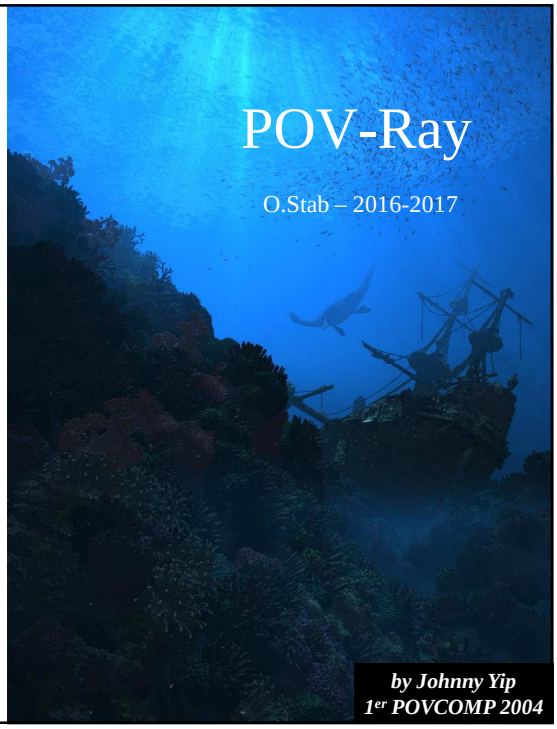
- Persistence Of Vision RAY-tracer
- free-ware
- Windows/Unix/ Linux/MAC
- www.povray.org
- v 3.6.2 Juin 2009
- v 3.7 SMP Nov 2013
- https://github.com/...

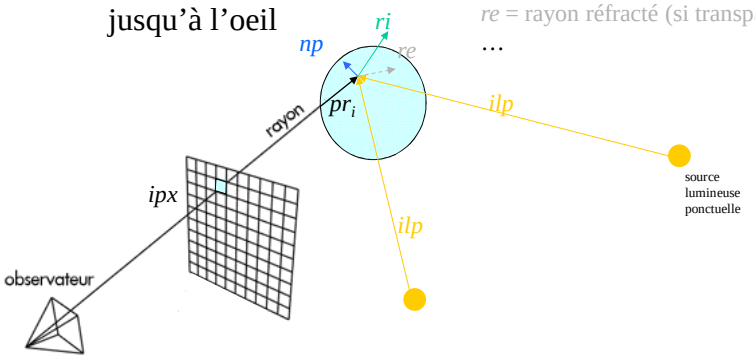


Introduction à PovRay

PovRay

Lancé de rayon
(Ray tracing)

Principe : modéliser le trajet des rayons de la lumière jusqu'à l'oeil



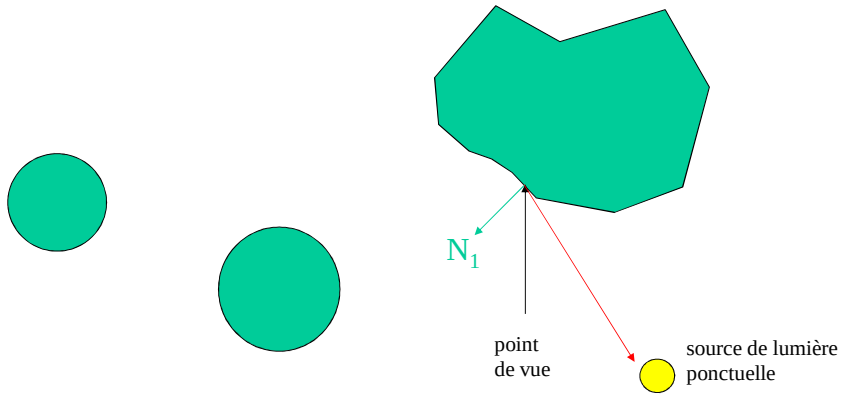
ipx = intensité lumineuse du pixel
r = lancé d'un rayon partant de *ipx*
pr_i = intersection de r avec la géométrie
np = normale en pr_i
ilp=éclairage de la surface en pr_i
ri = rayon réfléchi en pr_i (si miroir)
re = rayon réfracté (si transparence)
…

Le lancer de rayon

- 1968 : Algorithme proposé par Appel
- 1980 : David Kirk Buck (futur PovRay)
- 1991 : PovTeam
- …
- 2012 : Nvidia Kepler real-time…

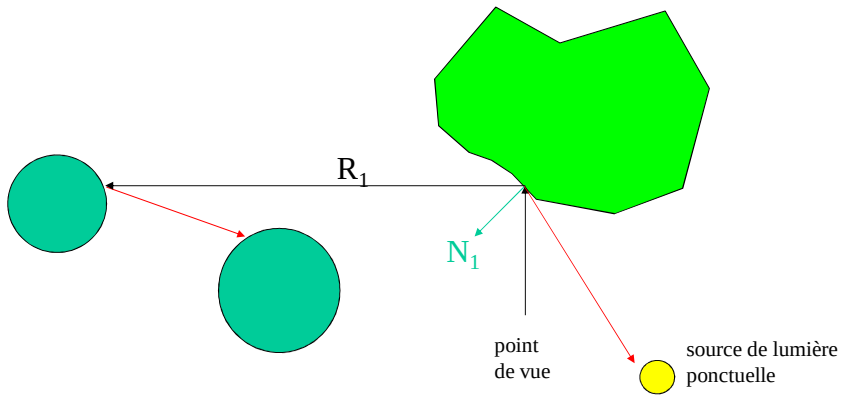
- Idée
  - facette = réflecteur parfait
  - inverser la circulation des rayons lumineux : de l'écran vers les sources de lumière

Principe : lancé de rayon



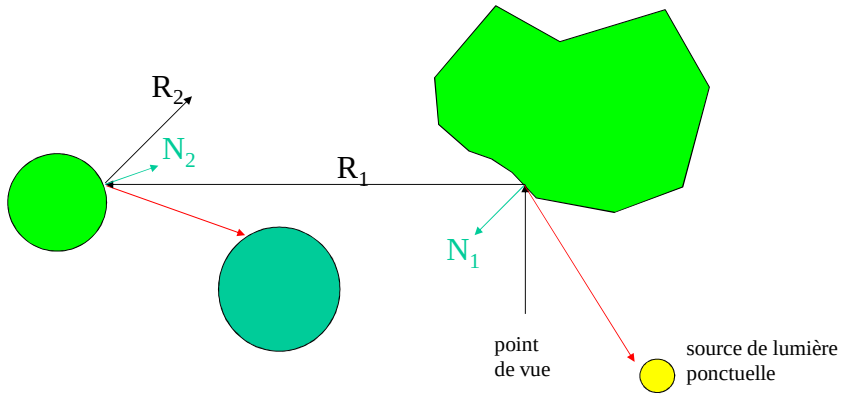
from CG principles & practice Foley, Van Dam …

Principe : lancé de rayon



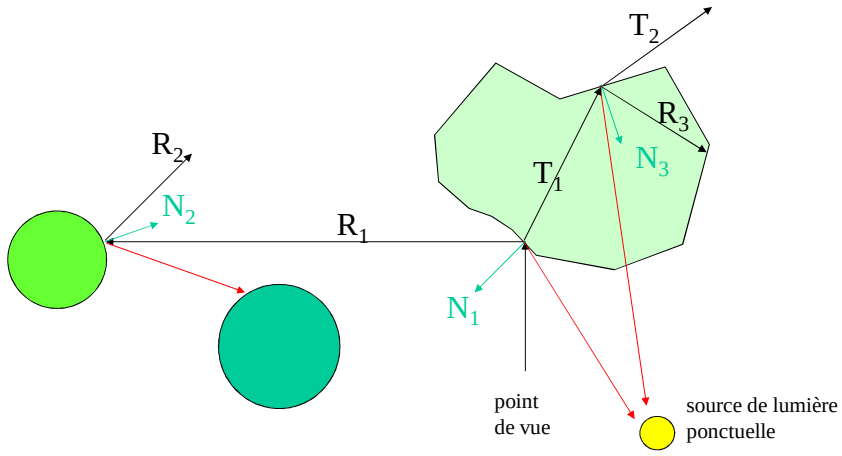
from CG principles & practice Foley, Van Dam …

Principe : lancé de rayon



from CG principles & practice Foley, Van Dam …

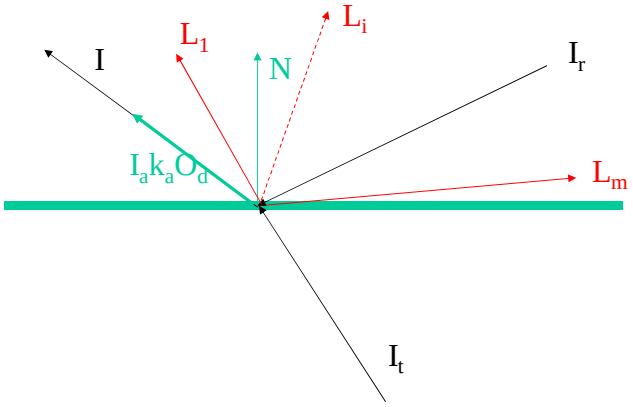
Principe : lancé de rayon



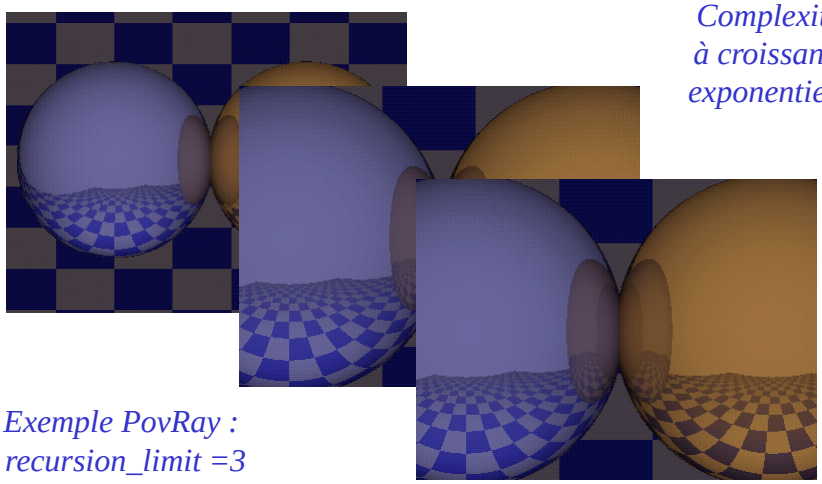
from CG principles & practice Foley, Van Dam …

Définition récursive

$$I_\lambda = \underbrace{I_{a\lambda} K_a O_{d\lambda}}_{\text{ambiante}} + \underbrace{\sum_{i=1}^{m} S_i f_{att\_i} I_{p\lambda i} \left[\underbrace{K_{di} O_{d\lambda} (\vec{N}.\vec{L}_i)}_{\text{diffuse}} + \underbrace{K_s (\vec{R}_i.\vec{V})^n}_{\text{spéculaire}} \right]}_{\text{sources}} + \underbrace{K_r I_{r\lambda}}_{\text{refléchie}} + \underbrace{K_t I_{t\lambda}}_{\text{réfractée}}$$

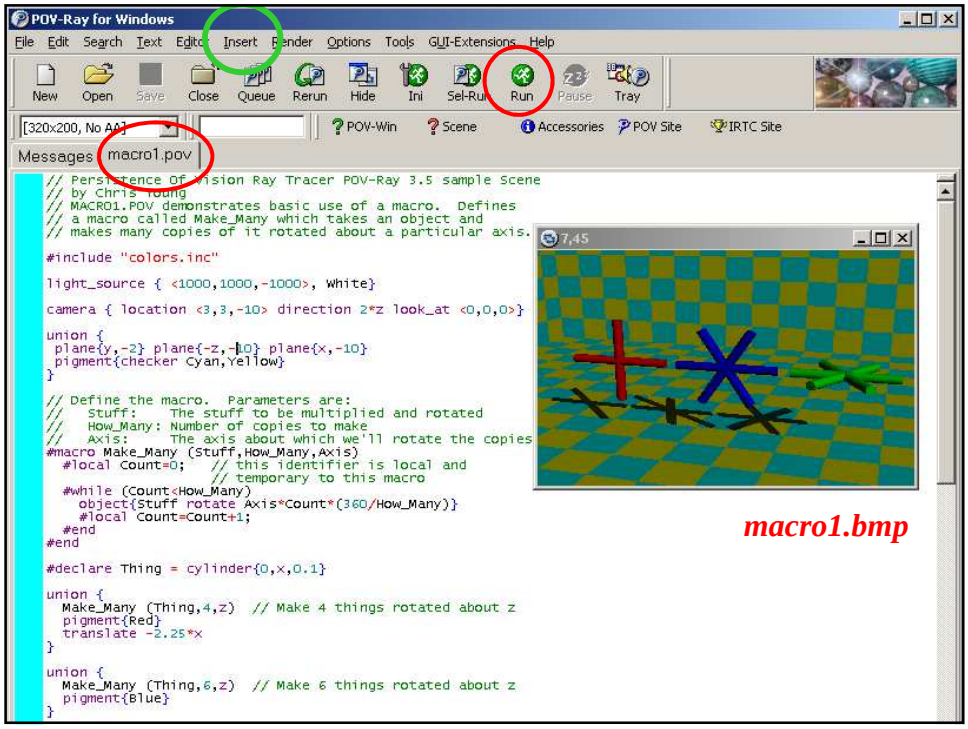


Jusqu'où ?



Complexité à croissance exponentielle

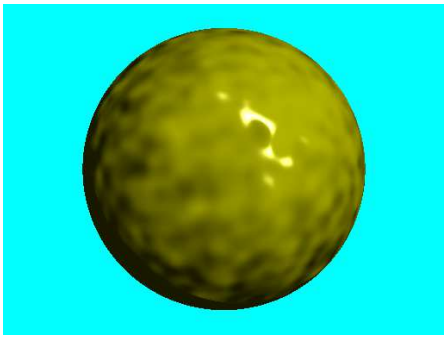
Exemple PovRay : recursion_limit =3 20 au maximum



II Le langage

Un langage de description de scène permet de générer des images

```
#include "colors.inc"    // Les fichiers d'include
#include "textures.inc"  // contiennent des elements
#include "shapes.inc"    // predefinis

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 }
   normal {bumps 0.4 scale 0.2}
   finish {phong 1}
   }
 }
```



Types, Opérateurs & fonctions

Réels : -2.4e-5	* , /, +, -	cos(A), log(A)…
Vecteurs : < réel,réel [,réel]*> < 1, 1> , < 1, 1, 1 > …	* , /, +, - (terme à terme)	vdot(V1,V2) vcross(V1,V2) …
Chaine de caractères : «colors.inc»		str(A,B,C), concat(...), …
Propositions : true yes on 1 false no off 0	&, | =, != , >=...	
…Objets géométriques…		

La syntaxe des objets

Type_objet { description_objet
[modificateur_objet]
}

// commentaire

*/*commentaire */*

 # directive

Exemple 1:

```
// une simple sphere translatée
#declare VTrans=<10,0,10>;
sphere { <0,0,0> 15
        translate Vtrans }
```

Exemple 2:

```
// une simple sphere colorée
#include "colors.inc"
sphere { <0,0,0> 15
        pigment {rgb Red} }
```

Scalaires & vecteurs

Les réels : # declare a = 2.4e-5 ; # declare b = sqrt(a);

Les points & vecteurs : < réel,réel [,réel]*> ;

```
#declare V1 = < a, -2.5 > ;  # declare b=V1.v; // En 2D : V.u,V.v
# declare V2= < V1.u, V1.v, b >; // en 3D
# declare V3 = V2 + <1,2,3>;
# local c=V3.z ; // (V.x,V.y,V.z)
# declare White = rgbt < 1, 1, 1, 0.5 > ; // en 4D
// ATTENTION aux noms réservés ; x, y, z, u, v, t…
```

Les fonctions :

```
# local V4 = vcross(V3,V4);          # declare d = vdot(V3,V4);
# local VN3 = vnormalize(V3);        # declare nv3 = vlength(V3);
```

Tableaux

```
// Declaration de tableau
# declare TAB = array [3]; // array [][][][][]
# local n = dimension_size( TAB,1 );
# declare TAB[0]=1; // de 0 à n-1
```

```
// Tableau de pigments
#declare TP = array [5] ;
#declare TP[4] = pigment { White }
```

```
// Tableau de vecteurs
#declare nb = 2;
#declare TV = array[nb] { < 1, -2.5 >, <0, 5.5> } ;
# declare TV[1] =  <TV[2].v, TV[2].u >;
```

```
// Grille
#declare TP = array [2][5] {
{ 1,2,3,4,5},
{2,3,4,5,6}};
#declare a=TP[1][0];
```

```
// Les éléments non initialisés sont non définis
#declare i = 2;
#ifndef ( TAB[i] )
#debug concat( "L element TAB["   , str(i,0,0), "] n est pas defini\n\n"  )
#end
```

Directives de programmation

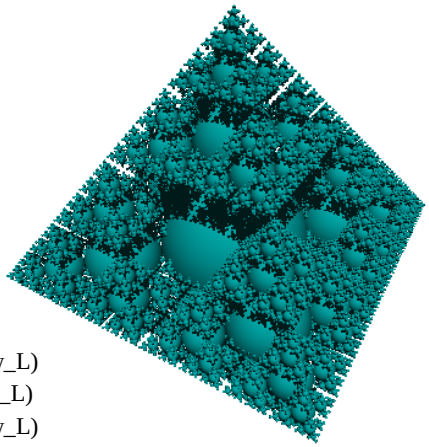
```
//exemple de boucle
#declare i = 0;
#while ( i < 5 )
#declare i = i+1;
…
#end
```

```
//branchement conditionnel
#switch( VALUE)
#case…
#break
#end
#if ( COND ) ... #else ...#end
```

```
//declaration de macro
#macro masphere(r)
sphere { < 0,r,0 > , r }
#end

masphere(10)
```

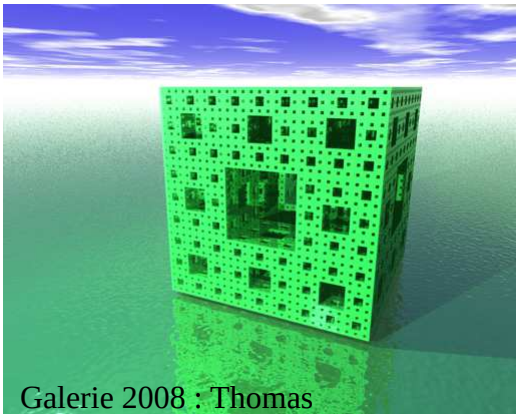
```
#declare masphere = sphere { < 0,0,0 > 15};

masphere
```

Les macros



```
#macro Pyramid(X,Y,Z,R,L)
 sphere { <X,Y,Z>,R}
  #if (L > 0)
    #local New_L = L - 1;
    #local New_R = R / 2;
    #local Pos   = New_R * 3;
    Pyramid(X+Pos,Y,Z,New_R,New_L)
    Pyramid(X-Pos,Y,Z,New_R,New_L)
    Pyramid(X,Y+Pos,Z,New_R,New_L)
    Pyramid(X,Y-Pos,Z,New_R,New_L)
    Pyramid(X,Y,Z+Pos,New_R,New_L)
    Pyramid(X,Y,Z-Pos,New_R,New_L)
  #end
#end
```

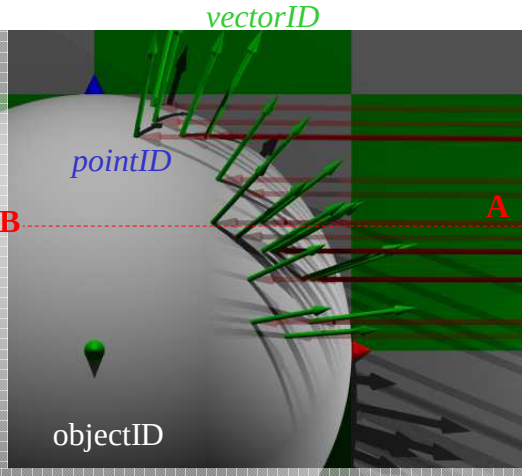
Exemple : le cube de Sierpinski



La fonction trace

Permet de faire :

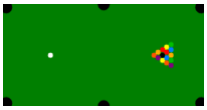
- du placement,
- des animations,
- des effets visuels
- …



pointID =trace(objectID,A,B,*vectorID*)

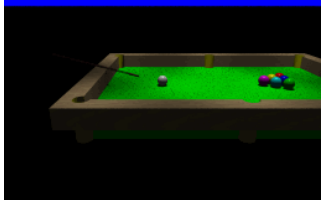
La fonction trace (exemples)

Animations

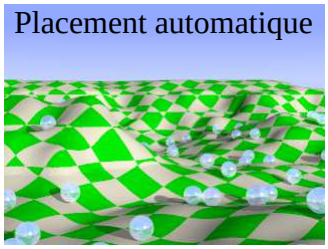


Mathematica
with povray

Yohan & Hugues 2009



Placement automatique



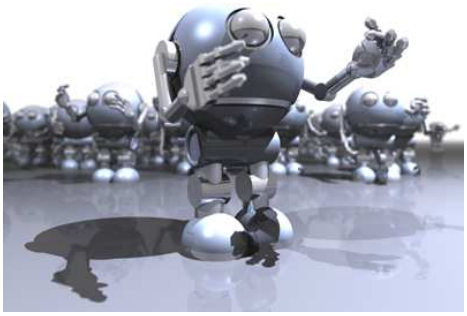
http://www.bugman123.com/Physics/

TP n° 2 : programmer une scène

III. La géométrie

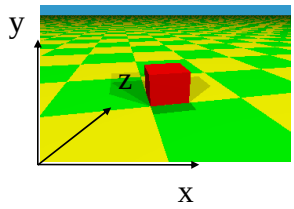
1. Positionnement et transformation

2. Formes
   - Primitives
   - Opérateurs
   - Constructeurs



Position et orientation

Repère indirect

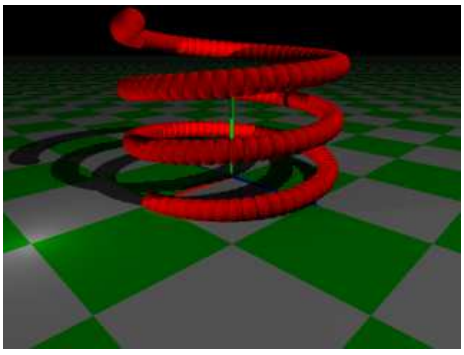


Les transformations

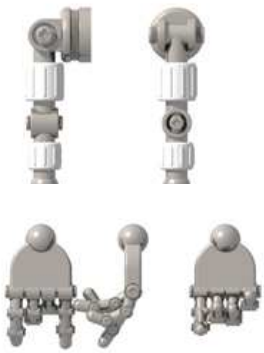
```
… // les transformations s 'appliquent à tout ce qui précède
union {
        box { <0,0,0> <107,48,11> }
        box { <0,0,0> < 11,48,25> translate <96,0,11> }
        translate <96,0,11>  // la translation est appliquée aux 2 boites
        } ...
```

translate{ vecteur3d }
scale {vecteur3d} Homothétie par rapport à l'origine
rotate {vecteur3d} Angles d'Euler
Conseils : scale, rotate puis translate T(R(S(o)))
matrix {...} Les matrices sont en coordonnées homogènes

Transformation : exemples



TP n° 4 : géométries « complexes »

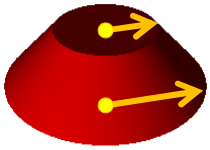


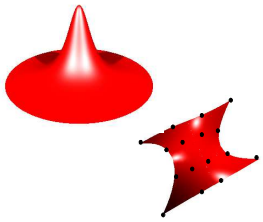
Bras, main articulés

Les primitives

- Volumes élémentaires (3D)
  - box{}, sphere {}, cone {}...
  - mesh{}…
- Primitives 2D
  - polygon{}, disq{}…
  - height_field{}…
- Les surfaces :
  - plane{}, poly{}, quadric{}…
  - bicubic_patch {}

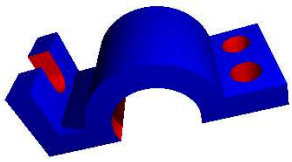
POV





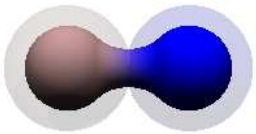
Les constructeurs

union{}, intersection{}, difference{},
objet{} ...



prism{},
sor{} *Surface Of Revolution*
lathe{} *Le contour peut être fermé*

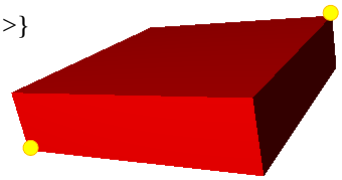
blob {
threshold *seuil_force*
sphere { < *Pcentre*>, *rayon_sphere* , strength *force_centre*}
sphere { < *Pcentre*>, *rayon_sphere* , strength *force_centre*}
cylinder {…}
...
}



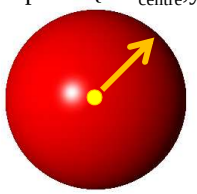
➔ *Shortcut to examples*

Volumes élémentaires

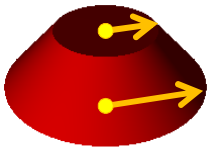
box {< x_{min}, y_{min}, z_{min} >,< x_{max}, y_{max}, z_{max} >}
// x_{min}, y_{min}, z_{min} : le coin « bas gauche »
// x_{max}, y_{max}, z_{max}: le coin oppose

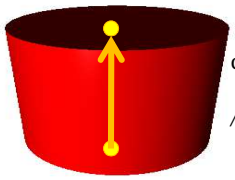


sphere { <x_{centre},y_{centre},z_{centre}>, rayon }



cone { <x_{base},y_{base},z_{base}>, rayon_base,
<x_{haut},y_{haut},z_{haut}>, rayon_haut }
// le cone reste perpendiculaire à l 'axe





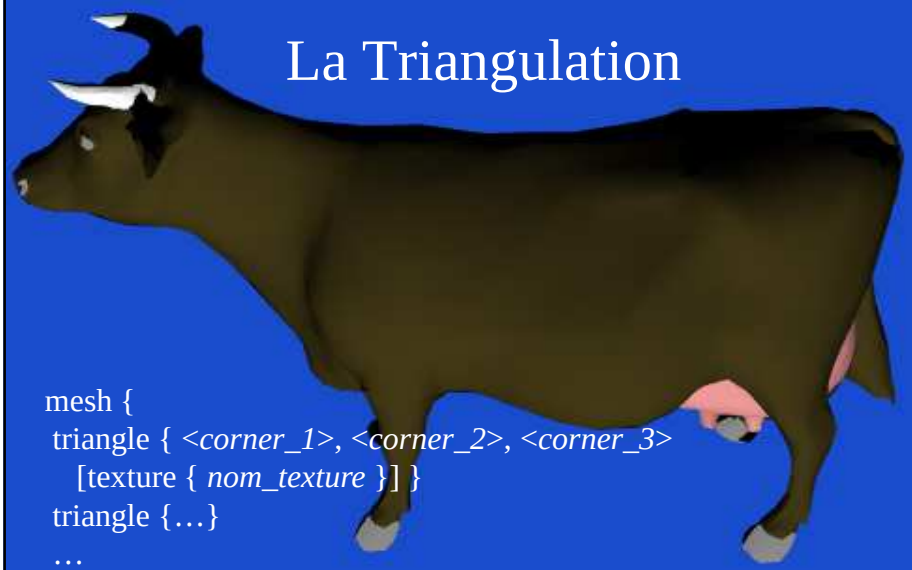
cylindre { <x_{base},y_{base},z_{base}>,
<x_{haut},y_{haut},z_{haut}>, rayon_haut }
// le cylindre reste perpendiculaire à l 'axe

Le polygones

Polygon { nb_points
< x_{11},y_{11} >, … , <, x_{1i},y_{1i} > … < x_{11},y_{11}>,
< x_{21},y_{21} >, … , <, x_{2i},y_{2i} > … < x_{21},y_{21}>,
< x_{j1},y_{j1} >, … , <, x_{ji},y_{ji} > … < x_{j1},y_{j1}>
}

Fichier : polygon.pov

La Triangulation



mesh {
 triangle { <*corner_1*>, <*corner_2*>, <*corner_3*>
  [texture { *nom_texture* }] }
 triangle {…}
 …
}

Fichier : cow2.pov

Les surfaces

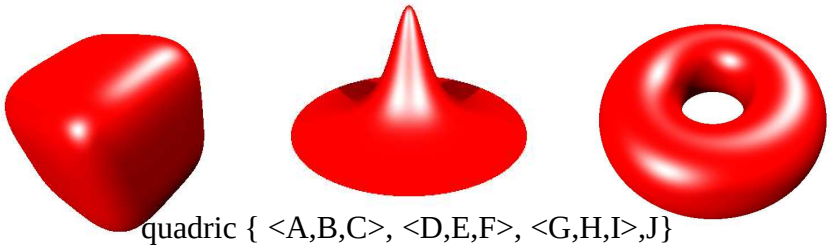
Surfaces plane :
plane { <normale> , distance }

Surfaces polynomiales :
poly { order, <T1, T2, …Tm> }
ou encore
cubic { <T1,… T20> }
quartic { <T1,…,T35> }

Quadric : ellipsoïdes, paraboloïdes…
quadric { <A,B,C>, <D,E,F>, <G,H,I>,J}

Surfaces quadriques

- équation algébrique: $Ax^2 + By^2 + Cz^2 + 2D\,xy + 2E\,yz + 2F\,xz + 2G\,x + 2H\,y + 2I\,z + J = 0$
- ellipsoïde, cylindre, paraboloïde, cône...



quadric { <A,B,C>, <D,E,F>, <G,H,I>,J}

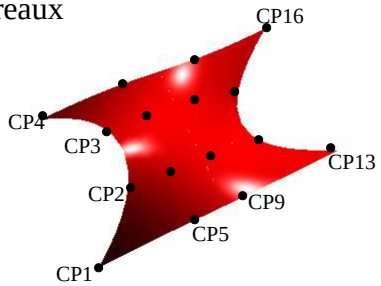
Carreau de Bezier 4x4

bicubic_patch {
type *1*
u_step *3*
v_steps *3*
}

$2^3 * 2^3$ petits carreaux

$$X(u,v) = \sum_{i=0}^{4} \sum_{j=0}^{4} X_{ij}\, P_{im}(u)\, P_{jn}(v)$$

<*CP1*>, <*CP2*>, <*CP3*>, <*CP4*>,
…
<*CP13*>, <*CP14*>, <*CP15*>, <*CP16*>
}



Courbes polynomiales : principe

Une courbe polynomiale paramétrique contrôlée par les points

$$\begin{cases} x(u) = a_0 * u + b_0 \\ y(u) = a_1 * u + b_1 \end{cases}$$

$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ $u=0$ $\quad$ $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ $u=1$

Résolution système linéaire

$$\begin{pmatrix} x(u) \\ y(u) \end{pmatrix} = \begin{pmatrix} x_1 - x_0 & x_0 \\ y_1 - y_0 & y_0 \end{pmatrix} * \begin{pmatrix} u \\ 1 \end{pmatrix}$$

P_{01} P_{11}

$$\begin{pmatrix} x(u) \\ y(u) \end{pmatrix} = \begin{pmatrix} x_0 & x_1 \\ y_0 & y_1 \end{pmatrix} * \begin{pmatrix} 1-u \\ u \end{pmatrix}$$

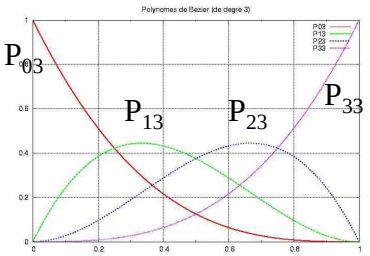
$$\begin{pmatrix} x(u) \\ y(u) \end{pmatrix} = \begin{pmatrix} x_0 & x_1 \\ y_0 & y_1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ u \end{pmatrix}$$

$$X(u) = \sum_{i=0}^{m} X_i \, P_{im}(u)$$

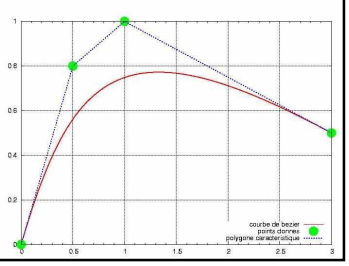
Exemple : la courbe de Bézier

$$X(u) = \sum_{i=0}^{m} X_i \, P_{im}(u)$$

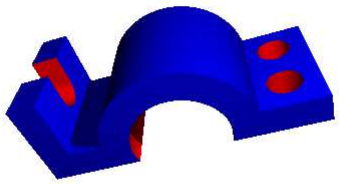
$$P_{im}(u) = (m!/((m-i)!\, i!))\, u^i (1-u)^{m-i}$$



$$\begin{pmatrix} x(u) \\ y(u) \end{pmatrix} = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{pmatrix} \begin{vmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ u \\ u^2 \\ u^3 \end{vmatrix}$$



Assemblage par opérations booléennes



```
…
difference {
    /* la partie bleue */
    union {
        union {
          box { <0,0,0> <107,48,11> }
          box { <0,0,0> < 11,48,25> translate <96,0,11> }
          }
        cylinder { <0,0,0> <0,48,0> 30 translate <53.5,0,5.5> }
        texture { pigment { red 0.0 green 0.0 blue 1.0 } }
        }
    /* la partie rouge */
    …
}
```

Fichier : frameguide.pov

union{}, intersection{}, difference{}, objet{} ...

Constructeurs : prism

Prismes : solide définis par translation d ’un contour

```
Prism {
  linear_sweep
  linear_spline
  0, 1 // hauteur (ymin, ymax)
  nb_pt,
<x11,z11> , … <x1i,z1i> // x1i=x11, z1i=z11
<x21,z21> , … <x2i,z2i> // x2i=x21, z2i=z21
…
[open]
}
```

conic_sweep
quadratic_spline | cubic_spline

Fichier : prisme.pov

Fichier : prisme_conic.pov

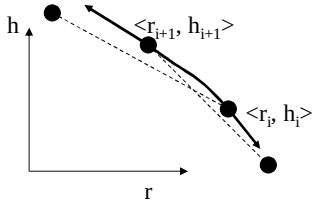
Constructeurs : S.O.R

S.O.R : Surface de révolution autour d ’un axe

```
sor {
  nb_pt,
  <r1, h1>
...
  <ri, hi>
  open
}
```

Fichier : sor1.pov

$r(h)^2 = a\,h^3 + b\,h^2 + c\,h + d$



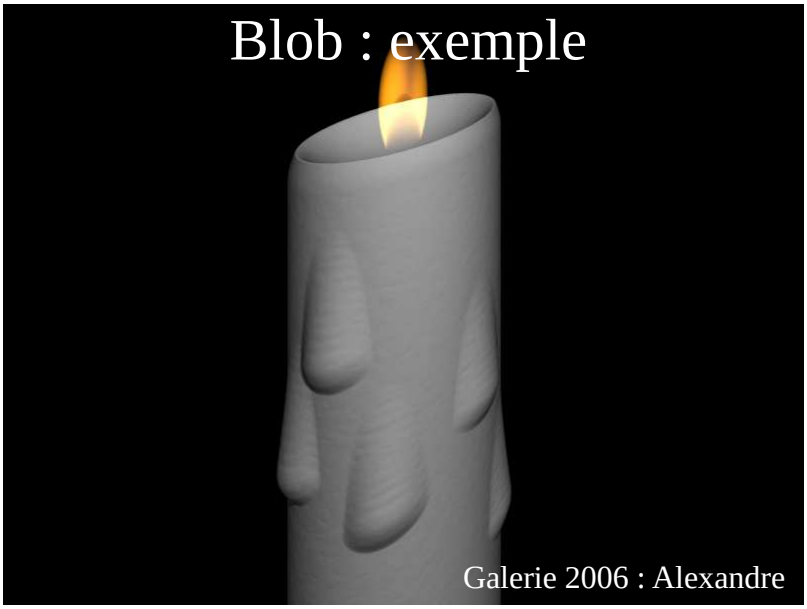
Constructeurs : Lathe

Lathe : Surface de révolution autour d ’un axe

```
lathe {
  linear_spline...
  nb_pt,
  <x1, z1>
...
  <xi, zi> // le contour peut être fermé
}
```

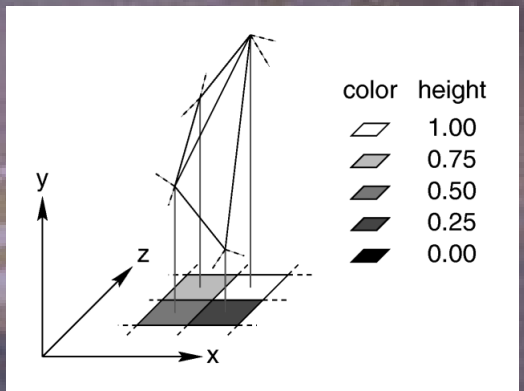
Fichier : lathe_ferme.pov

Blob : exemple

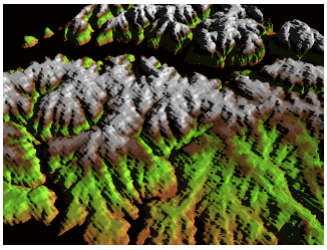


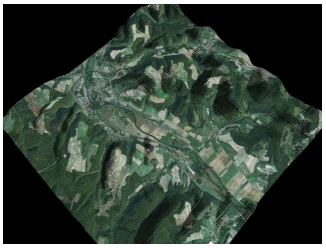
La grille

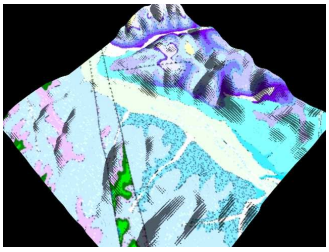
height_field { file_type « filename » }



Le « height field » : exemples





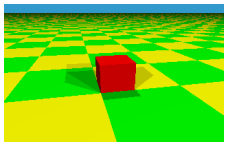


TP n° 5 : paysages

IV. L ’image

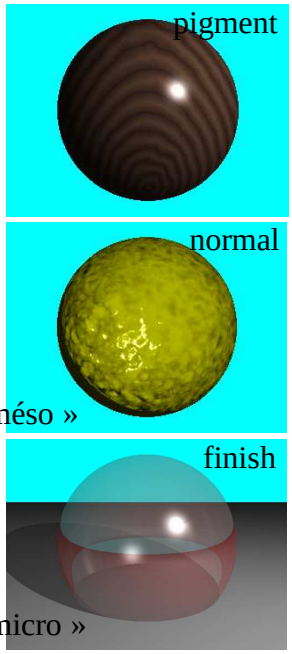
Le contexte

- La caméra
- L’éclairage



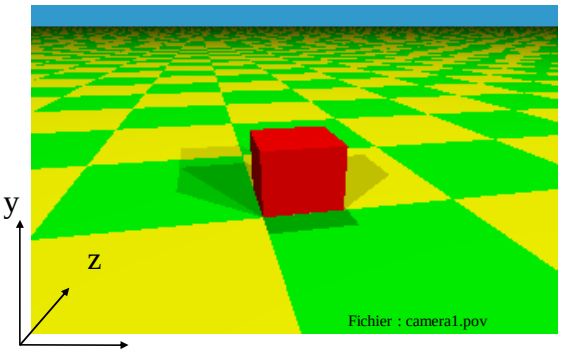
Les propriétés des surfaces (*texture*)

- La couleur (*pigment*)
- La texture (*normal*)
- L’aspect (*finish*)...



La caméra

Camera {
  [perspective]
  location <*point*>
  ...
  look_at <*point*>
}

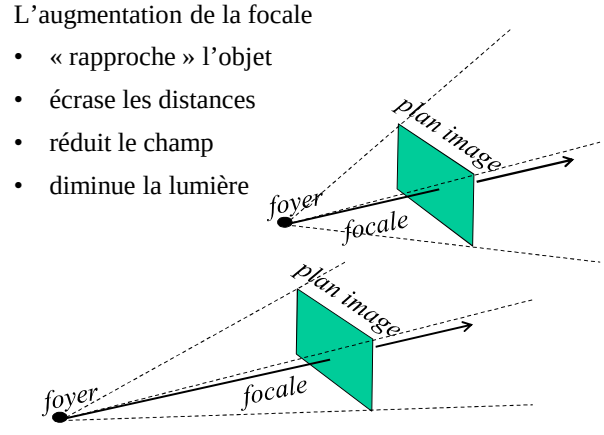


Exemple :
location <-1,5,-8>
look_at <1,1,1> //pointe vers le centre du cube <0,0,0> <2,2,2>.

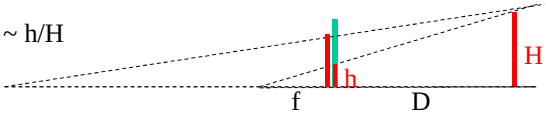
Le zoom, la focale

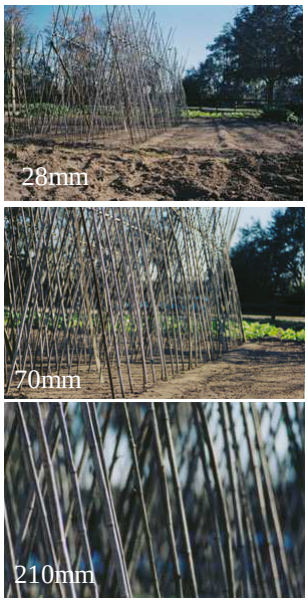
L'augmentation de la focale

- « rapproche » l'objet
- écrase les distances
- réduit le champ
- diminue la lumière



f/D ~ h/H

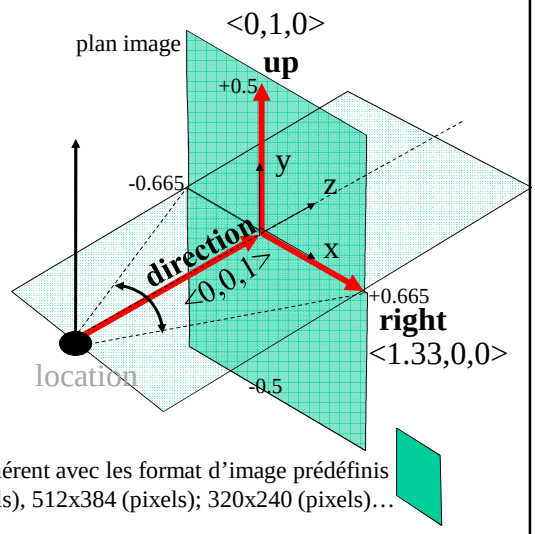




source wikipédia

Focale et “fenêtre” dans PovRay

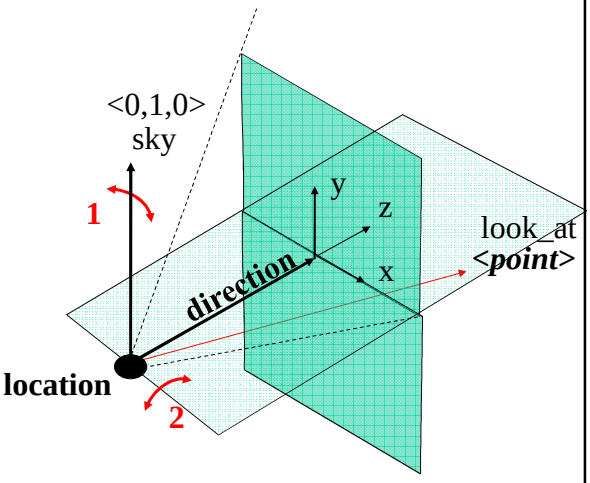
Camera {
perspective
location <*point*>
direction <*vecteur*>
right <*vecteur*>
up <*vecteur*>
sky <*vecteur*>
angle *float*
blur_samples *float*
…
look_at <*point*>
}



Cohérent avec les format d’image prédéfinis
640x480 (pixels), 512x384 (pixels); 320x240 (pixels)…

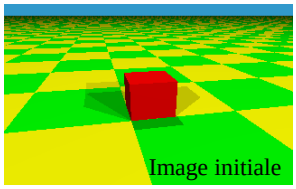
Placer la caméra

Camera {
...
location <*point*>
right <*vecteur*>
up <*vecteur*>
sky <*vecteur*>
angle *float*
blur_samples *float*
…
look_at <*point*>
}

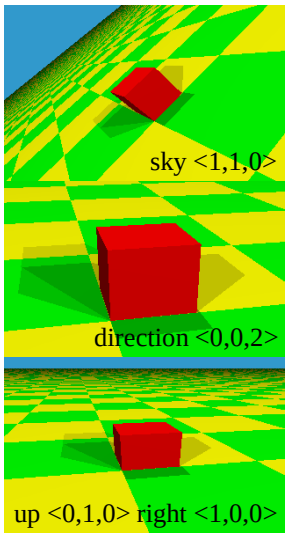


La caméra s’oriente pour « pointer » sur <**point**>
(rotation autour de <**sky**> puis autour de <**right**>)

Exemples



Modifications ?

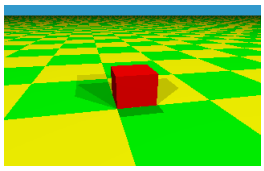


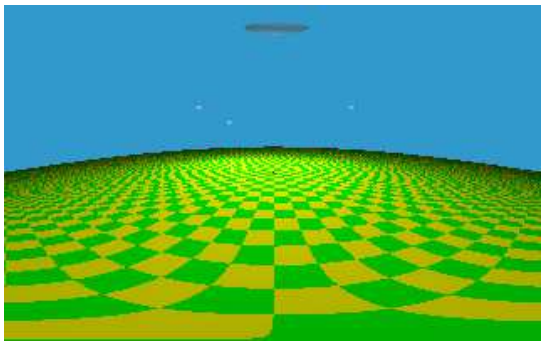
Camera {
- [perspective| orthographic
  fisheye|panoramic…]
- location *<point>*
- direction *<vecteur>*
- right *<vecteur>*
- up *<vecteur>*
- sky *<vecteur>*
- angle *float*
- blur_samples *float*
- …
- look_at *<point>*

}

L ’éclairage

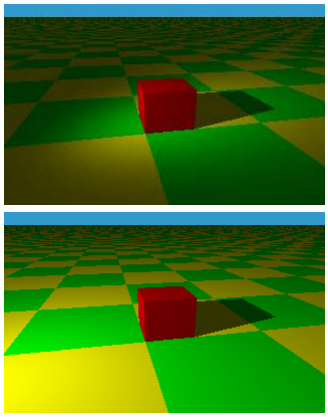
light_source {
<point> // location
color *<color>*
[looks_like {}]
}





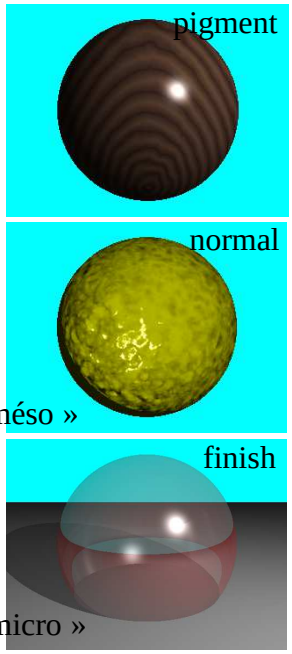
L ’éclairage


```
light_source {
<point> //location
color <color>
spotlight
point_at  <point>
radius     float
fall_off   float
…
}
```



La texture

```
texture {
   texture_identifier
   pigment {}
   normal {}
   finish {}
   transformations
}
```



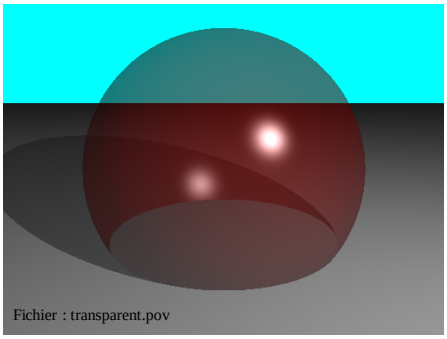
Le pigment

(texture)

Pigment {
pigment_identifier
pattern_type
pigment_modifier
}

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

pigment { color rgbt <1,0,0,0.7> }



Fichier : transparent.pov

Le pigment

(texture)

Pigment {
pigment_identifier
pattern_type
pigment_modifier }

wood

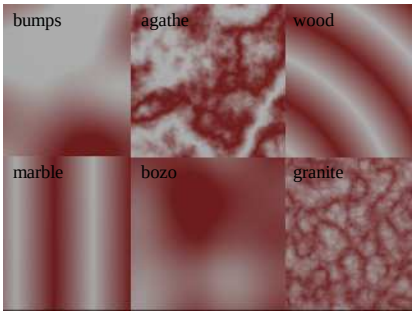
```
pigment {
   wood
   color_map {
   [0.0 color DarkTan ]
   [0.9 color DarkBrown]
   [1.0 color VeryDarkBrown]
   }
   turbulence 0.05
   scale <0.2, 0.3, 1>
   }
```

Fichier : wood.pov

Le “pattern” de pigment

Le « pattern » peut être :
un aplat : color une couleur
une répartition : agate, bozo, marble, wood, granite...
une alternance : brick ou checker 2 couleurs, hexagon 3 couleurs
une fonction : gradient x…

Le « pigment_modifier » :
color_map
pigment_map
image_map
quick_color

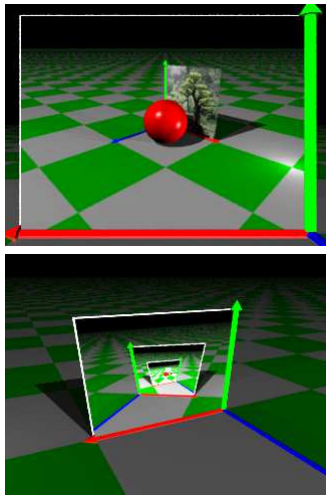


Fichier : pattern_pigment.pov

Le mapping d'une image

```
pigment {
        image_map { gif "logo.gif"
        map_type 0
        }}

pigment {
        image_map { gif "logo.gif"
        map_type 1
        filter 0, 0.5 // le fond
        filter 1, 1   // le logo
        }}
```

Mapping d'images : exemples



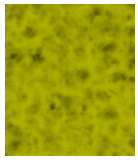
Samuel 2013

TP n°3 : projection d'images

normal

(texture)

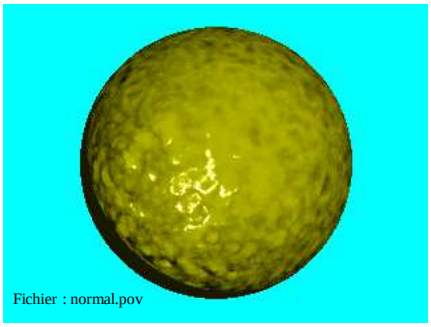
wrinkles



normal {
normal_identifier
pattern_type value
normal_modifier
transformation
}

La surface au niveau « meso »

normal {wrinkles 0.8 scale 0.2}
// rides, plis, (cratères)

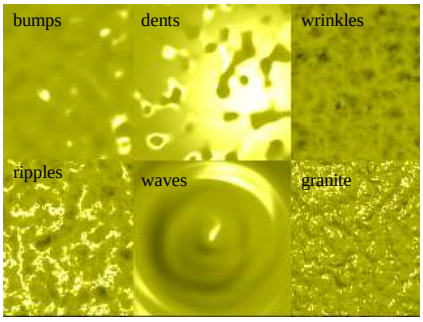


Le “pattern” de normal

Le « pattern » peut être de type :
bumps, dents (bosses), wrinkles (rides), ripples (ondulations), waves.

La valeur qui suit le pattern donne l ’intensité de la perturbation.

Le « normal_modifier » :
normal_map
slope_map
bump_map

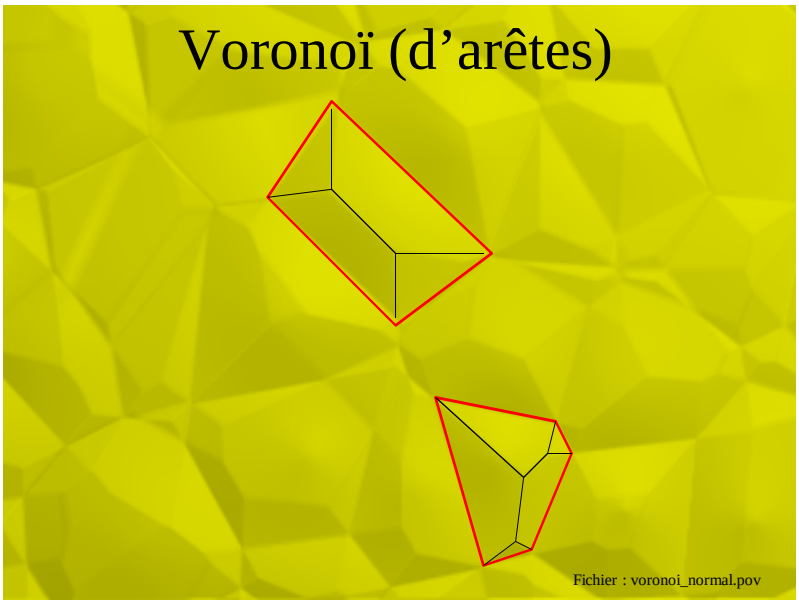


Fichier : pattern_normal.pov

La texture : les patterns

- 24 patterns (Version 3. de PovRay):
  - Agate, Average, Bozo, Brick, Bumps, Checker, Crackle,
  - Dents, Gradient, Granite, Hexagon, Leopard, Marble,
  - Onion, Quilted, Radial, Ripples, Spiral1, Spiral2, Spotted,
  - Waves, Wood, Wrinkles.
- utilisables pour :
  - « pigment »
  - « normal »
  - « texture_map »
- Modèles mathématiques :
  - Voronoi pour Crackle, Fractales pour Mandel,
  - du bruit pour granite wrinkles et agate…
  - répartitions régulières pour wood, waves, marbles, léopard...

Voronoï (d’arêtes)



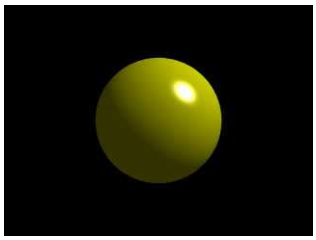
Finish (l'essentiel)

finish {
finish_identifier
...
[phong *float*]
[phong_size *float*]

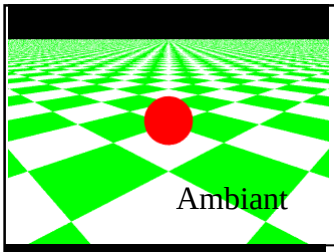
[reflection *float*]

...
}

La surface au niveau « micro »



finish permet de préciser:
- les réflexion diffuses
- la lumière d ’ambiance
- les reflets
- les réflexions spéculaires
- les réfractions

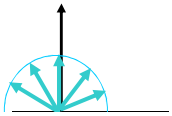


Diffuse la lumière ambiante

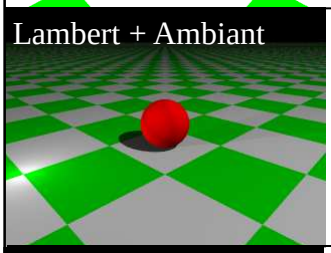
« Radiosité »

$$I_i = K_{ai} I_a$$

Réflexion



Pour K_{ai}=1



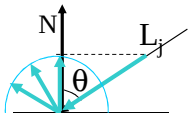
Réflexion diffuse

$$I_d = \max\left(K_d \sum_{sources} \langle N, L_j \rangle I_j, 0\right)$$

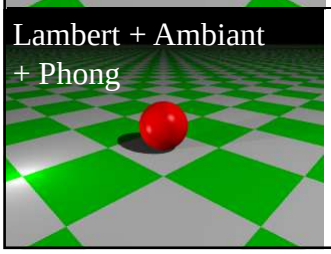
N normale à la surface

Lj rayon de la lumière j

Matériau Mat



Pour K_d=1

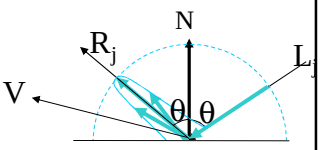


Réflexion spéculaire

$$I_s = \sum_{sources} K_s \langle R_j, V \rangle^n I_j$$

$$R_j = L_j - 2\langle N, L_j \rangle N$$

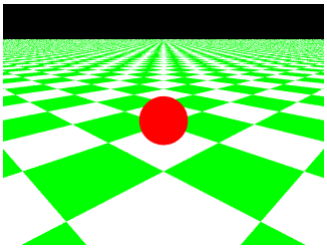
Brillant/Lisse



Pour K_s=1

Éclairage ambiant avec POV

```
#declare MYOBJ = object {
sphere { <0.5, R, 0.5>, R }
pigment {color rgb <1,0,0>}
finish { ambient 1 }
}
```



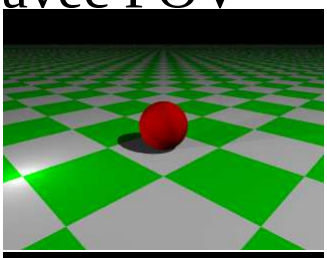
Intensité diffuse avec POV

```
#declare MYOBJ = object {
sphere { <0.5, R, 0.5>, R }
pigment {color rgb <1,0,0>}
}
```

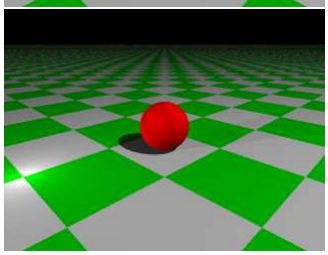
Par défaut :

diffuse = 0.6

ambient = 0.1



```
#declare MYOBJ = object {
sphere { <0.5, R, 0.5>, R }
pigment {color rgb <1,0,0>}
finish { ambient 0.3}
}
```

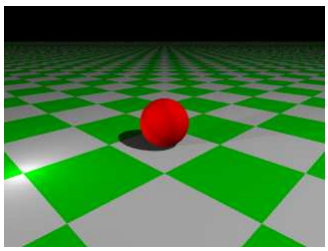


```
global_settings{  radiosity{…} }
```


Intensité spéculaire avec POV

```
#declare MYOBJ = object {
sphere { <0.5, R, 0.5>, R }
pigment {color rgb <1,0,0>}
finish { ambient 0.3}
}
```

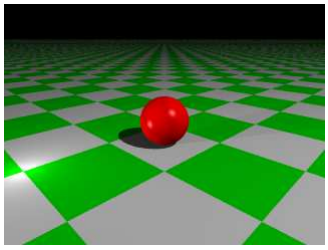
Lambert
+ Ambiant



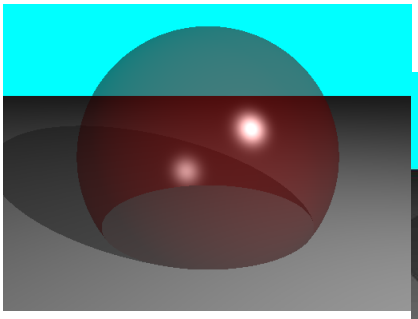
// I=Ks <Rj,V> ^n I=phong* <Rj,V> ^phong_size*

```
#declare MYOBJ = object {
sphere { <0.5, R, 0.5>, R }
pigment {color rgb <1,0,0>}
finish { ambient 0.3
   phong 0.5         // amount [0.0]
   phong_size 40     // default value (plastic)
} }
```

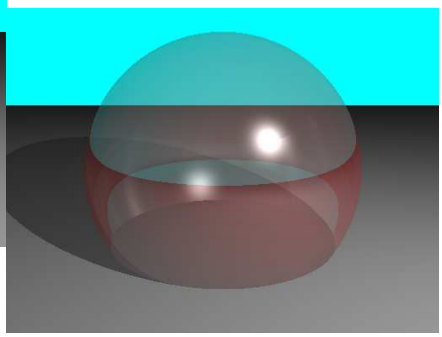
Lambert
+ Ambiant
+ Phong



Finish : réflexion (spéculaire)



Réflexion spéculaire



```
finish {
        specular { 1.0 }
        diffuse 0
        ambient 0 }
```

= *miroir parfait*

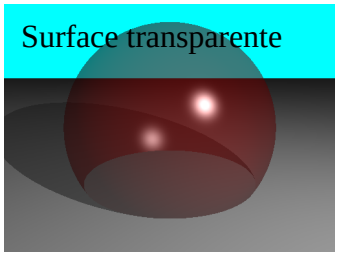
La réfraction dans Pov: interior

Objet
- Surface (texture : opaque/transparent)
- Intérieur (**interior**)
  - plein (mais transparent)
  - vide (**hollow**)

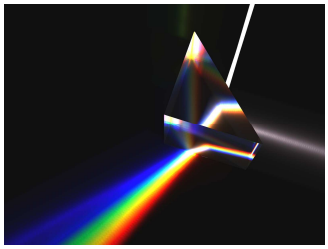
```
object { MyObject pigment {Clear } interior { ior 1.5 } }
```

ior = indice of refraction
- = 1 (pas de réfraction)
- = 1.33 pour l'eau (air/eau)
- = 1.5 pour le verre
- = 2.4 pour le diamant

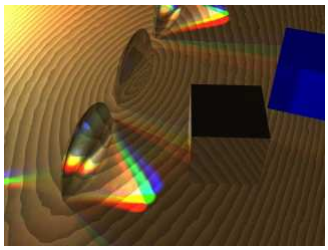
Surface transparente



Interior : dispersion



dispersion = rapport des indices de réfraction entre le violet et le rouge
= 1 (pas de dispersion)
= 1.01 à 1.1

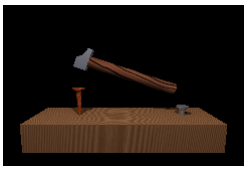


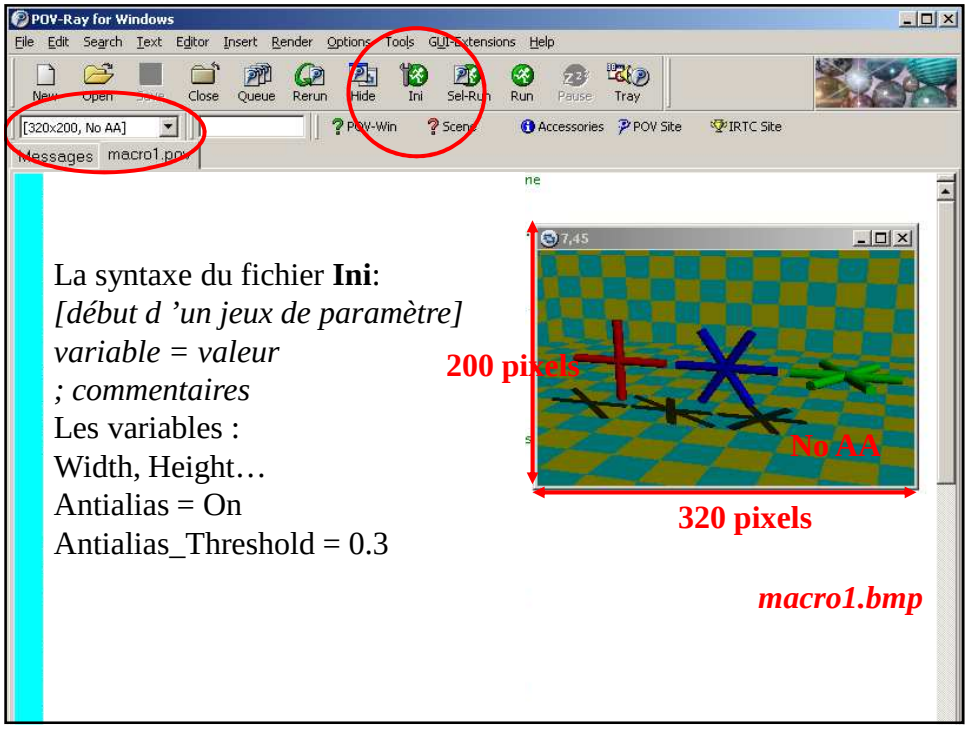
dispersion_samples = décomposition du spectre en
= 7 (par defaut)
= 2 à > 100

fade_ = atténuation de la lumière avec la distance….

$$attenuation = \frac{2}{1 + \left(\frac{d}{fadeXdistance}\right)^{fadeXpower}}$$

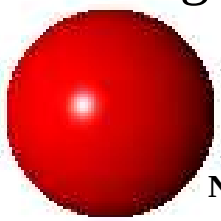
IV. Rendu, animation & logiciels



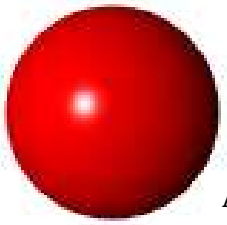


Adapter le rendu : antialiasing

// Activation
Antialias = On
// Seuil (de 0 à 3)
Antialias_Threshold = 0.3
// méthode récursive :
Sampling_Method =2
Antialias_Depth = 3 (max=9)
// ou méthode adaptative :
Sampling_Method =1
Antialias_Depth = 3 (max=9)



NA



AA 0.3

Animation

***scene*.pov**
Utilisez la variable : **clock**
Pour du paramétrage :
de positions/orientations
de textures, couleur...

```
#declare Angle = 70+30*clock;
…
rotate <0,0,Angle>
```

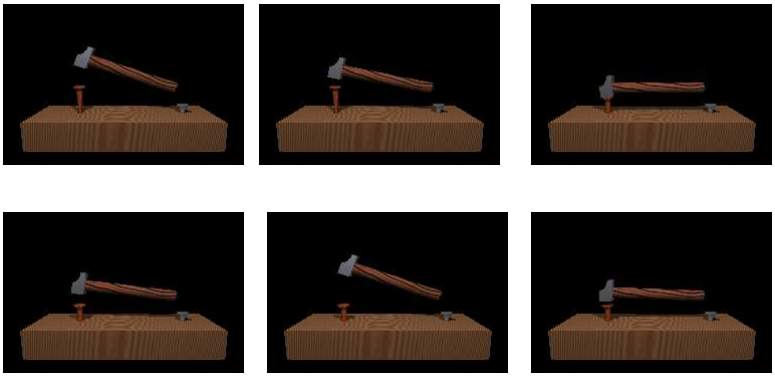
***scene*.ini**

Initial_Frame = 1
Final_Frame = 20

Initial_Clock = 0.0
Final_Clock = 2.0

Output_File_Name = *scene*.bmp

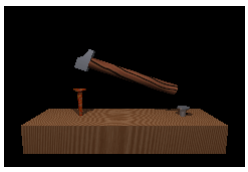
Animation



...

résultat 20 fichiers d ’image : *scene*01.bmp…*scene*20.bmp
pour des valeurs d ’horloge : 0.0, 0.105, …. , 2.0

Du bmp à la vidéo



2 logiciels pour assembler les images

Bmp2avi

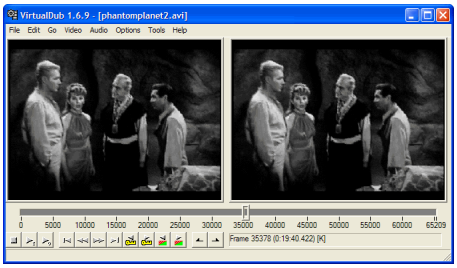
bmp2avi –o X.avi RepBitmaps/nomfichier

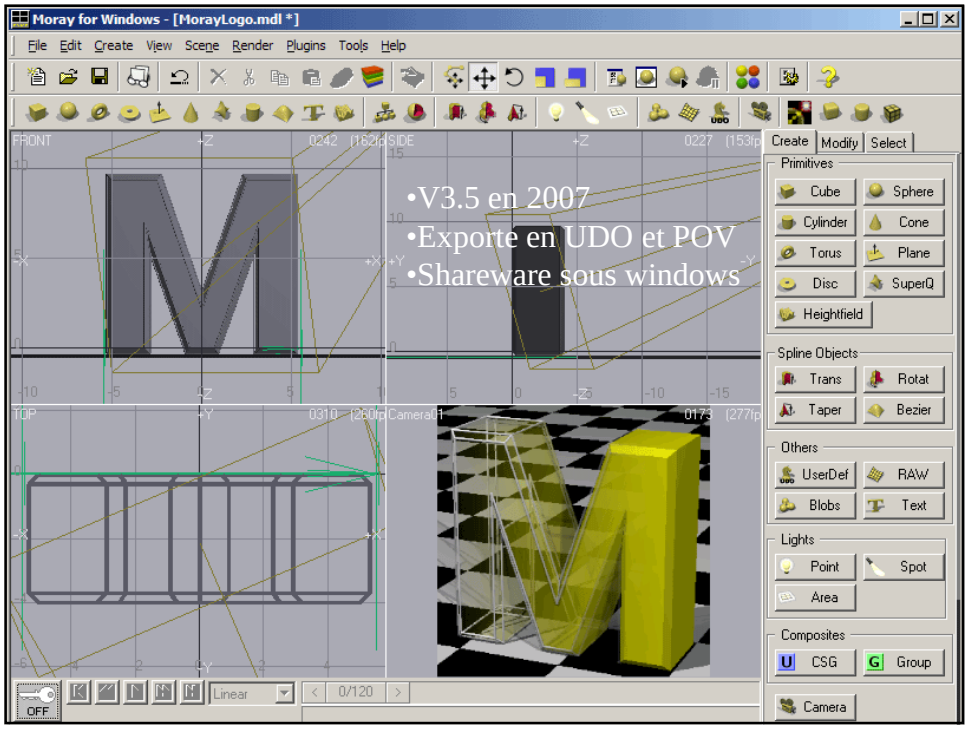
VirtualDub (GPL)

/File/Open video File (image sequence)

➔Save as AVI

➔Export/Animated Gif





DAZ 3D ➔

Sculptris ➔

Blender ➔

3DSmax® ➔

Wavefront .obj ➔

…

Convertisseur

Editeur « Matériaux »

Géométrie basique

➔ PovRay

➔ MoRay

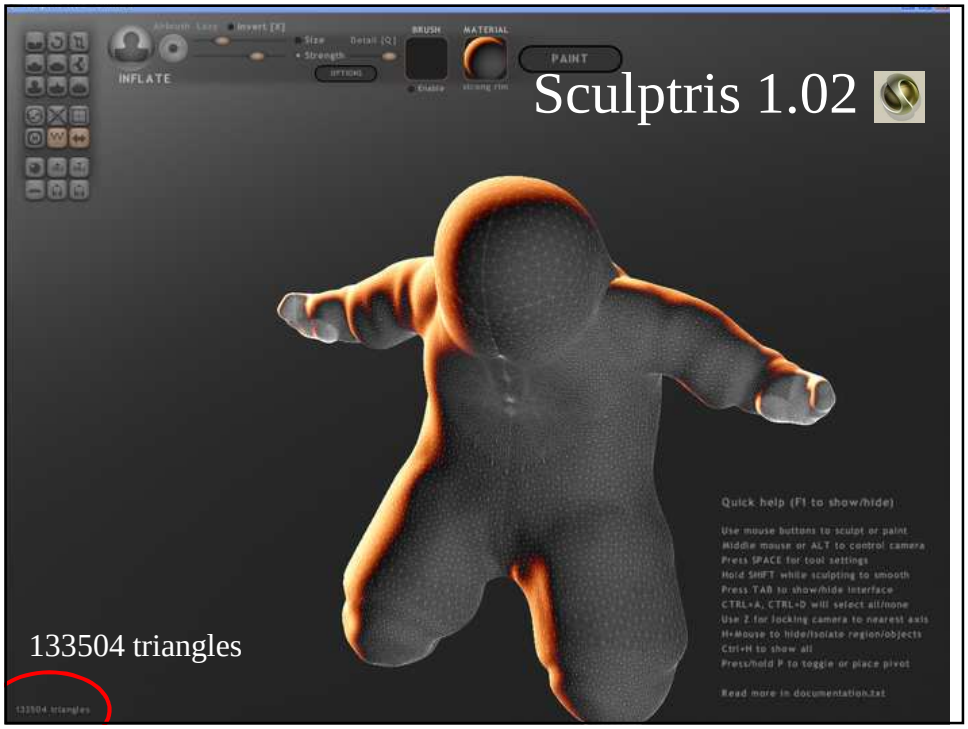
Wavefront .obj ➔ X

Sketchup ➔ PovRay

SU2POV 3.5

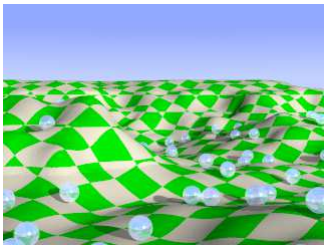
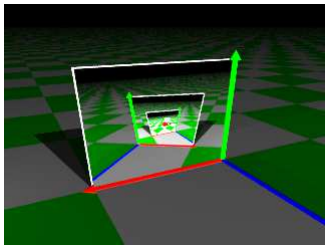
➔ PovRay

http://rhin.crai.archi.fr/rld/plugin_su2pov.php



VI. Travaux pratiques

http://people.mines-paristech.fr/olivier.stab/SIRV/

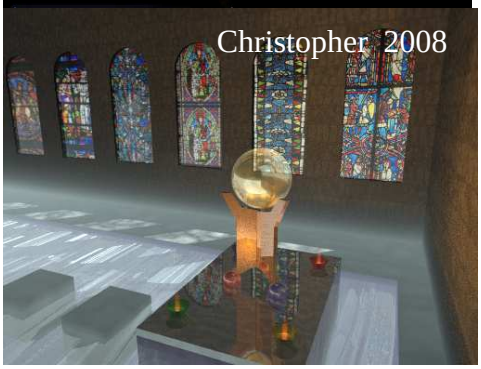

P I C H E

Grégoire 2015

VII. La galerie

Jacquemin, Guyot, Perrez 2004

Min 2014



Christopher 2008