

L3-Synthèse

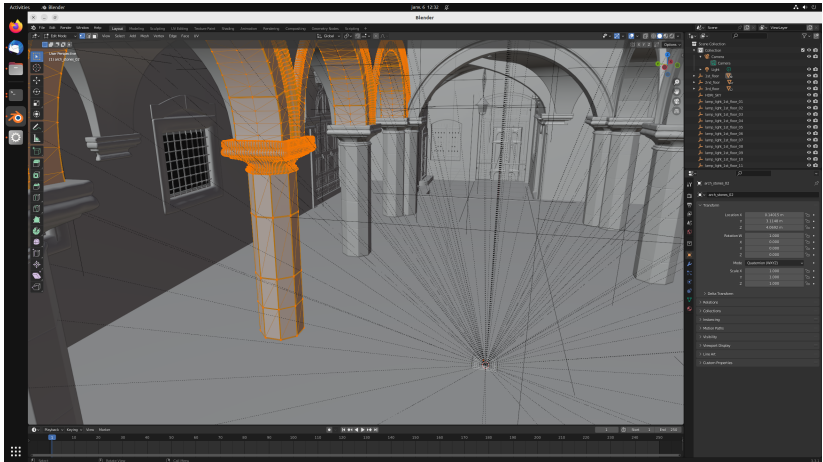
Lancer de rayons et rendu

J.C. lehl

February 6, 2025

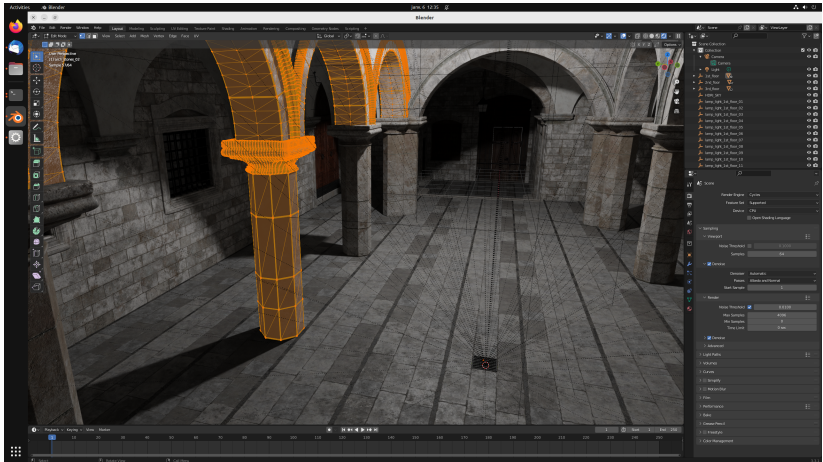
introduction
les détails...
et avec plusieurs objets ?
bilan

c'est quoi ?



introduction
les détails...
et avec plusieurs objets ?
bilan

c'est quoi ?



c'est quoi ?

en résumé :

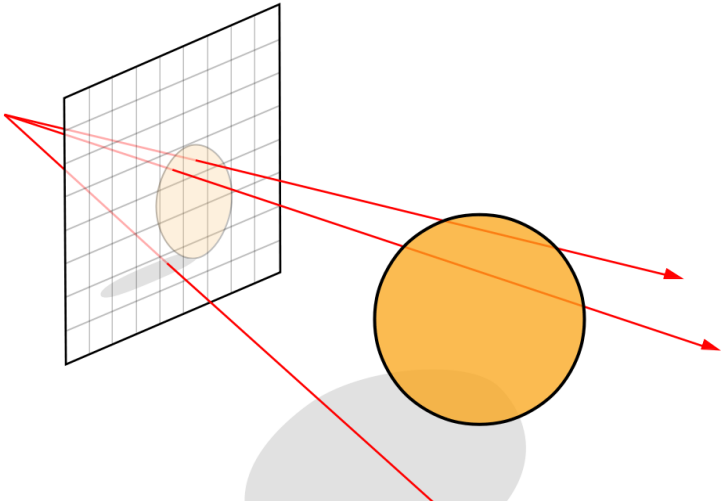
- ▶ construire une image,
- ▶ à partir d'un ensemble d'objets,
- ▶ (observés par une camera)
- ▶ (et éclairés par une ou plusieurs lumières)

comment ça marche ?

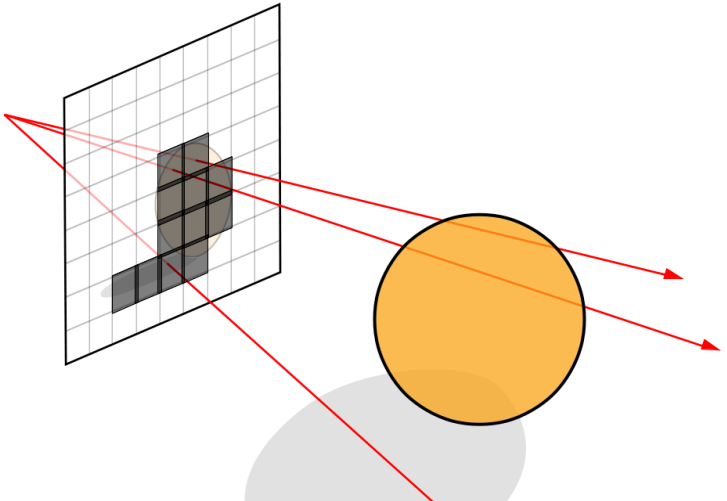
constuire une *image* :

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

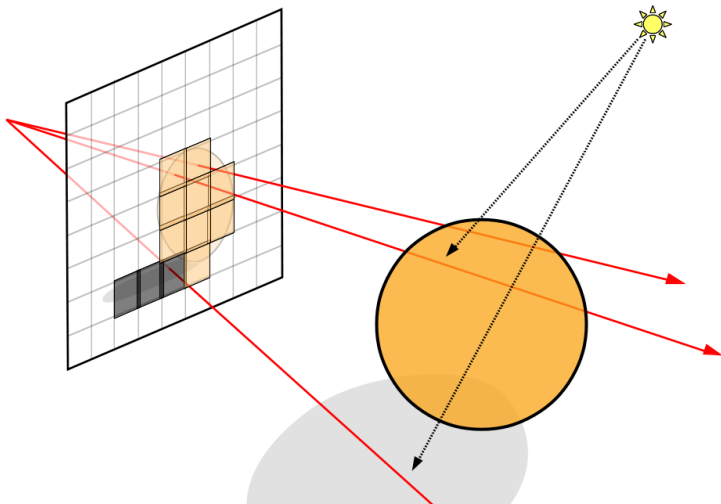
comment ça marche ?



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...

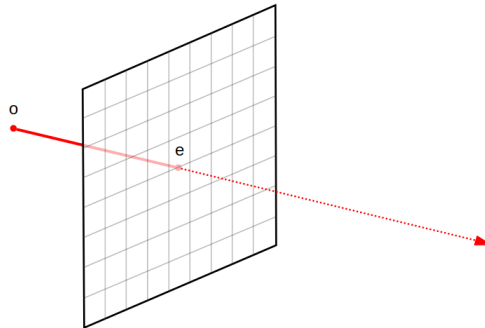
trouver l'objet visible :

- ▶ 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 (plutôt de la *camera*)

introduction
les détails...
et avec plusieurs objets ?
bilan

droite...
rayon !
intersection rayon / objet

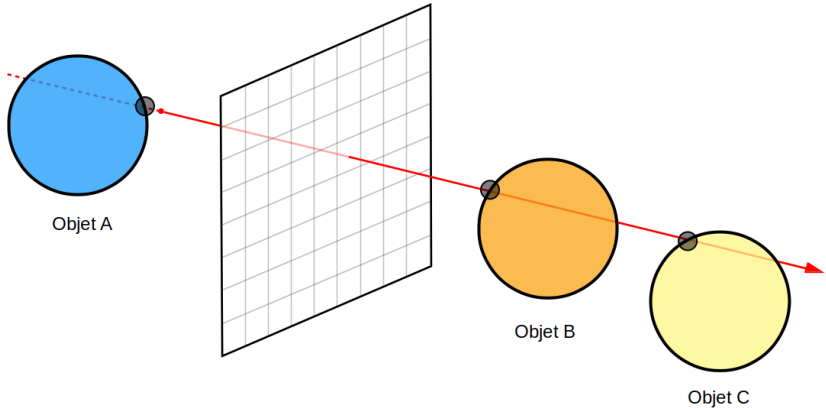
droite qui passe par un pixel ?



introduction
les détails...
et avec plusieurs objets ?
bilan

droite...
rayon !
intersection rayon / objet

quel objet ?



droite qui passe par le pixel...

une droite ?

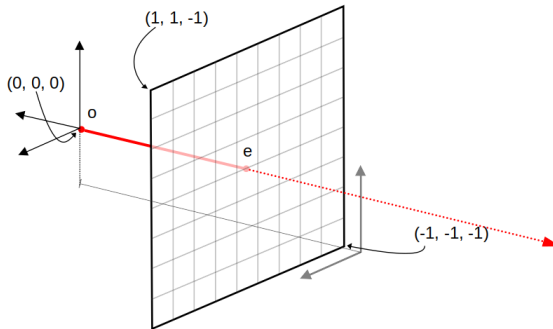
- ▶ comment décrire une droite ?
- ▶ avec 2 points ou 1 point et 1 direction,
- ▶ ?
- ▶ il va falloir aussi décrire une camera, comment on projette des objets 3d sur une image 2d...

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...

comment ça marche ?

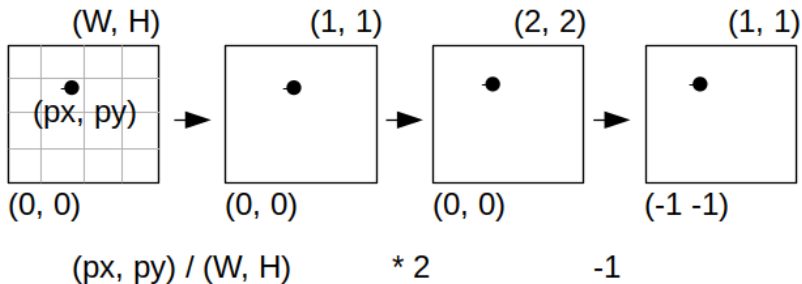


pixel et plan image

par convention :

- ▶ les points du plan image correspondent aux pixels de l'image,
- ▶ une image de resolution $W \times H$: 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);
}
```

droite ou rayon ?

droite qui passe par le pixel :

- ▶ on connaît 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 :

$$p(t) = o + t \cdot \vec{d} \text{ (avec } \vec{d} = e - o \text{) ou}$$

$$p(t) = o + t \cdot (e - o) = (1 - t) \cdot o + t \cdot e$$

rayon !

on utilise plutot :

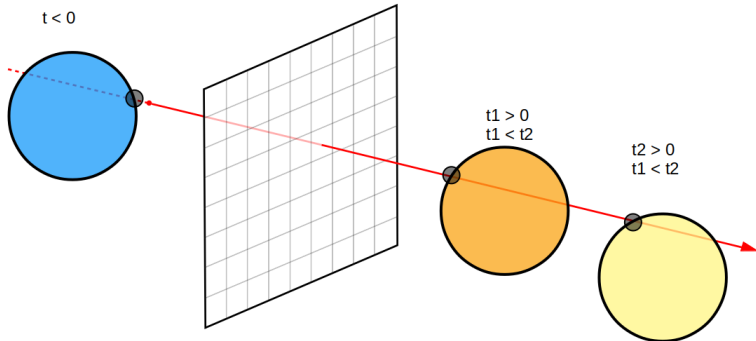
- ▶ $p(t) = o + t \cdot \vec{d}$
- ▶ si $t < 0$, le point se trouve avant l'origine de la droite (derrière l'origine),
- ▶ si $t > 0$, le point est après l'origine,
- ▶ si $t = 0$, le point est sur l'origine...

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 ...

introduction
les détails...
et avec plusieurs objets ?
bilan

droite...
rayon !
intersection rayon / objet

rayon et 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

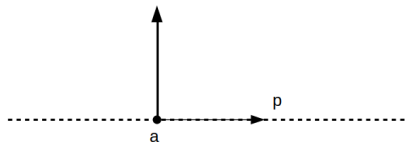
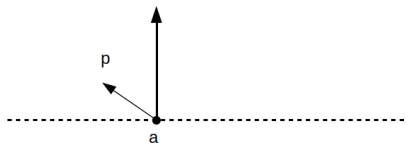
on veut calculer le point où le rayon traverse un plan...

- ▶ 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 propriété du produit scalaire entre 2 vecteurs :
- ▶ si le produit scalaire de 2 vecteurs est nul, les vecteurs sont perpendiculaires...

introduction
les détails...
et avec plusieurs objets ?
bilan

droite...
rayon !
intersection rayon / objet

intersection rayon / plan



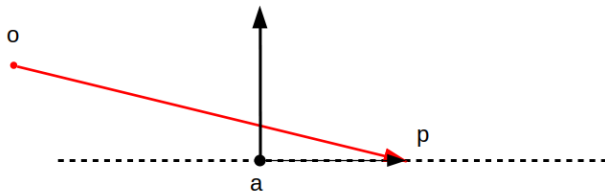
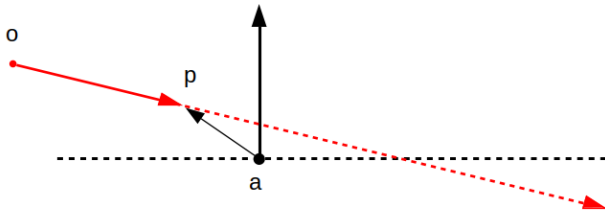
intersection rayon / plan

euh ?

- ▶ si un vecteur est la normale $\vec{n}$ du plan...
- ▶ et que l'on connaît 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;
```

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 :
 $\|p - c\| = r$, ils sont à la distance r de c ,
- ▶ 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

$$(p(t) - c) \cdot (p(t) - c) - r^2 = 0$$

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

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

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

$$\dots = 0$$

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

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 plutôt intuitif comme résultat :
- ▶ une droite traverse la sphère et donne 2 intersections,
- ▶ ou touche la sphère en un seul point,
- ▶ ou passe complètement à côté...

intersection rayon / sphère

rappels :

$$a = \vec{d} \cdot \vec{d}$$

$$b = 2(\vec{d} \cdot \vec{co})$$

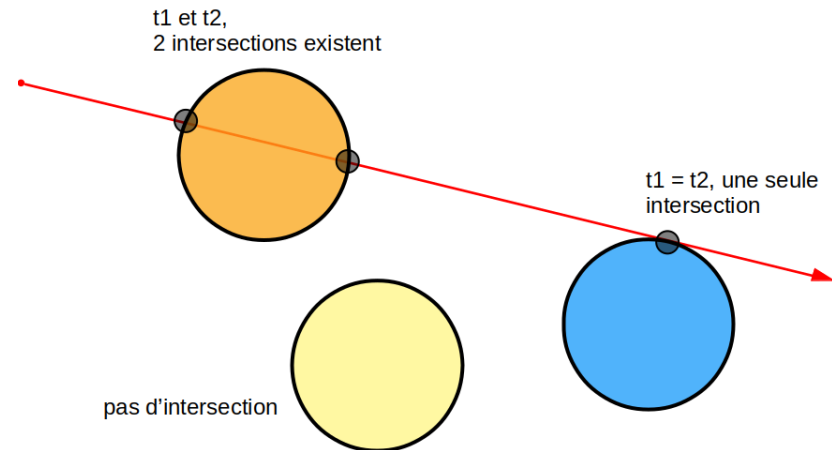
$$k = \vec{co} \cdot \vec{co} - r^2$$

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 :

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

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

intersection rayon / sphère



comment ça se code ?

```
#include "vec.h"

// sphere centre c, rayon r
Point c= { ... };
float r= ... ;

// intersection avec le rayon o, d
float a= ... ;
float b= ... ;
float k= ... ;

float dd= b*b - 4*a*k;
if(dd < 0)
    return "pas_touche";

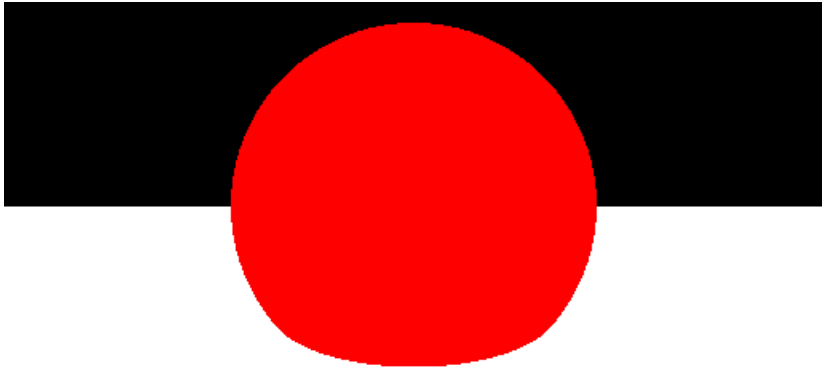
// calculer les 2 racines / intersections
float t1= ... ;
float t2= ... ;

// renvoyer l'intersection la plus proche de la camera...
return ...;
```

introduction
les détails...
et avec plusieurs objets ?
bilan

droite...
rayon !
intersection rayon / objet

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 !

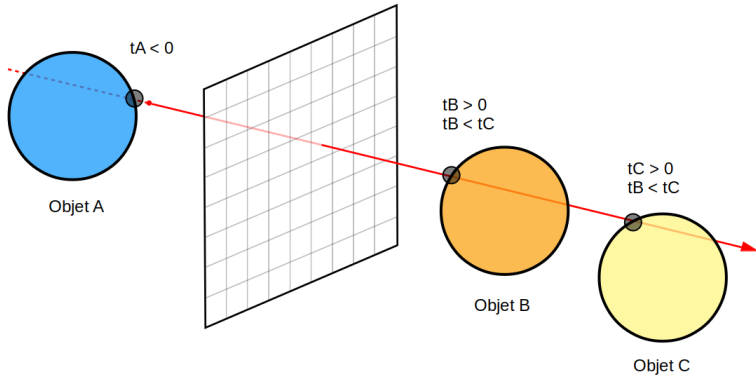
et avec plusieurs objets ?

intersection valide ?

- ▶ on ne s'intéresse qu'aux intersections *visibles*,
- ▶ pas 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 alors ?



et avec plusieurs objets ?

pour chaque objet :

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

on connaît l'objet visible pour le pixel !!

et avec plusieurs objets ?

attention :

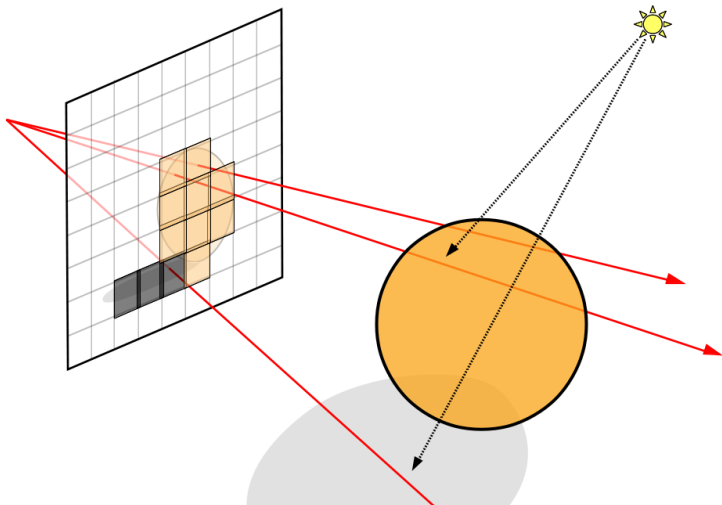
- ▶ même problème pour les objets qui calculent plusieurs points d'intersection,
- ▶ une sphère par exemple... (ou une boîte)
- ▶ on ne conserve que les intersections valides, avec $t > 0$

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, paraboloïde, hyperboloïde, 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](#).