POV-Ray

Un langage de description de scenes

Christian Nguyen

Département d'informatique
Université de Toulon et du Var

16 novembre 2009

Position, orientation et propriétés de la caméra, des sources lumineuses et des objets et leurs textures associées.

```
#include "colors.inc"    // The include files contain
#include "shapes.inc"    // pre-defined scene elements
#include "textures.inc"

camera {
  location  <0, 2, -3>
  look_at   <0, 1,  2>
}

light_source { <2, 4, -3> color White}

background { color Cyan }

sphere {
  <0, 1, 2>, 2
  texture {
    pigment {color Yellow}  // pre-defined in COLORS.INC
    normal {bumps 0.4   scale 0.2}
    finish {phong 1}
  }
}
```

La caméra (obligatoire mais non visible) ne correspond pas au modèle physique mais définit les caractéristiques du plan de projection.

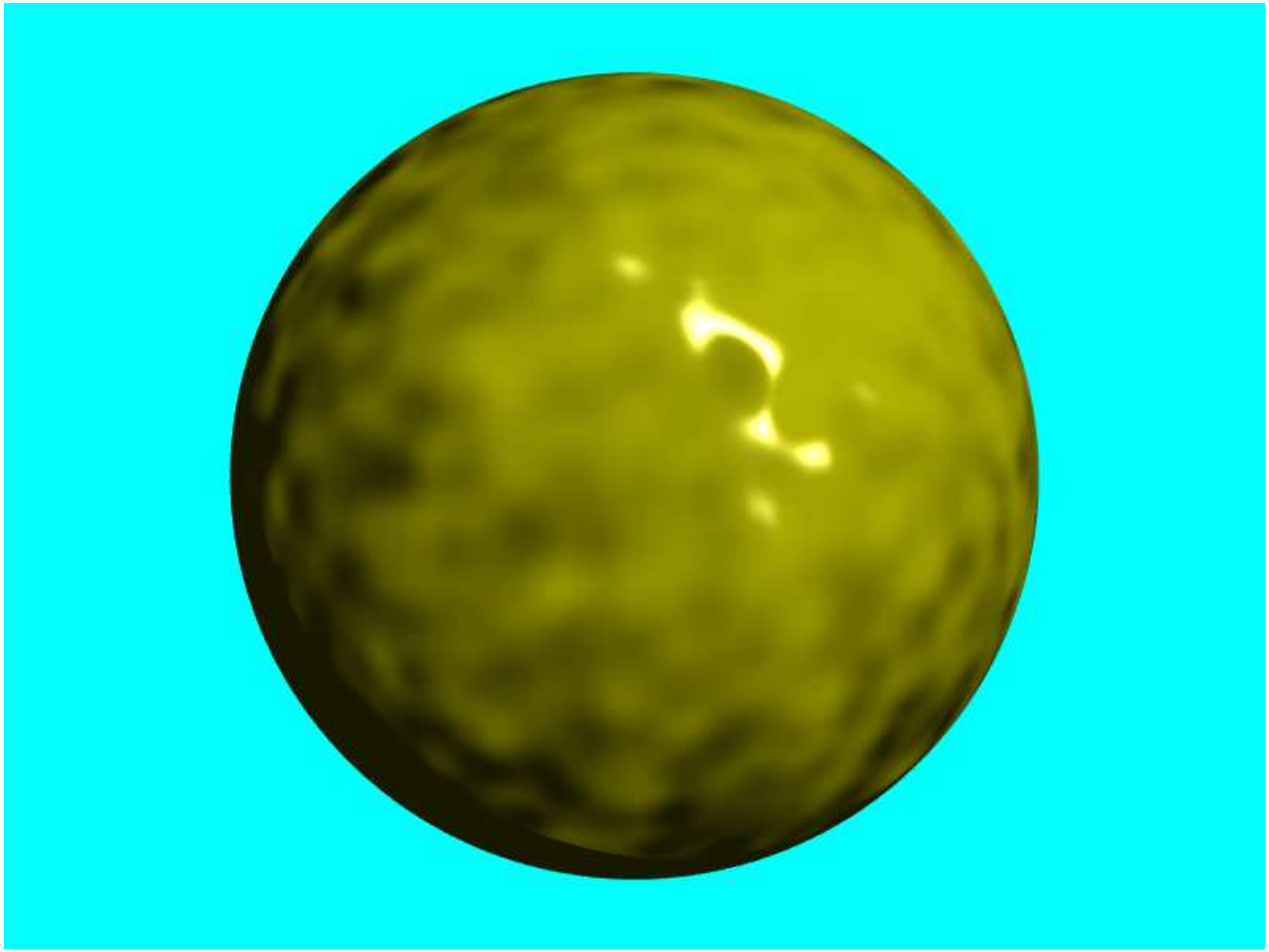
Les sources lumineuses n'ont pas de contour matériel (par défaut).

Une simple ligne de commande :

```
povray +L/usr/share/povray/include +Iexemple1.pov +V +W640 +H480
```

produira une image de nom `exemple1.png`.

Généralités



Eléments fondamentaux

Tous les objets de Pov suivent la même syntaxe, en distinguant majuscules de minuscules :

```
<type_objet> {
        <info_objet>
       [<modif_objet>]
}
```

Commentaires : tout ce qui suit les caractères `//`

Un point 3D est décrit par sa position `<x, y, z>`, définit dans un repère orthonormé indirect ("gauche").

La précision dépend du système mais elle est le plus souvent de 10^{-10}.

Objets de base

Ensemble de formes prédéfinies, volumes les plus simples : parallélépipède, cylindre, cône, plan et tore, par exemple :

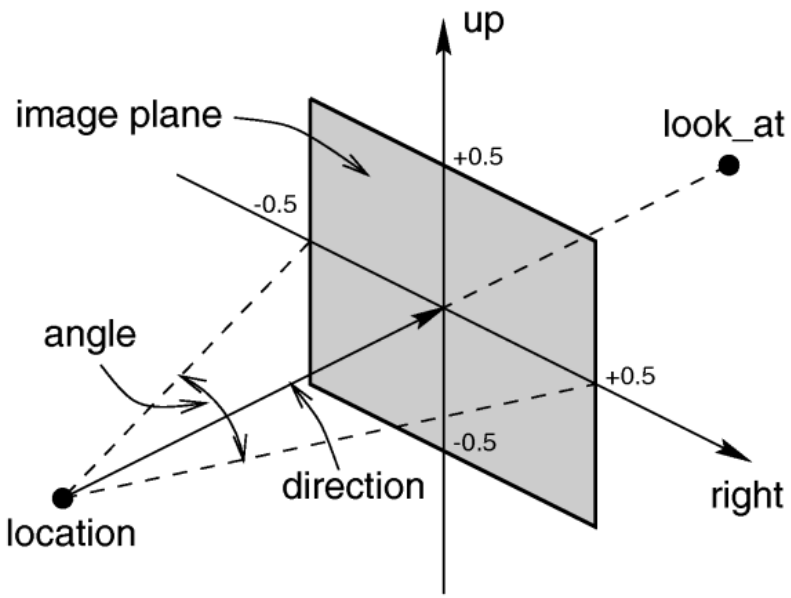
```
cone  <point1>, rayon1, <point2>, rayon2 [open]
```

Possibilité de créer de nouvelles formes et de les instancier :

```
#declare <id> = <element>
object  Ellipsoid
```

Tous les objets peuvent subir des transformations : translation (`translation <distance>`), rotation (`rotation <vecteur>`) et homothétie (`scale <vecteur>`).

Le modèle de caméra



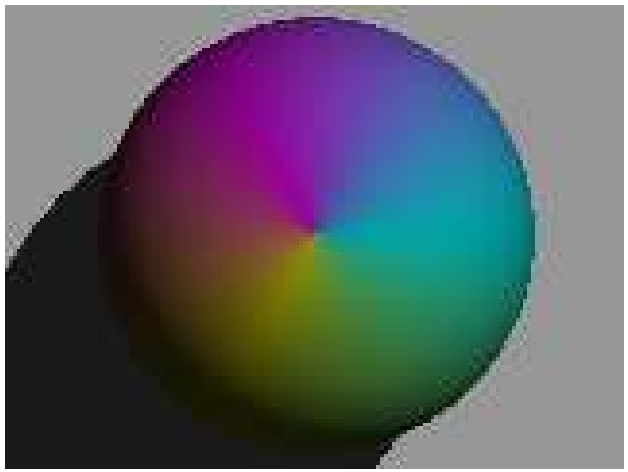

```
camera {
  focal_point <x, y, z>
  aperture n // aire de la focale = 1/n
  blur_samples i // nombre de rayons subdivisant un pixel
```

Textures de base

Propriétés de la matière (prédéfinies : `textures.inc`).

Déclarée par le mot-clé `texture` suivi de ses attributs.

Produit approximativement **un** changement (transition de couleur) pour une sphère de rayon 1.



Texture : pigmentation

Couleur ou motif de couleurs d'un matériau : type, table des couleurs, coefficient de turbulence et transformations.

Table des couleurs : jusqu'à 40 entrées, syntaxe :

```
color_map { [0.0 couleur1] ... [0.x couleuri] ...  [1.0 couleurf] }
```

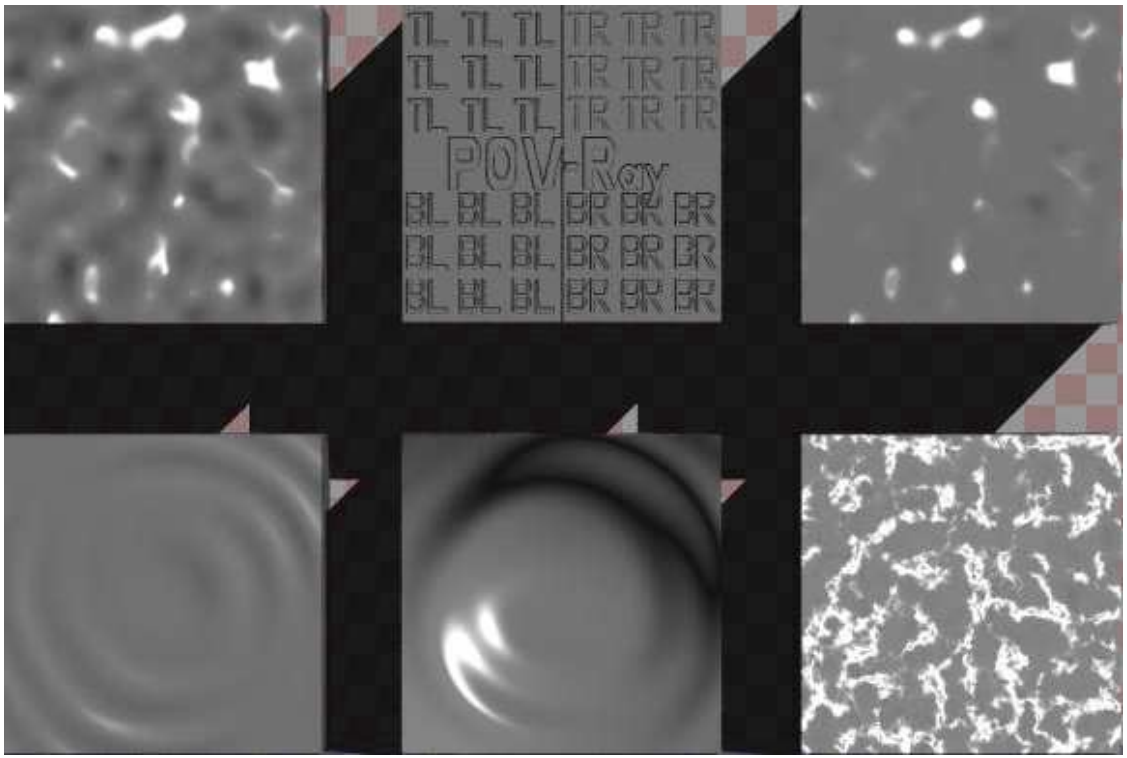
Motifs de couleur : fonctions mathématiques transformant les coordonnées (x, y, z) de chaque point de la surface en une valeur réelle $[0.0, 1.0]$.

Exemple :

```
pigment {
  checker color Blue color White
  scale 0.5
}
```

Texture : normale

Création peu coûteuse de surfaces irrégulières (`normal`). Par défaut, la normale permet de déterminer comment la lumière est réfléchie en chaque point d'un objet.



Texture : finition

Permet de définir les propriétés des surfaces non couvertes par les définitions de pigmentation et de normale (`finish`).

Paramètres les plus importants : `ambient`, `diffuse`, `phong` et `reflection`.

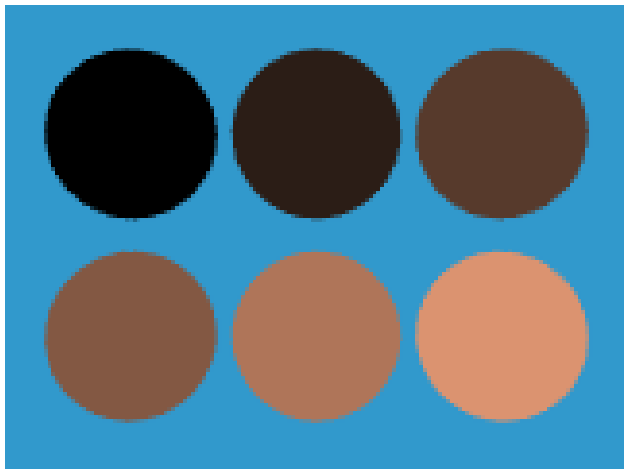
```
finish {
            ambient     a       // ambient factor [0,1], 0
            diffuse     d       // diffuse factor [0,1], 0.6
            phong       p       // phong highlight [0,1], 0
            reflection  r       // reflection factor [0,1], 0
       }
```

Finition : lumière ambiante

La luminosité ambiante est simulée par un coefficient de réflexion ambiante, qui mesure la quantité de lumière indirecte.

Il n'y a pas de source de lumière ambiante.

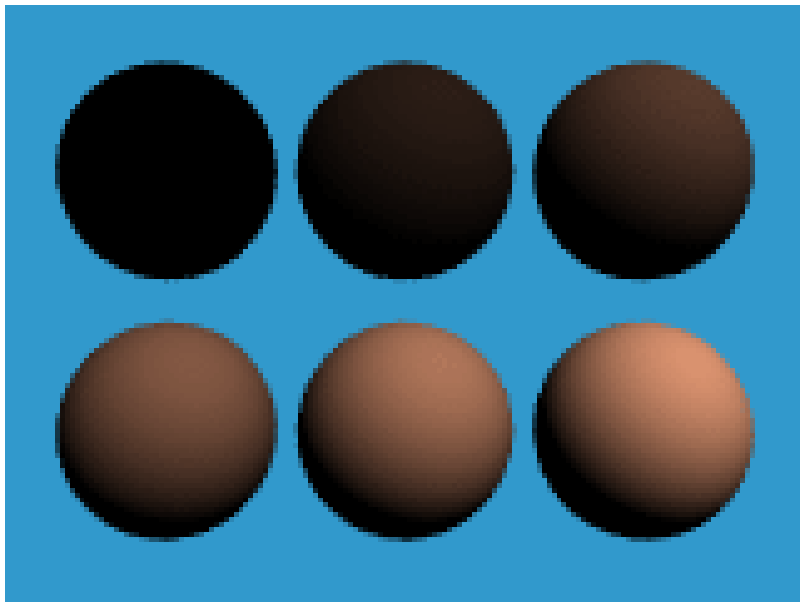
Dans l'exemple ci-dessous, $d = 0$, $a = 0.0, 0.2, \ldots, 1.0$, couleur "chair" :



Finition : réflexion diffuse

Réflexion dans toutes les directions d'une source de lumière directionnelle. Le coefficient détermine quel quantité de lumière visible provient directement d'une source lumineuse.

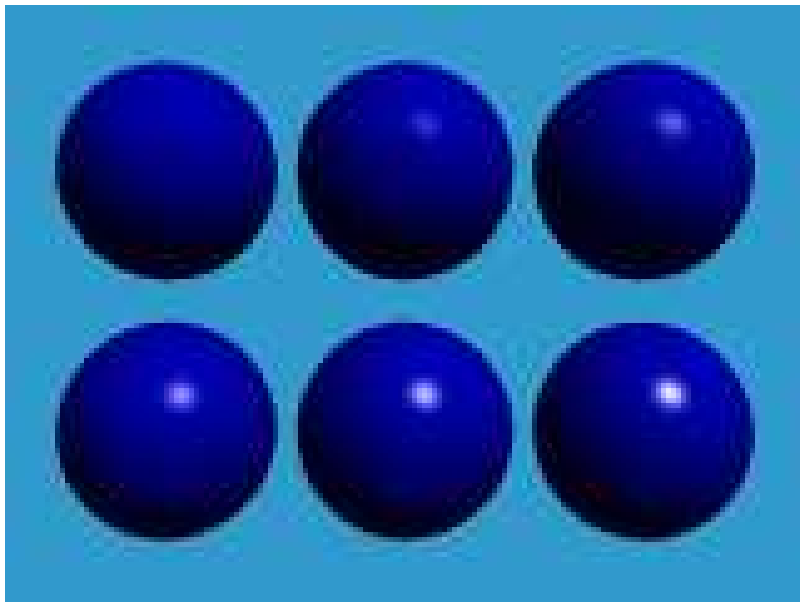
$d = 0.0, 0.2, \ldots, 1.0$:



Finition : réflexion spéculaire

Réflexion dans une direction privilégiée d'une source de lumière directionnelle (surfaces brillantes). Plus le coefficient est élevé, plus la lumière se réfléchie dans la direction de l'observateur.

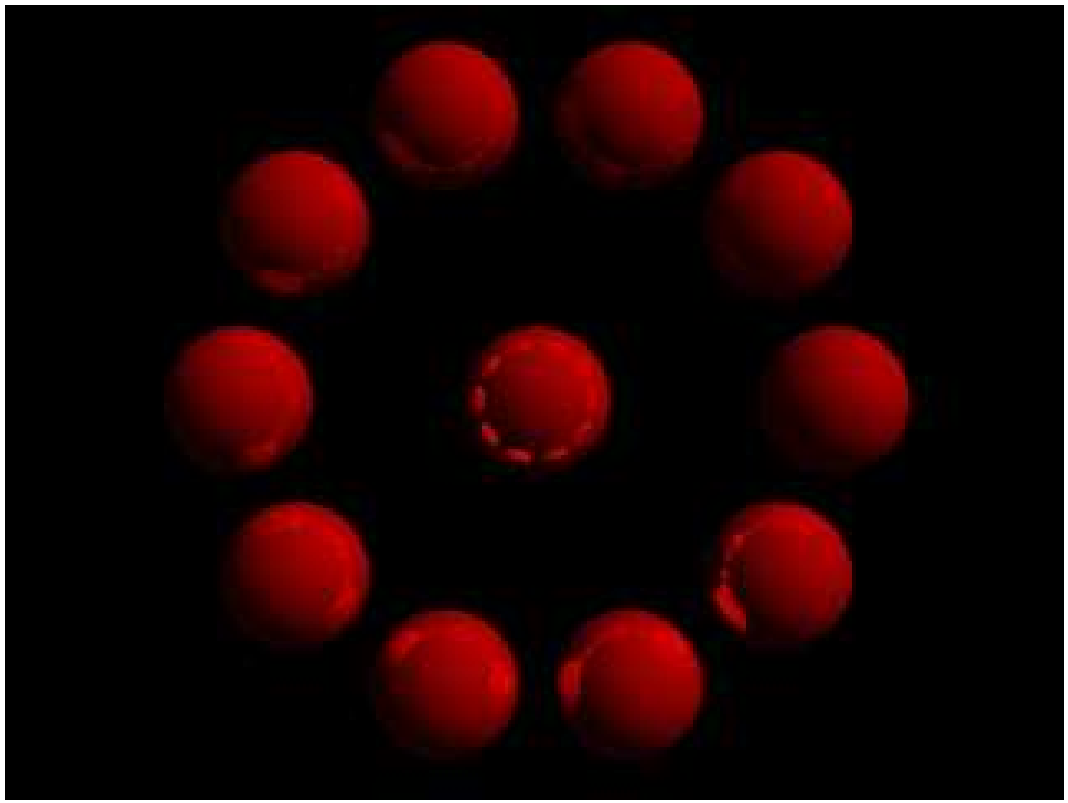
$p = 0.0, 0.2, \ldots, 1.0$:



Finition : reflets

Le mot-clé `reflection` détermine la capacité de la surface à refleter son environnement. C'est un paramètre coûteux.

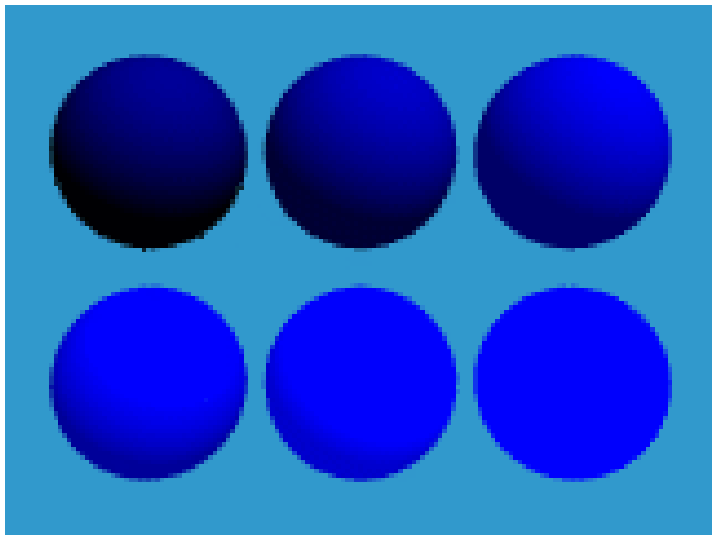
$r = 0.0, 0.1, \ldots, 1.0$:

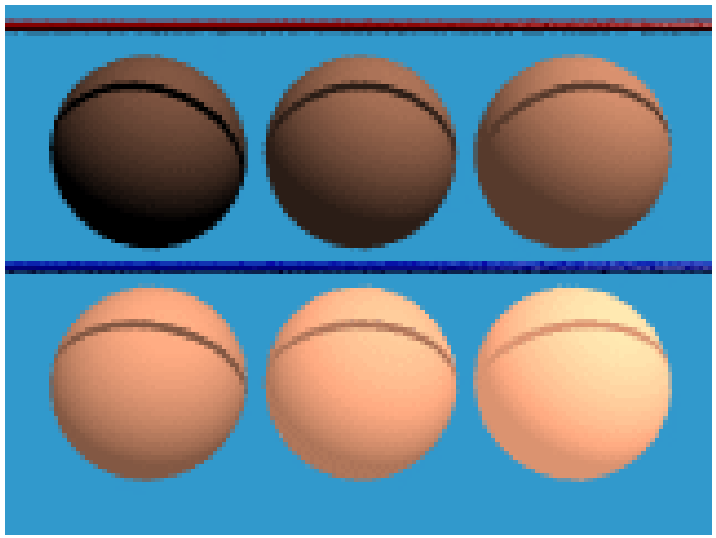


Texture : finition

Attention : la somme des valeurs des coefficients `ambient`, `diffuse` et `reflection` ne doit pas dépasser 1.

Dans le cas contraire, effet “délavé” :





Texture : finition

En tout généralité, jusqu'à 12 paramètres :

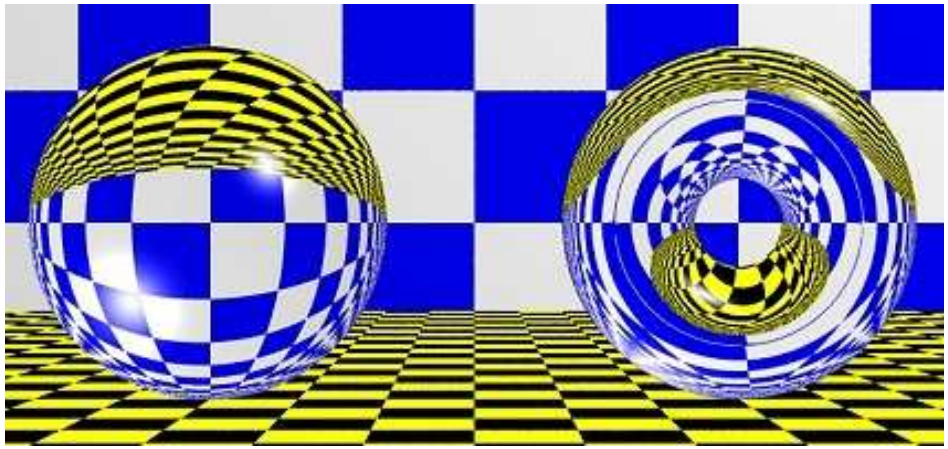
- contributions : `ambient`, `diffuse` et `reflection`,
- réflexion diffuse : `brilliance` ($I = \cos(\alpha_i^b)$), `crand`,
- réflexion spéculaire (`specular` et `roughness` ; `phong` et `phong_size` ; `metallic`),
- versions < 3.0 : refraction (`ior`, `refraction`).

Texture : réfraction

Mesure le degré de courbure des rayons lumineux dans un matériau (semi-)transparent par le paramètre `ior` (air $= 1.0$, eau $= 1.33$, verre $= 1.61$, diamant $= 2.47$, ...)

Dans les versions les plus récentes : `interior`

Couleur transparente : jouer sur le coefficient alpha dans le codage RVBA ou utiliser le mot-clé `filter` $\in [0, 1]$



Surfaces infinies orientées

Plan : `plane <normale>, deplacement` ou `plane <A, B, C>, D` :

$$Ax + By + Cz - D\sqrt{A^2 + B^2 + C^2} = 0$$

Quadrique : `quadric <A, B, C>, <D, E, F>, <G, H, I>, J` :

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0$$

Exemples : $x^2 + y^2 + z^2 - 1 = 0$, $x^2 + y^2 - 1 = 0$, ...

Cubique (polynôme du 3ième ordre) et quartique (polynôme du 4ième ordre : tore, ...).

`shapes.inc` : quadriques les plus connues.

Surfaces non orientées

triangle : `triangle <point1>, <point2>, <point3>`

triangle interpolé pour lequel il faut définir 3 normales

`smooth_triangle <pt1>, <n1>, <pt2>, <n2>, <pt3>, <n3>`

maillage 3D : `mesh <triangle|smooth_triangle texture ... >`

polygone : `polygon nb_pts, <pt1>, ..., <ptn>, <pt1>`

disque : `disc <centre>, <normale>, rayon_ext [, rayon_int]`

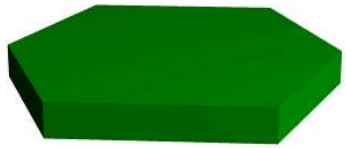
carreau bicubic (nombre de triangles $2(2^{u\_steps} \times 2^{v\_steps})$)

`bicubic_patch type 1 flatness <f> u_steps <u> v_steps <v> <PC1>, ..., <PC16>`

Primitives volumiques

Prismes : solides définis par translation d'un contour

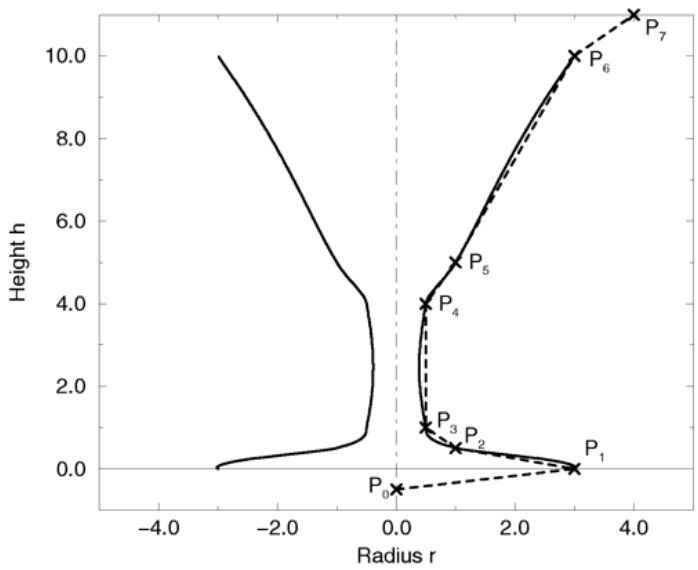
```
prism {
  linear_sweep
  linear_spline
  0, 1, // coord. du parcours en Y
  nb_pt, <x1, z1>, ..., <xi, zi> // xi = x1, zi = z1
```

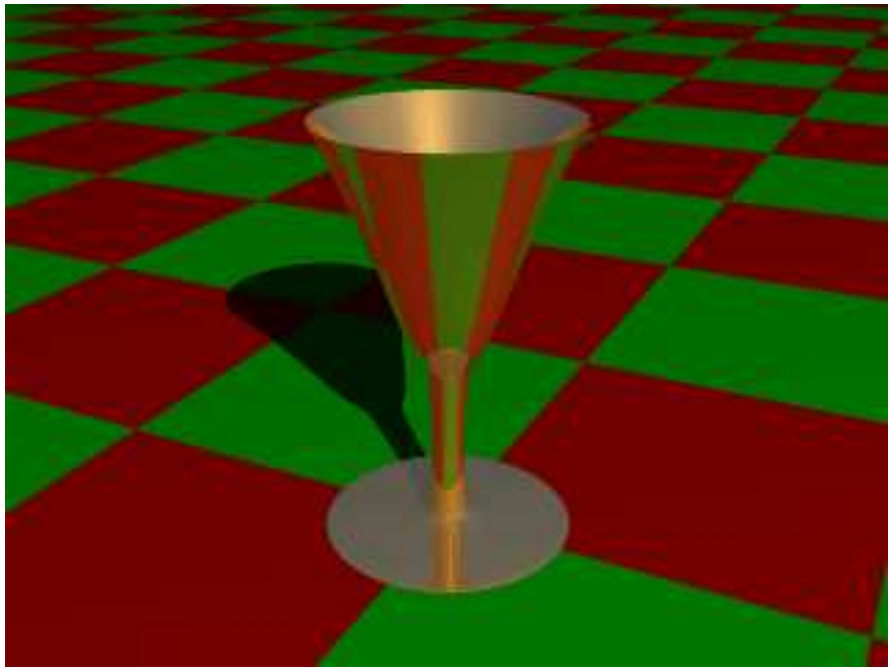


Surfaces de révolution : `sor` et `lathe` :

```
lathe {
  linear_spline // ou quadratic, cubic
  nb_pt, <x1, y1>, ..., <xi, yi>
```

Primitives volumiques

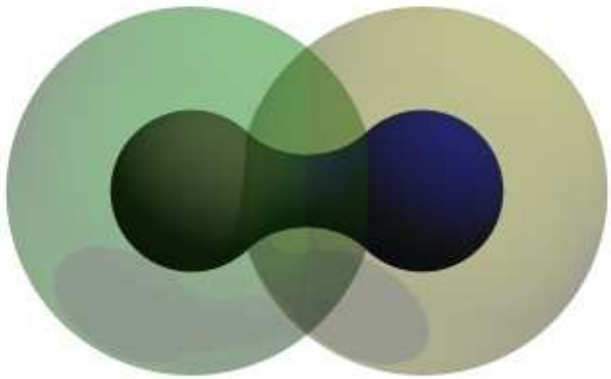




Primitives volumiques

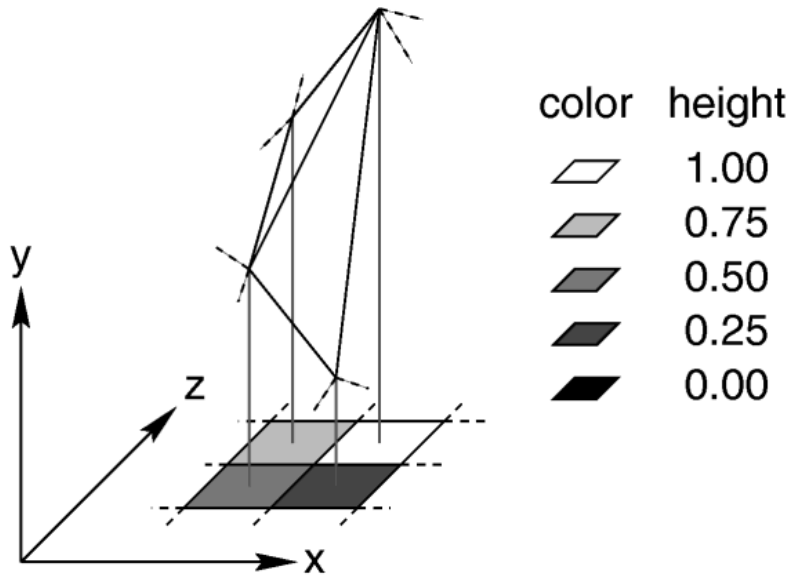
Blobs : points avec champ sphérique (ou cylindrique), maximum au centre et tombant à zéro à une distance égale au rayon. Le lieu des points où le champ est égal au paramètre *threshold* détermine la forme créée par la combinaison de ces champs.

```
blob {
  threshold .65
  sphere {<.5,0,0>, .8, 1 pigment {Blue}}
  sphere {<-.5,0,0>,.8, 1 pigment {Pink}}
  finish {phong 1}
}
```

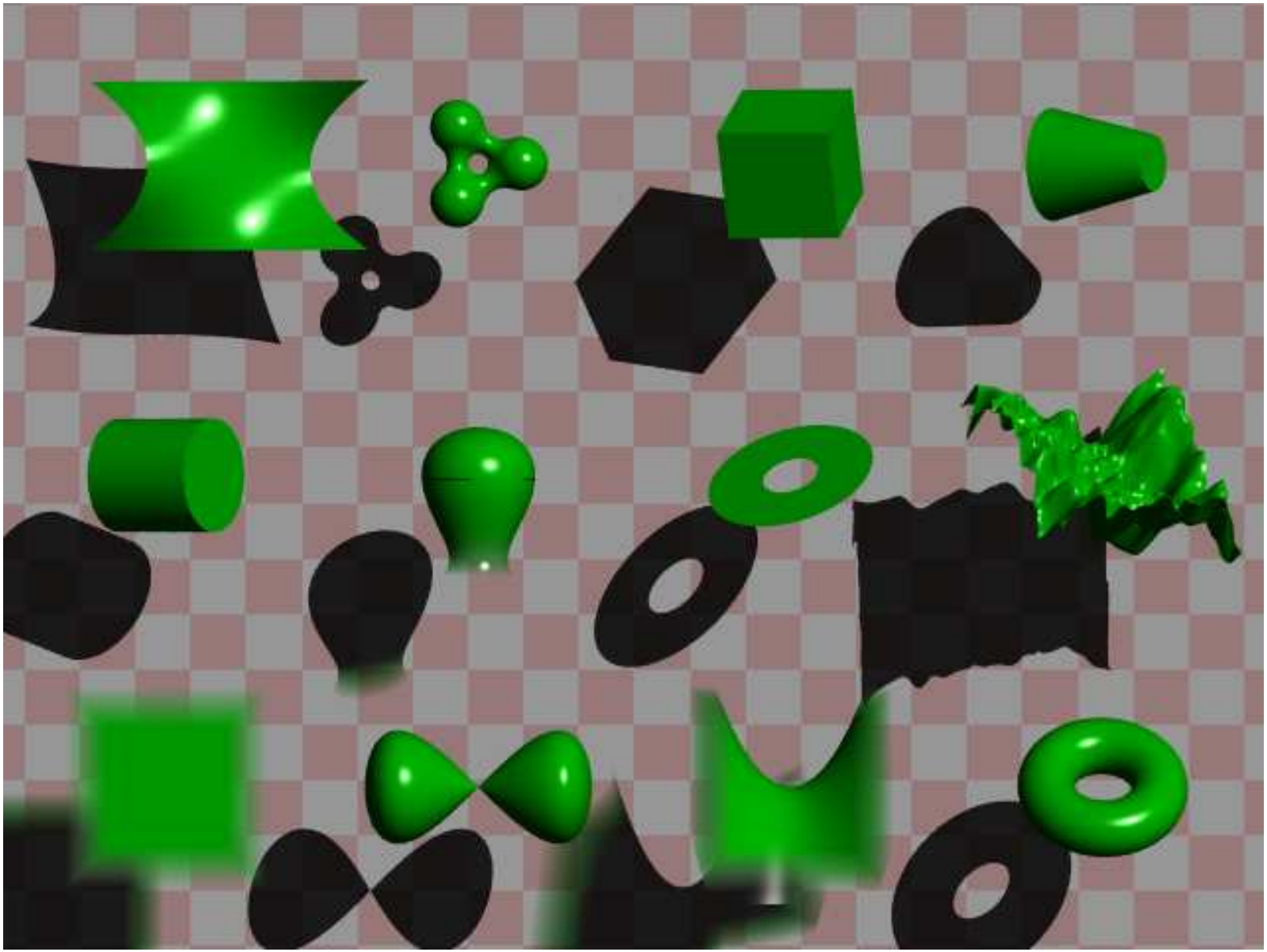


Primitives volumiques

Modèle numérique de terrain (*height_fields*) : à partir d'un fichier d'intensité déterminant l'ordonnée de la surface (2 triangles sont générés par pixel).



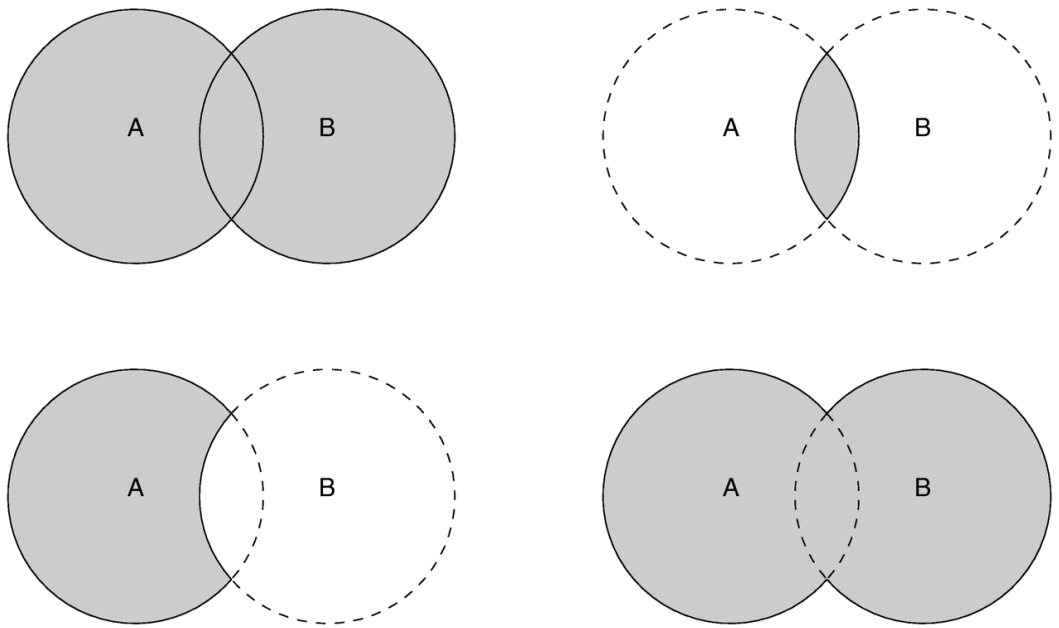
Récapitulatif



CSG

Le modèle CSG (*Constructive Solid Geometry*) : combinaison (booléenne) de primitives solides permettant la création d'objets plus complexes.

Les objets doivent délimiter deux demi-espaces (intérieur/extérieur). Pour les plans, ceux-ci sont définis grâce au vecteur normal.

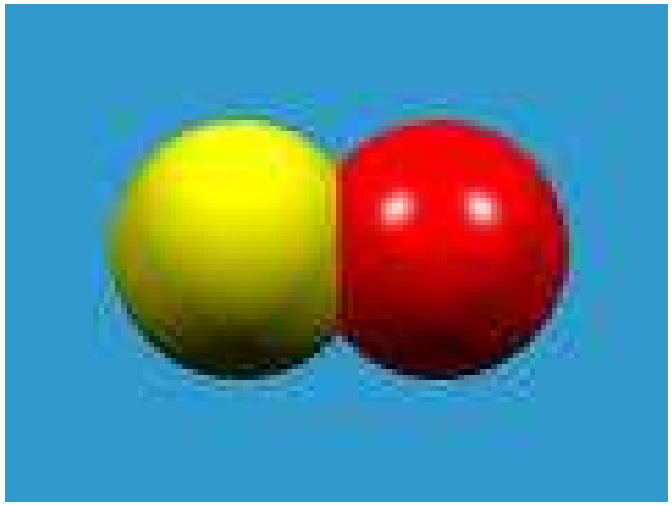


CSG : union

`union  shape/object_1 ... shape/object_n texture`

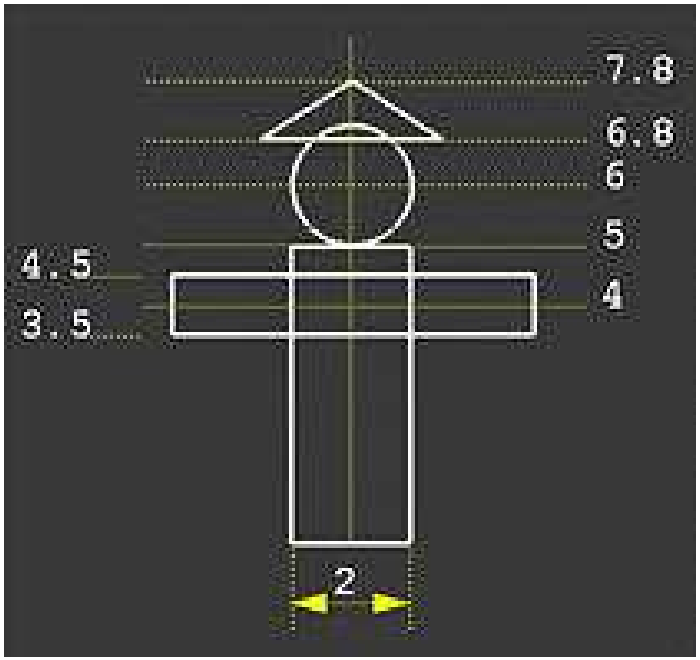
Création d'un unique objet qui peut subir des transformations géométriques. Par contre, chaque composant peut avoir sa propre texture ou prendra celle définie dans l'union si ce n'est pas le cas.

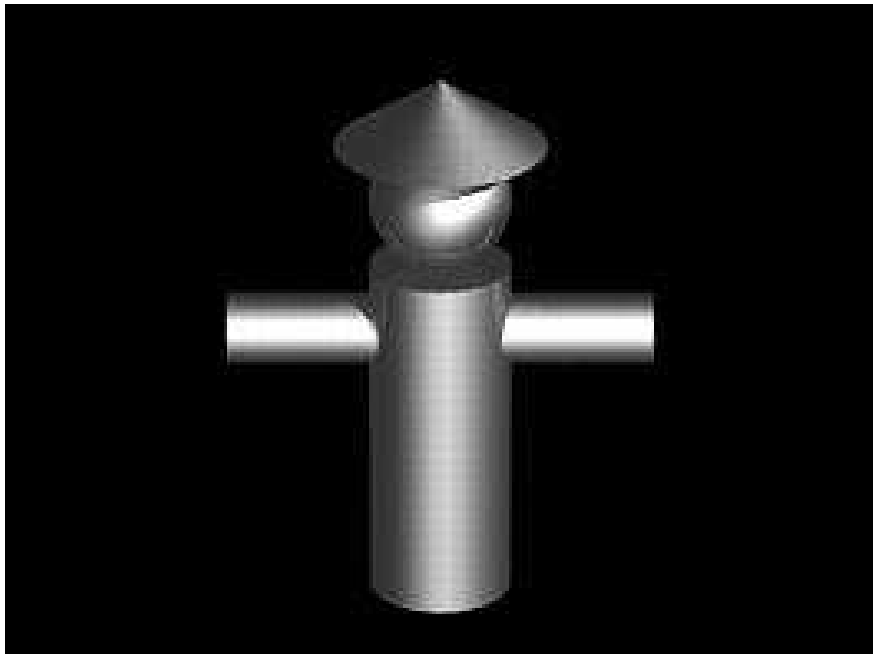
```
#declare TwoSphere =
  union {
    sphere {  x, 1.3  }
    sphere { -x, 1.3 pigment { color Yellow } }
  }

  object {
    TwoSphere
      pigment { color Red }
      finish { phong 1 ambient 0.2 }
  }
```



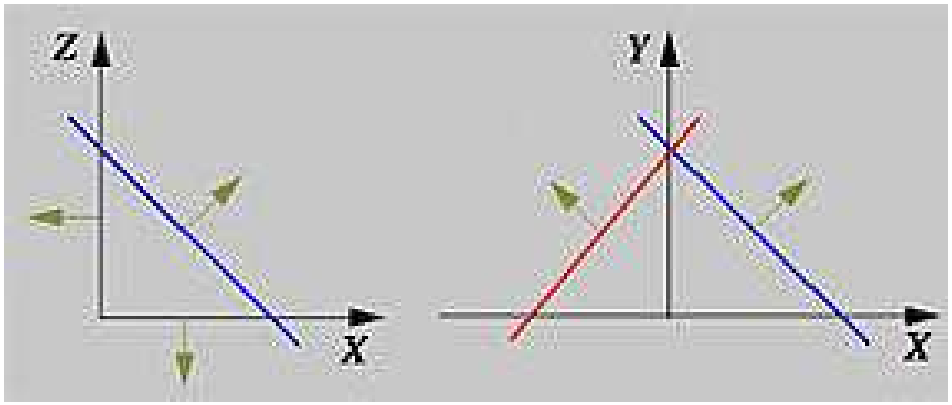
CSG : union

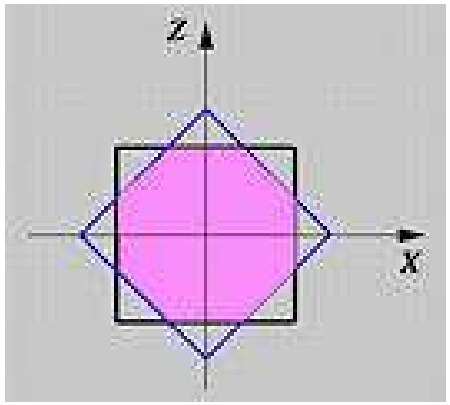
Un exemple, utilisant un cône, une sphère et deux cylindres :



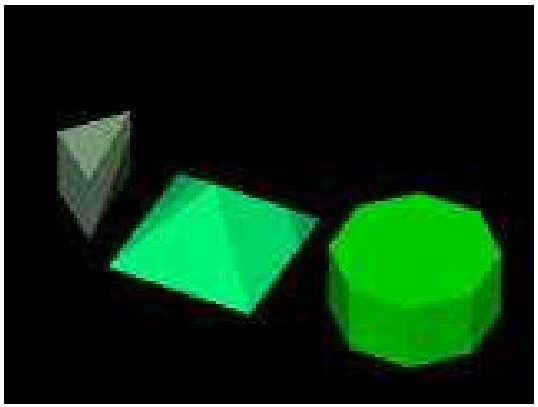


CSG : intersection

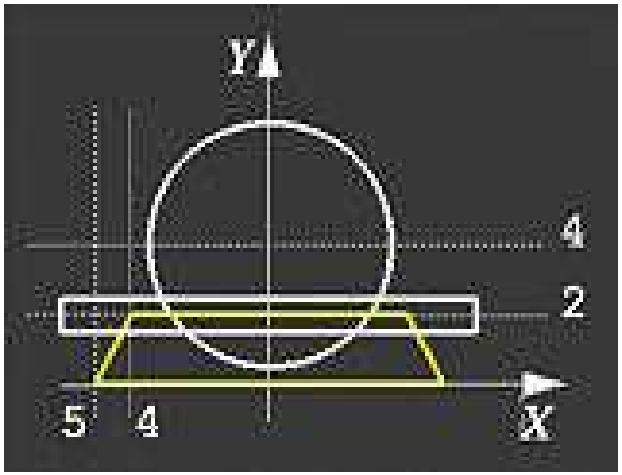





```
#declare Infinite_Block =
  intersection {
    plane { x, 1 }
    plane { x, 1 rotate 90*y }
    plane { x, 1 rotate 180*y }
    plane { x, 1 rotate 270*y }
  }

#declare AngularCylinder =
  intersection {
    object { Infinite_Block }
    object { Infinite_Block rotate 45*y }
    object { BoundingBox }
  }
```



CSG : différence



```
#declare Cone = cone { < 0, 0, 0 >, 5, < 0, 2, 0 >, 4 }
#declare Sphere = sphere { < 0, 4, 0 >, 3.5 }
#declare Cylinder = cylinder { < -6, 2, 0 >, < 6, 2, 0 >, 0.5 }
#declare Tray = difference {
  object { Cone }
  object { Sphere }
  object { Cylinder }
  object { Cylinder rotate 90*y }
}
```

Paramètres des volumes

Boîte englobante (*bounded by*) : la plupart des objets sont automatiquement encapsulés par une boîte englobante (*bounding slabs*).

Il n'en est rien pour les objets infinis ni les objets CSG contenant des primitives infinies. De plus, la boîte englobante n'est pas précise lorsque les opérations mises en jeu sont les intersections et les différences.

Les formes optimisées sont la sphère et la "boîte" mais le volume englobant peut être de n'importe quelle forme.

Autre optimisation : `bounded_by clipped_by`

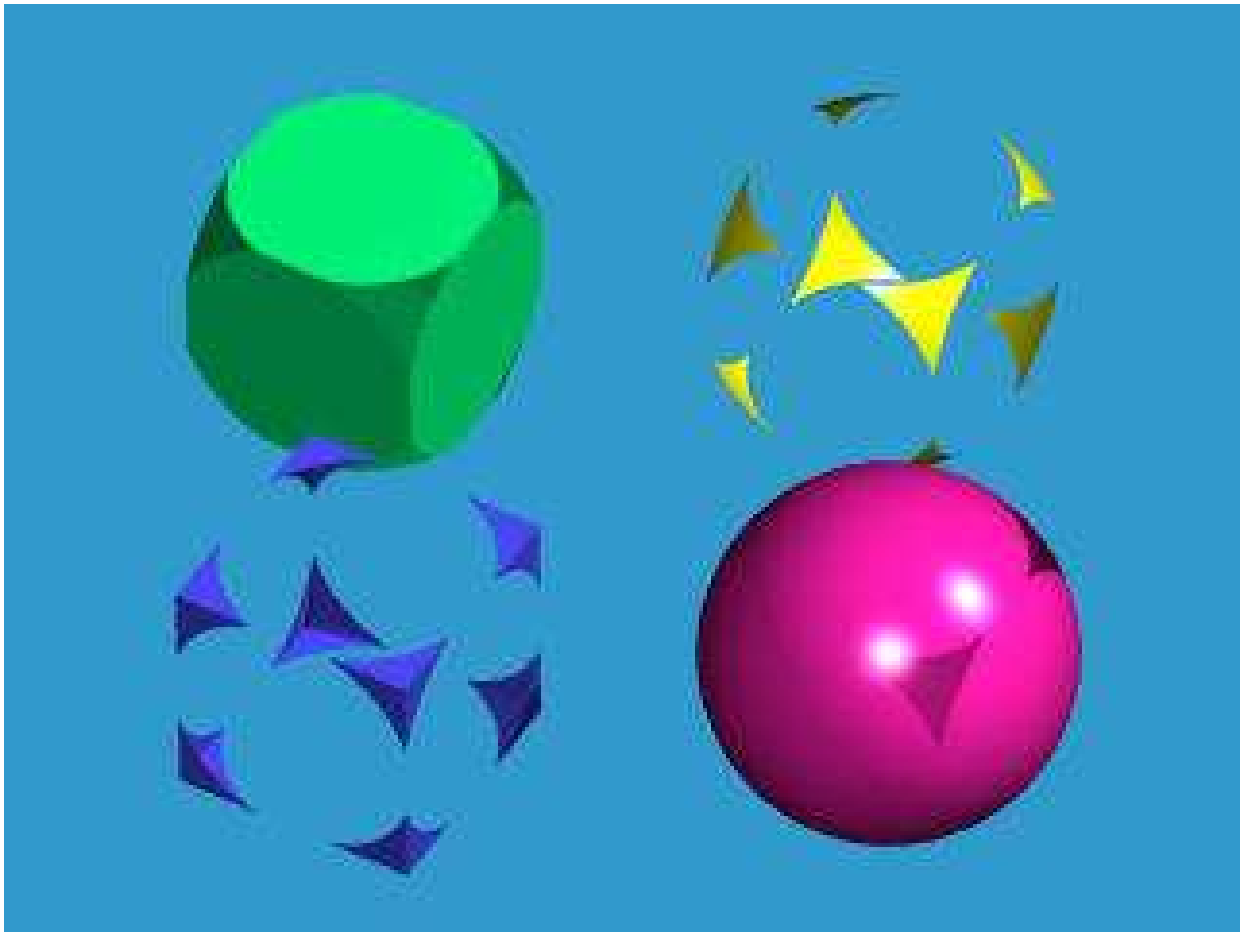
Paramètres des volumes

Forme de découpe (*clipped by*) : définit des parties d'un objet à supprimer dans le but de visualiser sa surface résultante (tout solide est définit par une surface).

Différence par rapport à l'opérateur booléen d'intersection : la surface résultante peut être ouverte.

Il peut être utile d'inverser (*inverse*) l'intérieur de l'extérieur dans ce contexte, pour pouvoir supprimer les parties d'objet qui sont à l'intérieur du solide de découpe.

Volume de découpe



Les textures complexes

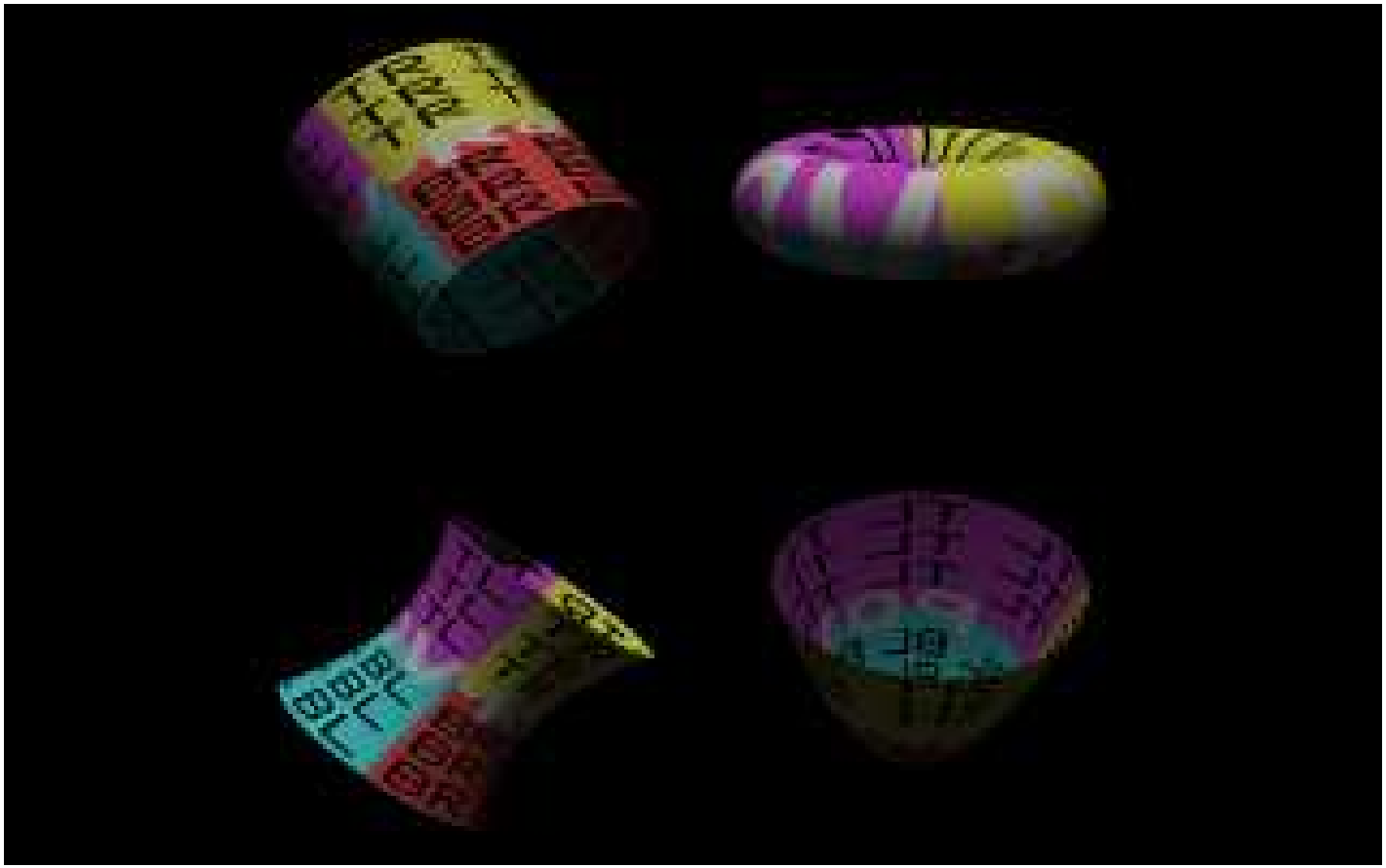
Le plaquage de textures (*image_map*) :

```
pigment  image_map  <type> "<nom>"
```

Types les plus courants : `gif`, `png` et `tga`. Par défaut l'image est projetée sur le plan XY suivant la direction $-Z$ sur un carré de côté 1.

Les options les plus utilisées sont : `map_type` qui définit le type de projection ; `filter` ou `transmit` qui permettent de rendre semi-transparentes certaines couleurs.

Image map



Les textures complexes

Les normales (2) : `bump_map` permet d'utiliser un fichier d'images qui définit la perturbation de la normale en un point via sa couleur associée dans le fichier.

Syntaxe : `bump_map  <type> "<nom>"` .

Un paramètre utile : `bump_size`, qui peut aller de 0.1 jusqu'à 5.0.

Caractéristiques internes

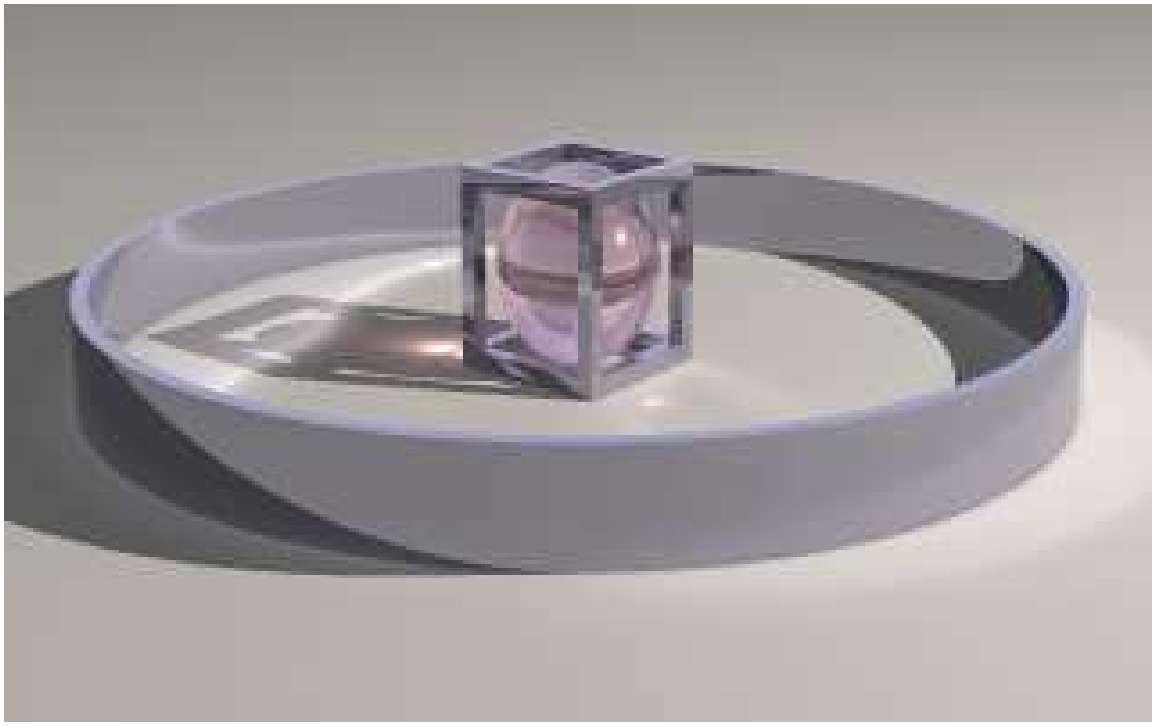
Depuis la version 3.1, certains paramètres de texture ont été regroupé dans la structure `interior`.

Celle-ci (resp. `texture`) décrit les propriétés “internes” (resp. de la surface) de la matière, nécessaire dans le cadre d'objets (semi) transparents. Ces paramètres sont : ior, caustics, dispersion, fade et MEDIA.

Le tout, `texture` et `interior`, est encapsulé dans la structure `material`

Paramètres de “interior”

`caustics` (*refractive caustics*) : applats de lumière dans les ombres portées lors du passage de la lumière à travers des objets translucides.



Paramètres de “interior”

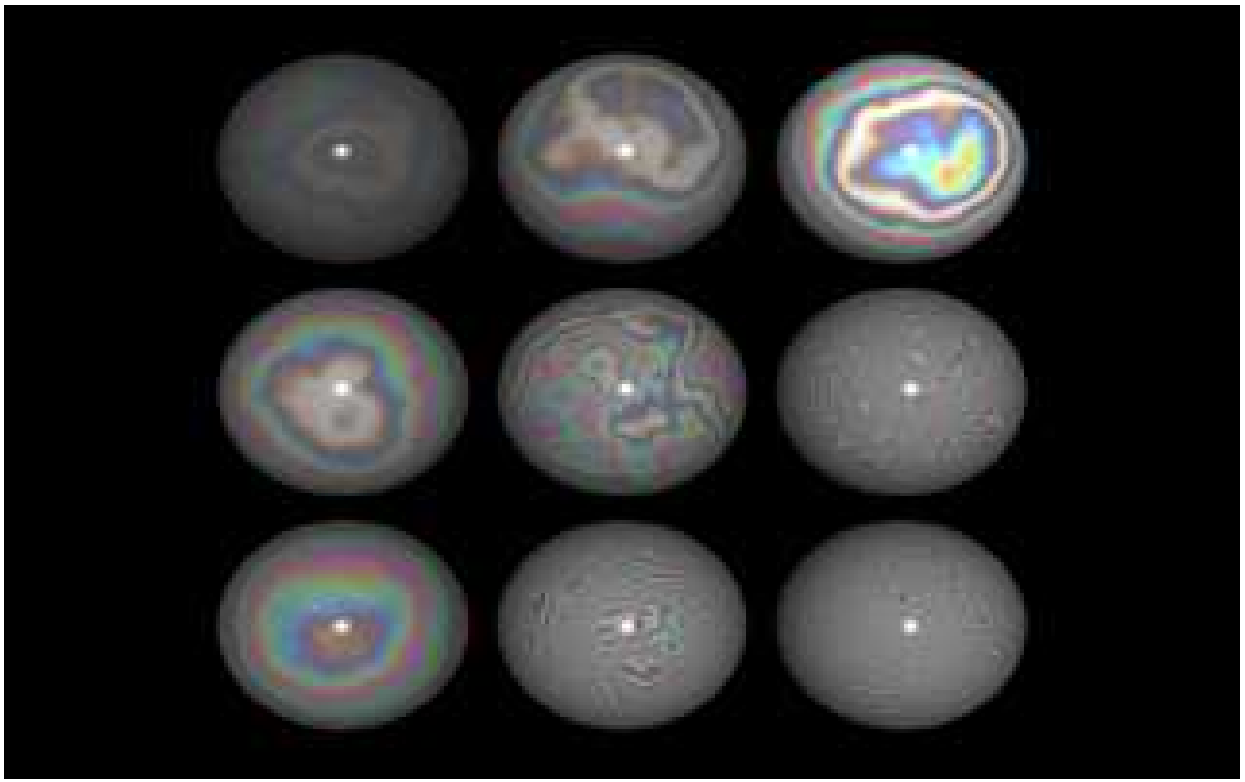
`fade` : attenuation de l'intensité lumineuse à travers un objet transparent présentant des impuretés :

$$\text{attenuation} = \frac{1}{1 + (\frac{d}{\text{fade\_distance}})^{\text{fade\_power}}}$$

Iridescence

C'est l'effet causé par exemple par de l'huile à la surface de l'eau.

Syntaxe : `irid  <quantite> thickness <t1> turbulence <t2>`



Les textures spéciales

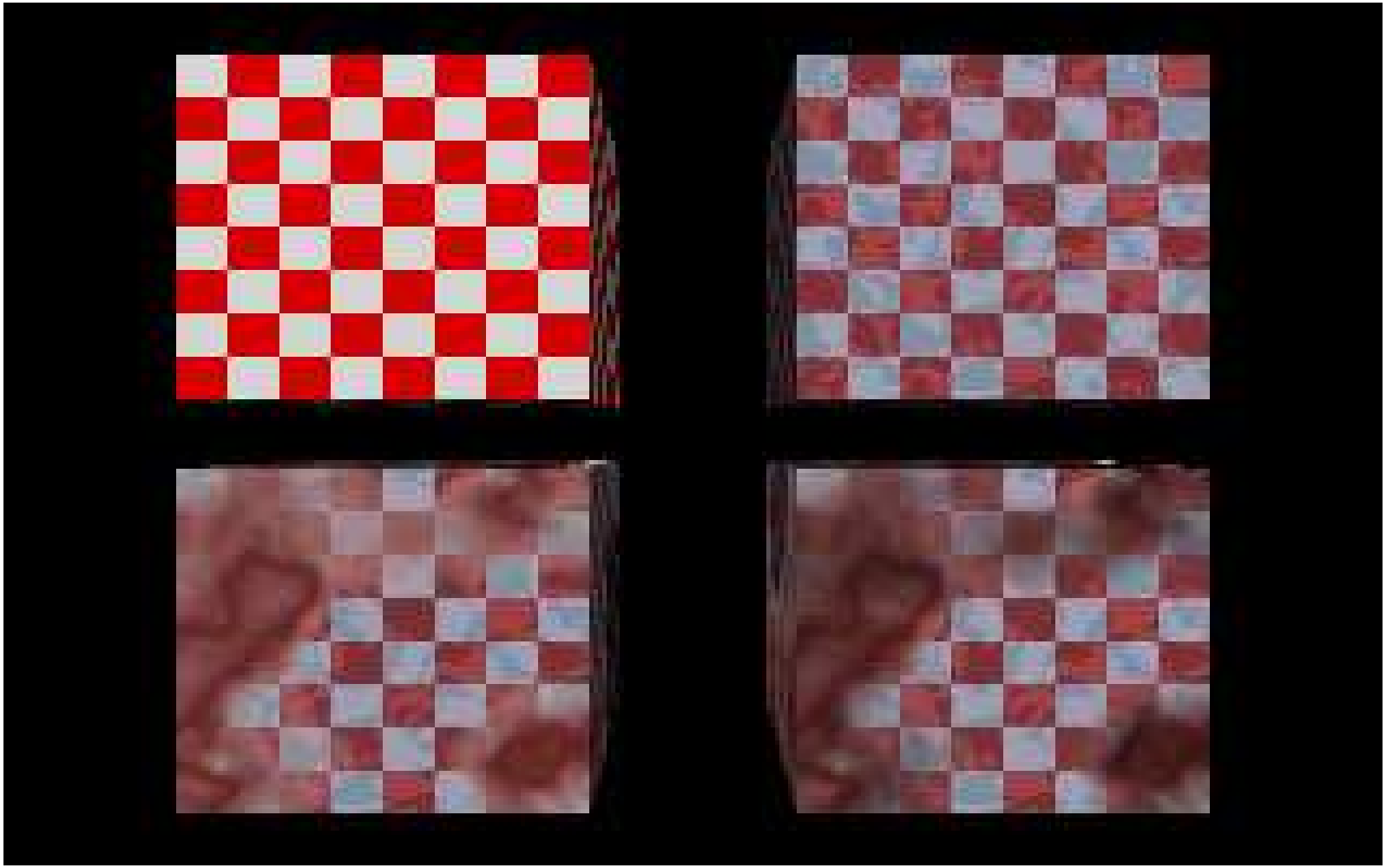
Deux types de textures : PNF (pigment, normale et finition unique) et spéciales (combinaisons de textures) :

- *layered texture* : plusieurs textures, partiellement transparentes et superposées (rem : `normal`).
- `texture_map` : fonction de répartition d'indices de réflexions différents le long d'une surface. Syntaxe : `texture <fonction>`

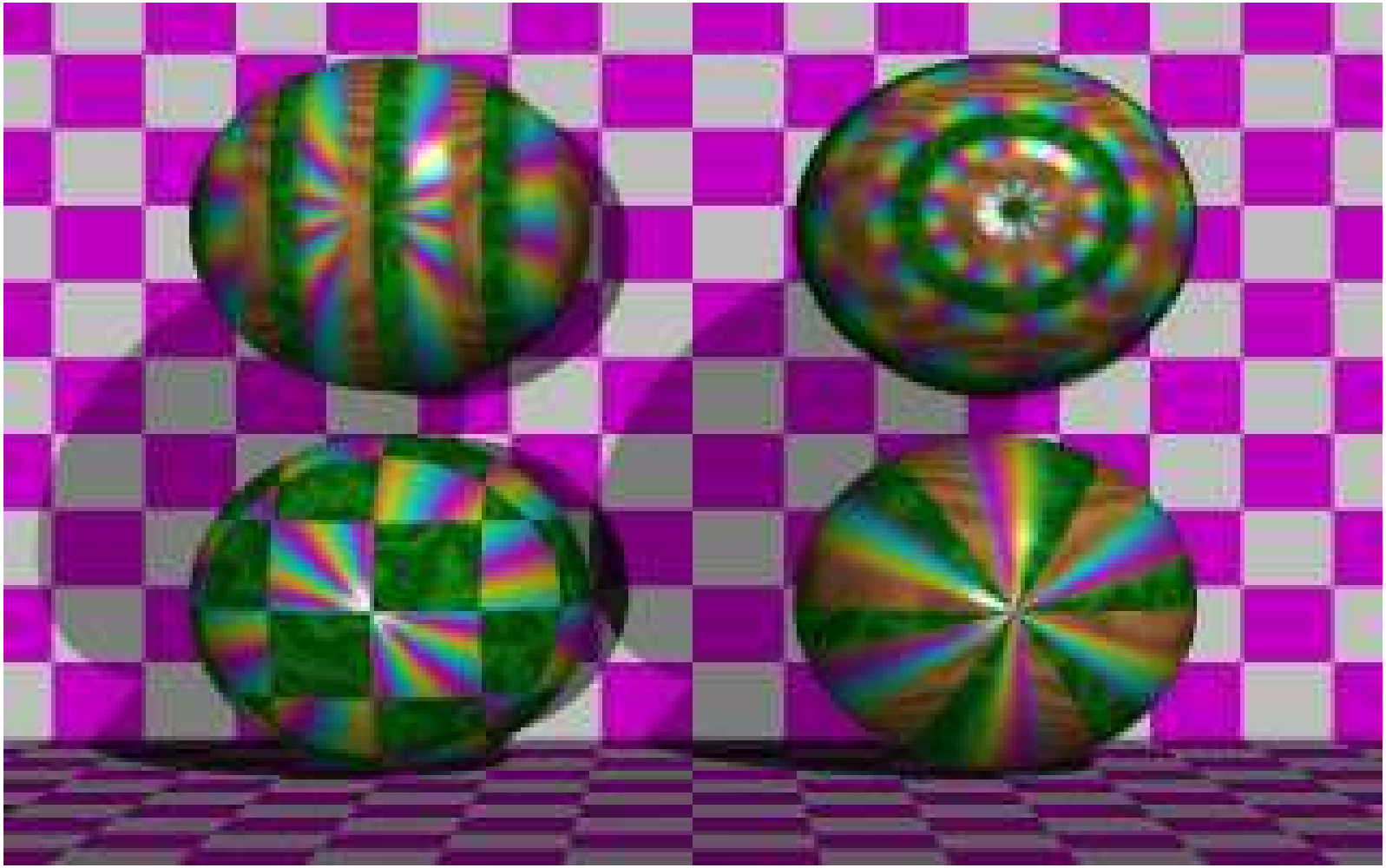
  `texture_map [0.0` t_1`] ... [1.0` t_n`]`
- `material_map` : textures multiples, fichier image définissant les textures à projeter sur un objet. Syntaxe : `texture material_map`

  `<file_type> <modifier> <textures_list>`

Layered texture



Texture map



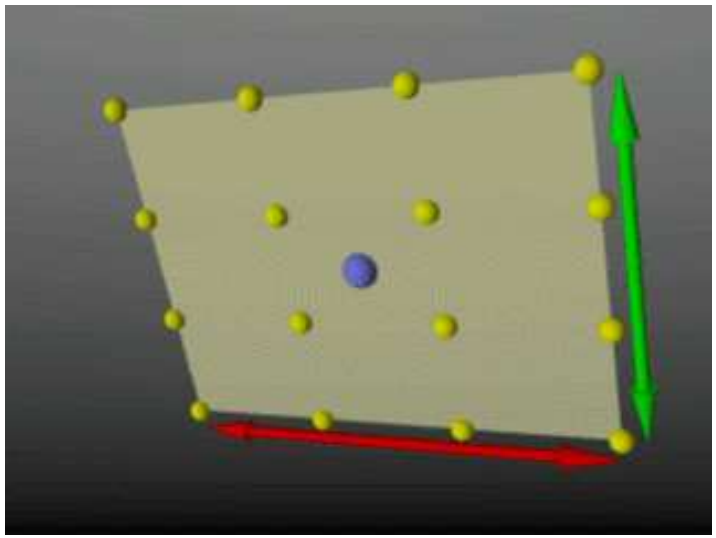
Material map

Modèles d'éclairement

Aire lumineuse (*area_light*) : source de lumière "étendue" pour des effets d'ombres douces :

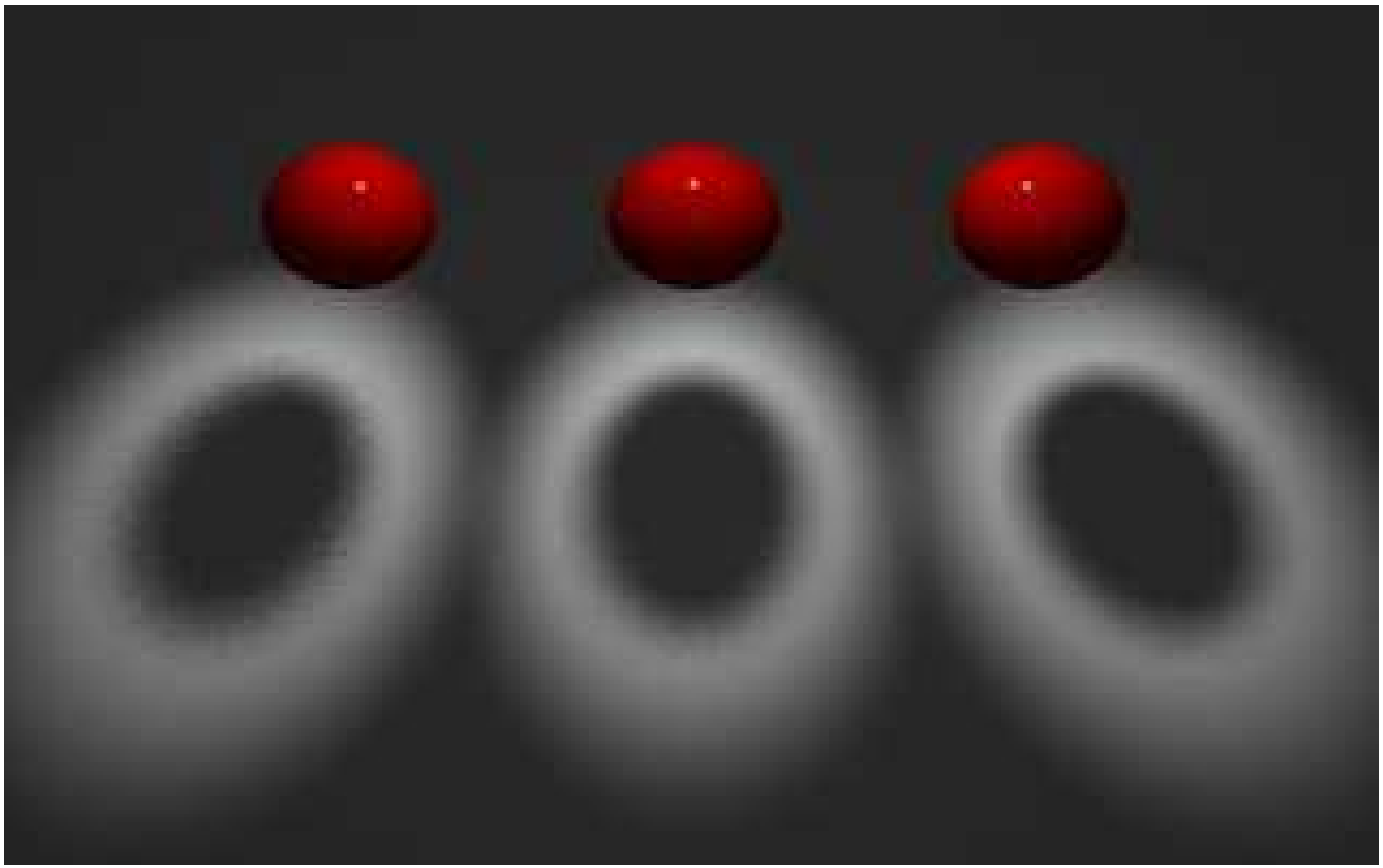
```
light_source { ... area_light <axe1>, <axe2>, <taille1>, <taille2>
               adaptative <entier> jitter <reel> }
```

Paramètres : *adaptative* pour un pré-calcul des rayons d'ombre ; *jitter* pour un lancé avec perturbation.



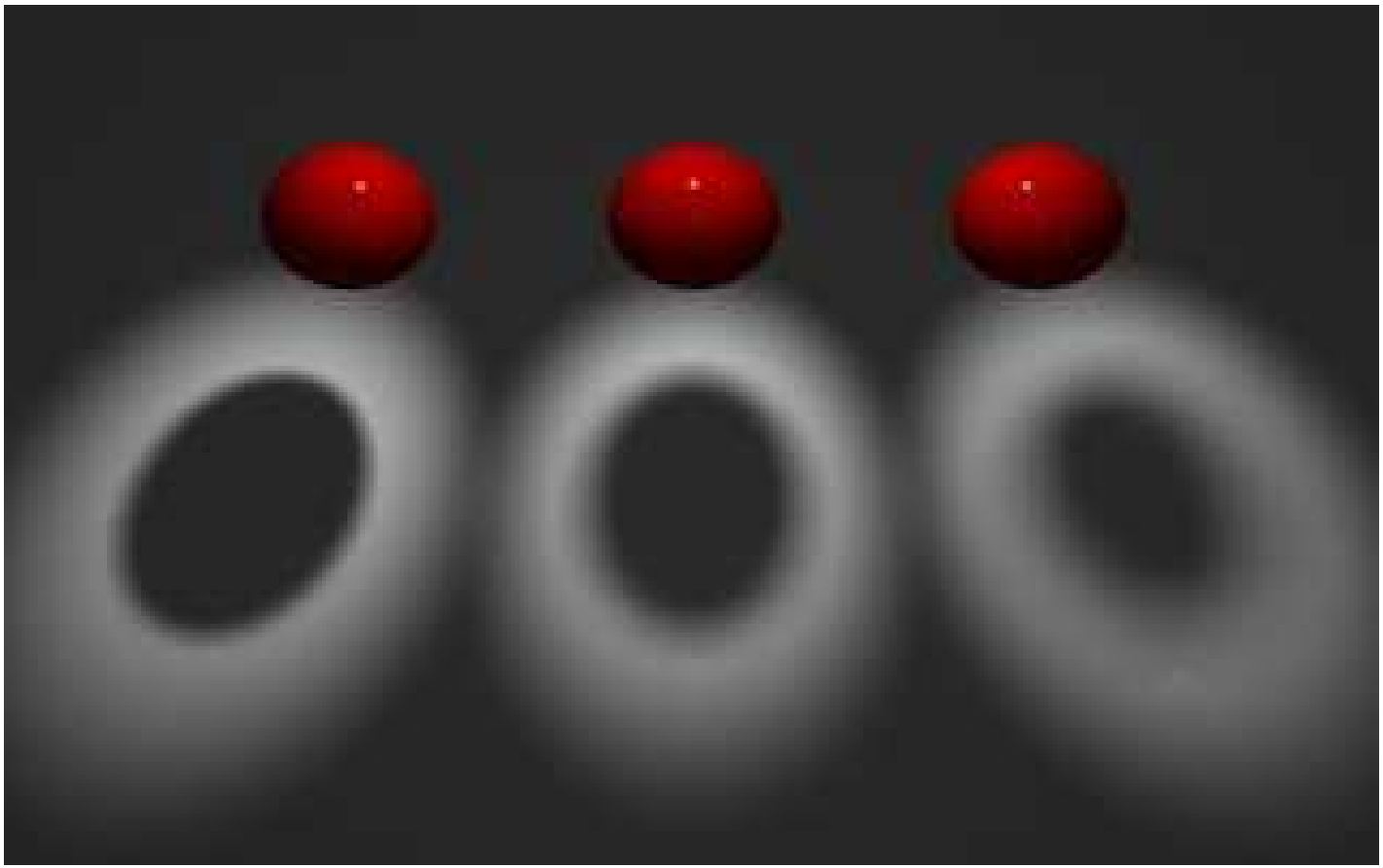
Aire lumineuse

Taille de la grille (3×3, 5×5, 17×17) :



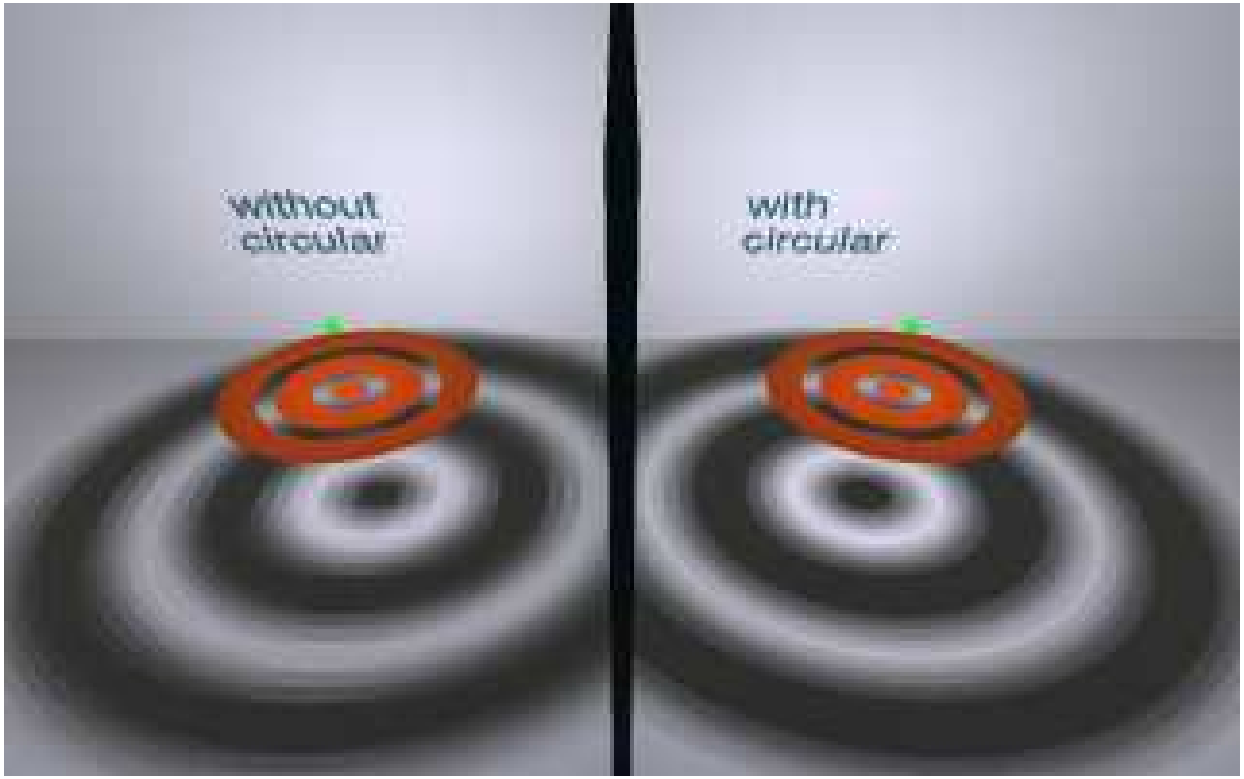
Aire lumineuse

Taille de l'aire (4×4, 8×8, 16×16) avec grille 17×17 :



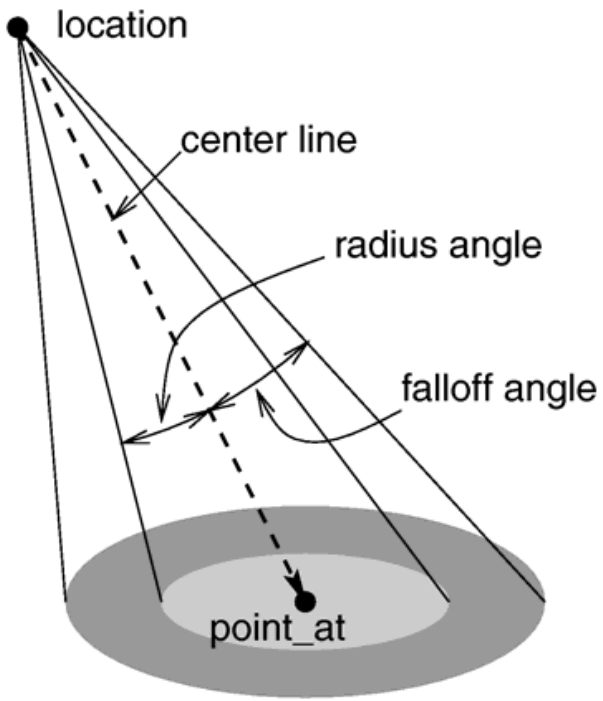
Modèles d'éclairement

Le cas des ombres circulaires :



Modèles d'éclairement

Cône de lumière (*spotlight*) ; paramètre : *falloff* définition de la frontière d'intensité lumineuse nulle.



Effets atmosphériques

Autrefois dans `atmosphere`, à présent dans `media` (au même titre que `halo`) : modèle particulaire.

Echantillonnage de la densité des particules le long de la trajectoire du rayon de vue. Trois types : dans un objet (`interior`), dans un milieu atmosphérique et local.

Trois types d'interaction pour les particules :

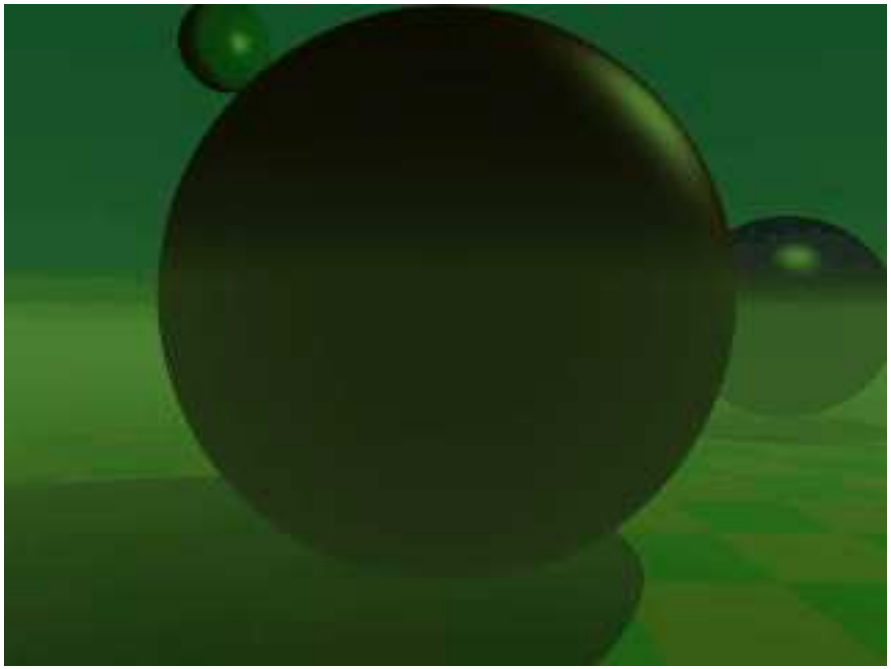
- absorption : couleur absorbée vue à travers le media,
- émission : couleur "émise" (cf. `ambient`),
- dispersion (*scattering*) : cf. réflexion diffuse.

Effets atmosphériques

Effets atmosphériques

Brouillard : `fog fog_type <t> distance <d> colour <c>`. S'applique à l'ensemble de la scène et ajoute sa couleur à celles des objets. Ratio de la couleur du brouillard sur celui de l'objet :

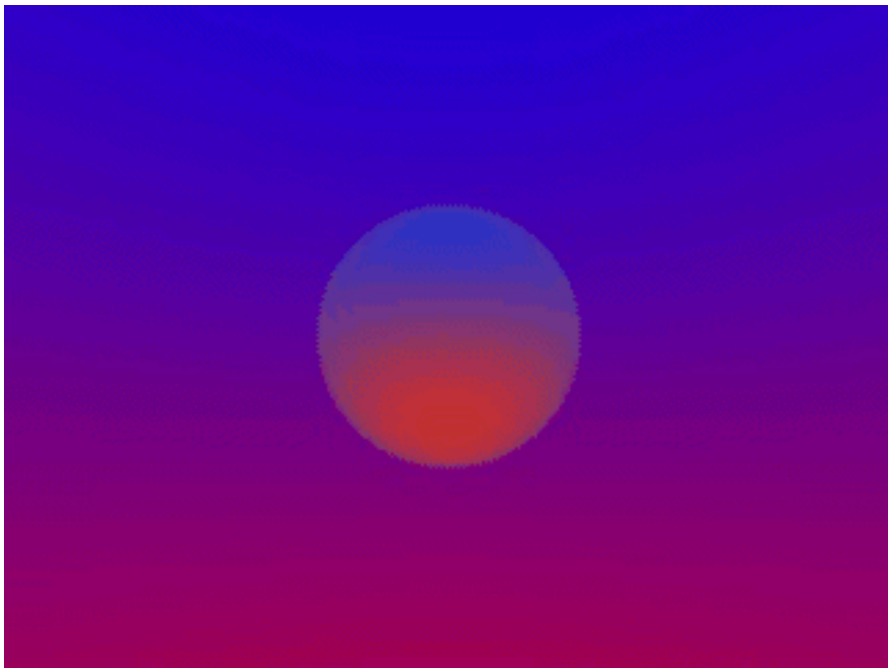
$$1 - e^{-\frac{d_o}{d_f}}$$



Objets naturels

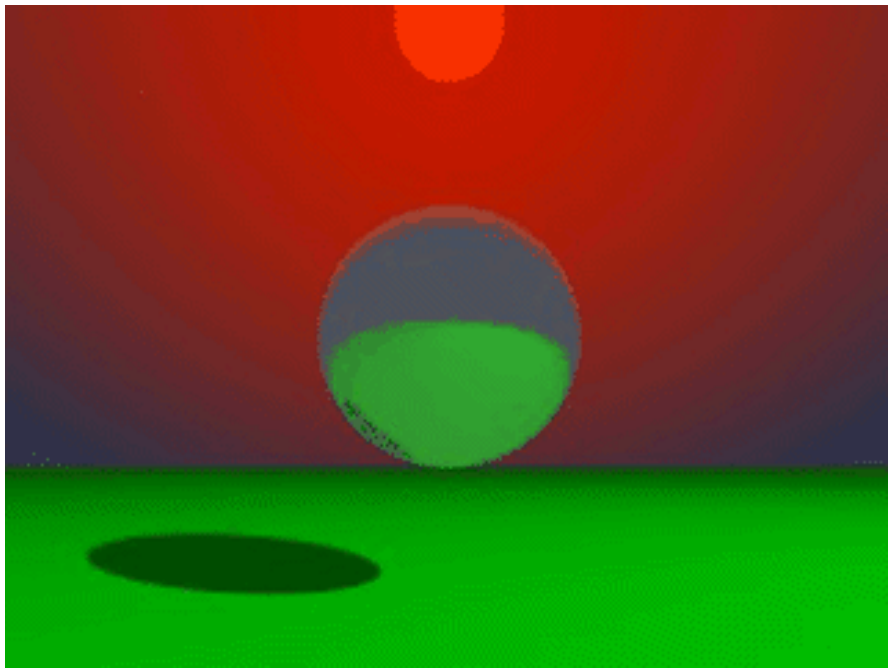
Ciel : `sky_sphere { pigment { <p> ... }`, paramètres : `gradient`; soleil `rotate -135*x`; nuages bozo avec composantes alpha à 1.

```
sky_sphere {
    pigment {
      gradient y
      color_map {
        [0 color Red]
        [1 color Blue]
      }
      scale 2
      translate -1
    }
  }
```



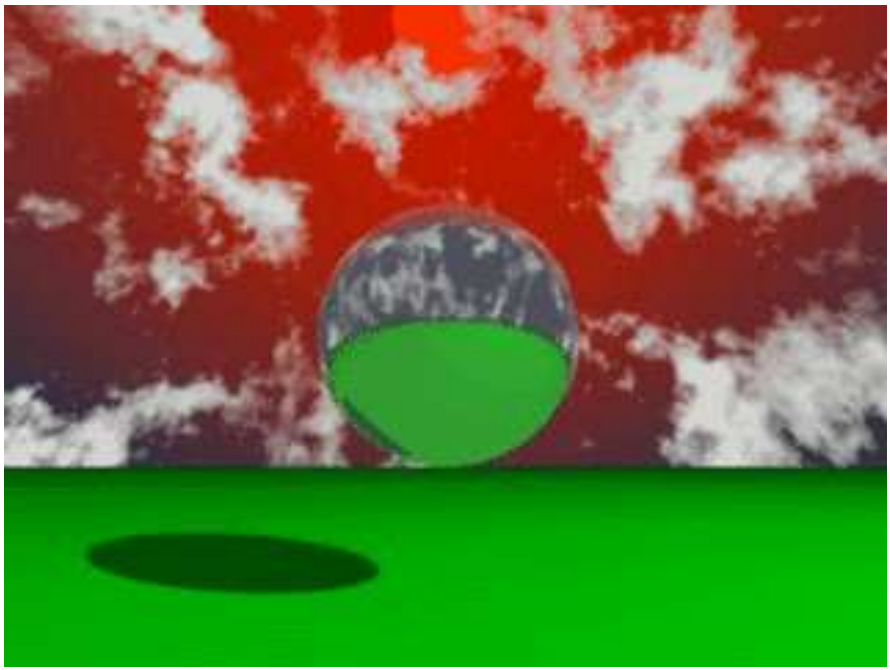
Le ciel (2)

```
sky_sphere {
  pigment {
    gradient y
    color_map {
      [0.000 0.002 color rgb <1.0, 0.2, 0.0>
                   color rgb <1.0, 0.2, 0.0>]
      [0.002 0.200 color rgb <0.8, 0.1, 0.0>
                   color rgb <0.2, 0.2, 0.3>]
    }
    scale 2
    translate -1
  }
  rotate -135*x
}
```



Le ciel (3)

```
sky_sphere {
  pigment {
  ...
  }
  pigment {
    bozo
    turbulence 0.65
    octaves 6
    omega 0.7
    lambda 2
    color_map {
      [0.0 0.1 color rgb <0.85, 0.85, 0.85>
               color rgb <0.75, 0.75, 0.75>]
      [0.1 0.5 color rgb <0.75, 0.75, 0.75>
               color rgbt <1, 1, 1, 1>]
      [0.5 1.0 color rgbt <1, 1, 1, 1>
               color rgbt <1, 1, 1, 1>]
    }
    scale <0.2, 0.5, 0.2>
  }
  rotate -135*x
}
```



Objets naturels

Arc en ciel : arc circulaire, de consistance équivalente à celle du brouillard. Syntaxe : `rainbow direction <d> angle <a> width <w> distance <d> color_map <c>`