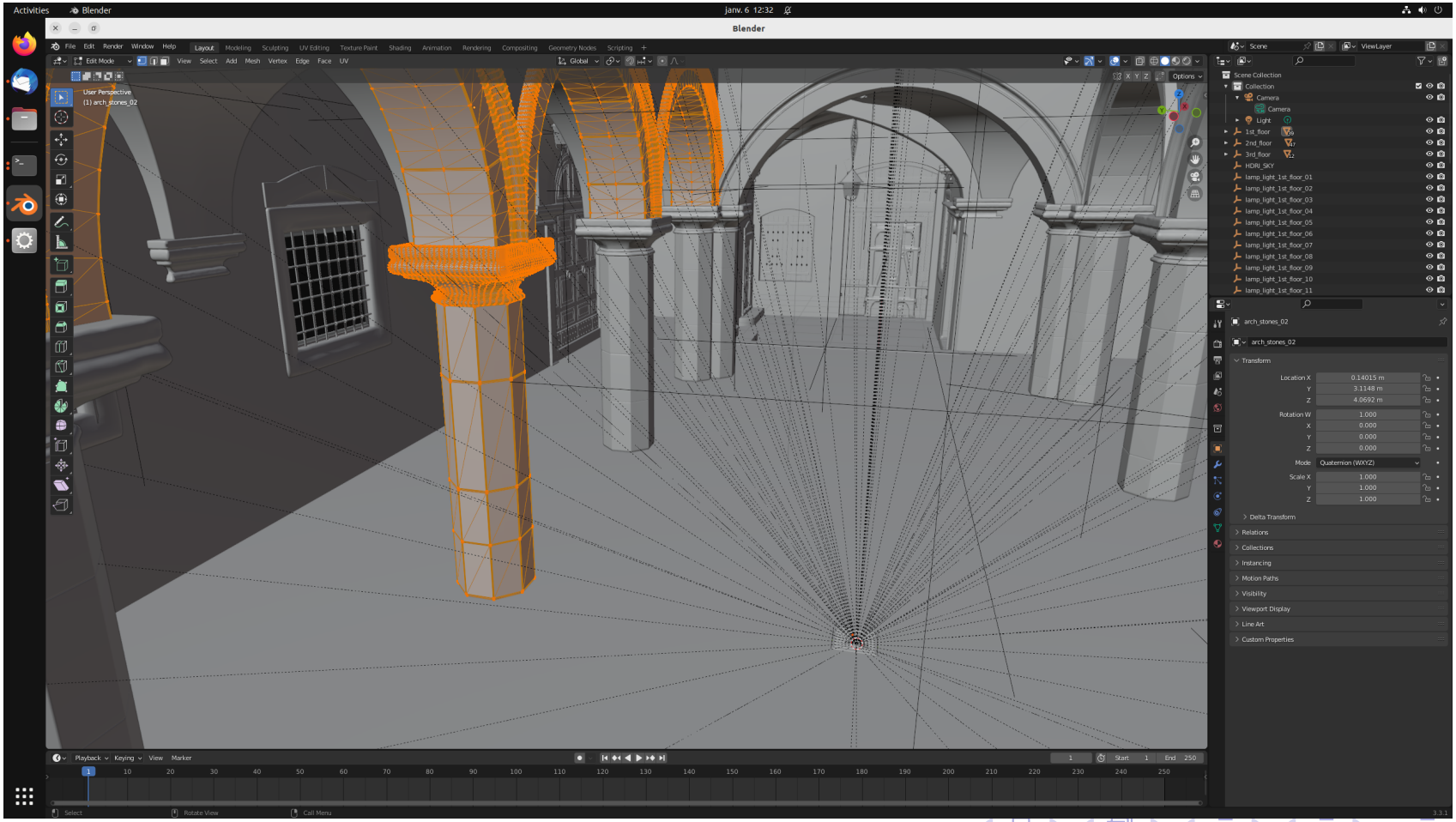
M1-Synthèse

Pipeline graphique

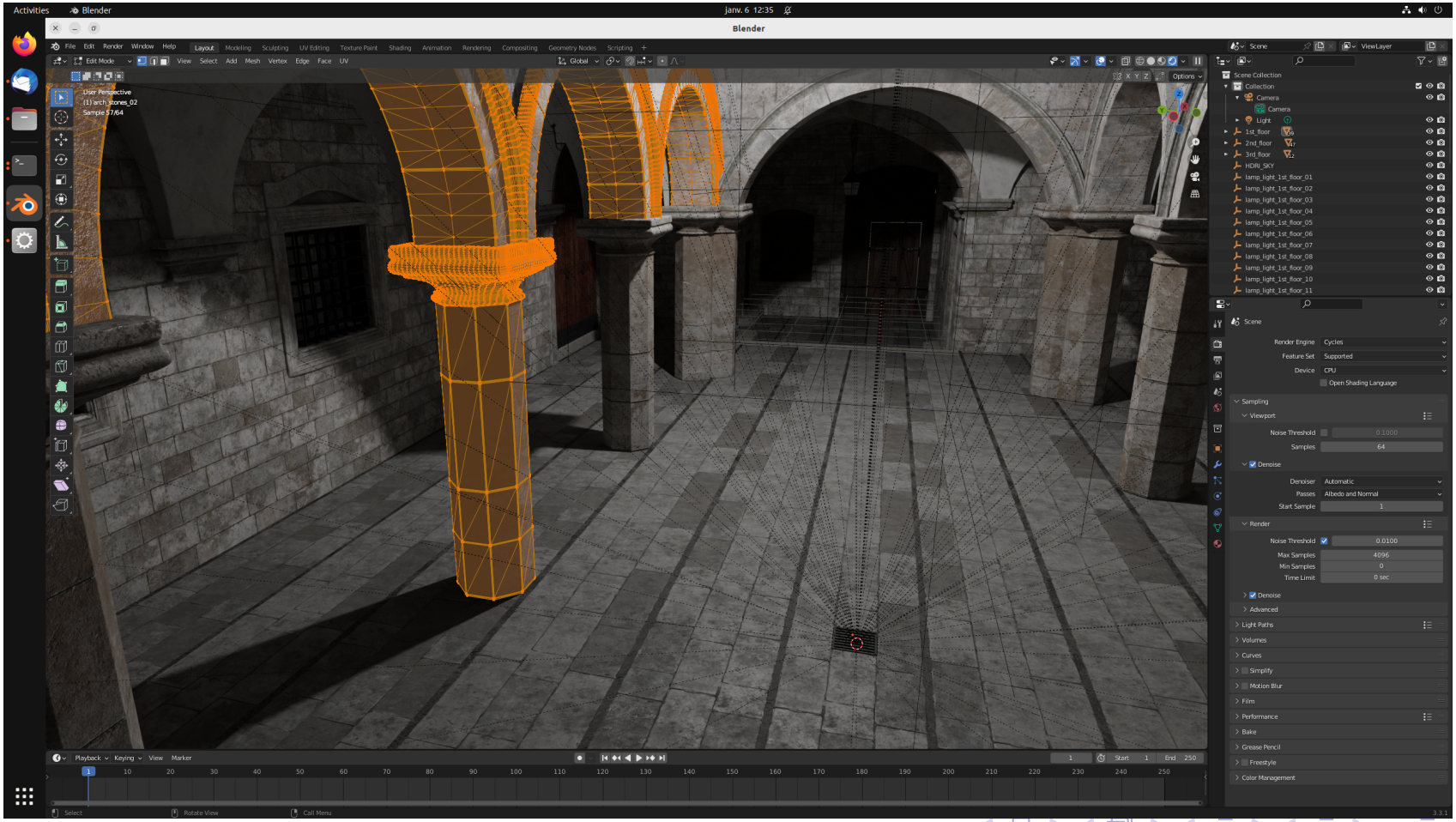
J.C. Iehl

March 8, 2023

Synthèse d'images : c'est quoi ?



Synthèse d'images : c'est quoi ?



Synthèse d'images : c'est quoi ?

en résumé :

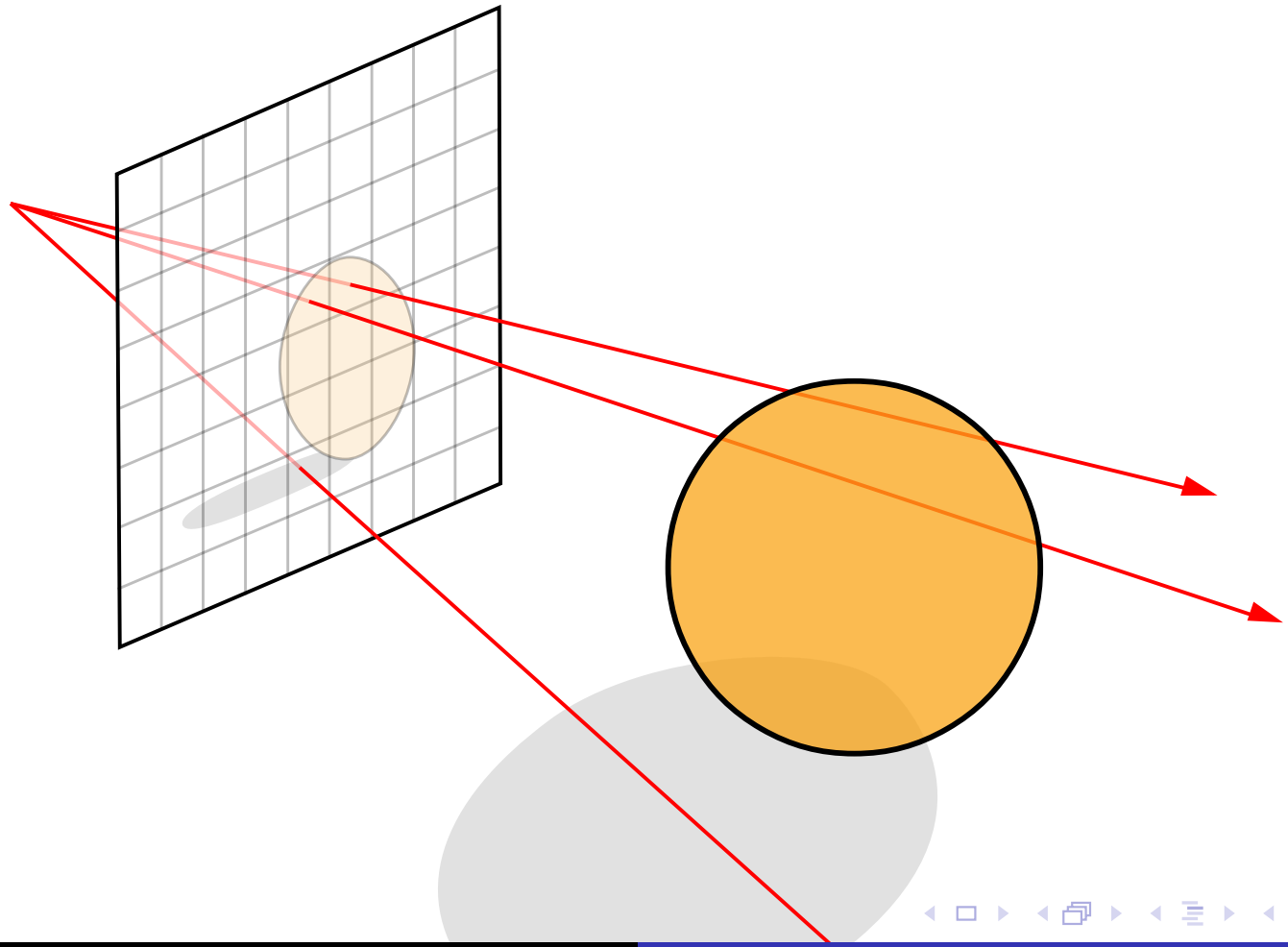
- construire une image,
- à partir d'un ensemble d'objets,
- (observés par une camera)
- (et éclairés par une ou plusieurs lumières)

comment ça marche ?

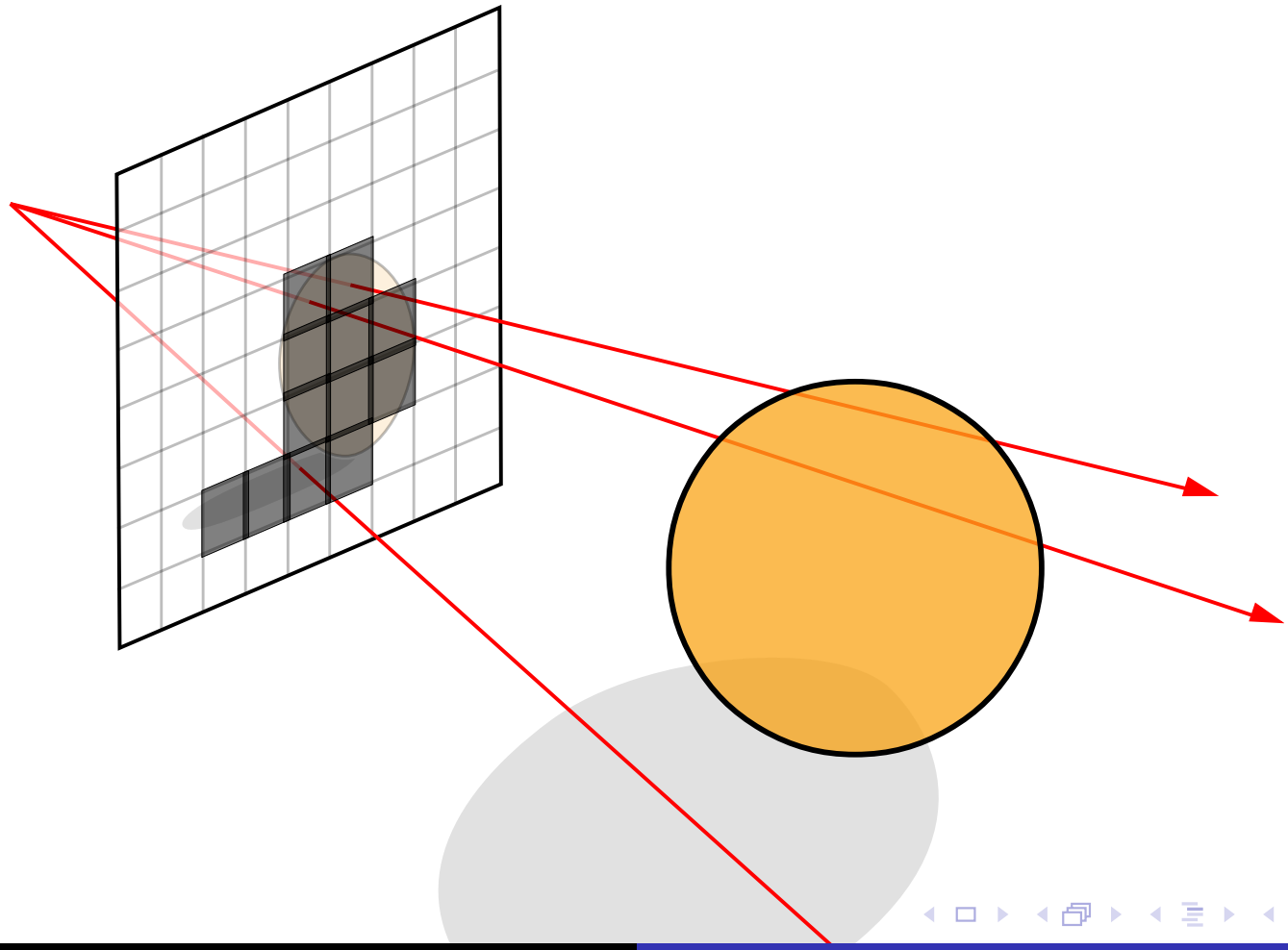
construire une *image* :

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

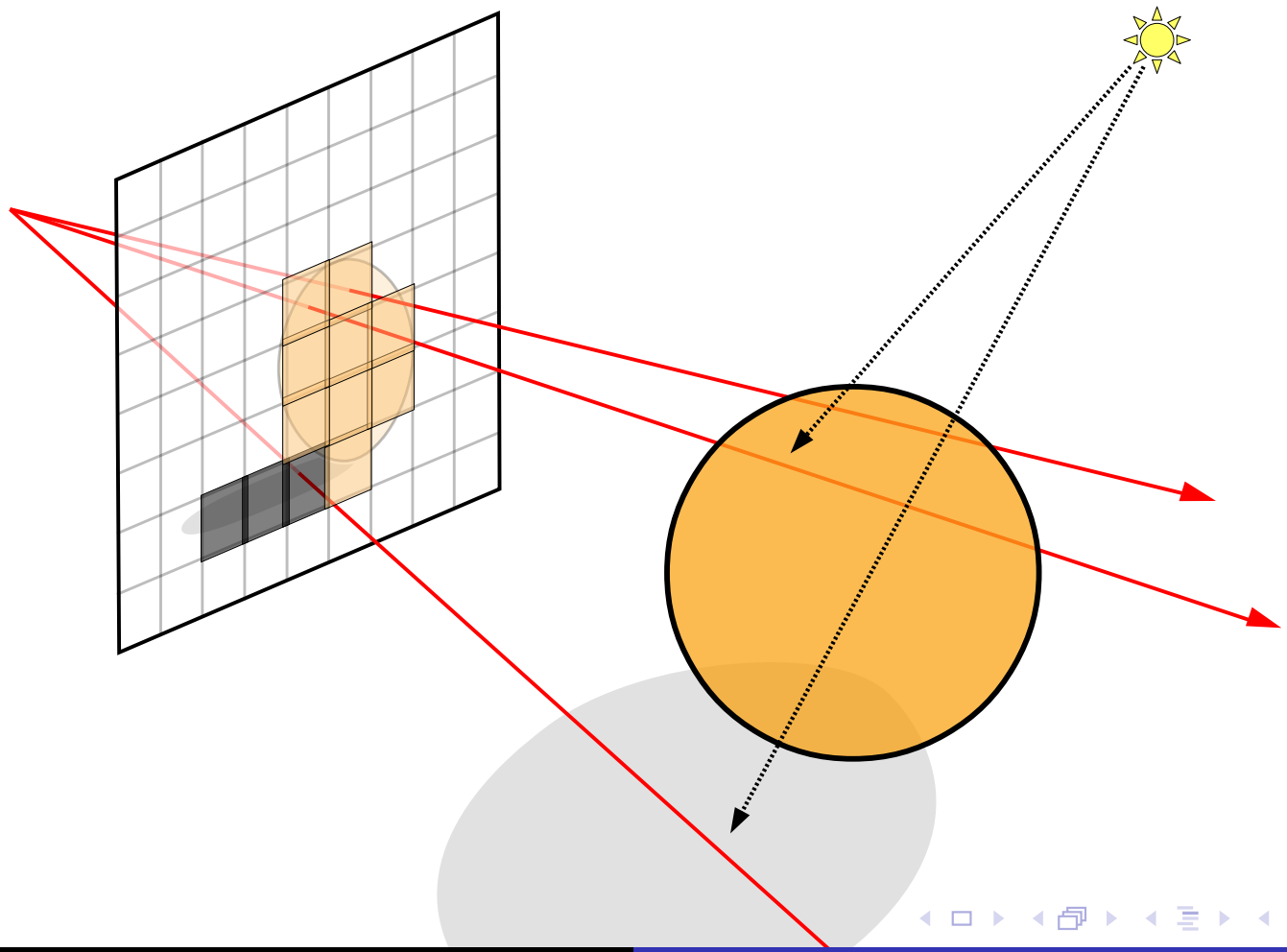
comment ça marche ?



comment ça marche ?



comment ça marche ?



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 M2,
- soit on programme tout, c'est techniquement plus simple,
- même s'il faut manipuler quelques détails pour obtenir le résultat...

quelques détails à régler...

trouver l'objet visible :

- devant la camera qui observe les objets...
- camera ?
- pour chaque pixel de l'image ?
- et quand il y a plusieurs objets qui se dessinent sur le meme pixel ?

intermède

avant de regarder le cas complet,
on va s'interresser à un cas simplifié.

version 2D :

- dessiner un triangle dont on connait la projection des sommets dans l'image...
- ??

algorithme diviser pour regner

simple :

- on sait modifier la couleur d'un pixel,
- donc : on sait dessiner un triangle de la taille d'un pixel,
- si le triangle est plus gros, on le découpe, en 4, par exemple...
- jusqu'a obtenir des triangles plus petits qu'un pixel...

autre solution : on peut aussi découper l'image / l'exterieur du triangle en blocs de pixels...

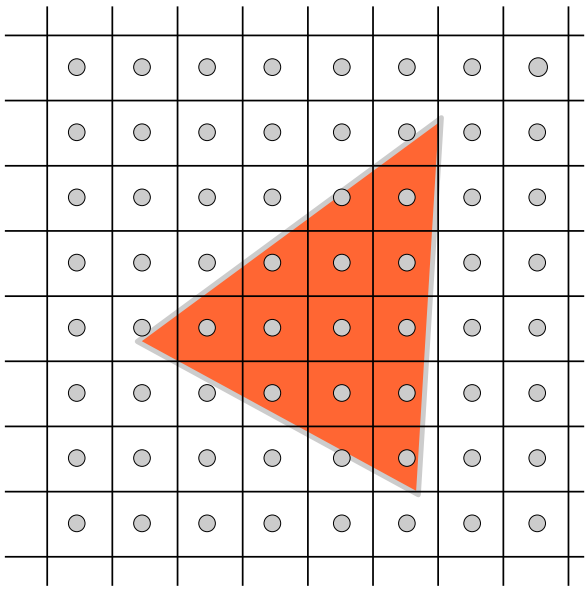
et pour de vrai ?

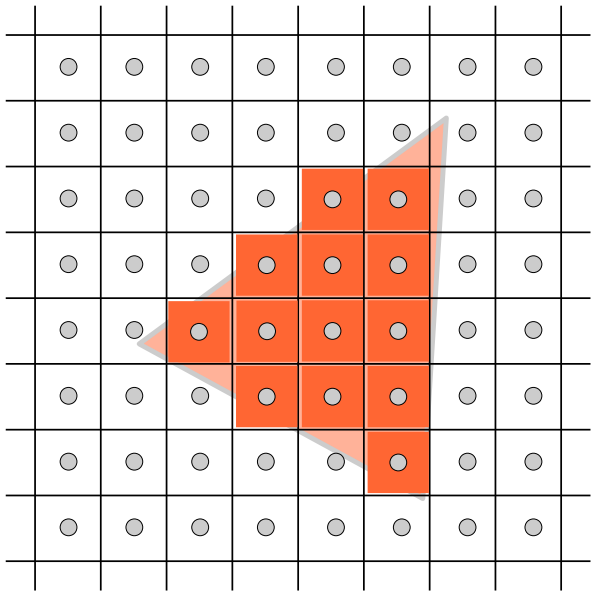
mais en pratique :

- on utilise un autre algorithme plus simple à paralléliser,
- plus simple de construire du materiel specialise, cf carte graphique...

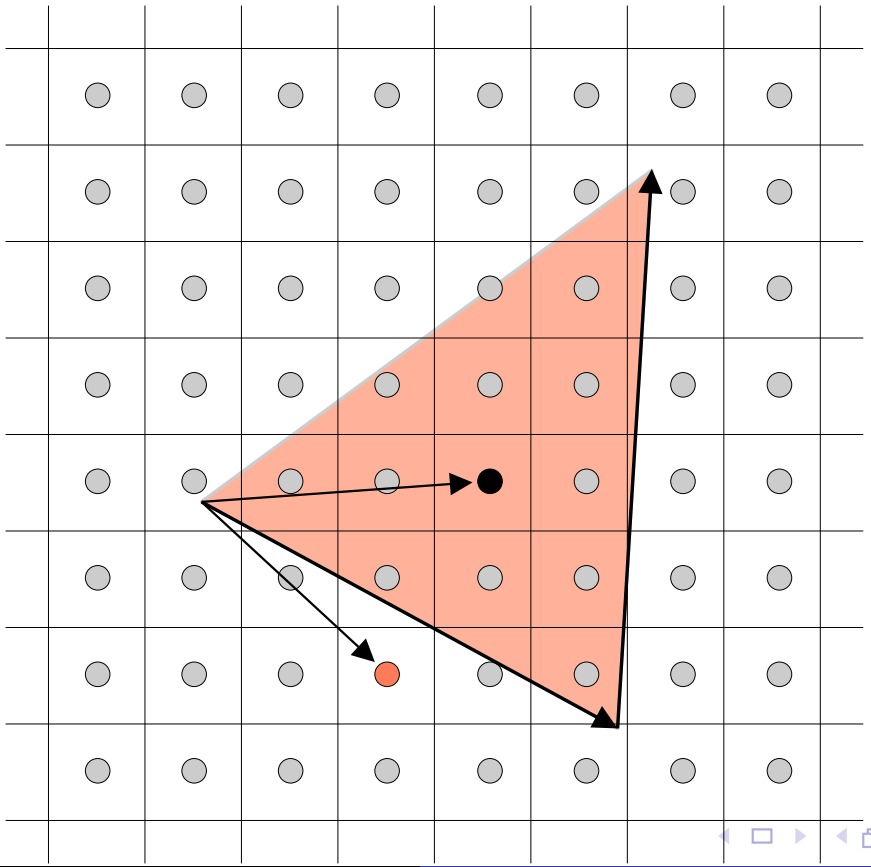
tester si un pixel de l'image, un point se trouve à l'interieur du triangle...
il suffit de tester tous les pixels de l'image...

et pour de vrai ?





et pour de vrai ?



et pour de vrai ?

test d'orientation

- si le point p est du "bon coté" de chaque arete, il est à l'interieur du triangle...
- ou si le triangle abp est orienté comme le triangle abc, + triangle bcp, + triangle cap

et alors ?

et alors ?

- si on ne dessinait qu'un seul triangle, le cours serait fini...
- mais que se passe-t-il lorsque plusieurs triangles se dessinent sur le meme pixel ?

on veut contruire une image qui represente les objets observes par la camera...

et alors ?

chaque pixel de l'image correspond à un objet visible par la camera,

si les objets sont opaques,
un pixel observe l'objet le plus proche de la camera...

il faut connaitre Z pour construire l'image des objets / triangles visibles...

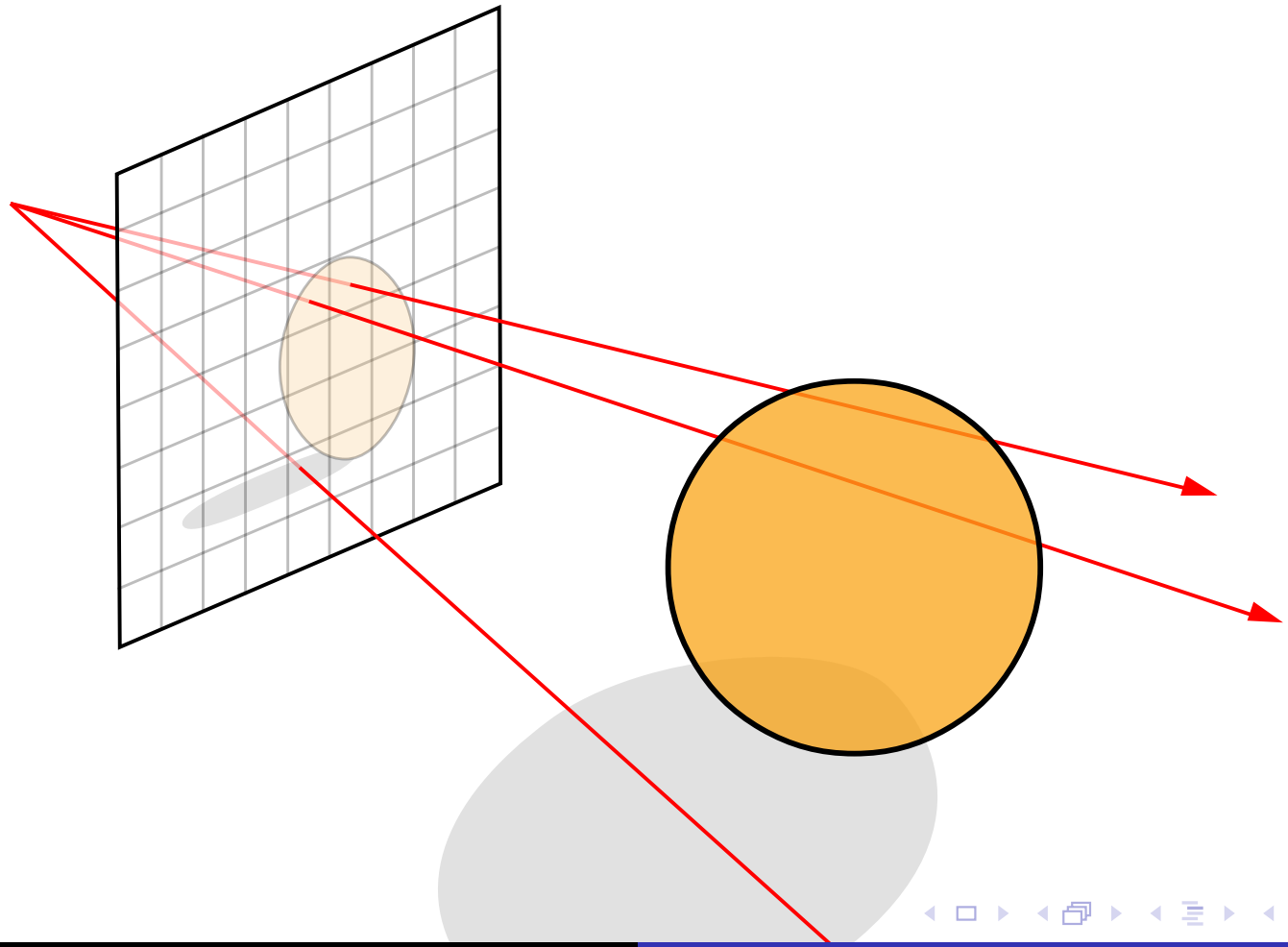
et il faut faire le test en 3D... mais il faut définir la camera.

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,
- la distance entre le plan image et la camera definit la projection (la taille du plan image aussi...)

pixel et plan image



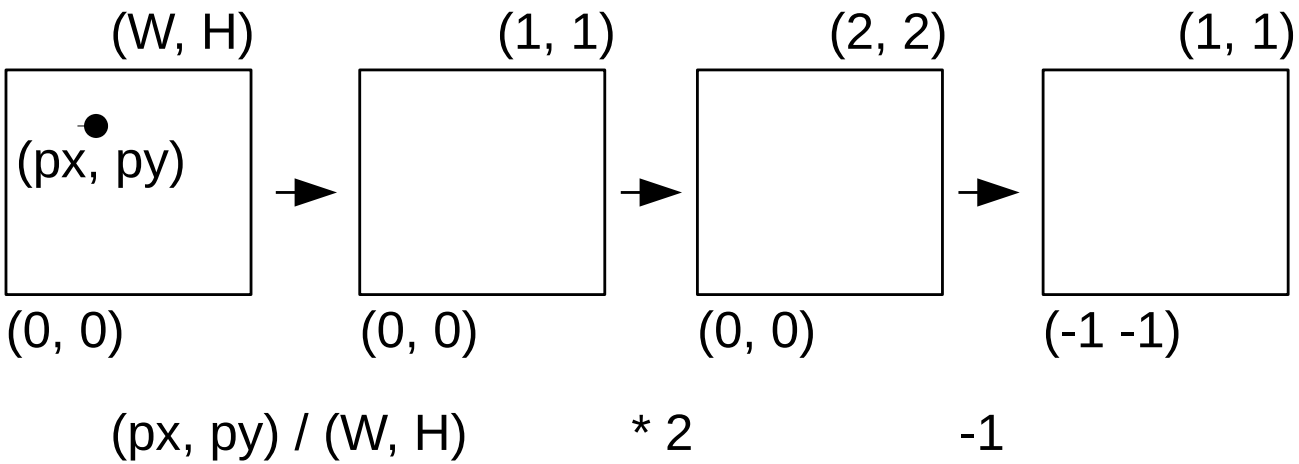
pixel et plan image

par convention :

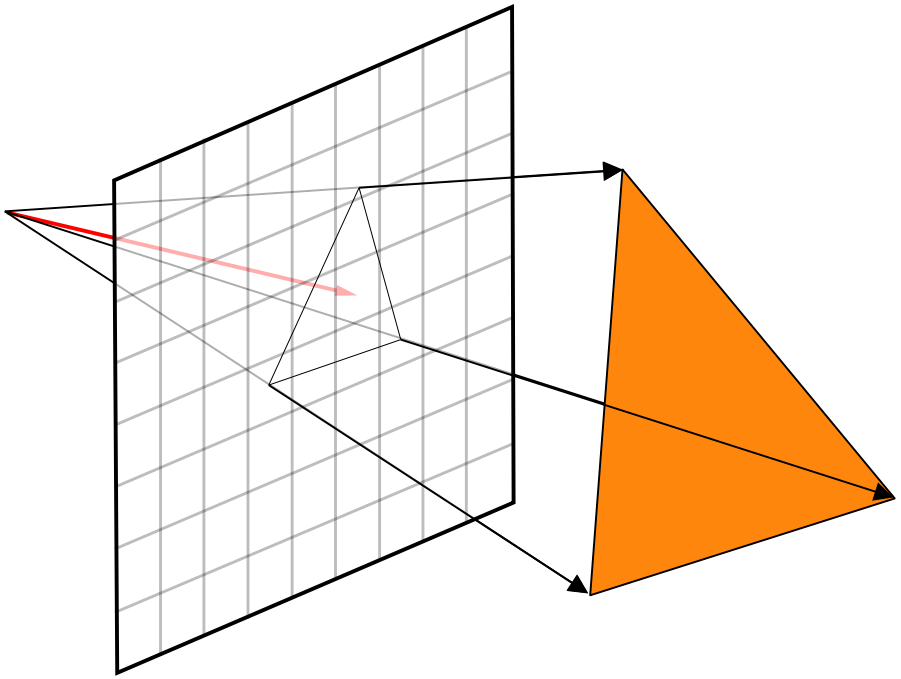
- les points du plan image correspondent aux pixels de l'image,
- une image de resolution WxH : H lignes de W pixels,
- le point (-1 -1) du plan image correspond au pixel (0, 0) en bas à gauche de l'image,
- le point (1 1) du plan image correspond au pixel (W, H) en haut à droite de l'image,
- quel point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

pixel et plan image

$$(x, y, -1) = (p_x/W, p_y/H) * 2 - 1$$

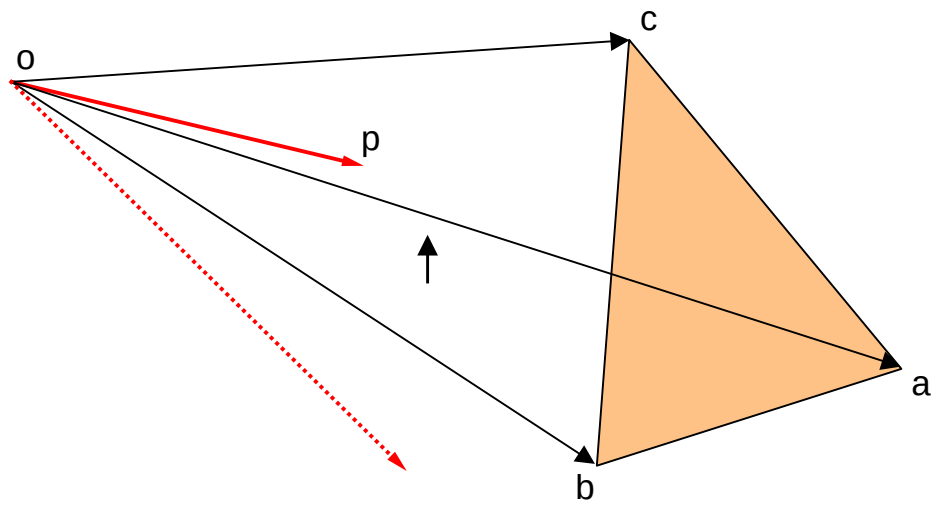


comment ça marche ?



comment ça marche ?

test d'orientation en 3D...



comment ça marche ?

orientation :

- si le vecteur $\vec{op}$ est du "bon cote" de chaque triangle, ie la droite $\vec{op}$ passe par l'interieur du triangle
- ou si le tetraedre oabp est orienté comme oabc, + tetraedre obcp, + tetraedre ocap

et alors ?

et alors ?

- c'est toujours un simple test d'inclusion ? on sait que p appartient, ou pas, à la projection du triangle.
- mais : on a aussi projette le triangle sur le plan image !

on veut connaitre la distance entre la camera et le point sur le triangle.

coordonnées barycentriques

on va interpoler la distance !

- ??
- interpoler sur un segment ok, mais sur un triangle ? comment on interpole ?
- sur un segment, on peut écrire $p = \lambda_a a + \lambda_b b$
- dans un triangle, on peut écrire $p = \lambda_a a + \lambda_b b + \lambda_c c$
- avec $\lambda_a + \lambda_b + ... = 1$

on peut utiliser le meme principe pour chaque attribut de sommet, couleur, normale, coordonnées de textures, etc... et la coordonnée Z des sommets !

coordonnées barycentriques

euh et c'est quoi λ ?

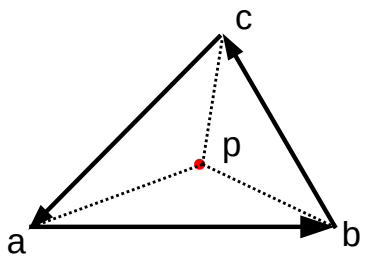
- facile, c'est l'aire normalisée des triangles abp, bcp, et cap en 2D,
- ou le volume normalisé des tetraedres oabp, obcp, ocap en 3D...
- et on vient de les calculer pour le test d'inclusion !
  (on a comparé leur signe... cf orientation)

coordonnées barycentriques

$$\lambda_a = \frac{bcp}{abc}$$

$$\lambda_b = \frac{cap}{abc}$$

$$\lambda_c = \frac{abp}{abc}$$



coordonnées barycentriques

rappel :

- aire du triangle abc : $\frac{1}{2}||\vec{ab} \times \vec{ac}||$
- volume du tetraedre oabc : $\frac{1}{6}||\vec{ao} \cdot (\vec{ab} \times \vec{ac})||$
- ou calculer le determinant d'une matrice contenant les coordonnées des sommets.

les détails des 384+36 manières de calculer la meme chose :
"Optimizing Ray-Triangle Intersection via Automated Search"
A. Kensler, P. Shirley, 2006

coordonnées barycentriques

avec o à l'origine du repère camera :

- $V_a(\vec{v}) = \vec{v} \cdot \vec{n_a}$
- $V_b(\vec{v}) = \vec{v} \cdot \vec{n_b}$
- $V_c(\vec{v}) = \vec{v} \cdot \vec{n_c}$
- $\lambda_a = \frac{V_a(\vec{op})}{V}$ et $V = V_a(\vec{op}) + V_b(\vec{op}) + V_c(\vec{op})$,
- $\lambda_b = \frac{V_b(\vec{op})}{V}$
- $\lambda_c = \frac{V_c(\vec{op})}{V}$

les notations exactes sont dans "3D Rasterization",
T. Davidovic, T. Engelhardt, I. Georgiev, 2012, section 3.2, page 3

coordonnées barycentriques

et alors ?

- on peut calculer la coordonnée z du point sur le triangle qui se projette en p :
- $z = \lambda_a a_z + \lambda_b b_z + \lambda_c c_z$
  avec les coordonnées des sommets du triangle.

on peut maintenant savoir quel triangle est le plus proche de la camera pour chaque pixel de l'image... et construire une image qui represente ce que voit la camera !

Rasterization et ZBuffer

en résumé :

- init( ) : ZBuffer[]= ∞, image[]= noir,
- pour chaque triangle :
- tester les pixels de l'image,
  est ce que $\vec{op}$ passe par le triangle ?
- calculer la distance (ou z), avec l'interpolation...
- si la distance est plus petite que ZBuffer[pixel]
- calculer la couleur du pixel,
- ZBuffer[pixel]= distance
- image[pixel]= couleur

Rasterization et ZBuffer

et ça marche ?

- oui ! avec pas mal d'astuces de calculs supplémentaires,
- tester l'inclusion d'un pixel dans le triangle ne demande que quelques additions et une division.

cf "3D Rasterization",
T. Davidovic, T. Engelhardt, I. Georgiev, 2012,
et
"Incremental and Hierarchical Hilbert Order Edge Equation Polygon Rasterization", M. McCool, C. Wales, . Moule, 2001

et alors ?

et alors ?

- on vient de calculer l'intersection d'une droite avec un triangle...
- pas de solution plus directe ??

ben si, c'est la base du lancer de rayon !

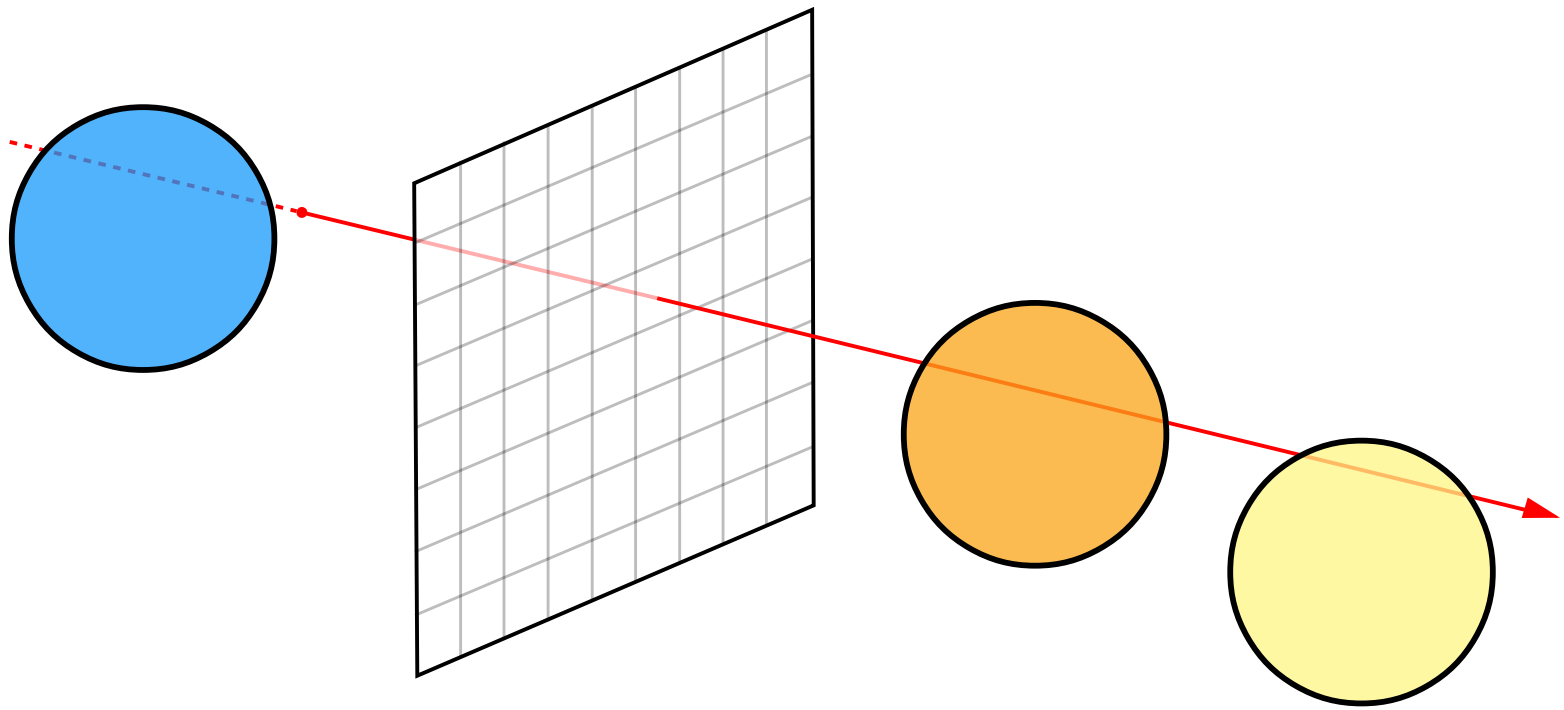
rayon ?

rayon ?

- on représente le point d'intersection sur la droite / le rayon :
- $p = o + t \cdot \vec{d}$

$t < 0$ si le point est derriere l'origine / la camera,
$t > 0$ si le point est devant l'origine...

rayon !



intersection rayon / triangle

intersection rayon :

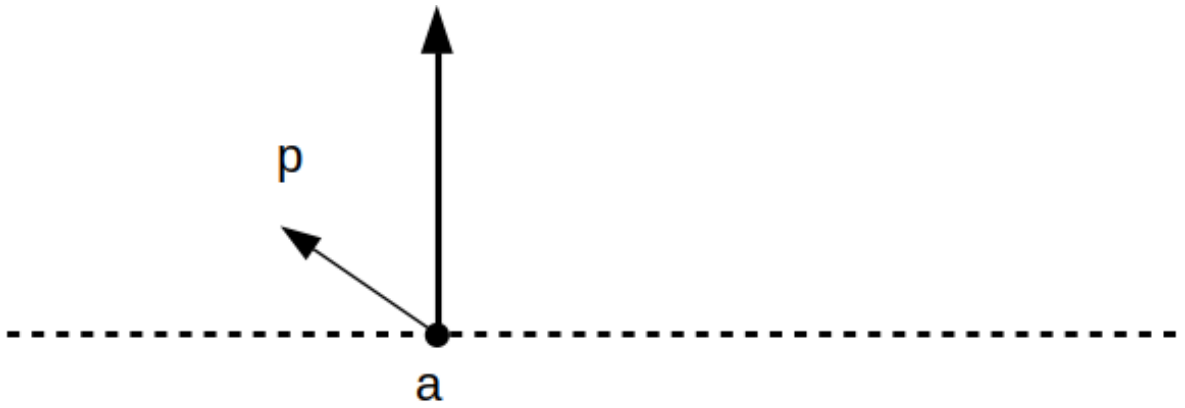
- avec le plan du triangle,
- et il ne reste plus qu'à verifier que ce point est à l'interieur du triangle... cf tests 2D !
- ou...

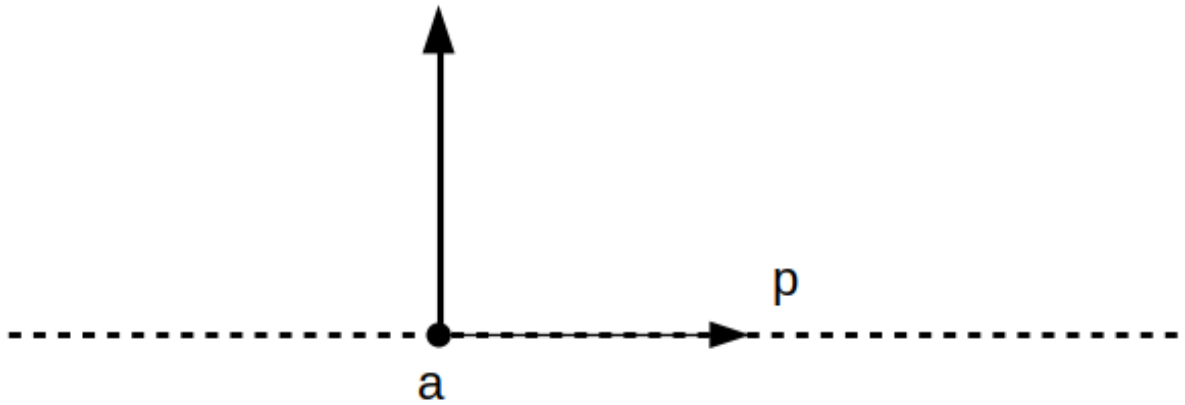
intersection rayon / plan

c'est quoi un plan ?

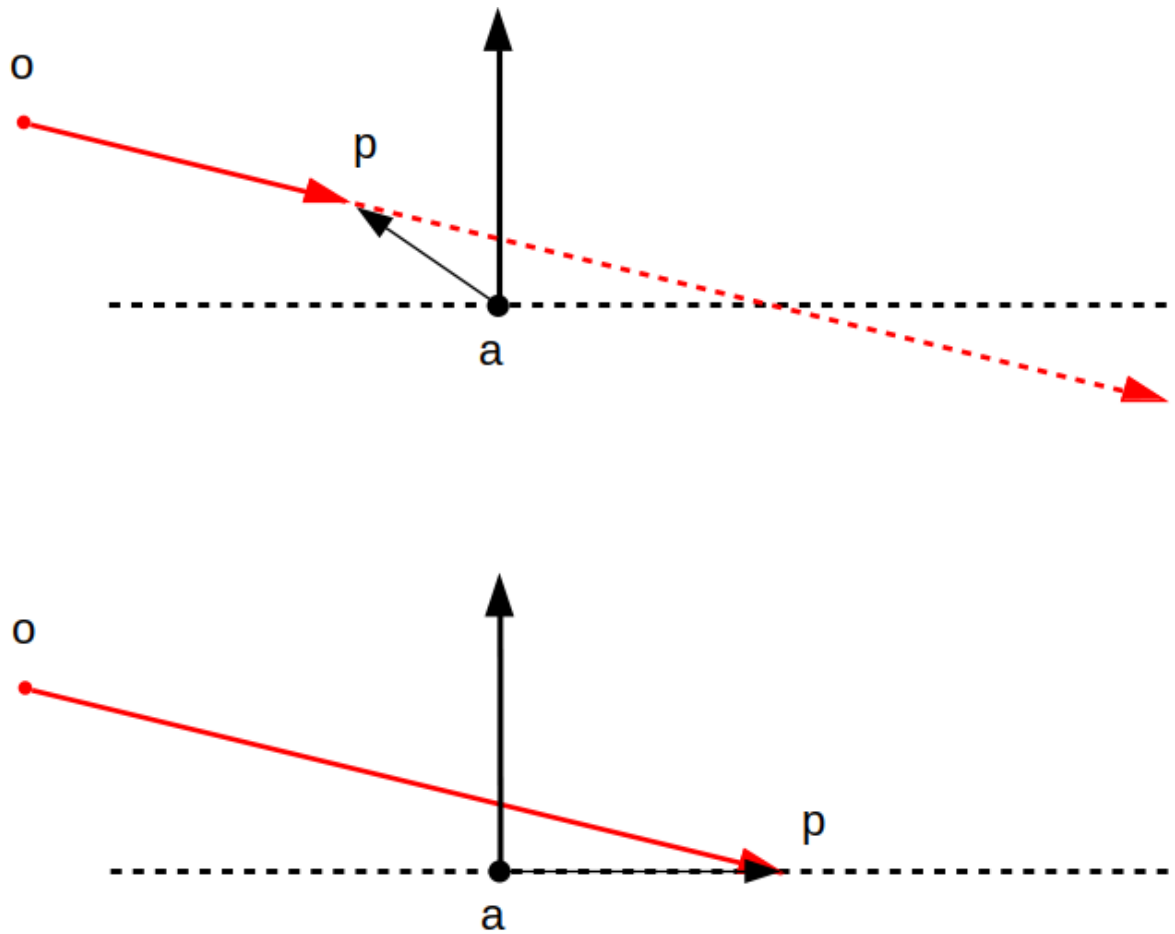
- on veut trouver un point qui est à la fois sur le rayon et dans le plan...
- comment déterminer qu'un point est sur un plan ?
- on utilise une propriété du produit scalaire de 2 vecteurs :
- si le produit scalaire de 2 vecteurs est nuls, les vecteurs sont perpendiculaires !

rayon !





rayon !



intersection rayon / plan

euh ?

- on connait la normale $\vec{n}$ du plan,
- et 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 !

intersection rayon / plan

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

intersection rayon / triangle

et ?

- maintenant on peut calculer : $p = o + t \cdot \vec{d}$
- et vérifier que p est bien dans le triangle abc... cf tests d'orientation 2D

mais : en général on calcule aussi les coodonnées barycentriques !

intersection rayon / triangle

mais pourquoi ?

- on a besoin d'interpoler les propriétés des sommets pour le reste des calculs... la normale en p, par exemple,
- et on peut réduire la quantité de calculs...
- ??
- $\lambda_a + \lambda_b + \lambda_c = 1$, donc $\lambda_a = 1 - \lambda_b - \lambda_c$
- si p est dans le triangle $0 < \lambda_a < 1$, $0 < \lambda_b < 1$ et $0 < \lambda_c < 1$
- on calcule seulement λ_b et λ_c, p est à l'intérieur si $0 < \lambda_b < 1$ et $0 < \lambda_c < 1$ et $0 < \lambda_b + \lambda_c < 1$

intersection rayon / triangle

pleins d'astuces de calculs :

par exemple, on peut transformer le rayon et le triangle dans une configuration simple :
"Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip", S. Woop, 2004

intersection rayon / triangle

pleins d'astuces de calculs :

ou se rendre compte, que l'on obtient un systeme lineaire 3 equations, 3 inconnues :
"Fast Minimum Storage Ray / Triangle Intersection", T. Moller, B. Trumbore, 1997

et que calculer le determinant de la matrice associée, se fait assez facilement avec des produits scalaires et vectoriels...

rappel : pleins de manieres d'exprimer les mêmes propriétés
"Optimizing Ray-Triangle Intersection via Automated Search"
A. Kensler, P. Shirley, 2006

et alors ?

et avec plusieurs intersections ?

- il suffit de garder l'intersection valide la plus petite / la plus proche de la camera

intersection valide : $t > 0$ ie la camera ne voit que les objets devant !

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 ?

quelles sont les différences ?

- rasterisation / fragmentation des triangles,
- vs lancer de rayon ?
- c'est presque les memes calculs !
- oui mais :
- 1 triangle à la fois pour la fragmentation,
- vs 1 pixel à la fois pour le lancer de rayons,
- origine commune / 1 camera pour la fragmentation,
- vs 1 origine par rayon...

et alors ?

quelles sont les différences ?

- materiel specialise tres tres rapide pour la rasterization / fragmentation,
- materiel assez rapide pour le lancer de rayon...
- uniquement des points, segments et des triangles
- vs pas mal d'objets differents...

mais : le lancer de rayon est beaucoup plus souple pour tester la visibilite et simuler la propagation de la lumière...

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.