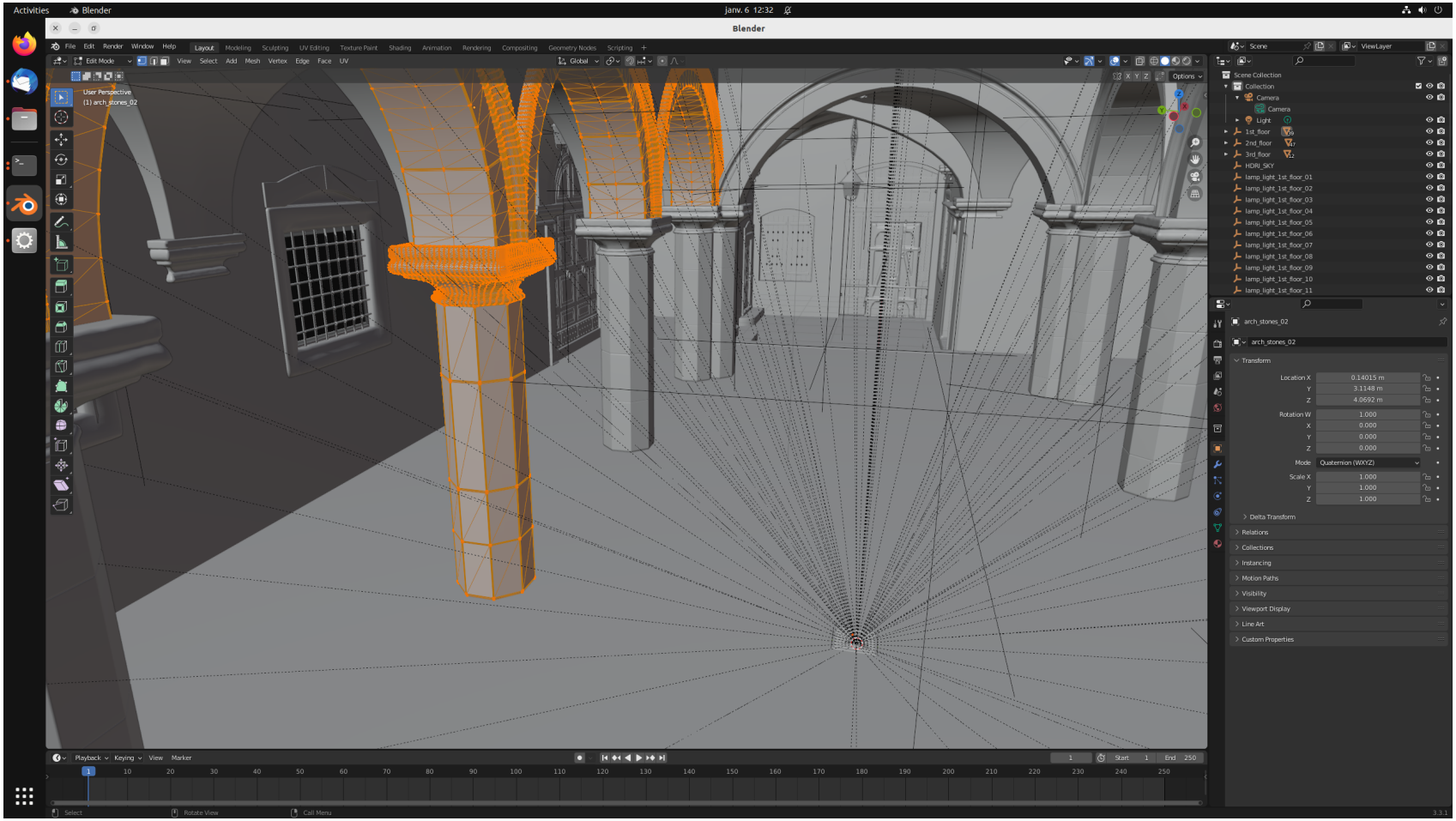
L3-Synthèse

Lancer de rayons et rendu

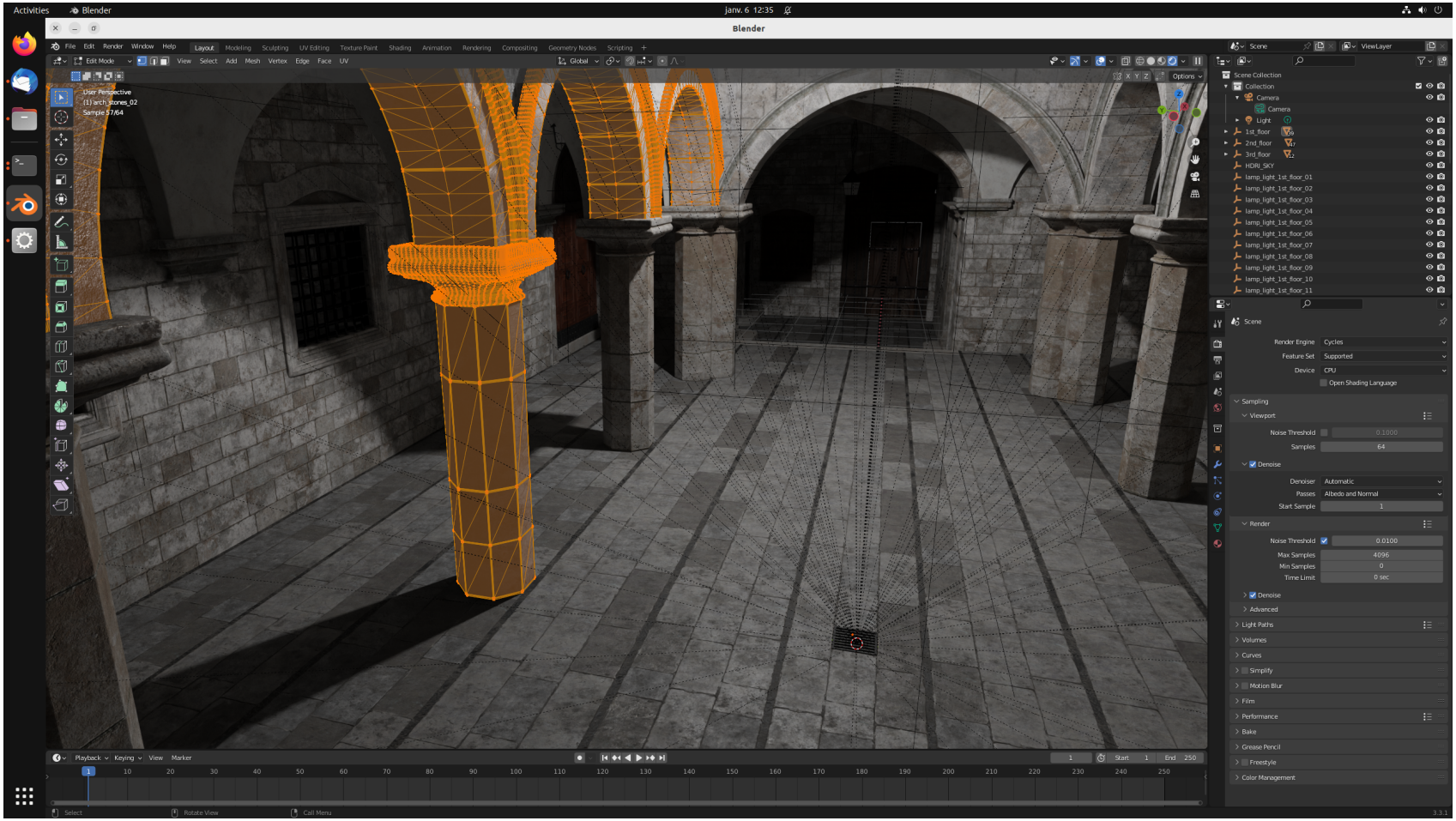
J.C. Iehl

January 19, 2023

c'est quoi ?


Activities
Blender
Janv. 6 12:32
Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Edit Mode View Select Add Mesh Vertex Edge Face UV
Global
Options
User Perspective
(1) arch_stones_02
Scene
ViewLayer
Scene Collection
Collection
Camera
Light
1st_floor
2nd_floor
3rd_floor
HDRI_SKY
lamp_light_1st_floor_01
lamp_light_1st_floor_02
lamp_light_1st_floor_03
lamp_light_1st_floor_04
lamp_light_1st_floor_05
lamp_light_1st_floor_06
lamp_light_1st_floor_07
lamp_light_1st_floor_08
lamp_light_1st_floor_09
lamp_light_1st_floor_10
lamp_light_1st_floor_11
arch_stones_02
Transform
Location X 0.14015 m
Y 3.1148 m
Z 4.0692 m
Rotation W 1.000
X 0.000
Y 0.000
Z 0.000
Mode Quaternion (WXYZ)
Scale X 1.000
Y 1.000
Z 1.000
Delta Transform
Relations
Collections
Instancing
Motion Paths
Visibility
Viewport Display
Line Art
Custom Properties
Playback Keying View Marker
Start 1 End 250
Select
Rotate View
Call Menu

c'est quoi ?


Activities
Blender
janv. 6 12:35
Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Edit Mode View Select Add Mesh Vertex Edge Face UV
Global
Options
User Perspective
(1) arch_stones_02
Sample 57/64
Scene
ViewLayer
Scene Collection
Collection
Camera
Light
1st_floor
2nd_floor
3rd_floor
HDRI_SKY
lamp_light_1st_floor_01
lamp_light_1st_floor_02
lamp_light_1st_floor_03
lamp_light_1st_floor_04
lamp_light_1st_floor_05
lamp_light_1st_floor_06
lamp_light_1st_floor_07
lamp_light_1st_floor_08
lamp_light_1st_floor_09
lamp_light_1st_floor_10
lamp_light_1st_floor_11
Render Engine Cycles
Feature Set Supported
Device CPU
Open Shading Language
Sampling
Viewport
Noise Threshold
Samples 64
Denoise
Denoiser Automatic
Passes Albedo and Normal
Start Sample 1
Render
Noise Threshold 0.0100
Max Samples 4096
Min Samples 0
Time Limit 0 sec
Denoise
Advanced
Light Paths
Volumes
Curves
Simplify
Motion Blur
Film
Performance
Bake
Grease Pencil
Freestyle
Color Management
Playback Keying View Marker
Start 1 End 250
Select
Rotate View
Call Menu

c'est quoi ?

en résumé :

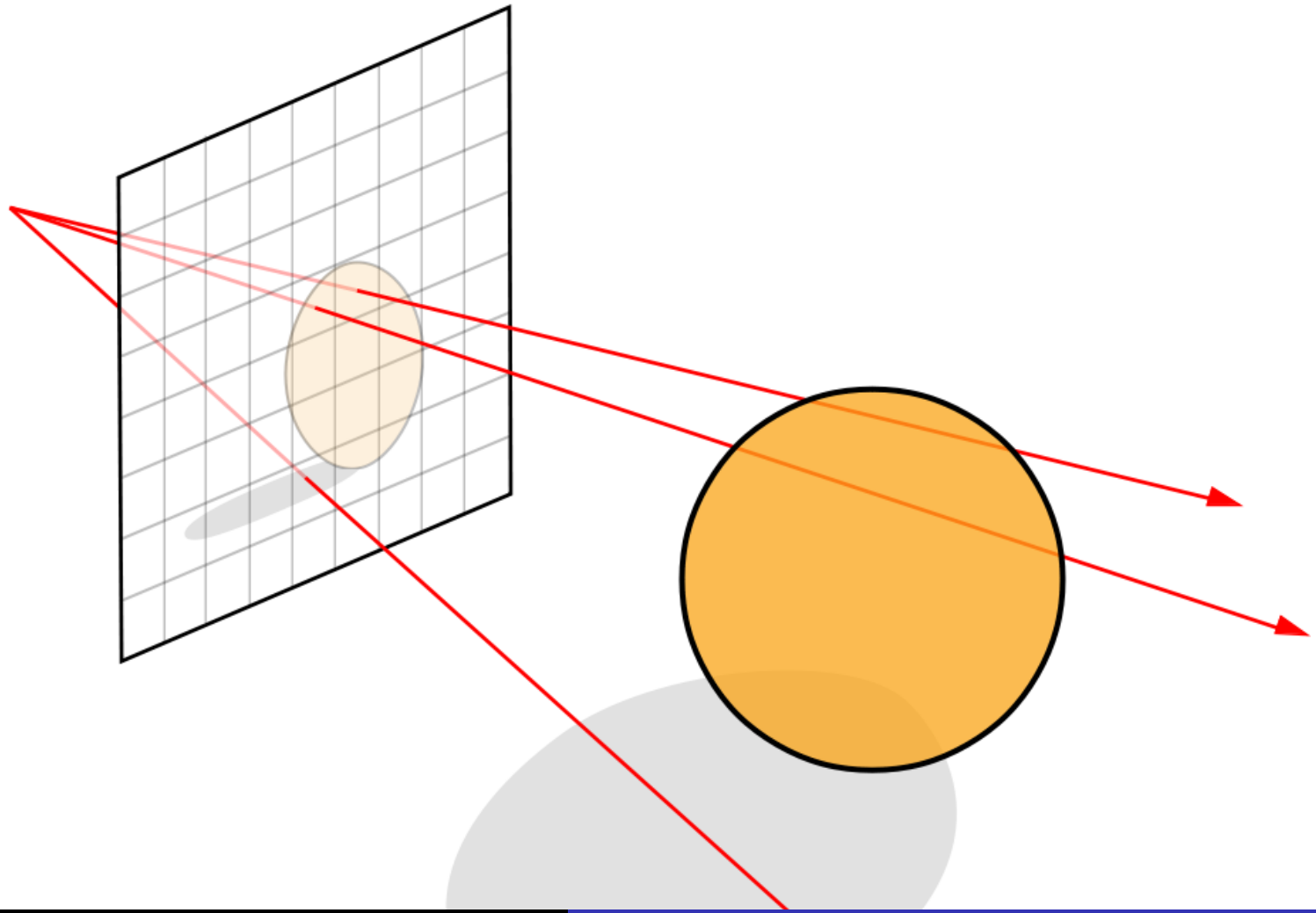
- construire une image,
- à partir d'un ensemble d'objets,
- (observés par une camera)
- (et éclairés par une ou plusieurs lumières)

comment ça marche ?

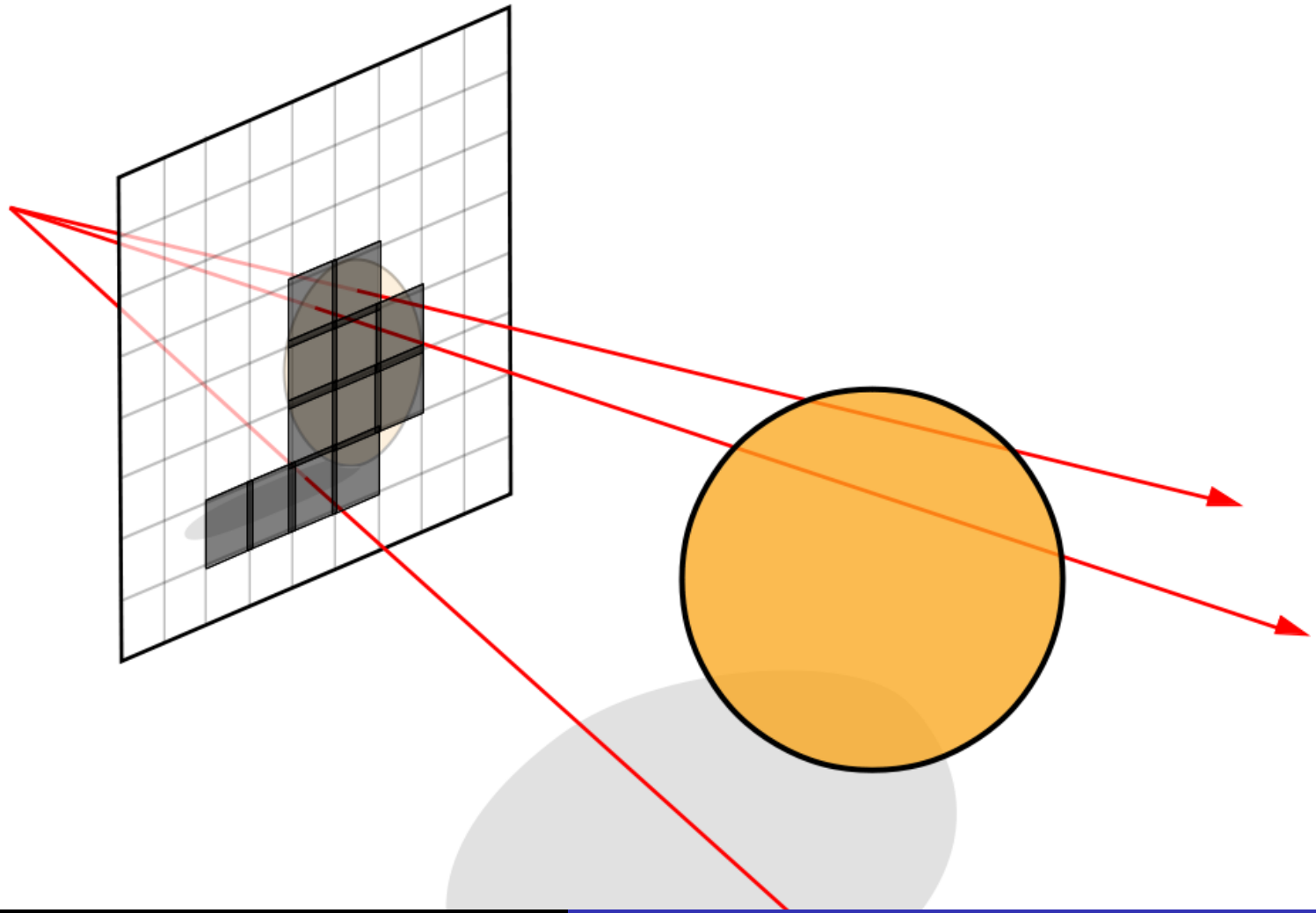
constuire une *image* :

- un ensemble de *pixels*,
- pour chaque pixel :
- trouver l'objet visible,
- trouver comment il est éclairé,
- calculer sa couleur...

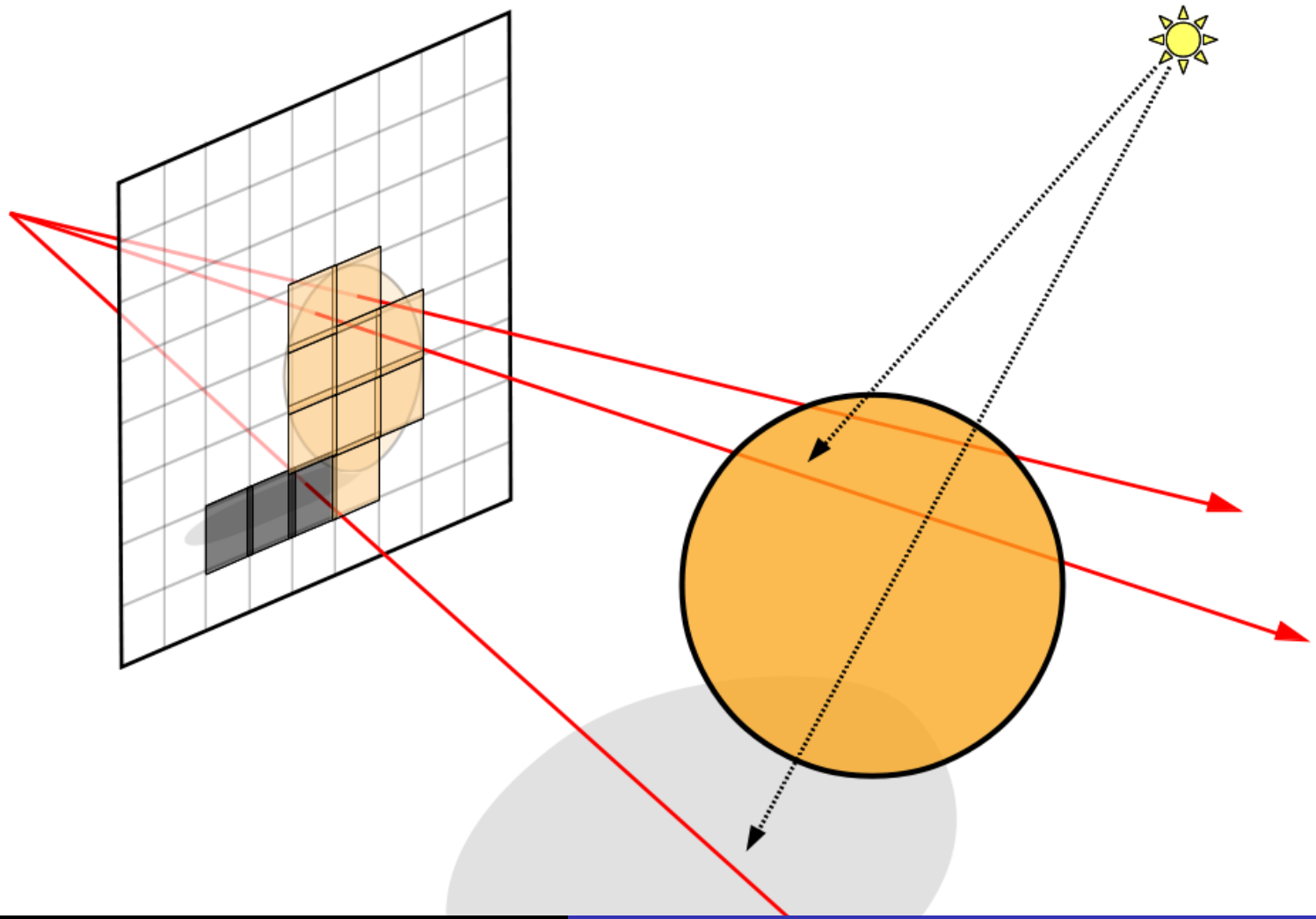
comment ça marche ?



comment ça marche ?



comment ça marche ?



comment ça se code ?

```
#include "color.h"
#include "image.h"
#include "image_io.h"

int main( )
{
    Image image(1024, 640);

    for(int py= 0; py < image.height(); py++)
    for(int px= 0; px < image.width(); px++)
    {
        Color pixel;
        // trouver l'objet visible pour le pixel
        // trouver comment il est eclaire
        // calculer sa couleur
        image(px, py)= pixel;
    }

    write_image(image, "image.png");
    return 0;
}
```

quelques détails à régler...

trouver l'objet visible ?

- soit on utilise une carte graphique avec openGL, par exemple, mais c'est assez pénible, cf cours de M1 et M2,
- soit on programme tout, c'est techniquement plus simple,
- même s'il faut manipuler pas mal de détails pour obtenir le résultat...

quelques détails à régler...

trouver l'objet visible :

- pour un pixel...
- ?
- facile, il se trouve sur la droite qui passe par le pixel,
- s'il y a plusieurs objets, on garde le premier / le plus proche du pixel (plutot de la *camera*)

droite qui passe par le pixel...

une droite ?

- comment décrire une droite ?
- avec 2 points ou 1 point et 1 direction,
- ?
- il va falloir aussi décrire une camera, comment on projette des objets 3d sur une image 2d...

camera

par convention :

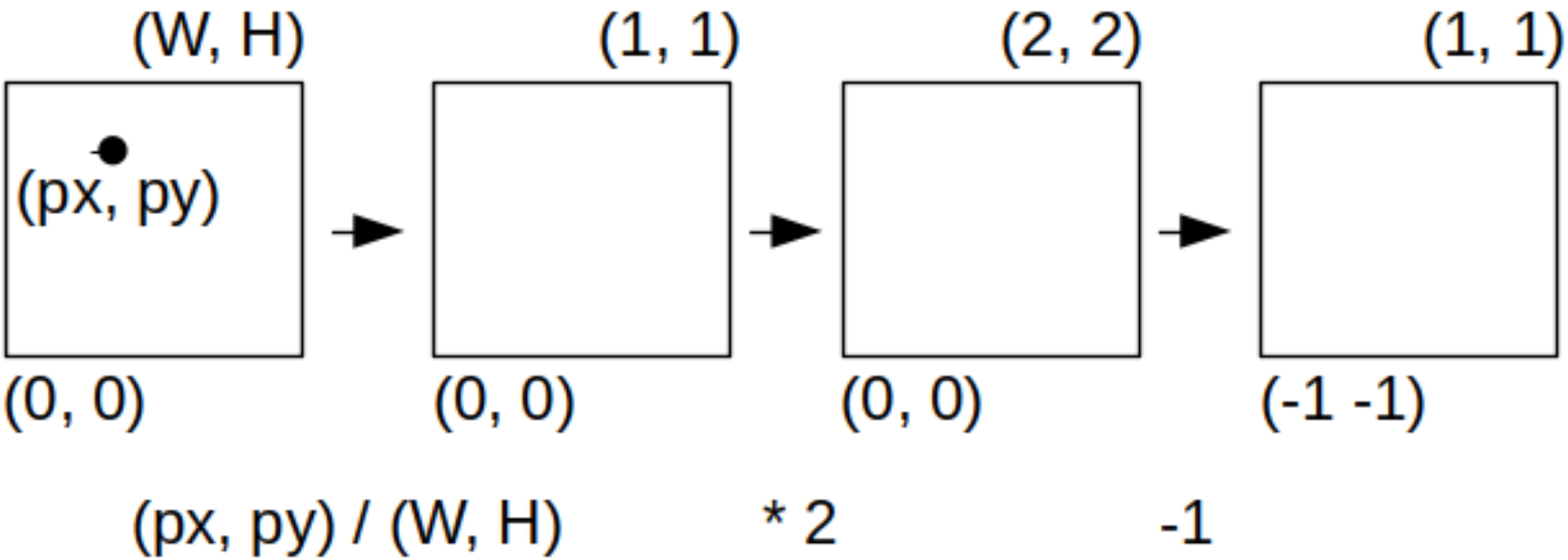
- la camera est placée à l'origine d'un repère,
- elle regarde dans la direction -z,
- le plan image, est un carre [-1 1] placé à z= -1
- ??
- pour décrire la droite qui passe par un pixel, on a besoin de 2 points :
- tous les rayons passent par (le centre de projection de) la camera,
- reste à calculer la position d'un pixel dans le plan image...

pixel et plan image

par convention :

- les points du plan image correspondent aux pixels de l'image,
- une image de resolution WxH : H lignes de W pixels,
- le point (-1 -1) du plan image correspond au pixel (0, 0) en bas à gauche de l'image,
- le point (1 1) du plan image correspond au pixel (W, H) en haut à droite de l'image,
- quel point du plan image correspond au pixel (p_x, p_y) ? (avec z= -1, par convention)

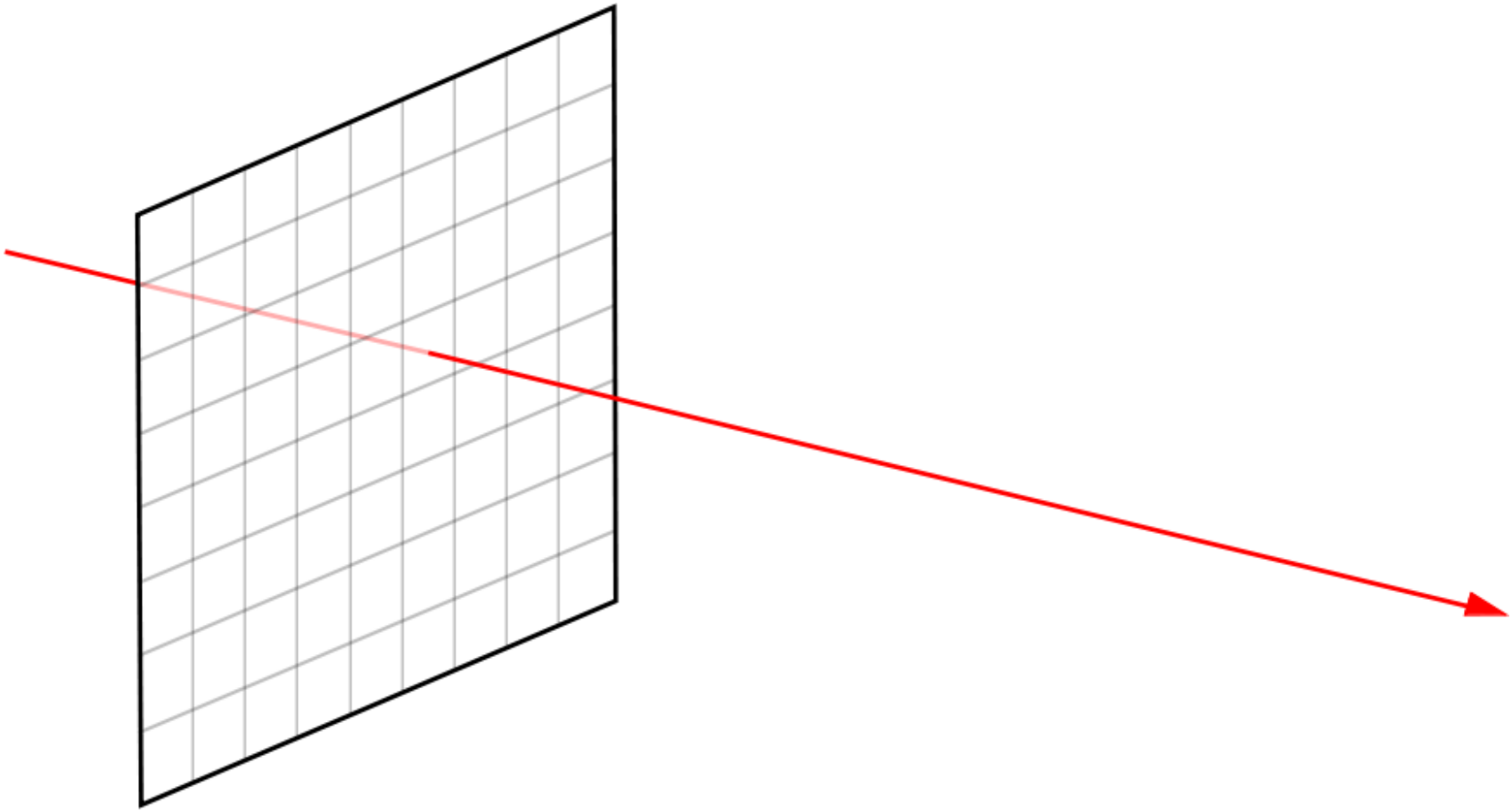
pixel et plan image



comment ça se code ?

```
#include "vec.h"

for(int py= 0; py < image.height(); py++)
for(int px= 0; px < image.width(); px++)
{
// trouver l'objet visible pour le pixel (px py)

    // point (x y z) du plan image
    float x= float(px) / float(image.width()) * 2 -1;
    float y= float(py) / float(image.height()) * 2 -1;
    float z= -1;

    // droite (o e) passant par le pixel (px py)
    Point o= Point(0, 0, 0);
    Point e= Point(x, y, z);
    Vector d= Vector(o, e);
}
```

comment ça marche ?



droite ou rayon ?

droite qui passe par le pixel :

- on connait 2 points, o l'origine / la camera,
- et e sur le plan image / le pixel de l'image,
- ou sont les autres points de la droite ?

le point p à la position t sur la droite peut s'écrire :
$p(t) = o + t \cdot \vec{d}$ (avec $\vec{d} = e - o$) ou
$p(t) = o + t \cdot (e - o) = (1 - t) \cdot o + t \cdot e$

rayon !

on utilise plutot :

- $p(t) = o + t \cdot \vec{d}$
- si $t < 0$, le point se trouve avant l'origine de la droite (derrière l'origine),
- si $t > 0$, le point est après l'origine,
- si $t = 0$, le point est sur l'origine...

pourquoi t ? lorsque plusieurs objets se trouvent sur la droite, il faut garder le plus près de l'origine, il suffit de comparer les valeurs de t...

intersection avec un rayon...

pourquoi t ?

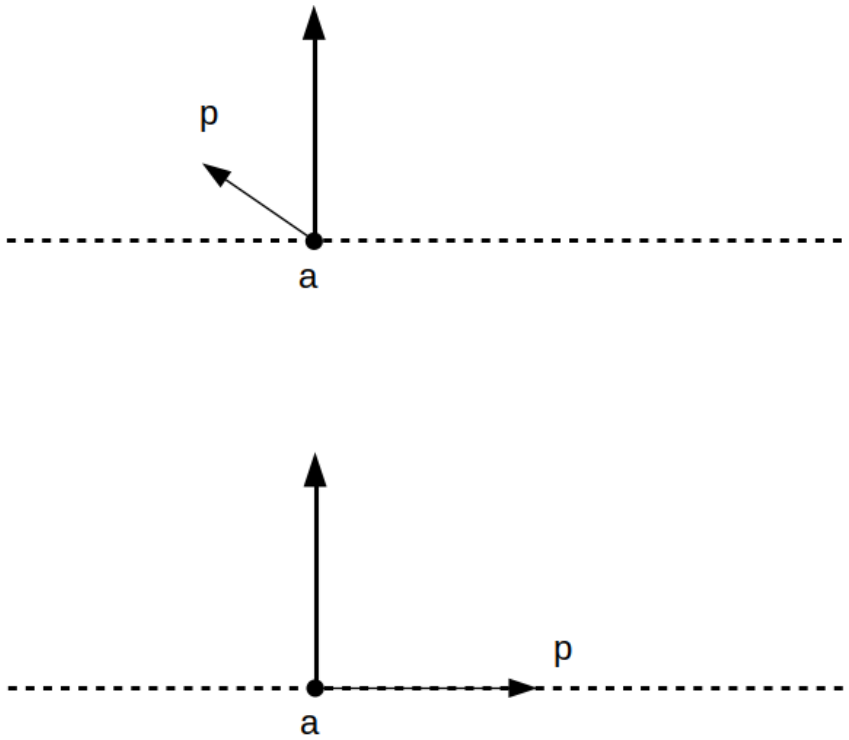
- et t représente aussi un point qui se trouve sur le rayon et à la surface de l'objet !
- c'est le point d'intersection entre le rayon et l'objet.

intersection rayon / plan

on veut calculer le point ou le rayon traverse un plan...

- c'est quoi un plan ?
- comment faire le calcul d'intersection ?
- très simplement :
- on écrit que le point sur le rayon fait aussi parti du plan, et on en déduit t !
- comment savoir qu'un point fait parti d'un plan ?
- on utilise une proprieté du produit scalaire entre 2 vecteurs :
- si le produit scalaire de 2 vecteurs est nul, les vecteurs sont perpendiculaires...

intersection rayon / plan



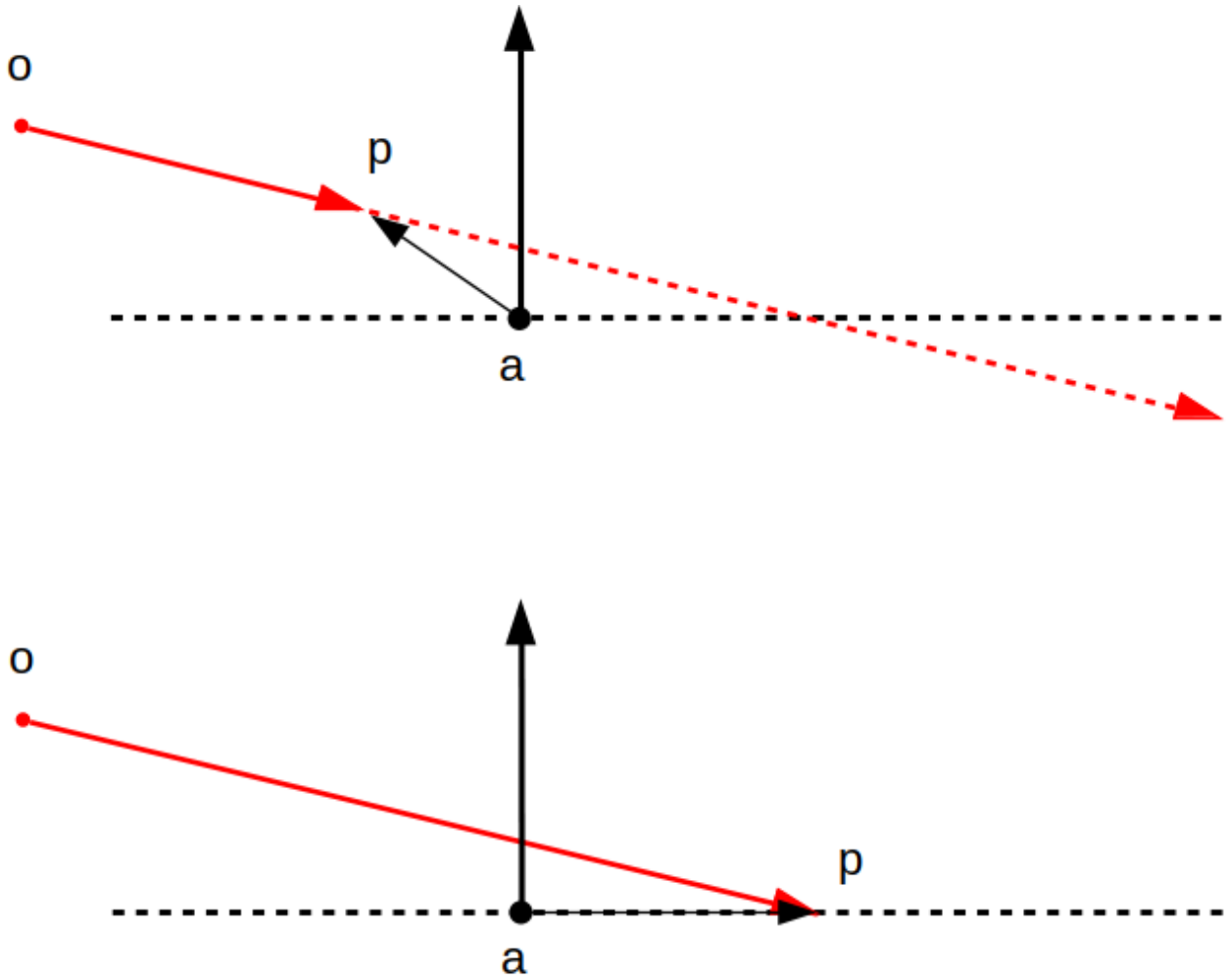
intersection rayon / plan

euh ?

- si un vecteur est la normale $\vec{n}$ du plan...
- et que l'on connait un point du plan a,
- il suffit de vérifier que le vecteur $\overrightarrow{ap}$ est perpendiculaire à n !
- si $\overrightarrow{ap} \cdot \vec{n} = 0$ le point sur le rayon est aussi dans le plan,
- il ne reste plus qu'à calculer la valeur de t !

rappel : quelques propriétés pratiques des produits scalaires et vectoriels, cf wikipedia

intersection rayon / plan



intersection rayon / plan

$$\vec{n} \cdot \overrightarrow{ap(t)} = 0$$
$$\vec{n} \cdot (o + t\vec{d} - a) = 0$$
$$\vec{n} \cdot ((o - a) + t\vec{d}) = 0$$
$$\vec{n} \cdot (\overrightarrow{ao} + t\vec{d}) = 0$$
$$\vec{n} \cdot \overrightarrow{ao} + \vec{n} \cdot t\vec{d} = 0$$
$$\vec{n} \cdot t\vec{d} = -\vec{n} \cdot \overrightarrow{ao}$$
$$t(\vec{n} \cdot \vec{d}) = -\vec{n} \cdot \overrightarrow{ao}$$
$$t = \frac{-\vec{n} \cdot \overrightarrow{ao}}{\vec{n} \cdot \vec{d}} = \frac{\vec{n} \cdot \overrightarrow{oa}}{\vec{n} \cdot \vec{d}}$$

comment ça se code ?

```
#include "vec.h"

// plan, point + normale
Point a= { ... };
Vector n= { ... };

// intersection avec le rayon o, d
float t= dot(n, Vector(o, a)) / dot(n, d);

// point d'intersection
Point p= o + t*d;
```

intersection rayon / triangle

et avec un triangle ?

- plusieurs solutions...
- la plus simple :
- on commence par calculer l'intersection entre le rayon et le plan du triangle,
- ensuite on vérifie que le point d'intersection se trouve à l'intérieur du triangle !

intersection rayon / triangle

rappels :

- normale du triangle = produit vectoriel des 2 arêtes du triangle
- longueur du produit vectoriel = 2 * aire du triangle
- triangle abc, $\vec{n} = \overrightarrow{ab} \times \overrightarrow{ac}$
- $aire(abc) = \frac{1}{2}||\vec{n}||$
- mais : attention à l'orientation des sommets...

et avec un sommet du triangle et sa normale, on peut calculer l'intersection du rayon et du plan du triangle...

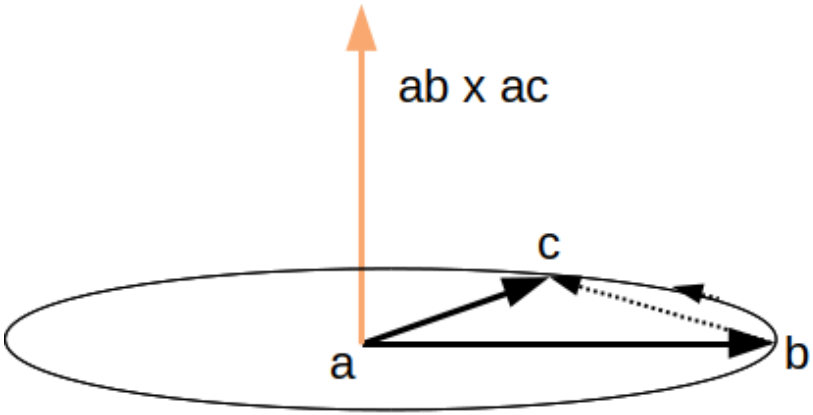
intersection rayon / triangle

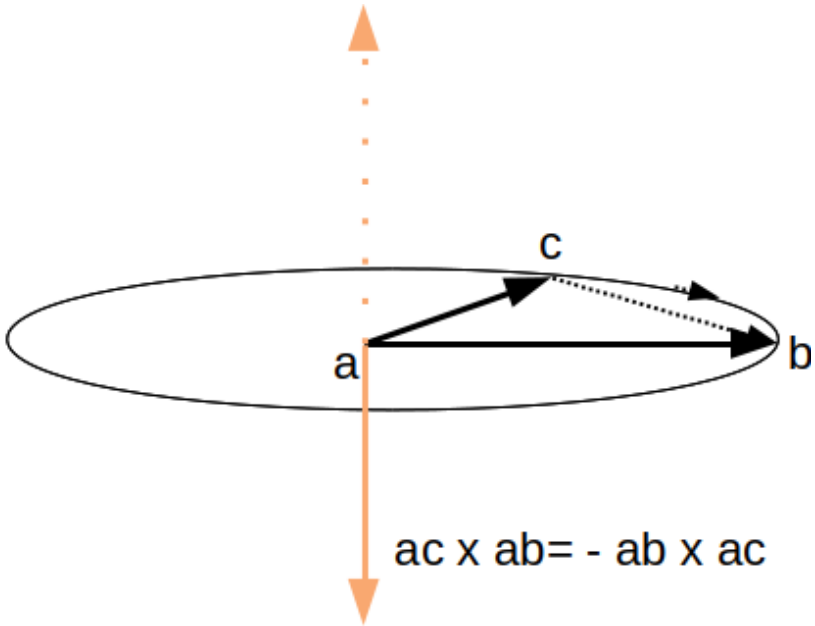
vérifier que le point est dans le triangle :

- comment ?
- comme les cartes graphiques ! vérifier que le point est du "bon" coté de chaque arête du triangle

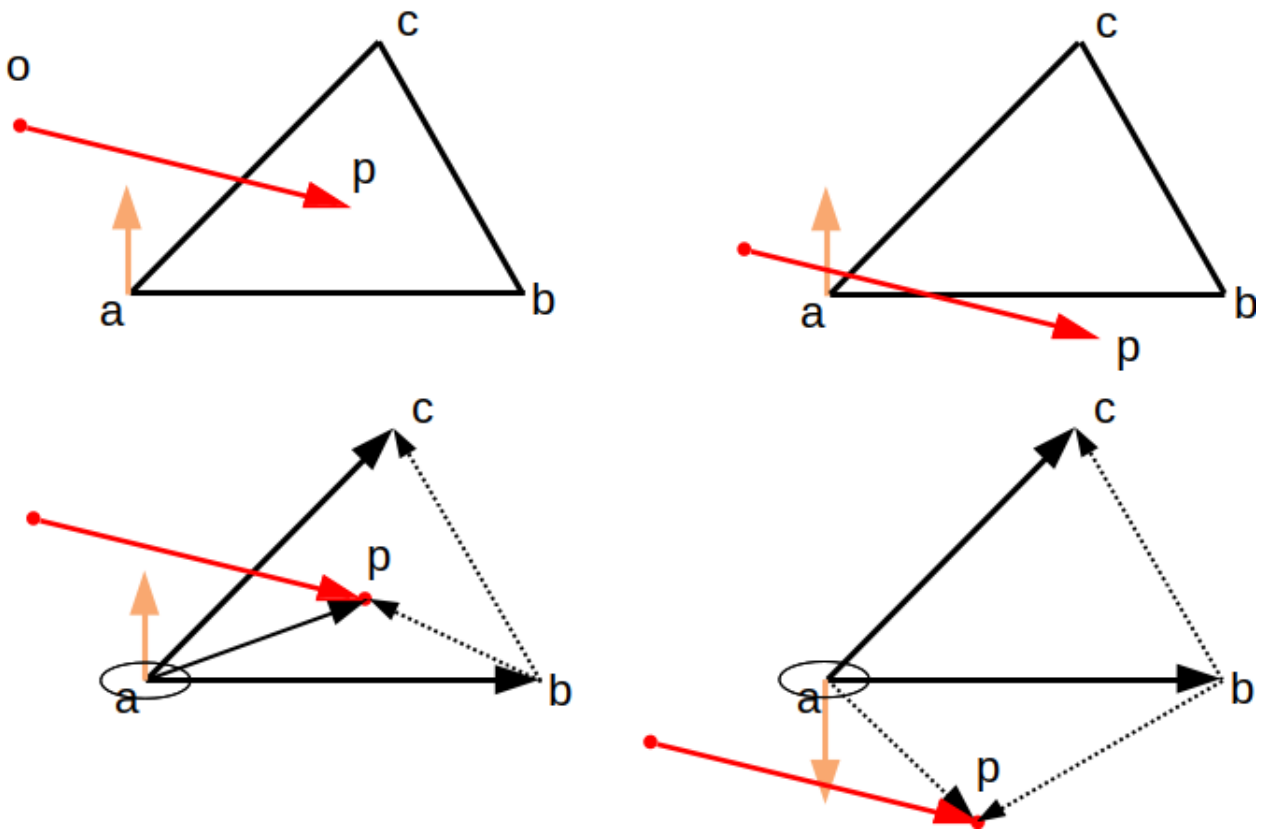
rappel : les triangles abc et acb n'ont pas les mêmes normales !!
même direction, mais sens opposés,
le produit scalaire change de signe, $(\overrightarrow{ab} \times \overrightarrow{ac}) \cdot \vec{n} = -(\overrightarrow{ac} \times \overrightarrow{ab}) \cdot \vec{n}$

intersection rayon / triangle





intersection rayon / triangle



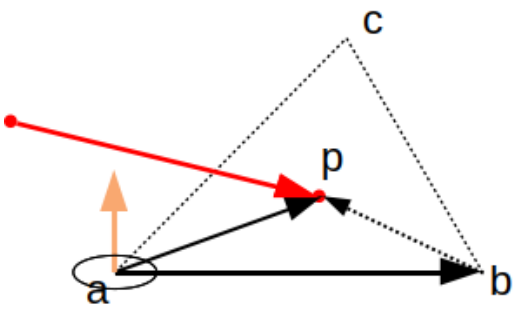
intersection rayon / triangle

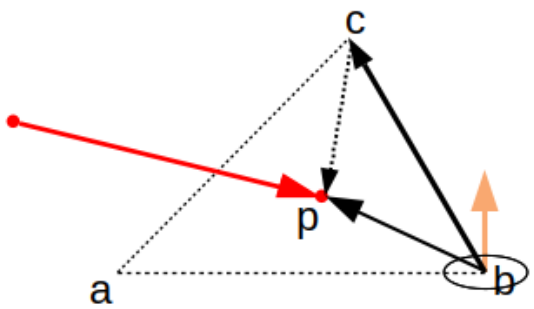
vérifier que le point est dans le triangle :

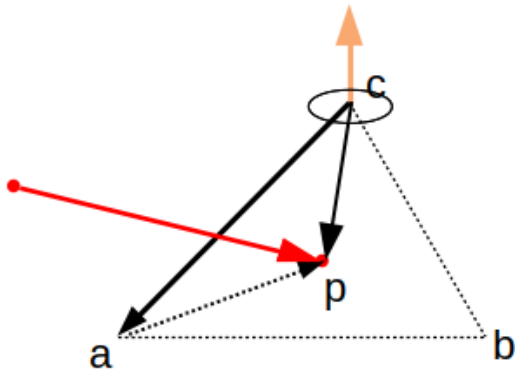
- construire un triangle entre le point p et chaque arête,
- vérifier qu'ils sont tous orientés comme le triangle abc :
- comparer les normales des triangles abp, bcp, et cap à la normale de abc.

rappel : le produit scalaire de 2 vecteurs est < 0 s'ils ne sont pas dans le même sens...

intersection rayon / triangle



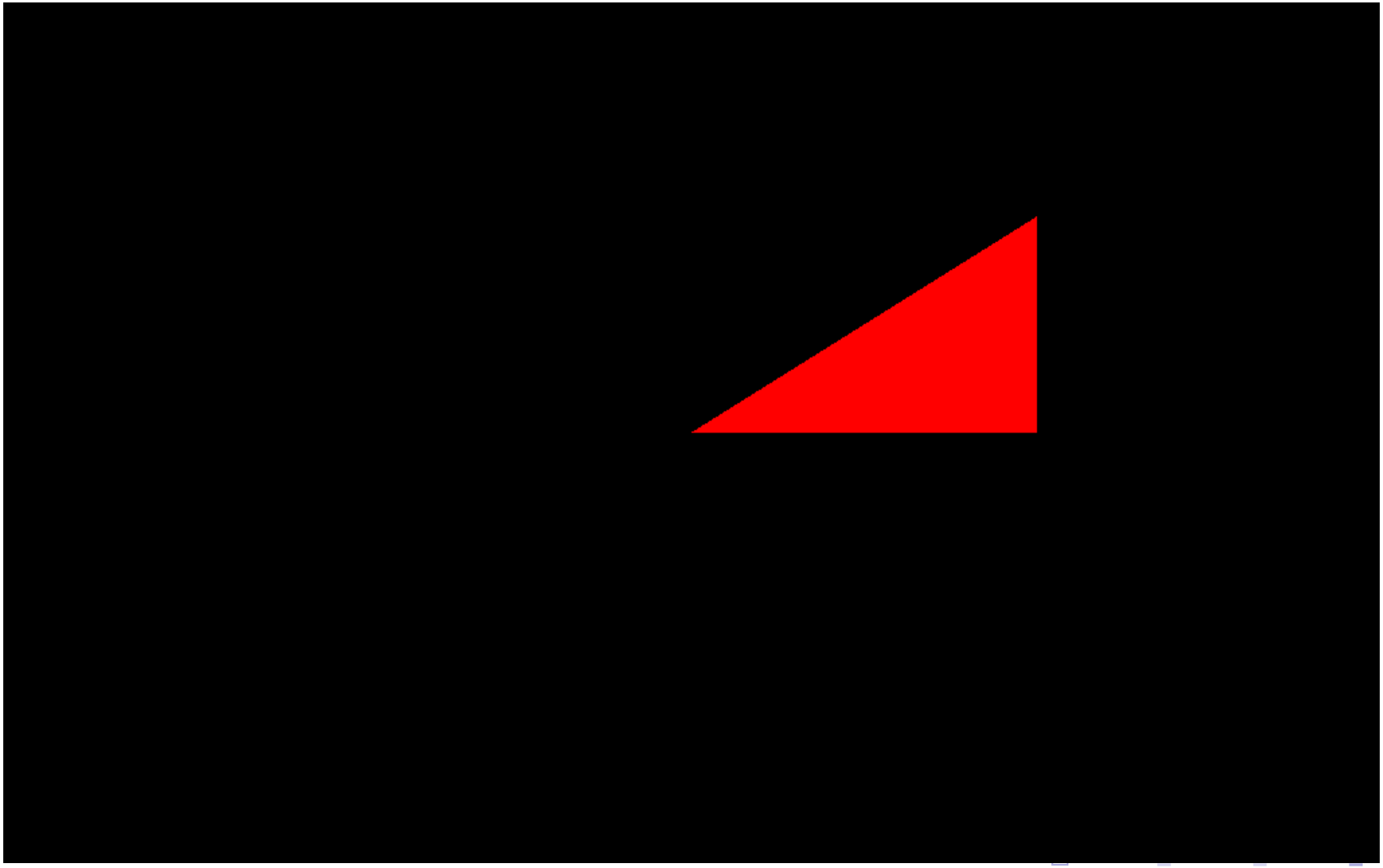




comment ça se code ?

```
#include "vec.h"

// triangle abc
Point a= { ... };
Point b= { ... };
Point c= { ... };
// et sa normale
Vector n= cross(Vector(a, b), Vector(a, c));

// intersection avec le rayon o, d
float t= dot(n, Vector(o, a)) / dot(n, d);

// point d'intersection
Point p= o + t*d;

// p a l'interieur ?
if(dot(n, cross(Vector(a, b), Vector(a, p))) < 0) return "dehors";
if(dot(n, cross(Vector(b, c), Vector(b, p))) < 0) return "dehors";
if(dot(n, cross(Vector(c, a), Vector(c, p))) < 0) return "dehors";

// les 3 tests sont positifs, p est a l'interieur...
return "touche"
```

et ça marche ?



et pour d'autres formes ?

pour d'autres formes :

- il faut décrire les points à la surface de l'objet,
- et ensuite trouver la position d'un point du rayon qui est aussi sur la surface de l'objet...

exemple : une sphère ?

intersection rayon / sphère

sphère de centre c et de rayon r :

- les points à la surface de la sphère vérifient :
  $||p - c|| = r$
- ou de manière équivalente :
  $||p - c||^2 = r^2$
- en utilisant une autre propriété du produit scalaire :
  $||\vec{v}||^2 = \vec{v} \cdot \vec{v}$
- $(p - c) \cdot (p - c) = r^2$
- il ne reste plus qu'à remplacer p par le point sur le rayon et à calculer t !

intersection rayon / sphère

$$(p(t) - c) \cdot (p(t) - c) - r^2 = 0$$
$$(o + t\vec{d} - c) \cdot (o + t\vec{d} - c) - r^2 = 0$$
$$(o - c + t\vec{d}) \cdot (o - c + t\vec{d}) - r^2 = 0$$
$$(\vec{co} + t\vec{d}) \cdot (\vec{co} + t\vec{d}) - r^2 = 0$$
$$\ldots = 0$$
$$(\vec{d} \cdot \vec{d})t^2 + (2\vec{d} \cdot \vec{co})t + \vec{co} \cdot \vec{co} - r^2 = 0$$

intersection rayon / sphère

et alors ?

- la solution est sous la forme : $at^2 + bt + k = 0$,
- il y a donc plusieurs cas possible, 2 intersections, 1 seule ou aucune,
- il suffit de calculer les racines du polynome,
- au final, c'est plutot intuitif comme résultat :
- une droite traverse la sphère et donne 2 intersections,
- ou touche la sphère en un seul point,
- ou passe complètement à coté...

intersection rayon / sphère

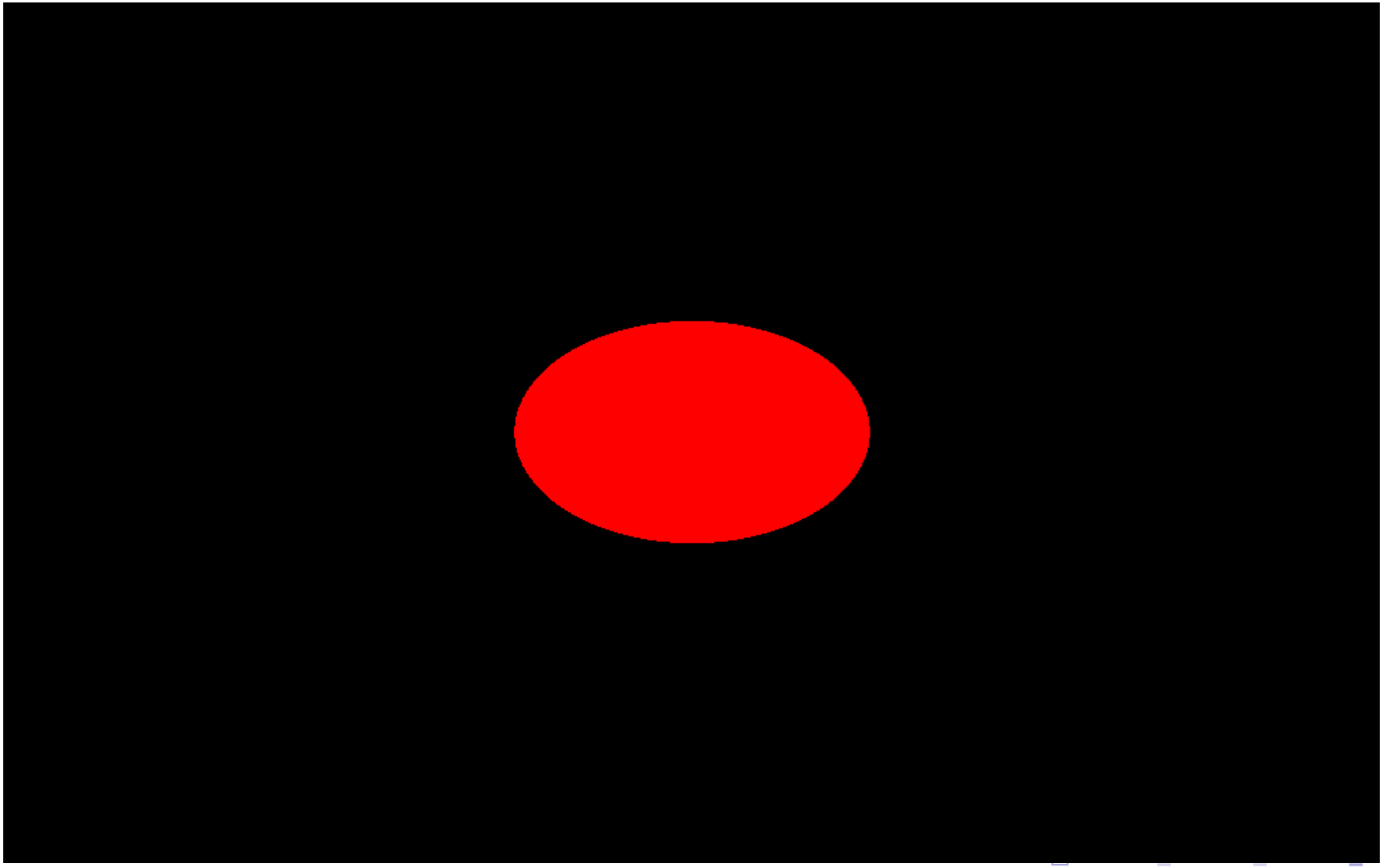
rappels :

$$a = \vec{d} \cdot \vec{d}$$
$$b = 2\vec{d} \cdot \overrightarrow{co}$$
$$k = \overrightarrow{co} \cdot \overrightarrow{co} - r^2$$

si $b^2 - 4ak < 0$, il n'y a pas de solution, le rayon passe à coté de la sphère, sinon il existe 2 solutions :

$$t_1 = \frac{-b + \sqrt{b^2 - 4ak}}{2a}$$
$$t_2 = \frac{-b - \sqrt{b^2 - 4ak}}{2a}$$

et ça marche ?

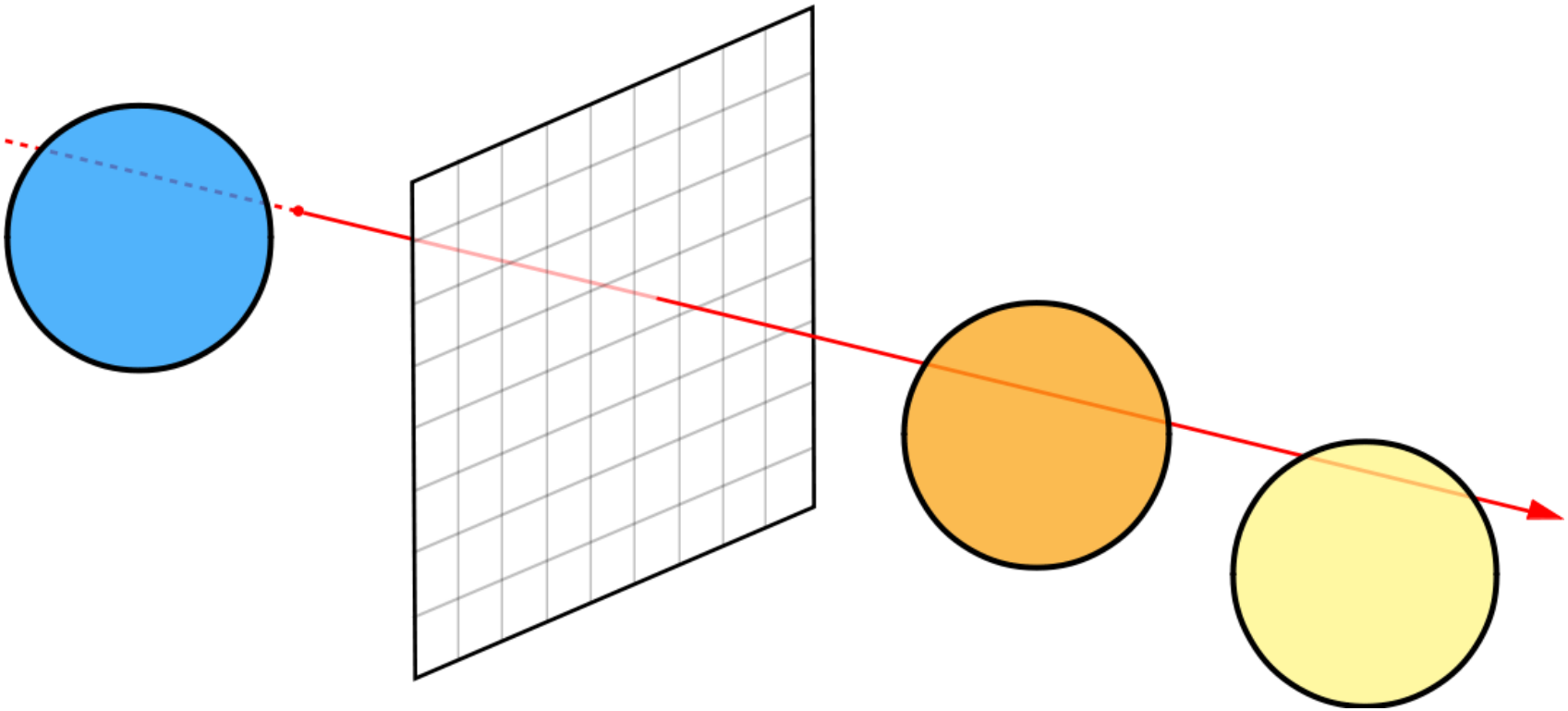


et avec plusieurs objets ?

il faut calculer toutes les intersections :

- et garder l'intersection *valide* la plus proche de la camera / de l'origine du rayon !

et avec plusieurs objets ?



et avec plusieurs objets ?

intersection valide ?

- on ne s'interresse qu'aux intersections *visibles*,
- pas à celles qui se trouvent derrière l'origine du rayon... (derrière la camera)
- si $t < 0$, l'intersection n'est pas valide.

les calculs d'intersections se font sur la droite infinie du rayon...
mais on ne garde que les intersections valides / visibles / devant...

et avec plusieurs objets ?

pour chaque objet :

- si l'intersection est valide,
- et plus proche que celle deja trouvée,
- conserver l'intersection.

on connait l'objet visible pour le pixel !!

comment ça se code ?

trouver le minimum d'un ensemble de valeurs...

```
#include <limits>
#include "color.h"

const float inf= std::numeric_limits<float>::infinity();  //  infini

for(int py= 0; py < image.height(); py++)
for(int px= 0; px < image.width(); px++)
{
  { ... }
  Color color= Black();
  float plus_proche= inf;
  for(int i= 0; i < n; i++)
  {
    float intersection= { ... };
    // calculer l'intersection avec le ieme objet

    if(intersection > 0 && intersection < plus_proche)
    {
      plus_proche= intersection;
      color= { ... };  // couleur du ieme objet
    }
  }

  image(px, py)= color;
}
```

comment ça se code ?

les intersections peuvent renvoyer *inf* pour indiquer que le rayon ne touche pas un objet... exemple avec le triangle

```
#include <limits>

const float inf= std::numeric_limits<float>::infinity();  //  infini

float intersect_triangle( ... )
{
  float t= { ... };

  if(dot(n, cross(Vector(a, b), Vector(a, p))) < 0)
    return inf; // return "dehors";

  // tester les autres aretes du triangle

  return t;     // return "touche"
}
```

comment ça se code ?

on peut aussi renvoyer inf si l'intersection n'est pas valide...

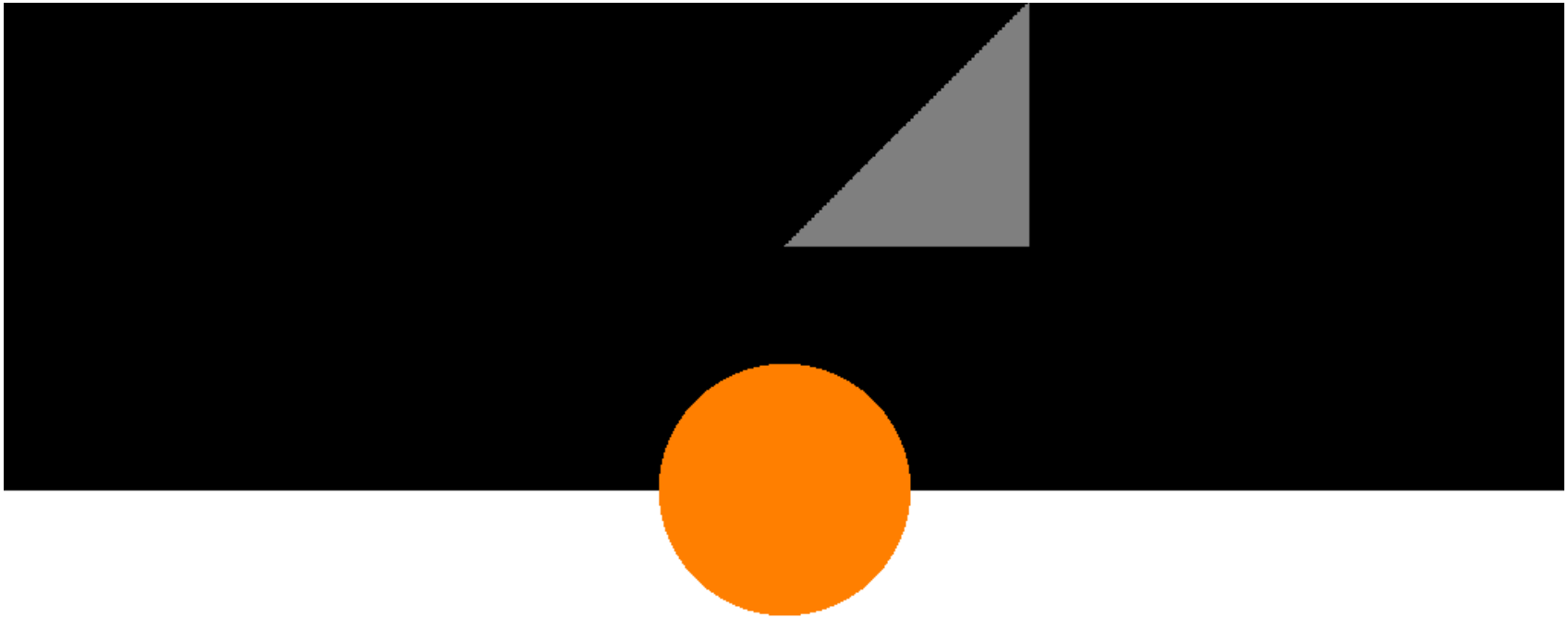
```
#include <limits>

const float inf= std::numeric_limits<float>::infinity();  //  infini

float intersect_triangle( ... )
{
  float t= { ... };

  if(dot(n, cross(Vector(a, b), Vector(a, p))) < 0)
    return inf; // return "dehors";

  // tester les autres aretes du triangle

  if(t < 0) return inf; // pas valide
  return t;     // return "touche"
}
```

comment ça se code ?

... et simplifier (un peu) la boucle

```
#include <limits>
#include "color.h"

const float inf= std::numeric_limits<float>::infinity();  // infini

for(int py= 0; py < image.height(); py++)
for(int px= 0; px < image.width(); px++)
{
  { ... }
  Color color= Black();
  float plus_proche= inf;
  for(int i= 0; i < n; i++)
  {
    float intersection= { ... };
    // calculer l'intersection avec le ieme objet, ou inf

    if(intersection < plus_proche)
    {
      plus_proche= intersection;
      color= { ... };  // couleur du ieme objet
    }
  }

  image(px, py)= color;
}
```

et ça marche ?

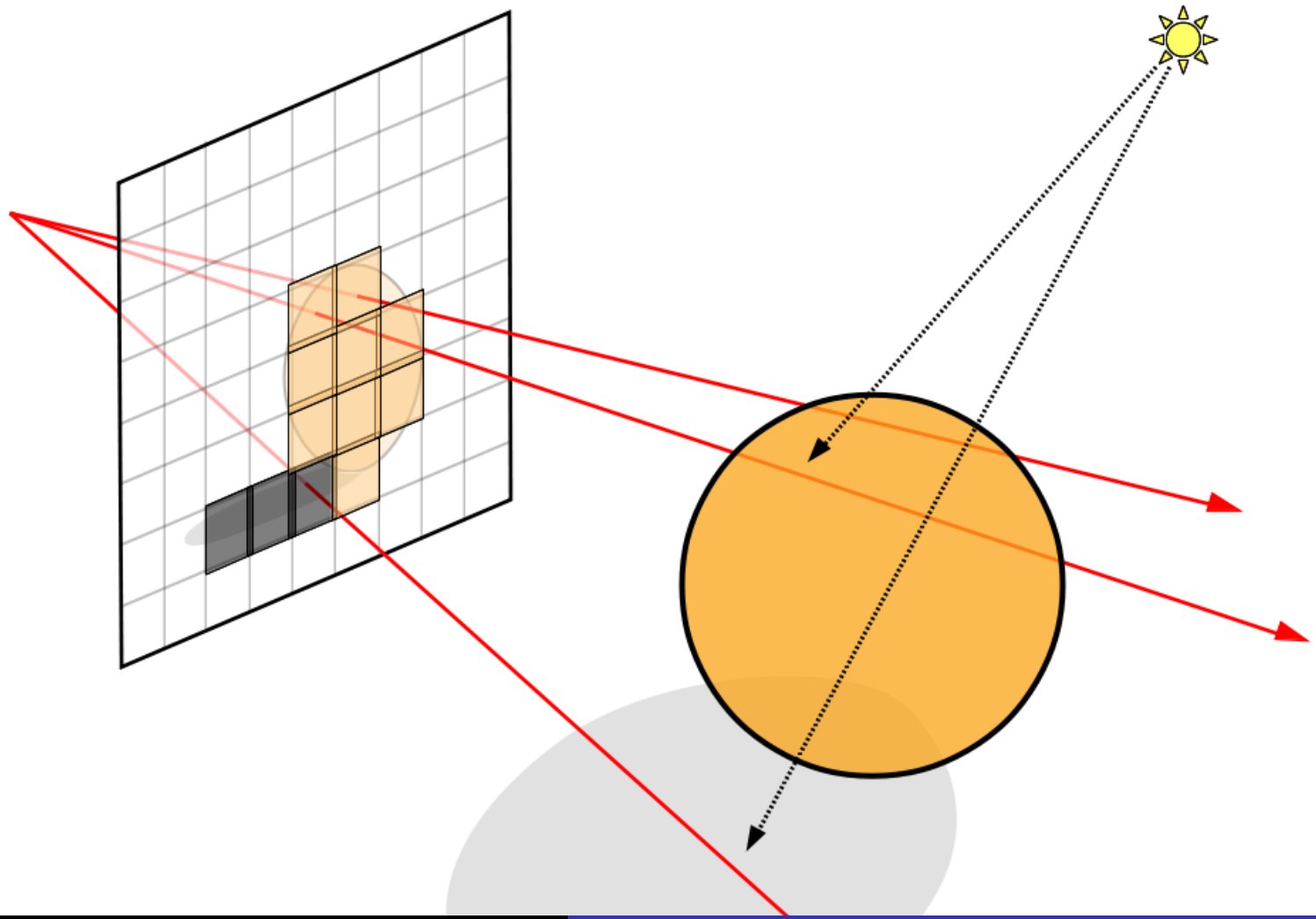


et alors ?

en résumé :

- camera qui observe des objets,
- plan image,
- 1 rayon par pixel,
- intersections,
- garder l'intersection la plus proche de la camera,
- colorier le pixel en fonction de l'intersection...

et alors ?



et alors ?

la suite :

- les ombres,
- les lumières,
- la couleur des objets éclairés / à l'ombre...

et alors ?

simplifications :

- les rayons et les objets sont décrits dans le repère de la camera,
- habituellement, cf principes du lancer de rayon,
  on place les objets et la camera dans le repère du *monde*,
- et il faut transformer les coordonnées entre les différents repères, cf matrices de transformations,
- la camera / le plan image est également défini par des matrices,
- plus simple pour démarrer...

et alors ?

simplifications :

- on peut calculer l'intersection avec pas mal d'autres formes,
- cf PBRT, un (gros) bouquin de référence,
- cube (aligné sur les axes) / voxel (minecraft ?),
- sphère,
- cylindre,
- disque,
- cone, paraboloide, hyperboloide, etc,
- courbe / ruban, utilisé pour les cheveux, la fourrure, l'herbe...

et alors ?

alternatives :

- on peut aussi définir les objets différemment,
- en utilisant une fonction de distance
  (entre un point de l'espace et l'objet),
- et en marchant le long du rayon jusqu'à trouver l'intersection,
- voir le cours de L2 graphique, par exemple,
- et i. quillez / shadertoy.