plateau  translate <  0,90,  0> }
      object { pied     translate <-70, 0,-45> }
      object { pied     translate < 70, 0,-45> }
      object { pied     translate <-70, 0, 45> }
      object { pied     translate < 70, 0, 45> }
   }


object { table }
#end


// ---------------------------------------- Le ciel -----------------------------------------------------
#if (TRACE_CIEL)

sky_sphere {
   pigment {
      wrinkles
      turbulence 0.3
      omega 0.707
      octaves 5
      color_map {
         [0.0 color DustyRose * 2.5]
         [0.2 color Orange ]
         [0.8 color SlateBlue * 0.25]
```

```
            [1.0 color SkyBlue]
        }
        scale <0.5, 0.1, 1000>
    }
}
#end



// ---------------------------------- Les verres, la paille---------------------------------------------
#if (TRACE_VERRES)

#declare verre =
    difference {
        cylinder { <0,0,0> <0,15,0> 3 }
        cylinder { <0,1,0> <0,16,0> 2.8 }
        texture { NBglass }
    }

#declare paille =
    difference {
        cylinder { <0,0,0>  <0,20,0> 0.30 }
        cylinder { <0,-1,0> <0,21,0> 0.28 }
        texture { Candy_Cane }
    }


#declare verre_rempli =
union {
    object { verre }
    object { cylinder {<0,1,0> <0,12,0> 2.8 } texture { NBwinebottle } }
    object { paille rotate z*10 translate <1,1,0> }
}

#declare verre_rempli2 =
union {
    object { verre }
    object { cylinder { <0,1,0> <0,6,0> 2.8 } texture { Ruby_Glass } }
    scale <1.25,0.6,1.25>
}

#declare Yellow_Glass=
texture {
    NBglass
    pigment { rgbf <0.98, 0.98, 0.1, 0.9> }
}
#declare M_Yellow_Glass = material {texture {Yellow_Glass} interior {Glass_Interior}}

#declare verre_rempli3 =
union {
    object { verre }
    object { cylinder { <0,1,0> <0,12.5,0> 2.8 } material { M_Yellow_Glass } }
    scale <1.25,0.6,1.25>
}

object { verre              translate <20,92.5,-15> }
object { verre_rempli       translate verrePos }
object { verre_rempli2      translate <27,92.5,-26> }
object { verre_rempli3      translate <48,92.5,-30> }
#end
```

```
// ---------------------------------- Le titre ------------------------------------------------------
#if (TRACE_TITRE)

#declare zBrightGold = color red 0.95 green 0.55 blue 0.10;

text {
   ttf "times.ttf" "L'image de synth\u00E8se en relief, c'est facile avec" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <2, 3, 3>
   rotate <0, -10, 0>
   translate <18, 121, -40>
   no_shadow
}

text {
   ttf "times.ttf" "StereoPOV ! ! !" 0.3, 0
   pigment { zBrightGold }
   finish { reflection .25 specular 1 }
   scale <3, 5, 10>
   rotate <0, -5, 0>
   translate <28, 114.5, -50>
   no_shadow
}
#end
```

Fichier Etape_16.ini : fichier de contrôle de l'animation

```
; ------------------------------------------------------------------------------------------------------
;                        Stéréo-Club Français - Séance Technique du 21 mars 2007
;                         Premiers pas en images de synthèse - Pierre Meindre
;
; Etape 16  : Animation et lenticulaire (2)
; ------------------------------------------------------------------------------------------------------

;Antialias=Off

;Antialias_Threshold=0.2
;Antialias_Depth=3
Jitter=off

; Nom du fichier POV à utiliser
Input_File_Name=Etape_16.pov

; Numéro de la première image et numéro de la dernière image
Initial_Frame =  1
Final_Frame   = 20

; Valeurs initiale et finale de l'horloge
Initial_Clock = 0
Final_Clock   = 1
```

Fichier InterlacesImages.vbs : script d'entrelacement d'image pour MyAlbum

```
' ------------------------------------------------------------------------------
'                              InterlacesImages.vbs
' This script will generate a vertically interlaced picture.
' The source pictures will be the selected pictures in the current album.
```

```
' Note: this script is constantly using the Clipboard for the temporary pictures, so
'       use of the clipboard should be avoided while the script is running.
' ------------------------------------------------------------------------------

Option Explicit

dim alb, nbPic, nbSelPic, pic, newPic, i, selPics(), k, wP, hP, w, h, bOK, iPic

app.ClearTrace
set alb = app.GetCurrentAlbum
nbPic = alb.nbPicture
nbSelPic = alb.nbSelectedPicture
if nbSelPic >= 2 then
  app.Trace nbSelPic & " pictures selected", &hff0000, TRACE_INFORMATION
  redim selPics(nbSelPic)

  k = 1
  bOK = true
  for i = 0 to nbPic-1
    Set pic = alb.GetPicture(i)
    if pic.bSelected then
      app.Trace "Checking picture '" & pic & "'..."
      if k = 1 then ' First picture
        wP = pic.w
        hP = pic.h
        app.Trace "Picture size is: " & wP & "x" & hP, -1, TRACE_ARROW
      else
        if wP <> pic.w or hP <> pic.h then
          app.Trace "Error: all pictures must have the same size", 255, TRACE_ERROR
          bOK = false
          exit for
        end if
      end if
      Set selPics(k) = pic
      pic.Load True
      k = k + 1

    end if
  next
  if bOK then
    set newPic = alb.NewPicture( wP, hP, 32, &hffffff )
    app.Trace "Interlacing pictures...", &hff0000, TRACE_WAIT
    for i = 0 to wP-1
            iPic = (i mod nbSelPic) + 1
            'app.trace iPic
            bOK = selPics(iPic).CopyRect( i, 0, i, hP-1, 0 )
            bOK = newPic.Paste( i, 0 )
            'if i > 10 then exit for
            if (i mod 50) = 0 then     app.Trace "  Interlacing (" & CInt((i/wP)*100) & "%)...", &hff0000,
TRACE_WAIT
    next
    for iPic = 1 to nbSelPic
            selPics(iPic).Load false
            set selPics(iPic) = Nothing
    next
    set newPic = Nothing
  end if
else
  app.Trace "Error: at least two pictures must be selected in the album", 255, TRACE_ERROR
end if
alb.redraw
app.Trace "Done !!!", -1, TRACE_GREENDOT
```