Documentation MLpov 0.81

Table of Contents

Informations

MLpov est une version non officielle de POV-Ray 3.5 [http://www.povray.org] (surtout un prétexte pour aller voir

comment POV-ray marchait de l'intérieur :) N'hésitez pas à me faire part des bugs (patchs très peu testés), de vos remarques, suggestions, ou magnifiques images qui utiliseraient un de ces patchs, sur le newsgroup francophone de POV-ray.

Pour utiliser les nouvelles fonctionnalités vous devez inclure

```
#version unofficial mlpov 0.8;
```
dans votre scène.

AOI (Angle of Incidence)

Description

Un nouveau pattern, "angle d'incidence". Ce pattern retourne une valeur comprise entre 0 et 1 suivant l'angle entre la normale (normale perturbée) d'un objet et, au choix, un vecteur "point fixe - point d'intersection" ou le rayon de vue.

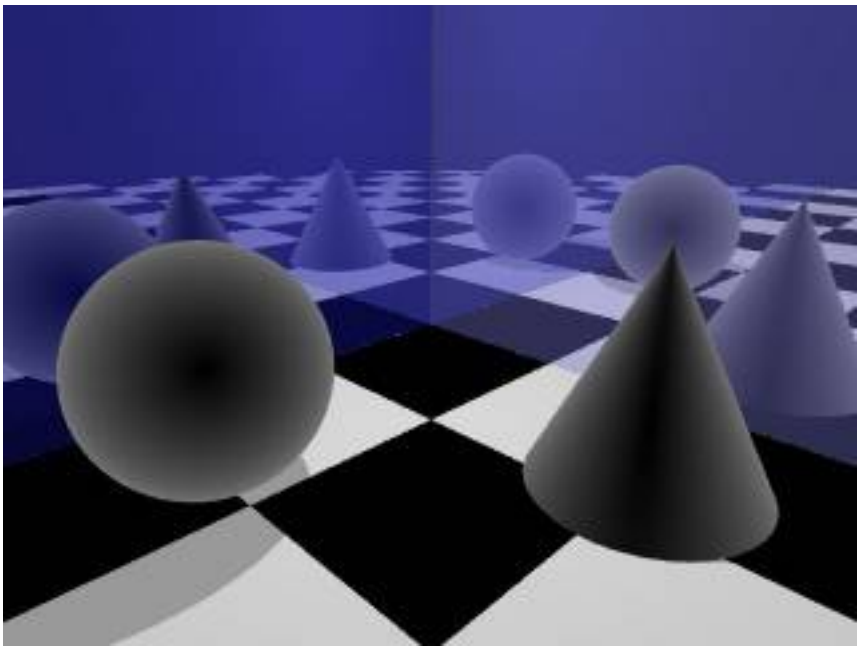
Syntaxe

```
pigment { aoi [point] .... }
```

si aucun *point* n'est spécifié le pattern est calculé par rapport au rayon incident.

Warning
Si le rayon incident est utilisé les valeurs de retour sont comprises entre 0 et 0.5 !

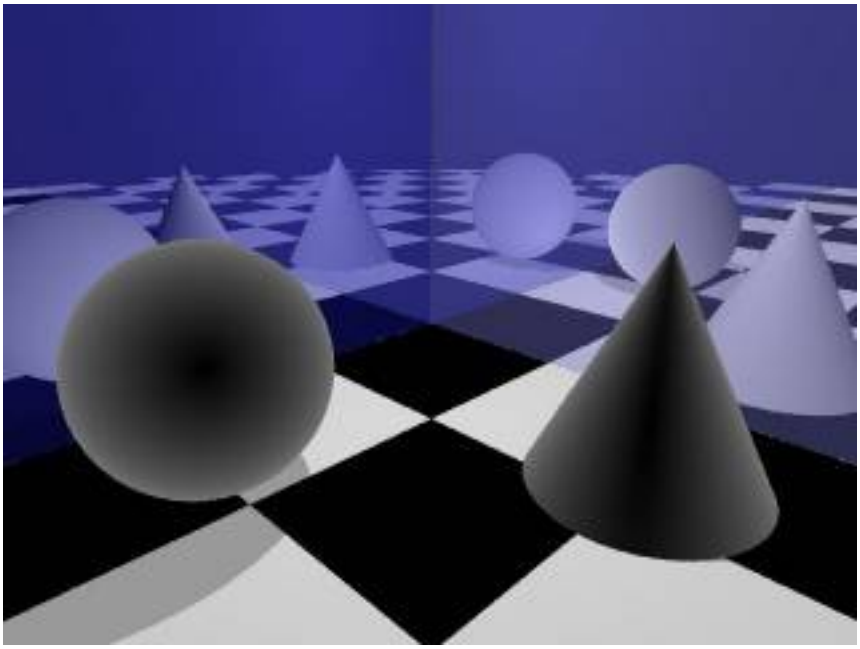
Exemples



Pattern aoi sans argument

```
pigment { aoi }
```

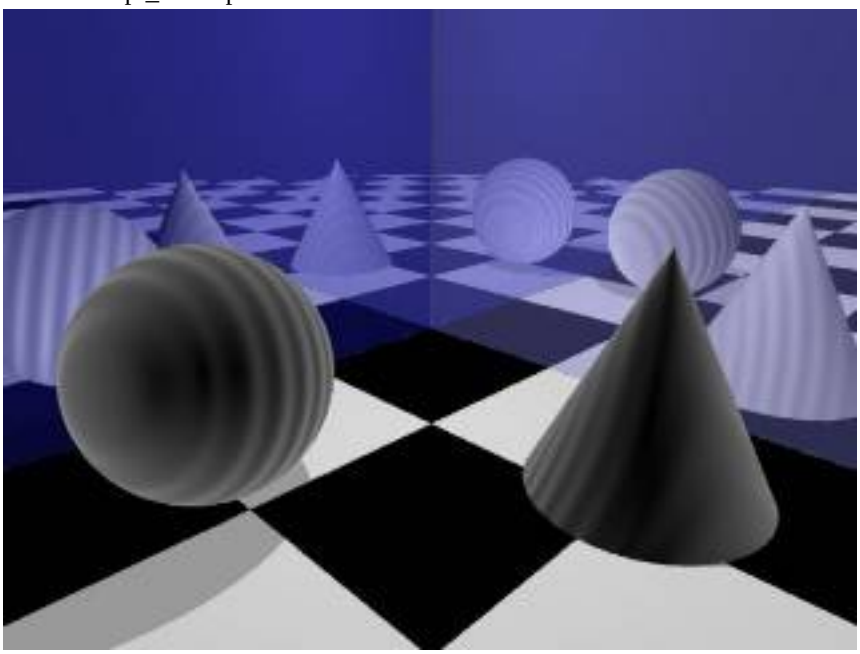
Pas de *point* précisé, le pattern utilise le rayon incident.



Pattern aoi avec point = position caméra

```
pigment { aoi pt_cam }
```

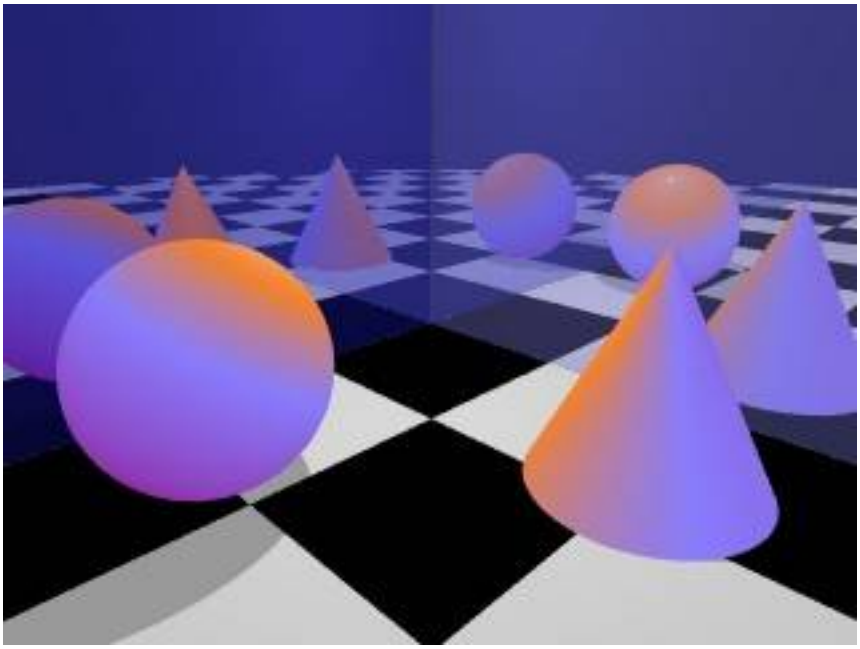
pt_cam est le point ou est placée la caméra. L'angle est donc calculé entre le vecteur normal à la surface de l'objet et le vecteur "pt_cam - point d'intersection".



Pattern aoi avec normale perturbée

```
pigment { aoi pt_cam } normal { ripples .6 }
```

Le pattern prend en compte la normale perturbée.



Pattern aoi avec une color_map

```
pigment { aoi pt_lum color_map { ... } }
```

Un exemple avec un autre point pour le calcul et une color_map.

Program

Look for #ifdef AOI_PATCH in sources

A new vector (incident ray direction) was added to the intersection struct (istk_entry). It is assigned at the beginning of Determine_Apparent_Colour.

Possible improvements/problems

- An option could allow to return "cos(angle)" instead of "angle"

Projection

Description

Un nouveau pattern. Hmm, pas facile à décrire, donc voilà l'algorithme de calcul : à partir du point à évaluer un rayon est lancé (soit dans une direction donnée, soit vers un point, soit suivant la normale de l'objet) et le pattern retourne 1 si ce rayon rencontre un objet fourni, 0 sinon.

Syntaxe

```
pigment { projection objet,pointv [blur blur_intensité,nb_samples] .... }
pigment { projection parallel objet,dirv [blur blur_intensité,nb_samples] .... }
pigment { projection normal objet [blur blur_intensité,nb_samples] .... }
```

si le mot clé *parallel* est utilisé le pattern prend *dirv* comme direction de lancé de rayon (Cf. exemple avec parallel), sinon *pointv* est le point vers lequel le rayon est envoyé (Cf. exemple avec pointv) . Si le mot clé *normal* est utilisé, la normale de l'objet est utilisée comme direction (Cf. exemple avec normal). *objet* est l'objet pour le test d'intersection.

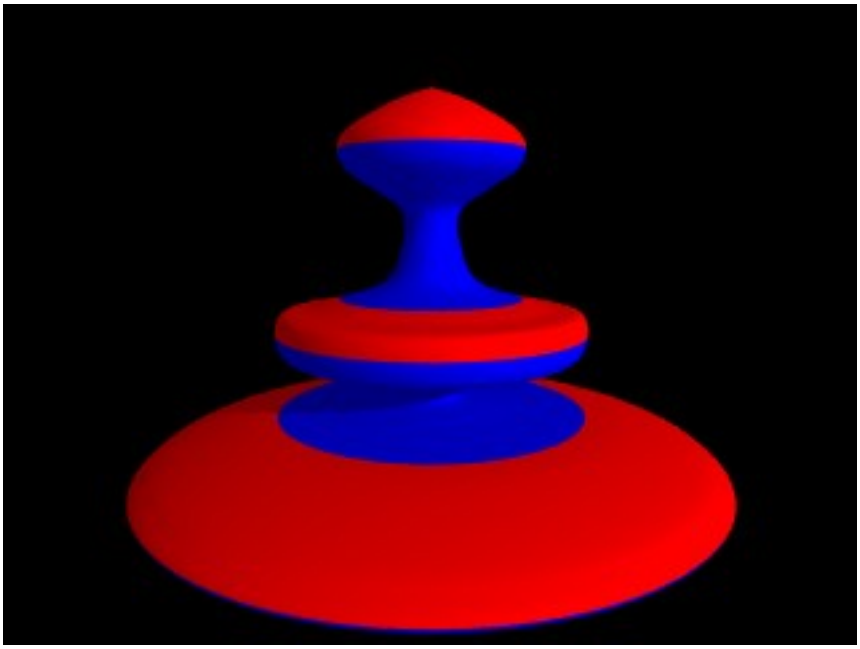
Avec l'option *blur* : *nb_samples* rayons sont envoyés, plus ou moins modifiés par rapport au rayon normal suivant *blur_intensité*, et le pattern donne la valeur moyenne du nombre d'intersections avec l'objet.

> **Note**
> Ce pattern peut être utilisé dans texture_map etc.

Exemples

Avec parallel

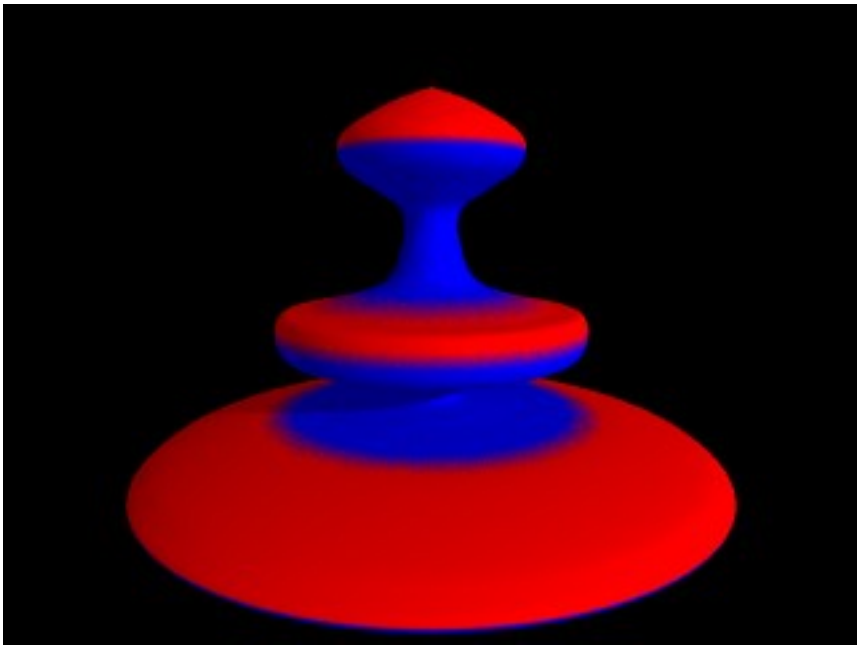
Un exemple en utilisant l'objet à texturer comme object pour le pattern projection.



Pattern projection

```
#declare obj = object {  .. }

object
{
  obj
  pigment { projection parallel obj,<0,1,0>
            color_map { [0 color rgb <1,0,0>][1 color rgb <0,0,1>] } }
}
```

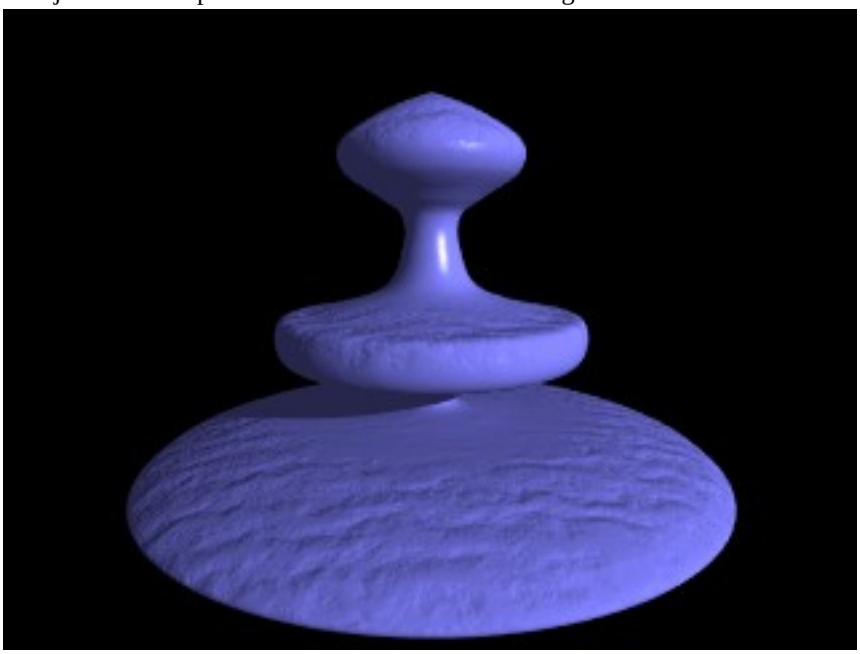
Rayons parallèles envoyés suivant l'axe y. Si le rayon touche l'objet couleur bleue, sinon rouge.



Pattern projection avec blur

```
object
{
  obj
  pigment { projection parallel obj,<0,1,0> blur 0.2,10
            color_map { [0 color rgb <1,0,0>][1 color rgb <0,0,1>] } }
}
```

On ajoute du blur pour lisser la frontière des deux régions.



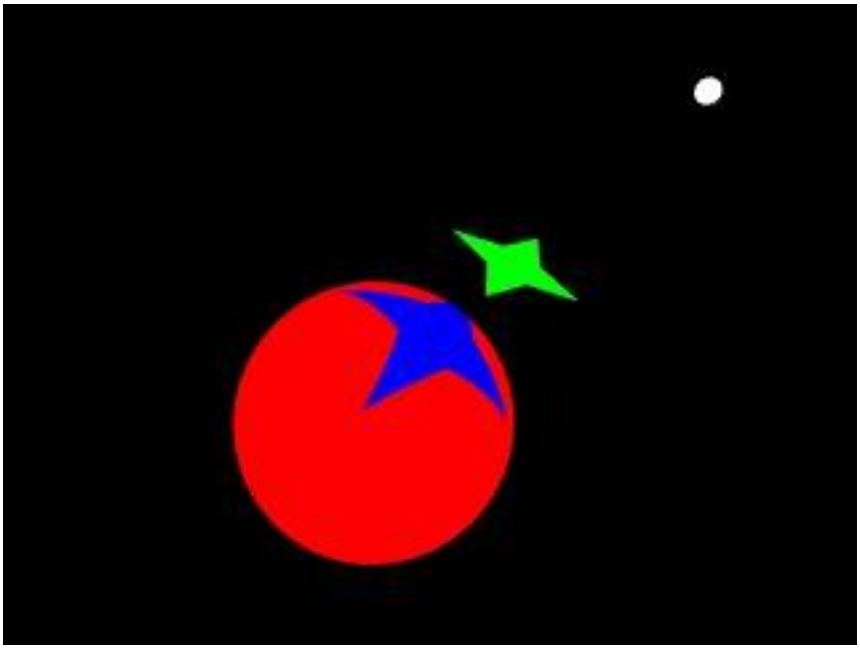
Pattern projection avec texture_map

```
object
```

```
{
  obj
  texture
  {
    projection parallel obj,<0,1,0> blur .2,10
    texture_map
    {
      [0 txt_abime]
      [1 txt_ok]
    }
  }
}
```

La même chose avec une texture map et deux textures suivant la région.

Avec pointv

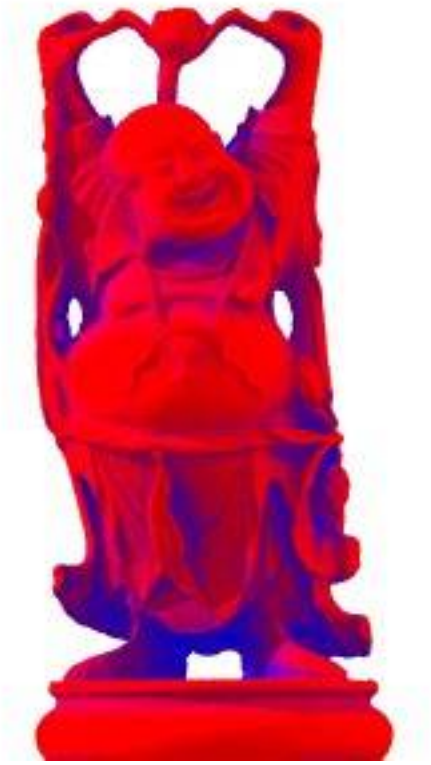


Pattern projection avec un point

```
#declare etoile = object { .. }
#declare pt_lum = <...>;

sphere
{
  0,1
  pigment { projection etoile,pt_lum
            color_map { [0 color rgb <1,0,0>][1 color rgb <0,0,1>] } }
}
```

Avec normal

Le mot clé *normal*, couplé à un *blur 1*, permet de construire une sorte de pattern d'accessibilité.



Pattern projection avec normal

```
object
{
  buddha
  texture
  {
    projection normal buddha blur 1,60
    texture_map { [0 TRouge] [1 TBleu] }
  }
}
```

Et la même chose avec deux textures adaptées :

Pattern projection avec normal et textures

Program

Look for #ifdef PROJECTION_PATCH in sources

The blur uses the "trig method" to pick a random point on the surface of a unit sphere.

Possible improvements/problems

- Faster blurring with an adaptive number of samples
- A *`distance`* keyword to specify a maximum for the distance of intersection (if the intersection occurs more far than distance then it does not count)
- Not only projection of an object but projection of its pigment ?

Shadow_pigment

Description

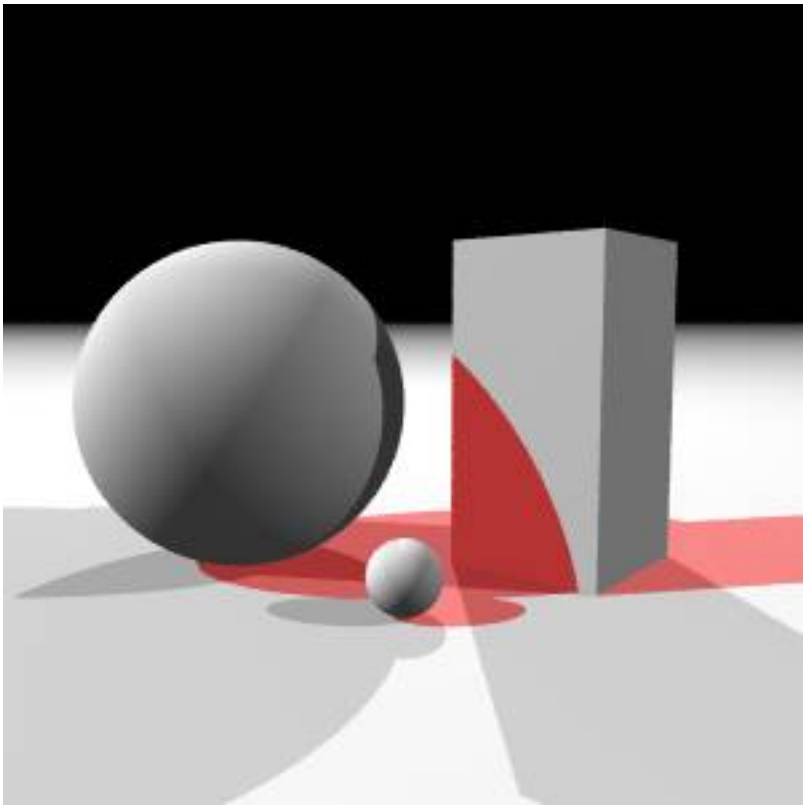
Ce patch ajoute un paramètre à une lumière : un pigment pour les ombres qu'elle projette. (Comme tout les autres patchs ici c'est du bricolage, et en particulier fonctionnement non garanti si utilisation avec transparence, photons et autres bizarreries :).

Syntaxe

```
light_source { ... shadow_pigment { pigment_ombre } ... }
```

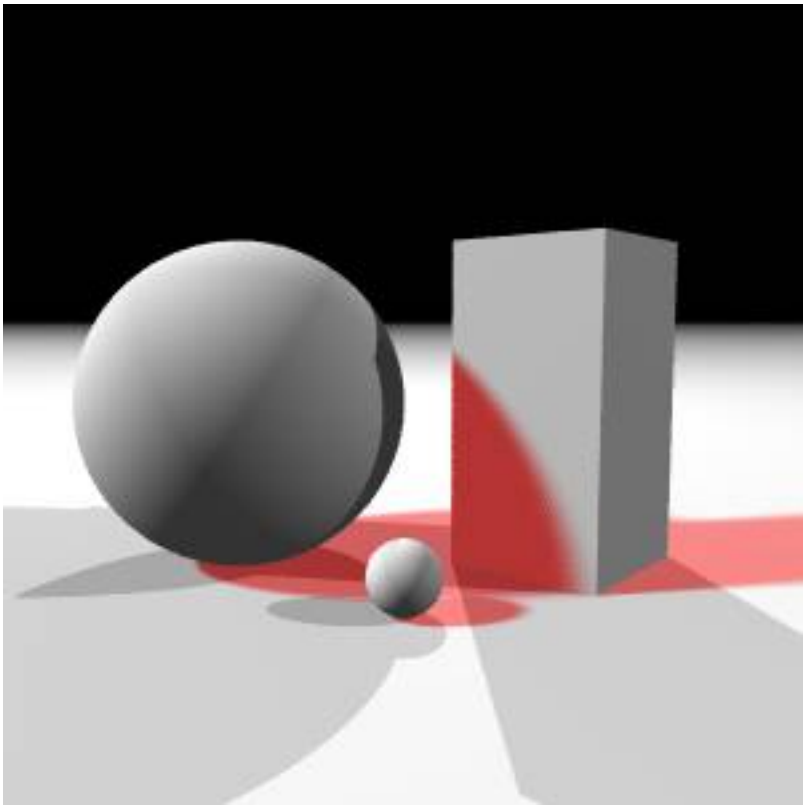
Les ombres créées par cette light_source seront de couleur définie par *pigment_ombre*.

Exemples



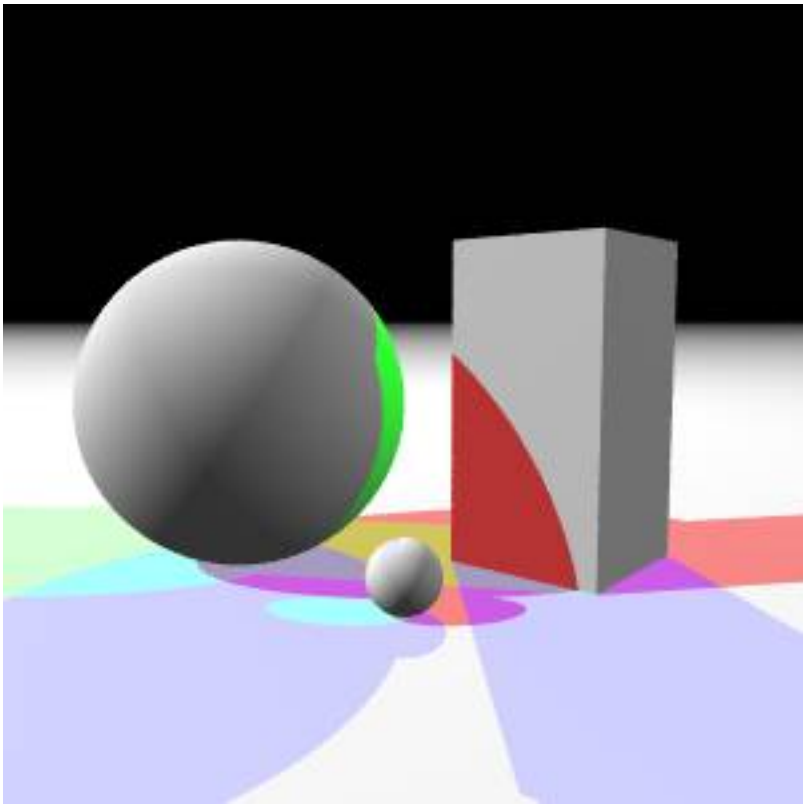
Shadow_pigment

```
camera { location <0,1.5,-5> up y right x look_at <0,1,0> }

// Shadow_pigment light_source
light_source
{
  <-50,35,-10> color rgb 1
  shadow_pigment { color rgb <1,0,0> } }

// Light_sources normales
light_source { <50,30,-10> .4 }
light_source { <0,10,10> .3 }

sphere { <-1,1,0>, 1 pigment { color rgb 1 } finish { ambient .2 diffuse .8 } }
box { <-0.5,0,-0.5>,<.5,2,.5> rotate 40*y translate 1*x pigment { color rgb 1 } finish { ambient .2
sphere { <0,0.2,-1>, .2 pigment { color rgb 1 } finish { ambient .2 diffuse .8 } }

plane { y,0 pigment { color rgb 1 } finish { ambient .2 diffuse .8 } }
```



Shadow_pigment avec area_light

Pareil que ci-dessus mais avec une area_light.



Shadow_pigment avec plusieurs lights

```
light_source { <50,30,-10> .4 shadow_pigment { color rgb <0,1,0> }}
light_source { <0,10,10> .3 shadow_pigment { color rgb <0,0,1> }}
```

Pareil en ajoutant des shadow_pigment aux deux autres light_source.

Program

Look for #ifdef SHADOW_PIGMENT_PATCH in sources

A pointer to a pigment was added into the light struct (Light_Source_Struct).

Possible improvements/problems

- There are certainly situations not very well handled with this (exotic) patch. Needs more testing

Bokeh_pigment

Description

Ce patch modifie le calcul du focal blur.

Syntaxe

```
camera { ... bokeh_pigment { pigment_densite } ... }
```

Les rayons envoyés pour le calcul du focal blur sont répartis en fonction de la valeur du pigment *`pigment_densite`* pris dans un carré [0,1]x[0,1] en x,y. La densité est la moyenne des trois composantes R,G et B de la couleur du pigment. Par défaut dans povray les rayons sont choisis dans [0,1]x[0,1] uniformément.

Exemples

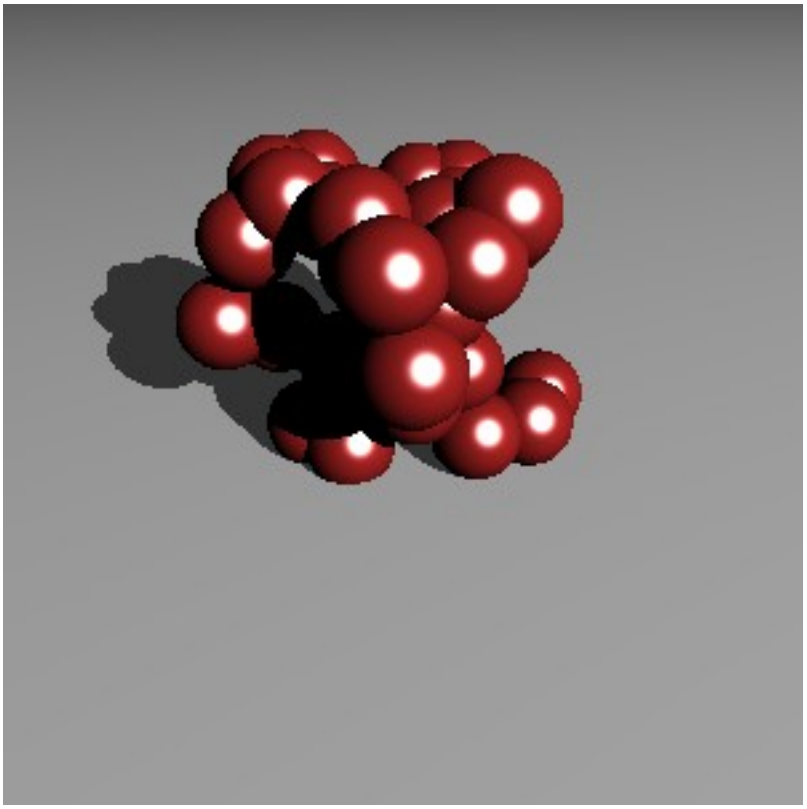
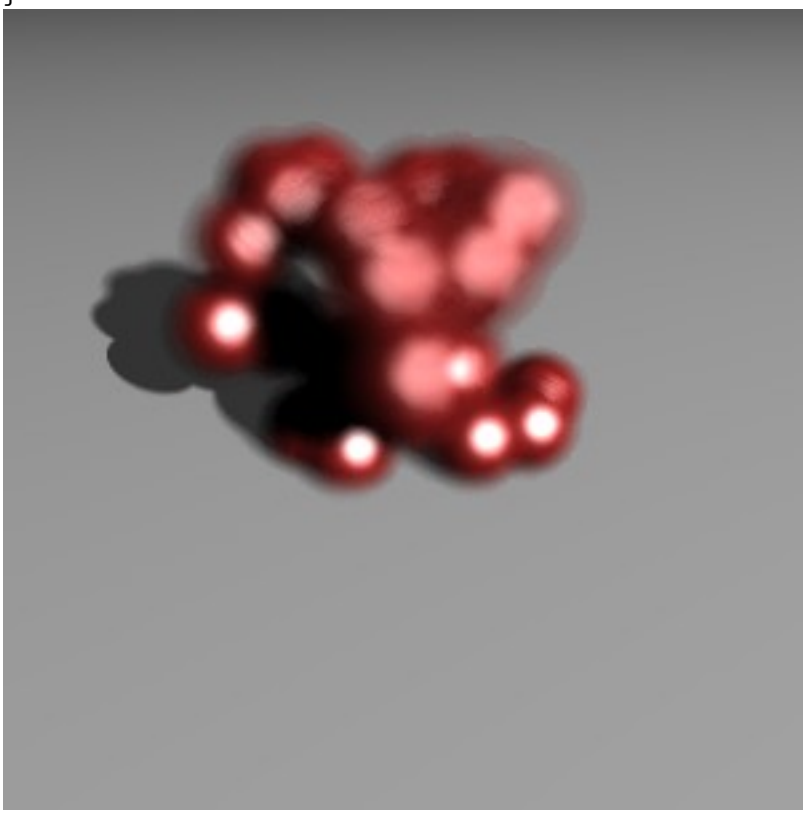


Image initiale

une image de test.

Focal blur de pov

```
camera
{
  location <5,5,-5> up y right x look_at <0,1,0>
  focal_point <0,0,4> aperture 3 blur_samples 4000
}
```

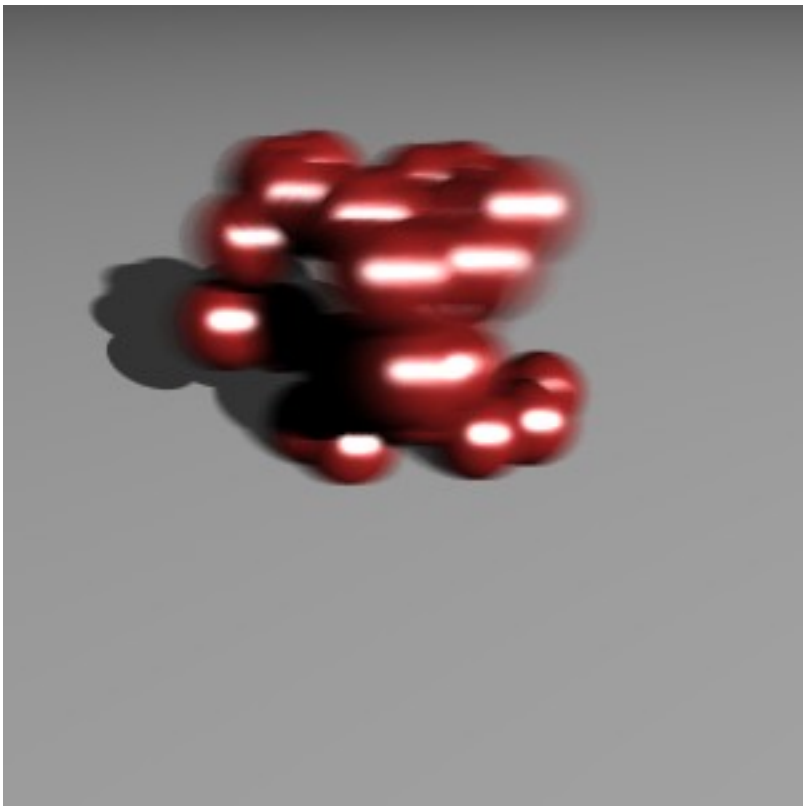


Focal et bokeh avec un pigment hexagonal

```
camera
{
  location <5,5,-5> up y right x look_at <0,1,0>
  focal_point <0,0,4> aperture 3 blur_samples 4000
  bokeh_pigment
  {
    hexagon color rgb 1 color rgb 0 color rgb 0    rotate 90*x  scale .5  translate <.5 ,.5 ,0>
  }
}
```

et le pigment correspondant :

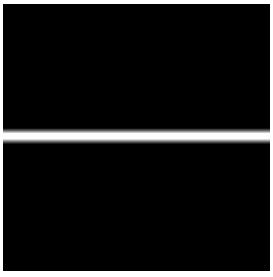
Le pigment hexagonal



Focal et bokeh avec un autre pigment

```
camera
{
  location <5,5,-5> up y right x look_at <0,1,0>
  focal_point <0,0,4> aperture 3 blur_samples 4000
  bokeh_pigment
  {
    gradient y color_map { [0.02 color rgb 1][0.05 color rgb 0] } scale .55 translate <.5,.5,0>
  }
}
```

et le pigment correspondant :



Le pigment gradient

Program

Look for #ifdef BOKEH_PATCH in sources

The samples used in the focal blur are found by dividing [0,1]x[0,1] in a grid of size x=size y=sqrt(nb_samples/4). Then for each sub square we randomly choose a number of samples proportional to the value of the pigment in this sub square (pigment computed at the center of the sub square). If the pigment is uniform we have 4 random samples by sub squares.

In function focal_blur, a Clip_Colour(C, C) was commented to have better results.

Possible improvements/problems

- Some problems can arise for pigment with fast variations
- Should use a function rather than a pigment to decribe density
- Needs fixing for adaptive number of samples (bokehsamples array)
- Needs a better way to find the samples !

Synthese

Description

Ce patch permet la création d'une nouvelle image de texture à partir d'un échantillon. Codé d'après un papier [http://www.cs.berkeley.edu/%7Eefros/research/quilting.html] de Alexei A. Efros et Willian T. Freeman. Le principe est de prendre des petits rectangles de l'image échantillon et de les 'coller' au mieux pour former l'image finale. On peut ainsi générer des textures de toutes tailles, ainsi que des textures non identiques du fait que le choix des blocs à coller comporte une part de hasard.

Syntaxe

```
image_map { synthese taillex,tailley blockx,blocky R1 image_type image_file ... }
```

taillex et *tailley* donnent la résolution de l'image à créer. *blockx* et *blocky* la taille des rectangles à prendre dans l'échantillon de texture. *R1* est un identificateur obtenu à l'aide de seed (#declare R1=seed(1234);), c'est lui qui est utilisé pour la partie aléatoire de l'algorithme. *image_type* et *image_file* donnent le fichier image échantillon (exemple: tga "modele.tga". Il faut une image 24 bits). A la suite vous pouvez spécifier les modificateurs habituels d'une *image_map* (*once*, *map_type* etc).

Le temps de synthèse est très dépendant de la taille de l'échantillon, il vaut donc mieux choisir de petites images (moins de 200x200). Le choix de *blockx,blocky* dépend de l'image modèle, de la taille des structures de la texture.

> **Note**
> Comme le temps de calcul est assez important il vaux mieux générer l'image, la sauver dans un bitmap et puis l'inclure comme une image_map classique ensuite.

Exemples

Voici les résultats de la synthèse pour quelques textures. Chaque texture échantillon est de taille environ 192x192, et pour chacune d'entre elles deux images de taille 400x400 sont générées : une avec des blocs 40x40 et une avec des blocs 20x20 (La plus réussie est celle présentée).

Pour certains cas où les résultats sont mauvais il doit être possible d'obtenir mieux en choisissant plus subtilement la

taille des blocs !

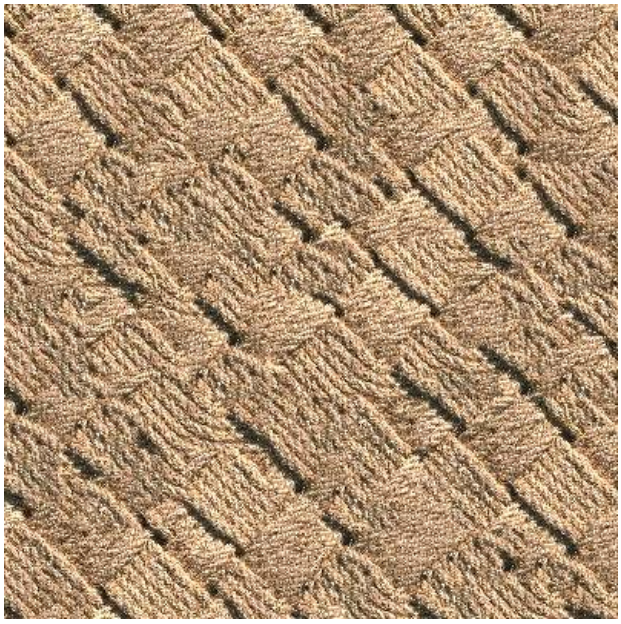
Le script pour générer ces images est le suivant :

```
#version Unofficial mlpov 0.8;

camera {
  orthographic
  location <0.5,0.5,-2> right 1*x up 1*y look_at <0.5,0.5,0>
}

#declare R1=seed(1234);

plane {
  z,0
  pigment {
    image_map { synthese 400,400 20,20 R1 sys "vistex01.bmp" once }
  }
  finish { ambient 1 }
}
```

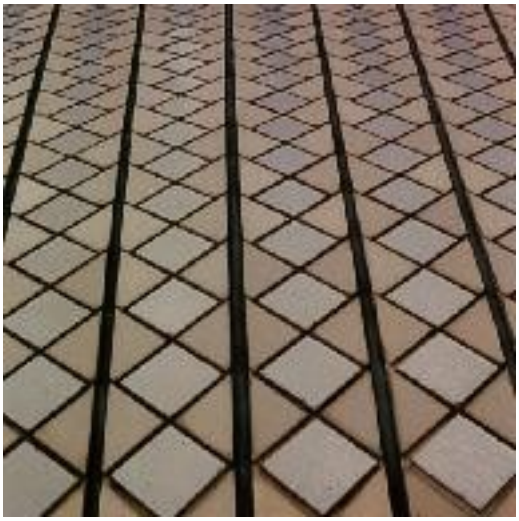
Table 1. Exemples

Enchantillon	Synthese
	

Enchantillon	Synthese
	
	

Enchantillon	Synthese
	
	

Enchantillon	Synthese
	
	

Enchantillon	Synthese
	
	

Program

Look for #ifdef SYNTHESE_PATCH in sources

The patch build a tga image at parsing.

Possible improvements/problems

- Only works with 24 bits per pixel images (but should not be too difficult to extend)

- Doesn't really work with large samples
- Needs better coding, speed optimizations..

HDR (High Dynamic Range)

Description

Les formats d'image 24 bits n'offrent que 256 niveaux pour les composantes rouge, vert, bleu d'un pixel. Paul Debevec [http://http.cs.berkeley.edu/%7Edebevec/Research/HDR/] a proposé un format HDR (High Dynamic Range) où chaque composante est codée sur 4 octets au lieu de 1. L'image comporte ainsi une bien plus grande dynamique d'intensité des couleurs. Ce patch permet d'utiliser ces fichiers .hdr. Le code de lecture des images est adapté du source de vrpic [http://excamera.com/articles/9/vrpic.html] de James Bowman, un outil de visualisation des .hdr pour linux.

Ce patch ajoute de plus une nouvelle map_type (map_type 7) qui repositionne un environnement HDR [http://www.debevec.org/Probes/] sur une sphère. Ce nouveau mapping couplé à la radiosité permet un rendu dit HDRI (High Dynamic Range Illumination).

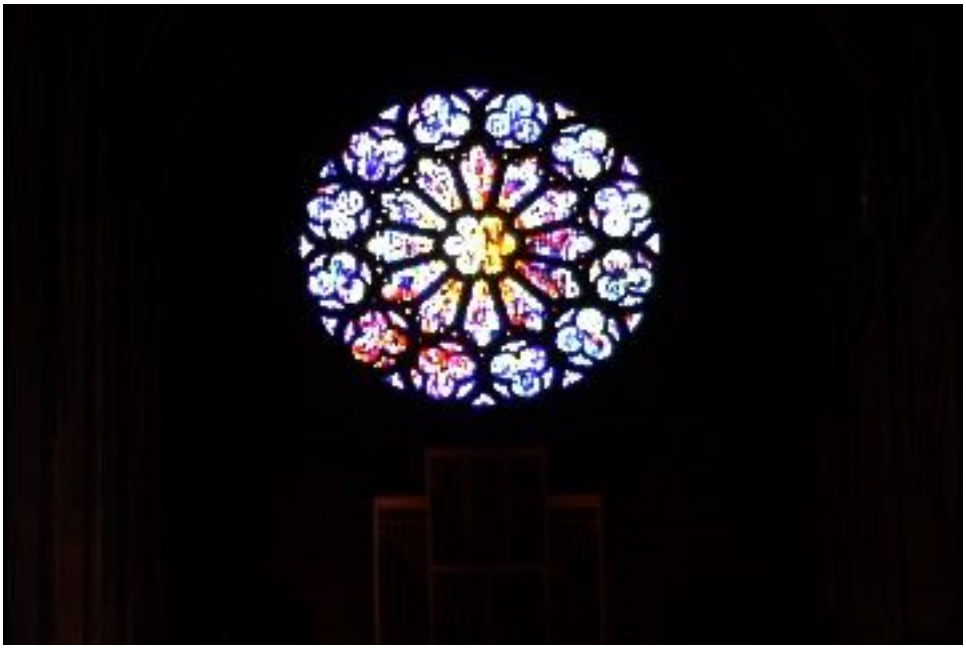
Syntaxe

```
image_map { hdr .... [map_type 7]}
```

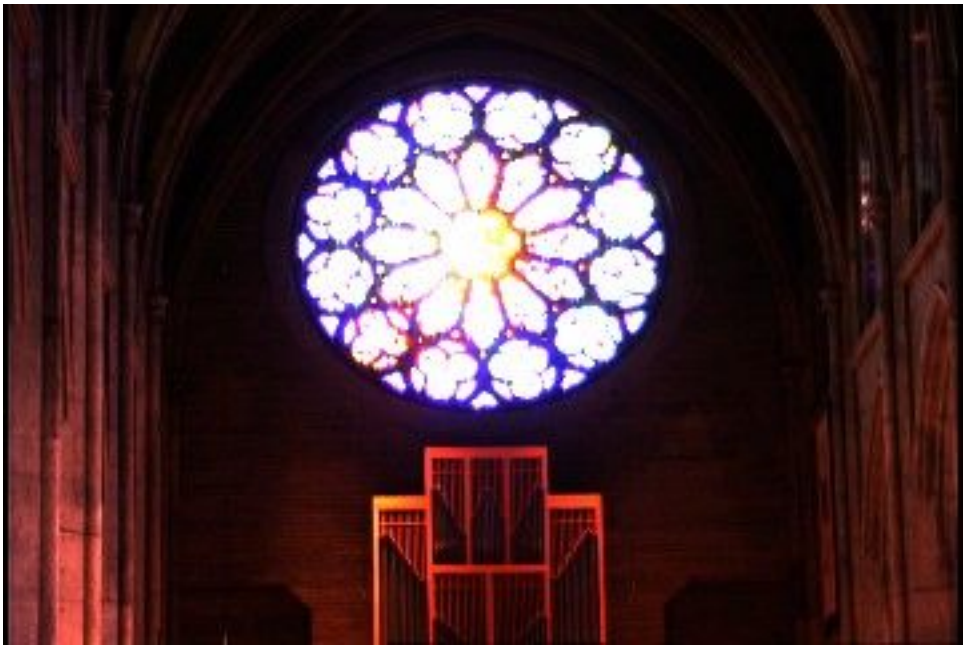
Toutes les options d'*`image_map`* (*`interpolate`*, ..) restent valides.

Exemples

Exemple pour visualiser la dynamique en jouant sur le paramètre *`ambient`*.



Ambient 1



Ambient 10

Ambient 30

```
camera
{
  orthographic
  location <.5,.5,-5> right 1*x up 1*y look_at <.5,.5,0>
}

plane
{
  z,0
  pigment { image_map { hdr "rosette.hdr" once interpolate 2} }
  finish { ambient 0.5 diffuse 0 }
```

```
}
```

Program

Look for #ifdef HDR_PATCH in sources

A new struct with 3 float (IMAGE_FLOAT_LINE) was added to image_struct. A new constant ISFLOATIMAGE for File_Type. The map_type 7 is calculated by function hdr_image_map in image.cpp.

Possible improvements/problems

- There are other (better ?) HDR formats (with TIFF structure for instance)
- An option to output a render in hdr format would be nice

Finish maps

Description

Ce patch permet de spécifier une fonction au lieu d'un float ou d'une couleur pour les paramètres *finish* d'une texture. Les paramètres du *finish* sont alors évalués à chaque point d'intersection, lors du rendu.

Syntaxe

Dans la partie *finish* d'une texture, au lieu d'un float ou d'une couleur on peut donner soit un identificateur de fonction, soit une déclaration de fonction. Attention il faut que la fonction retourne une valeur/une couleur qui corresponde au type du paramètre.

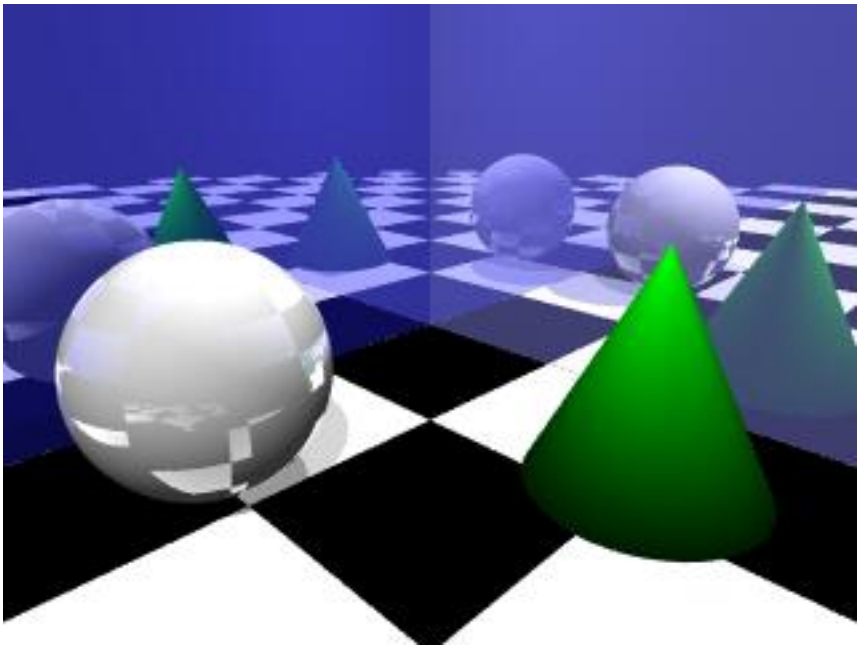
```
finish {
    [uv_mapping]
    ambient COLOR | VECTOR_FUNCTION_IDENT | VECTOR_FUNCTION
    diffuse Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
    brilliance Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
    phong Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
    phong_size Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
    specular Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
    roughness Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION (Cf. warning)
    reflection COLOR | VECTOR_FUNCTION_IDENT | VECTOR_FUNCTION
    reflection {
      Color_Reflection_Min | VECTOR_FUNCTION_IDENT | VECTOR_FUNCTION
      ,
      Color_Reflection_Max | VECTOR_FUNCTION_IDENT | VECTOR_FUNCTION
      falloff FLOAT_FALLOFF | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
      exponent FLOAT_EXPONENT | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION (Cf. warning)
    }
    irid {
      Irid_Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
      thickness Amount | FLOAT_FUNCTION_IDENT | FLOAT_FUNCTION
    }
}
```

Si *uv_mapping* est spécifié dans le bloc *finish*, l'évaluation se fait après passage des coordonnées en u,v.

Warning

Pour *roughness* et *exponent* il faut une fonction qui retourne 1/*roughness* et 1/*exponent*.

Exemples



Finish maps

```
#declare fct = function { pigment { checker color rgb 0 color rgb 1 scale .1 } }

sphere { ...
  finish {
    uv_mapping
    reflection fct
  }

cone { ...
  finish {
    diffuse function {y}
  }
}
```

Program

Look for #ifdef FINISH_MAPS_PATCH in sources

The patch adds 3 new members to the FINISH struct : a number which indicates the number of evaluations required, a boolean to know if we must use uv_mapping, and a pointer to an array of a structure with an offset (offset to the member of the finish struct to modify) and a FUNCTION_PTR. This array is filled during the parsing, and sorted by function (so that after, if the same function is used by more than one member of finish, it is only evaluated once).

Possible improvements/problems

- No tests for several declaration of the same member of finish
- The functions are evaluated at beginning of shader calculations while some of them will not be needed later (for example no need to compute specular if it appears that the object is in shadow)

Correctifs

Divers correctifs proposés sur le newsgroup povray.programming.

Bicubic

Un patch [http://news.povray.org/3db42c7c%40news.povray.org] de Massimo Valentini. Fichier pov exemple : bicubic.pov

Superllipsoid

Un patch [http://news.povray.org/3dae8b49%40news.povray.org] de Massimo Valentini. Fichier pov exemple : superllipsoid.pov

Copy normal accuracy

Un patch [http://news.povray.org/3d9f8dce%40news.povray.org] de John Haggerty. Fichier pov exemple : accuracy.pov

Copy flag

Un patch [http://news.povray.org/3d81cb77%40news.povray.org] de Massimo Valentini.

Spline et function

Un patch [http://news.povray.org/3d579399%40news.povray.org] de Thorsten Froehlich. Fichier pov exemple : spline.pov

Julia

Un patch [http://news.povray.org/3dc7fc83%40news.povray.org] de Massimo Valentini. Fichier pov exemple : julia.pov

Sphere_sweep

Un patch [http://news.povray.org/3dd4ca43%40news.povray.org] de Massimo Valentini. Fichier pov exemple : sphere_sweep.pov

Aspect ratio

Un patch [http://news.povray.org/3de51e29%241%40news.povray.org] de Anders K.