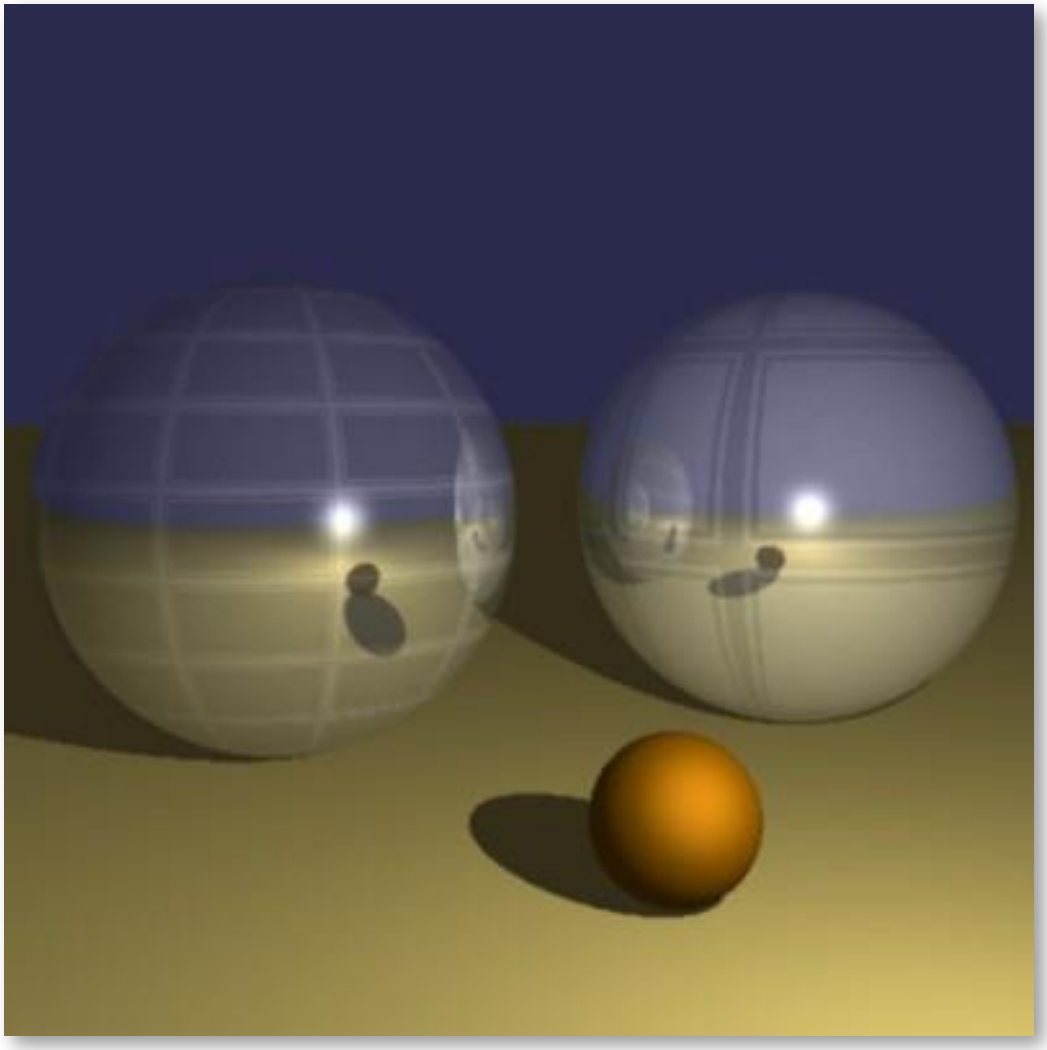
Projet : Lancer de rayons



Projet

Requis

- Path-tracing basique
- Ombrage basique
- Lumière étendues/Ombres douces
- Réflexions
- Maillages
- Réfraction ~10/20
- Kd-tree pour l'intersection rayon triangle

Au choix, effets supplémentaires :

- Profondeur de champs
- Textures
- Cartes de normales
- Photon mapping etc…

Rendus

2 rendus intermédiaires

- mi novembre
- début décembre

Rendu final en janvier

Génération des rayons primaires

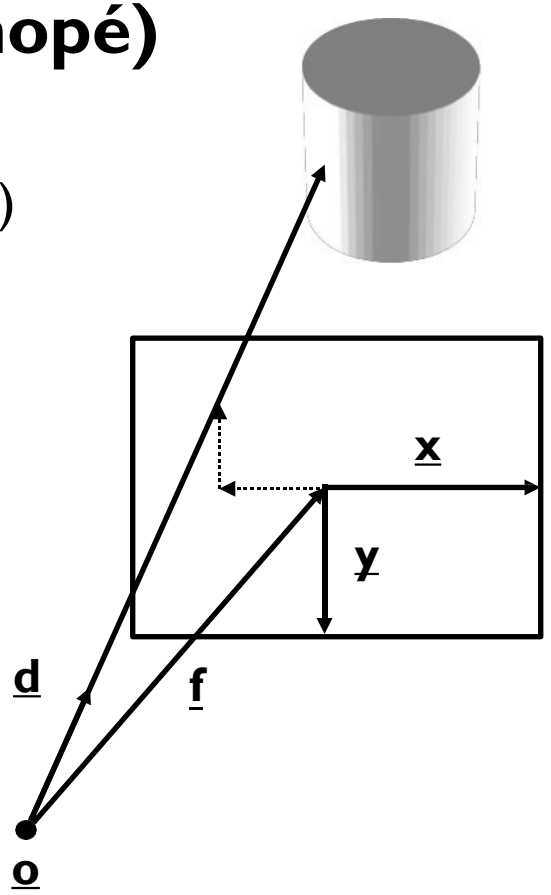
Un rayon : $\underline{r}(t) = \underline{o} + t\ \underline{d}$

- origine $\underline{o}=(o\underline{x}, o\underline{y}, o\underline{z})$, direction $\underline{d}=(d\underline{x}, d\underline{y}, d\underline{z})$ avec $||\underline{d}||=1$

Hypothèse :"Pinhole camera" (sténopé)

- $\underline{o}$: origine (point de vue)
- $\underline{f}$: axe optique (vecteur vers le centre de l'image)
- $\underline{x}, \underline{y}$: axes sur l'image
- width, height : résolution de l'image

```
for(i=0; i<width; i++)
  for(j=0; j<height; j++)
  {
    d = f + 2(i/width  - 0.5)x
          + 2(j/height - 0.5)y;
    d = d/|d|; // normalisation
    color = ray_tracing(o, d);
    write_pixel(i,j,color);
  }
```



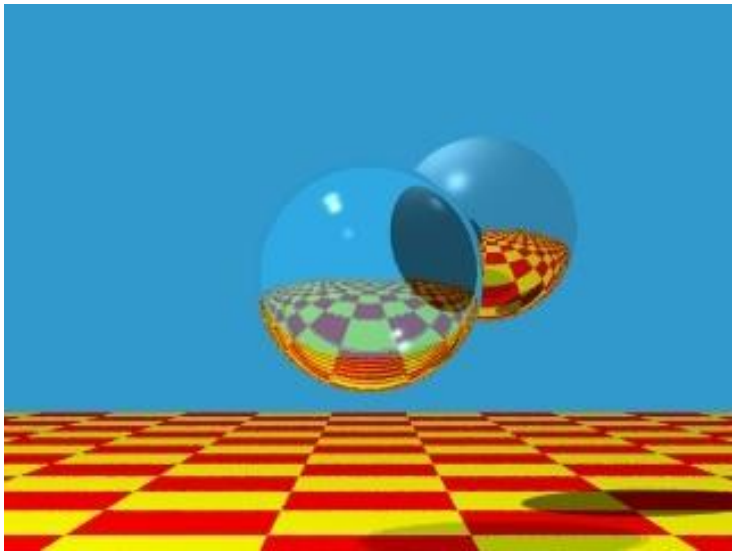
Extension du modèle

[Turner Whitted 1980]

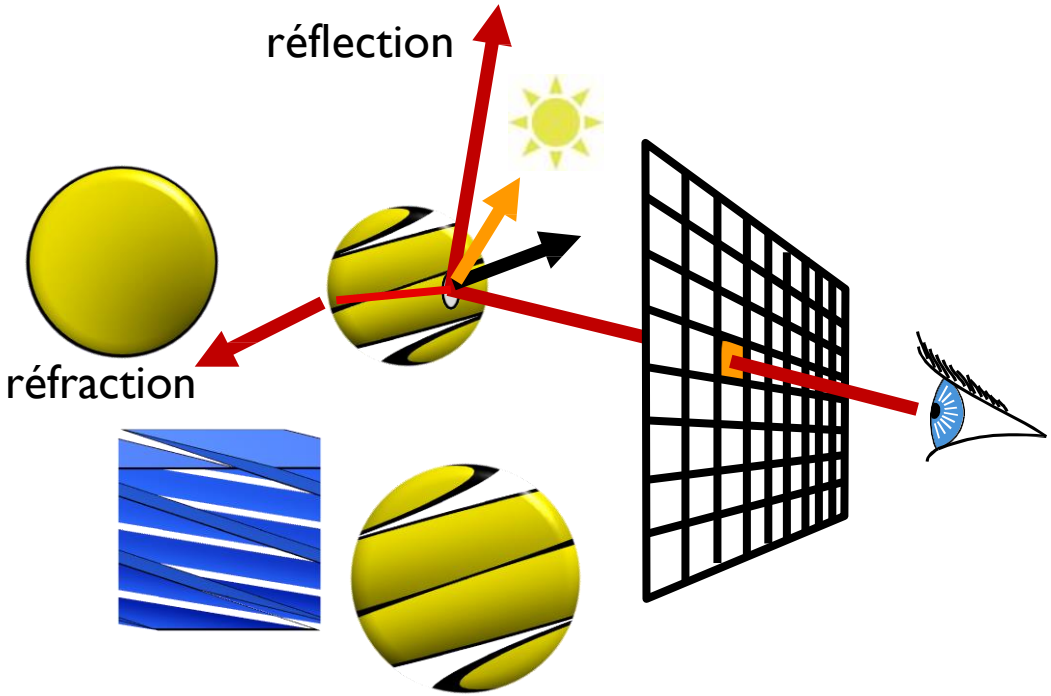
Trois nouveaux rayons sont générés :

- un rayon **réfracté***,
- un rayon **réfléchi***,
- un rayon **d'ombre**

$\Longrightarrow$ **lancer de rayon récursif**

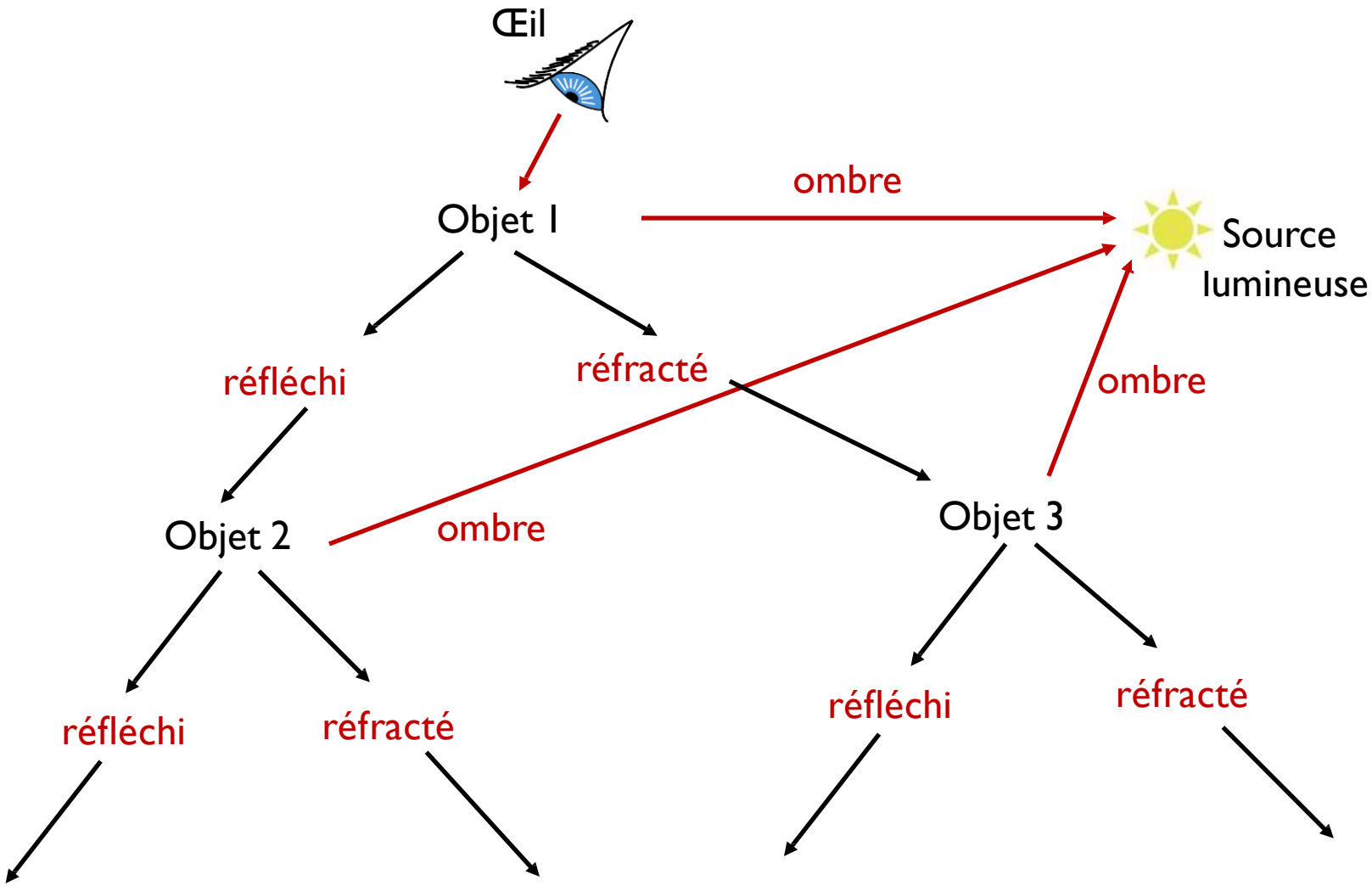


En 1980, 74 min de calcul.

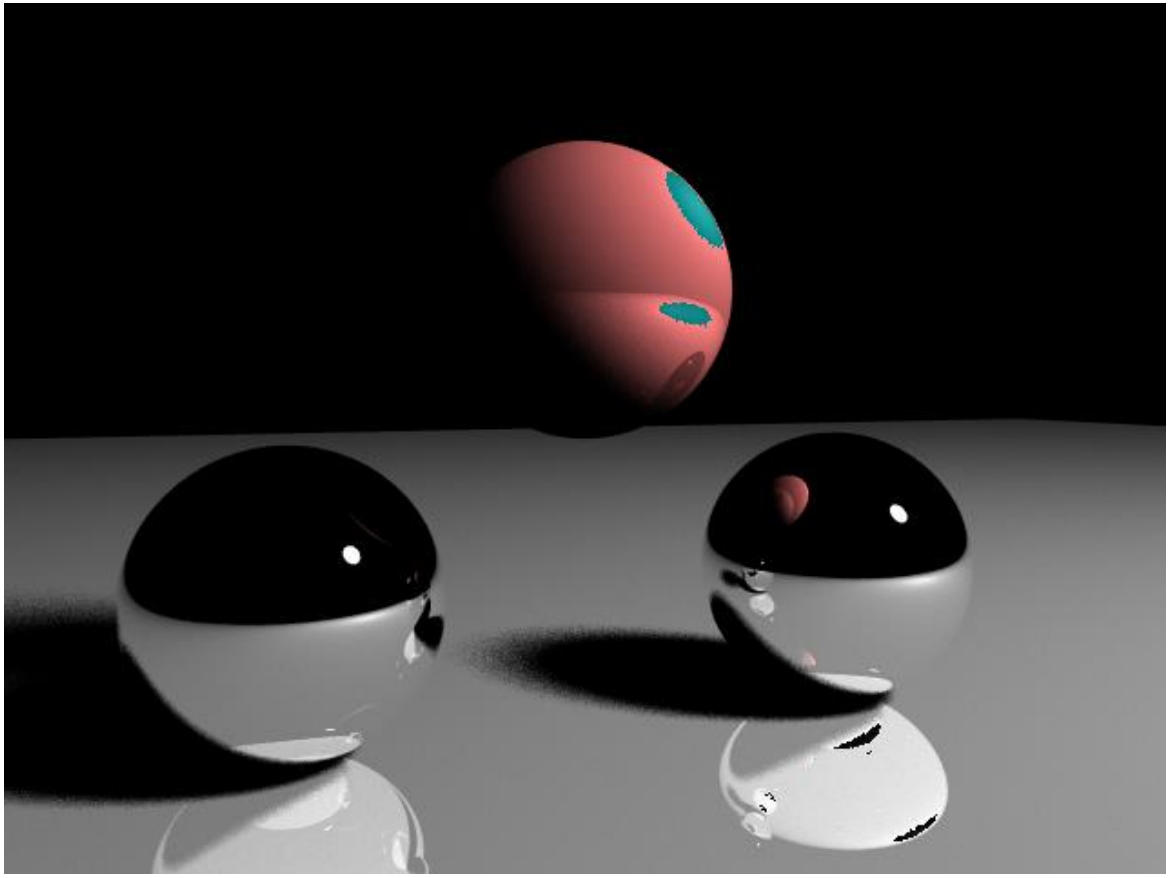


(*) formules réflexion et réfraction

L'arbre des rayons

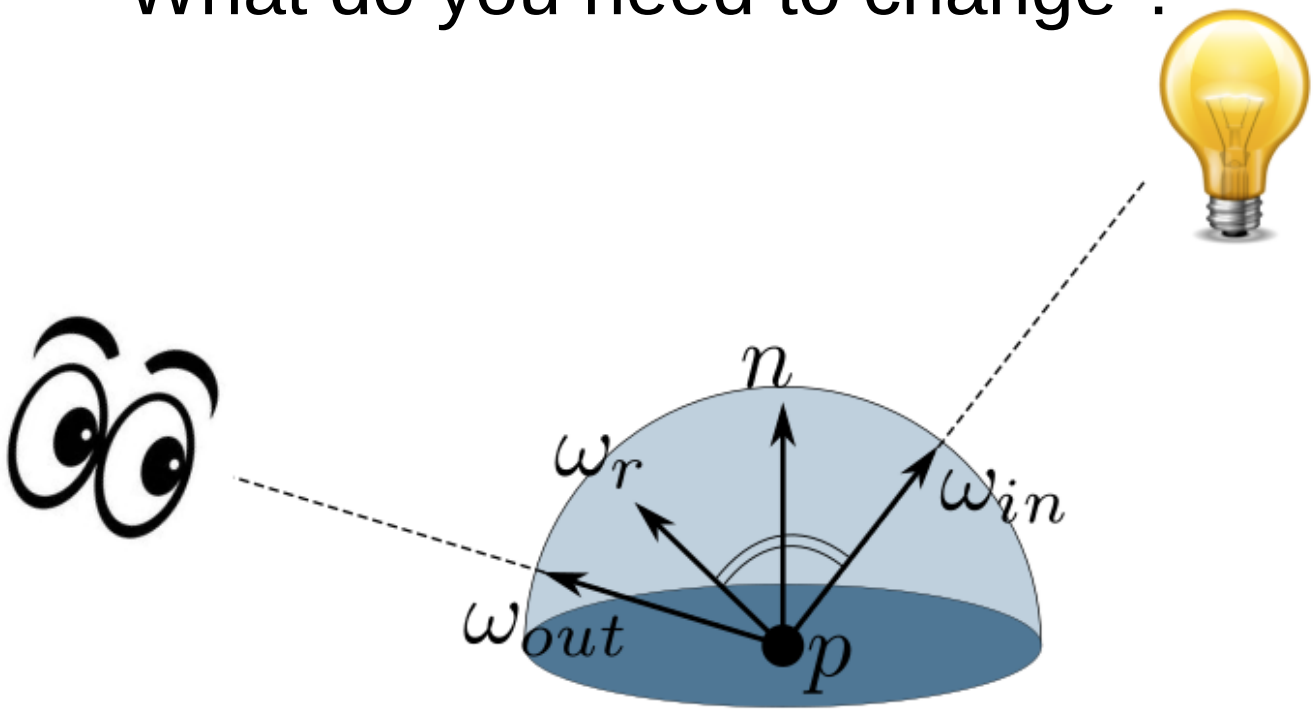


Reflections



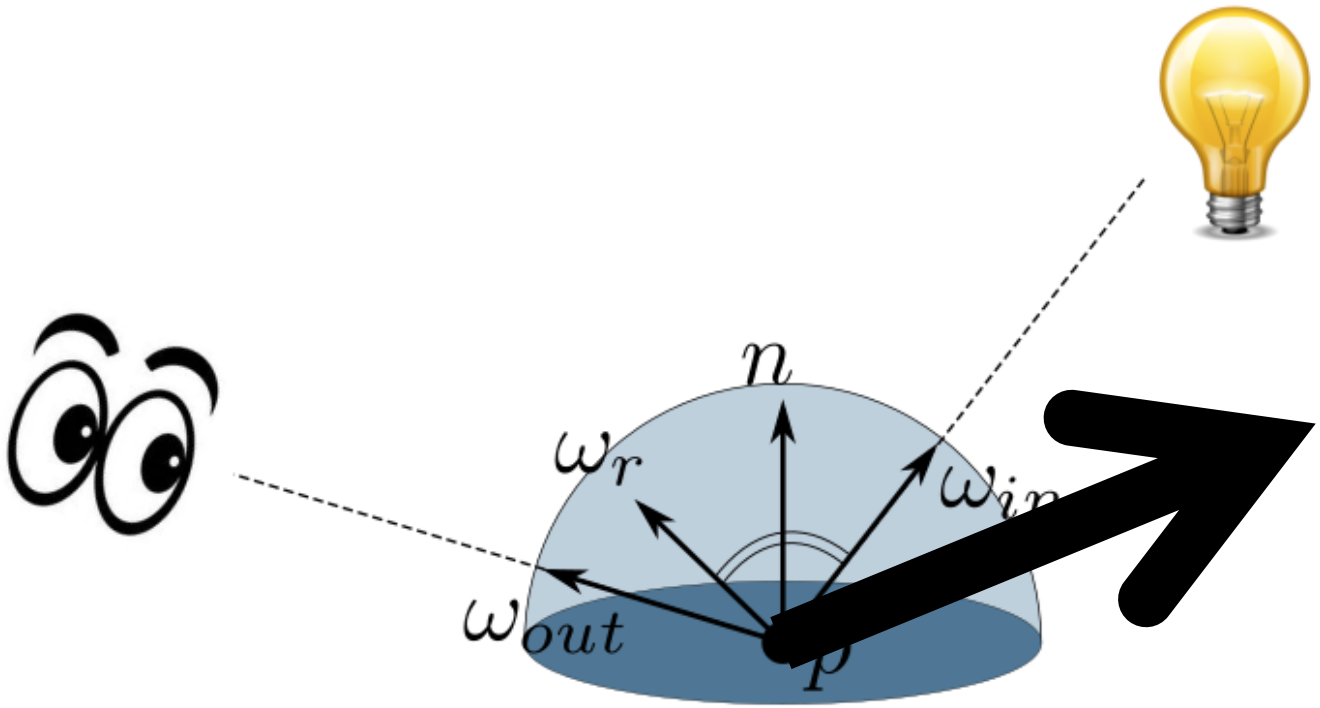
Reflections

What do you need to change ?



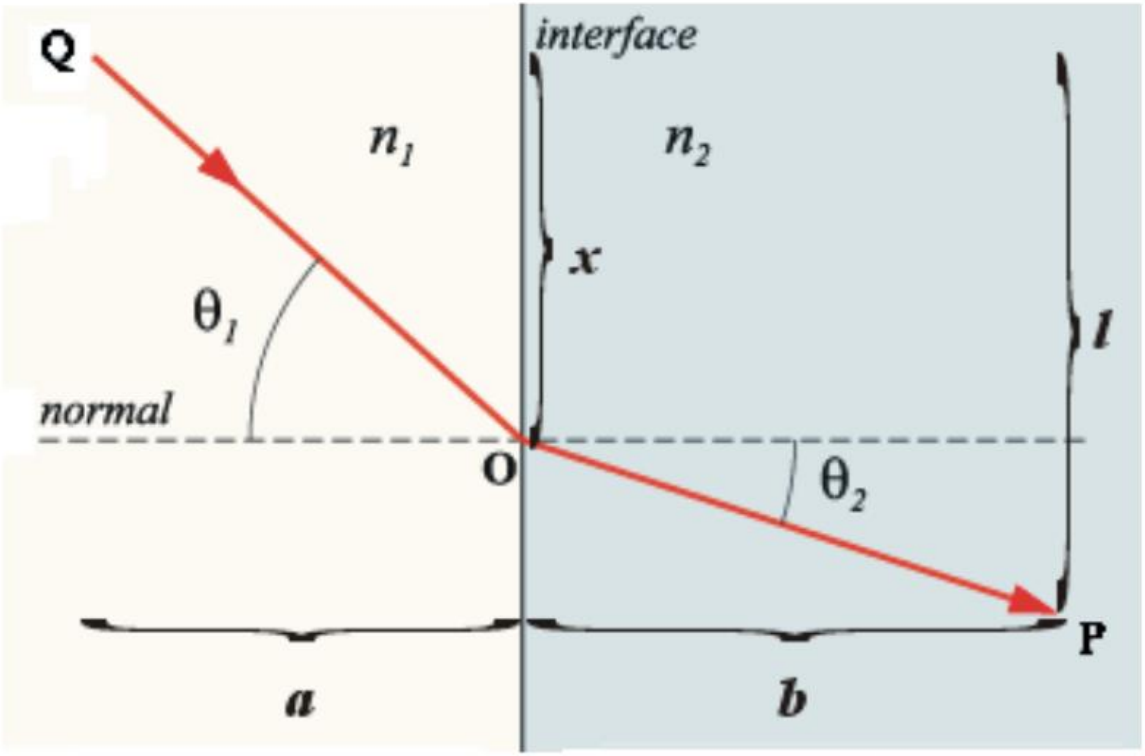
$$L(p,\omega_{out}) = L_e(p,\omega_{out}) + \int_{\omega_{in}\in\mathcal{H}(p,n)} L(p,\omega_{in}) * brdf(p,\omega_{in},\omega_{out}).\ \underbrace{(n\cdot\omega_{in})}_{\text{produit scalaire}}\ .d\omega_{in}$$

Reflections



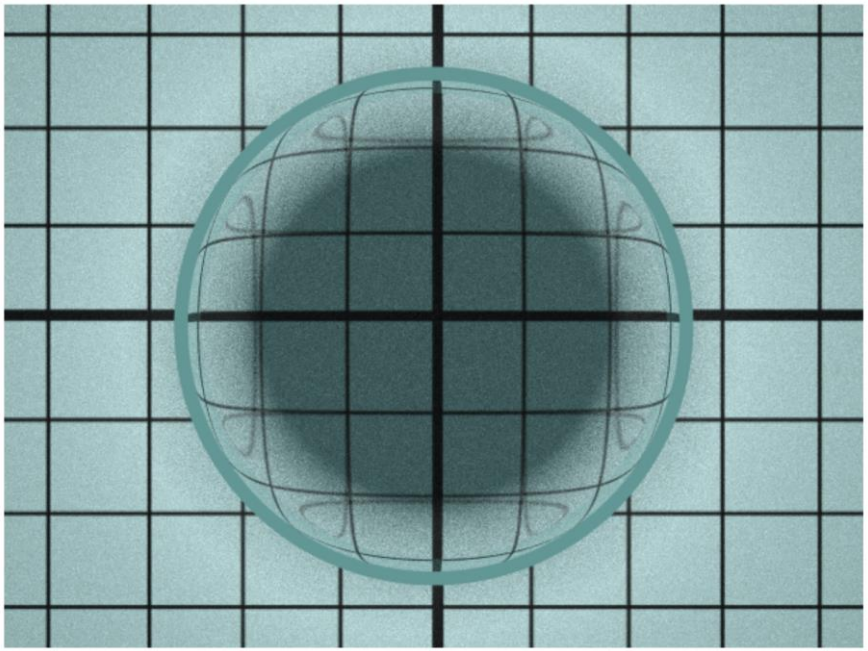
$$L(p, \omega_{out}) = L_e(p, \omega_{out}) + \int_{\omega_{in} \in \mathcal{H}(p,n)} (p, \omega_{in} * brd p, \omega_{in}, \omega_{out}). \underbrace{(n \cdot \omega_{in})}_{\text{produit scalaire}} .d\omega_{in}$$

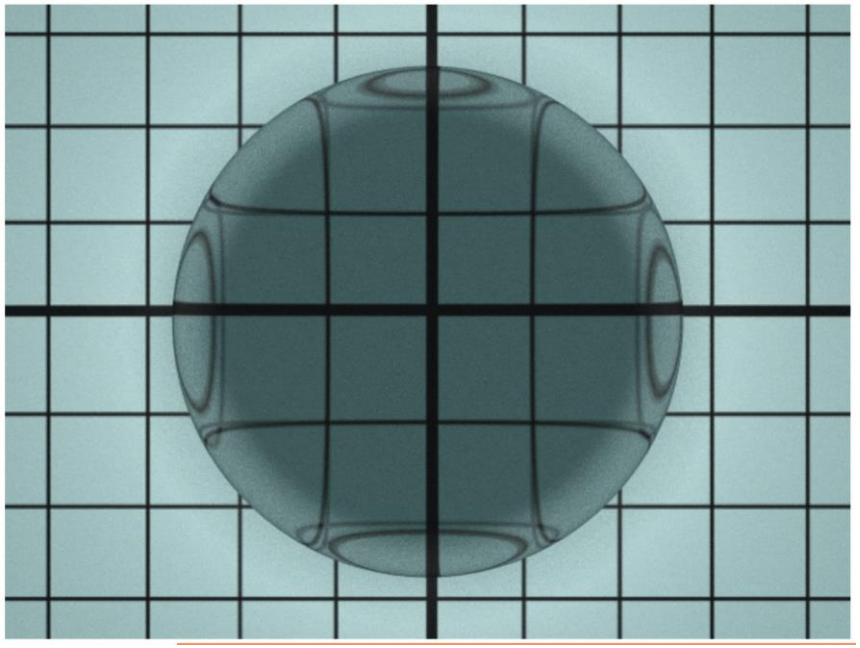
Refractions



Law of sines

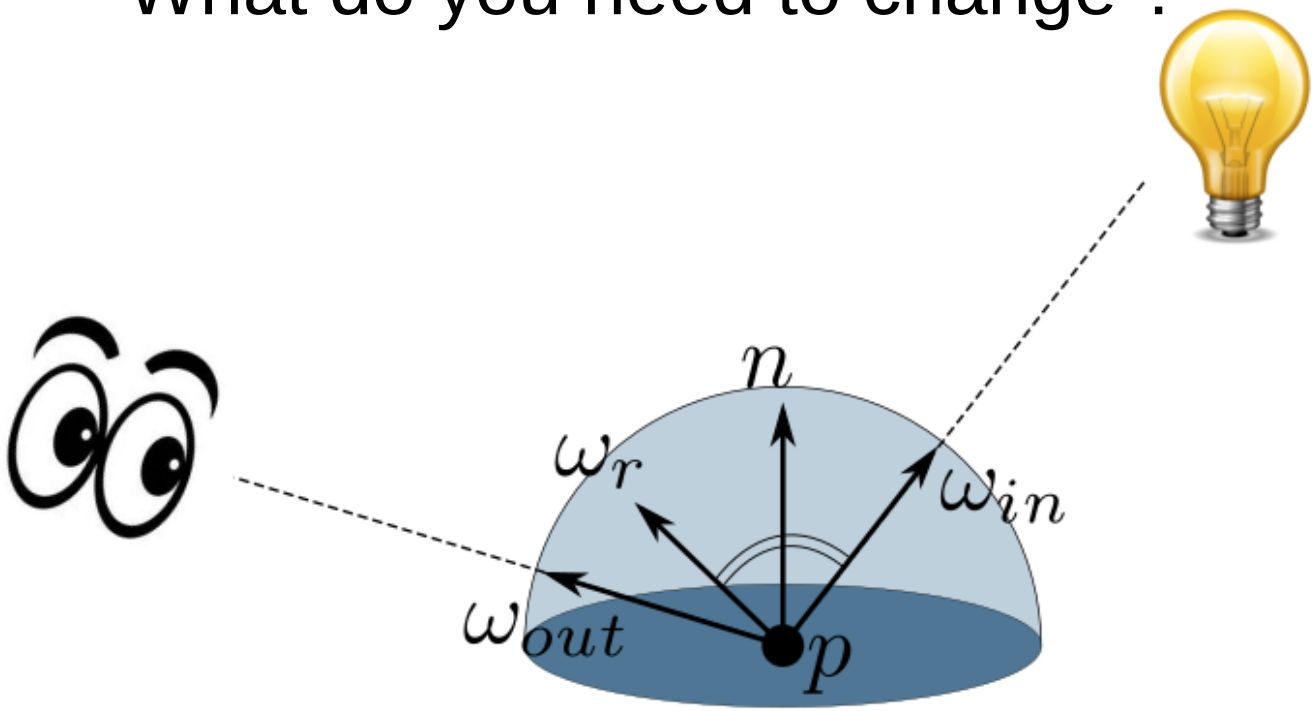
Refractions





Refractions

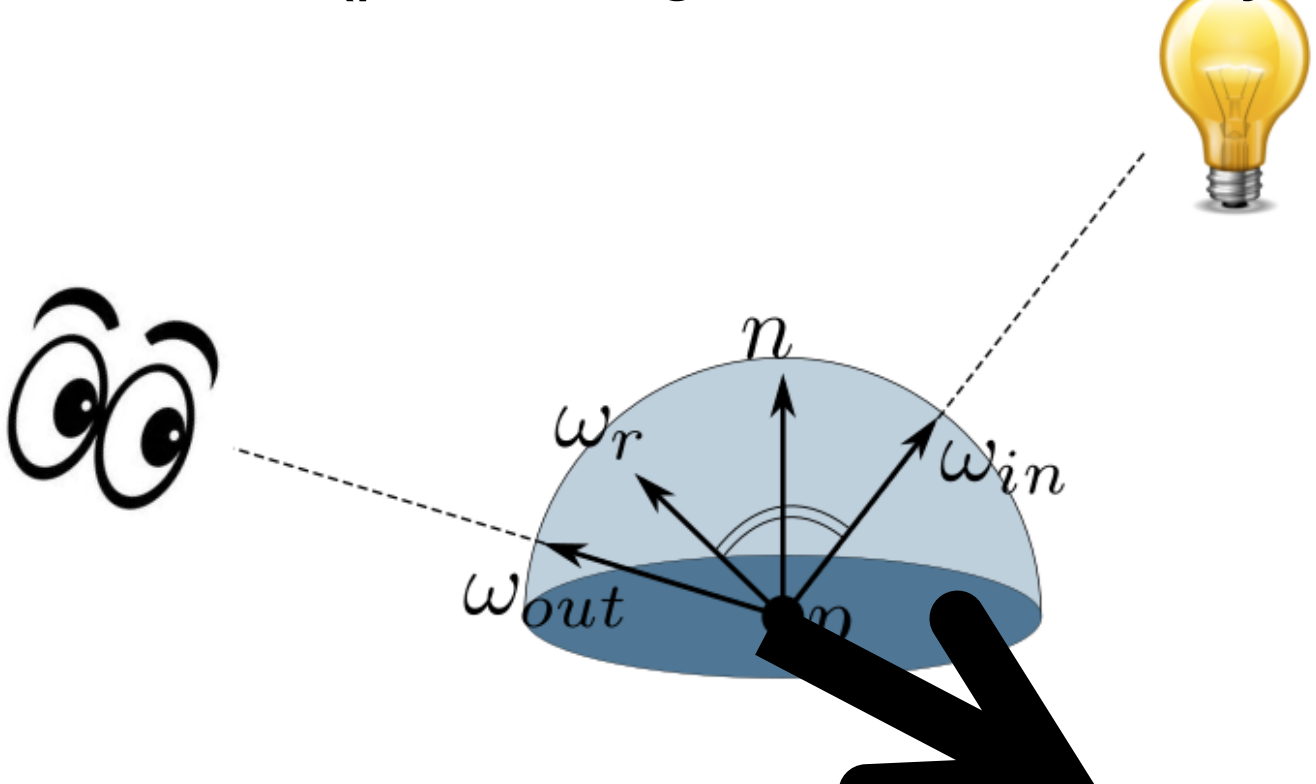
What do you need to change ?



$$L(p,\omega_{out}) = L_e(p,\omega_{out}) + \int_{\omega_{in}\in\mathcal{H}(p,n)} L(p,\omega_{in}) * brdf(p,\omega_{in},\omega_{out}).\ \underbrace{(n\cdot\omega_{in})}_{\text{produit scalaire}}\ .d\omega_{in}$$

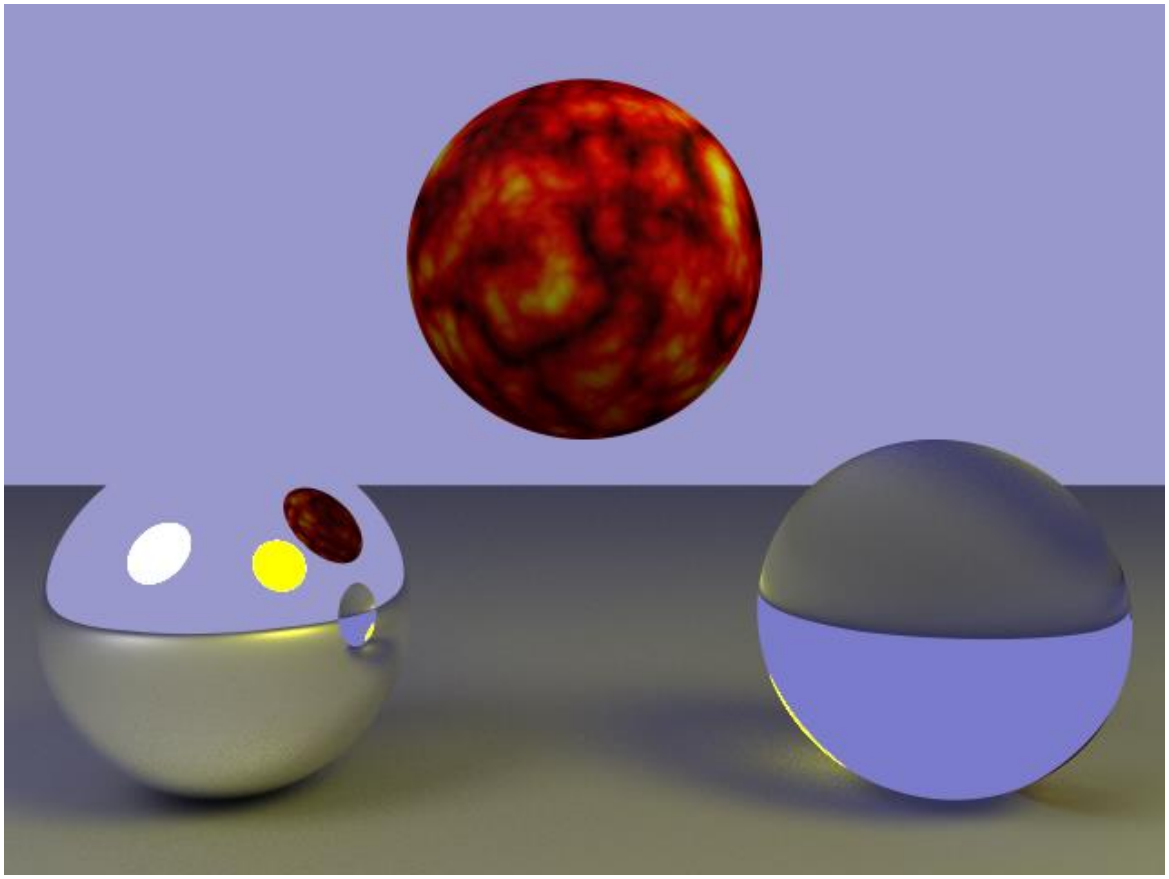
Refractions

on the other side (percentage is controlled by the transp



$$L(p,\omega_{out}) = L_e(p,\omega_{out}) + \int_{\omega_{in}\in\mathcal{H}(p,n)} L(p,\omega_{in}) * brdf(p,\omega_{in},\omega_{out}).\ \underbrace{(n\cdot\omega_{in})}_{\text{produit scalaire}}\ .d\omega_{in}$$

Refractions

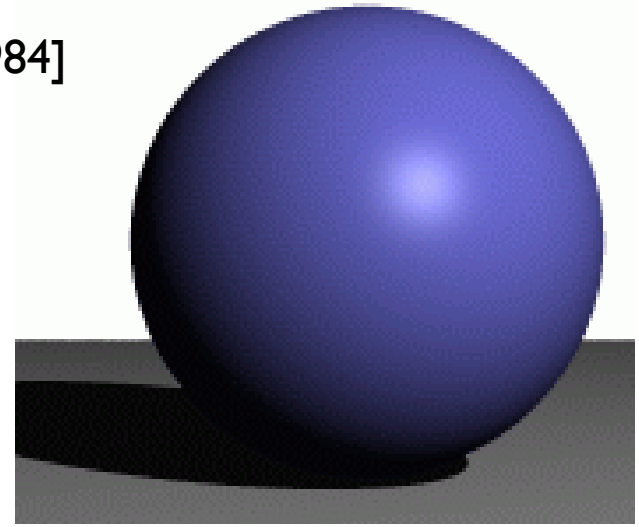


Create caustics

Encore plus de rayons

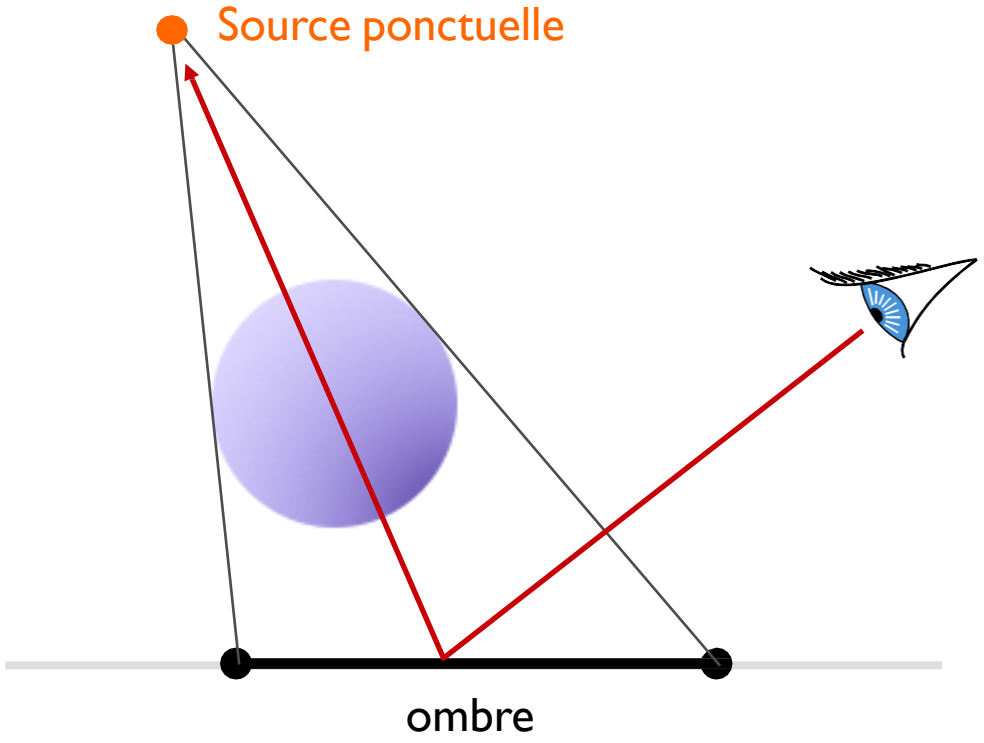
[Cook et al.1984]

source ponctuelle



Ombres douces

plusieurs rayons par source de lumière étendue

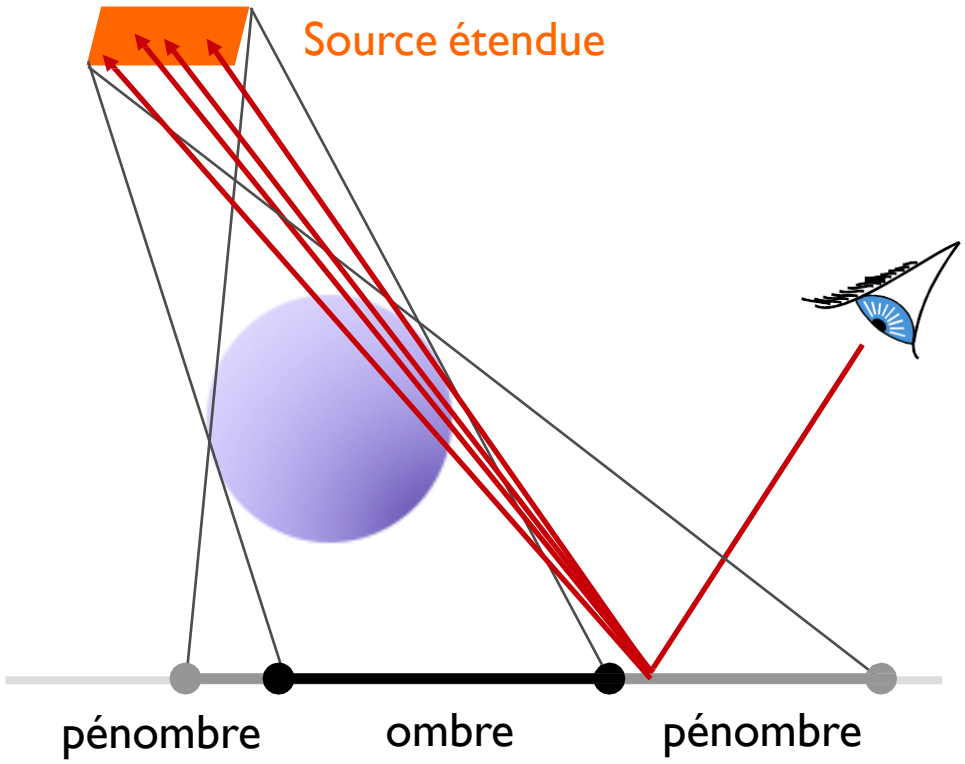


Encore plus de rayons

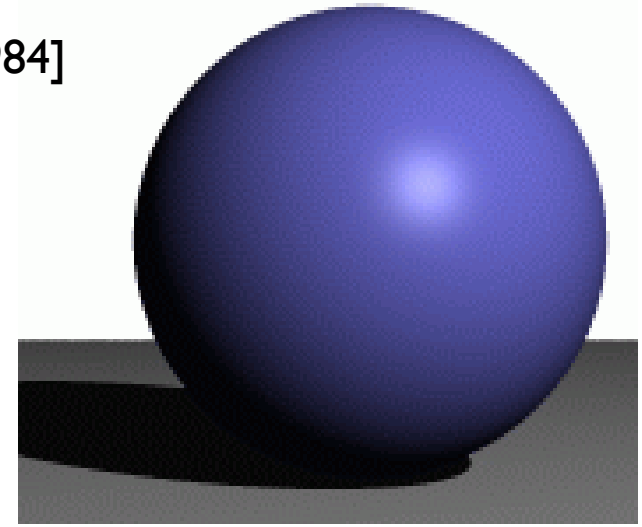
[Cook et al.1984]

Ombres douces

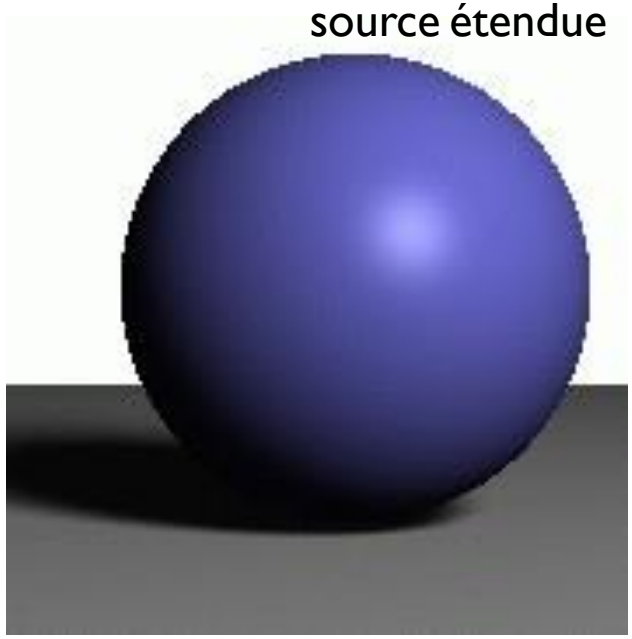
plusieurs rayons par source de lumière étendue



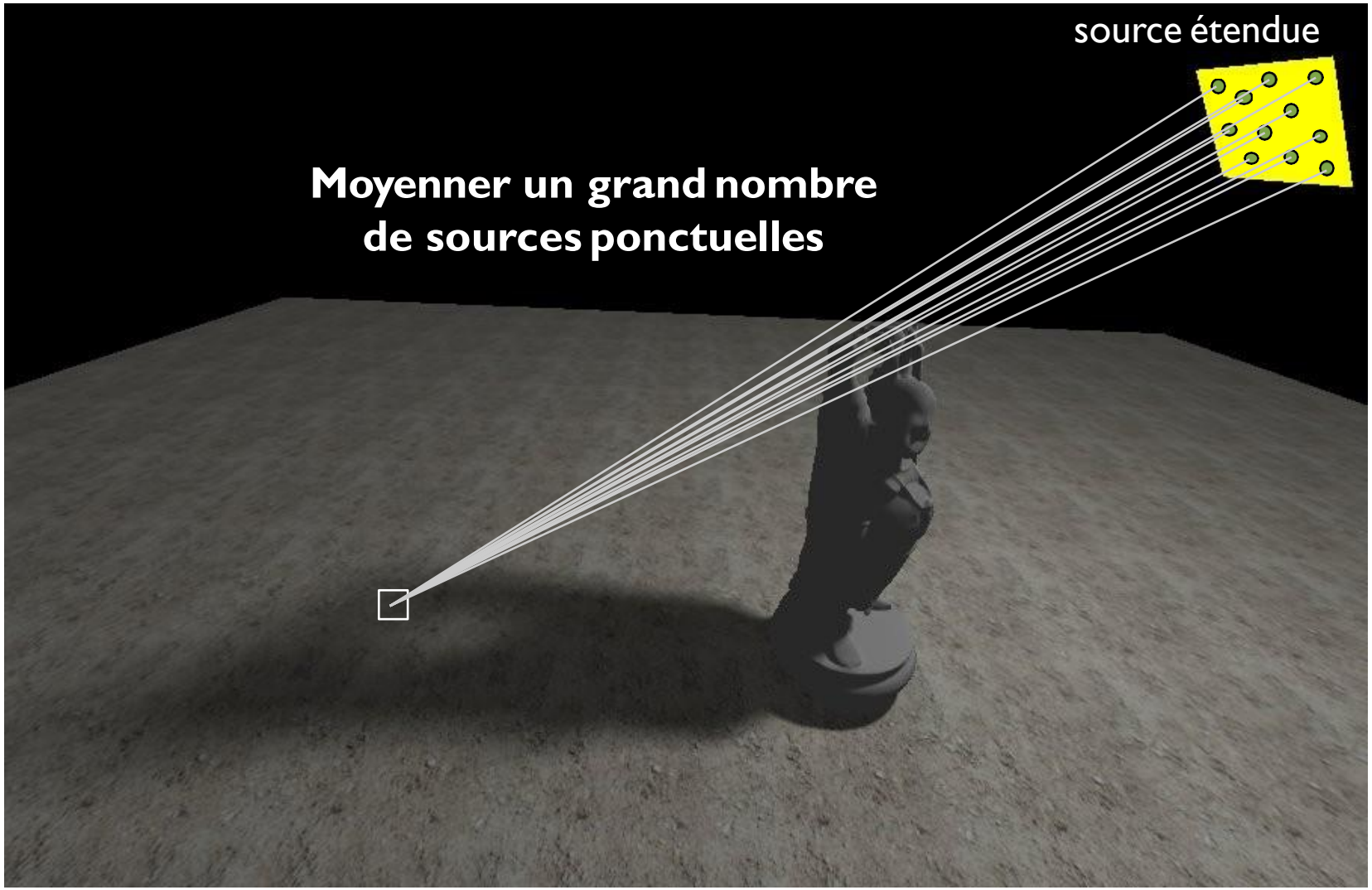
source ponctuelle



source étendue



Ombres douces

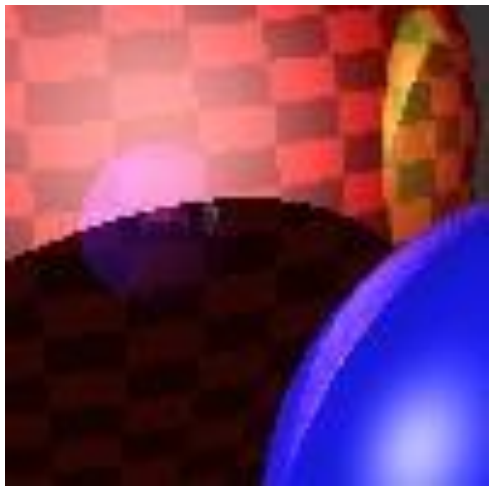


Encore plus de rayons

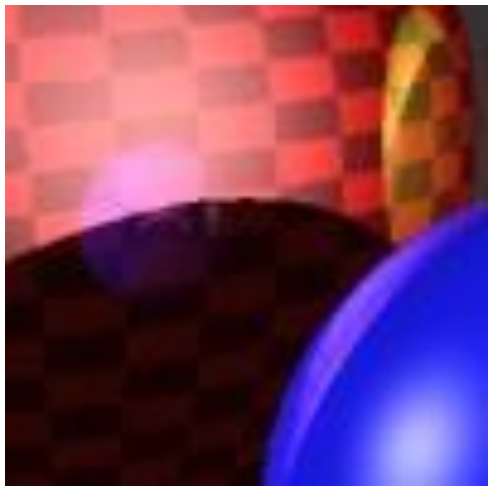
[Cook et al.1984]

Anti-aliasing

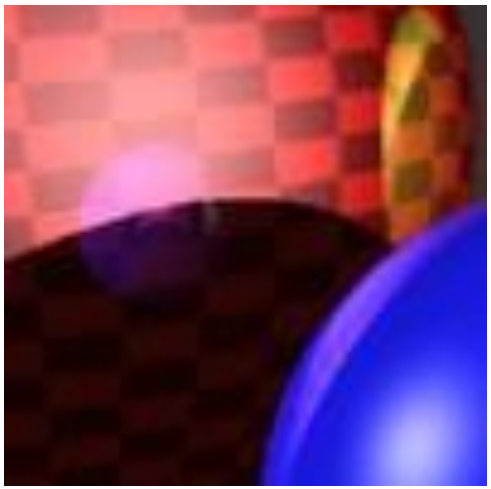
- 1 pixel ≠ 1 point ⟹ 1 pixel = petit élément de surface
- idéalement : intégrer sur toute la surface du pixel
- en pratique : plusieurs rayons par pixel et moyenner



1 rayon



2 rayons



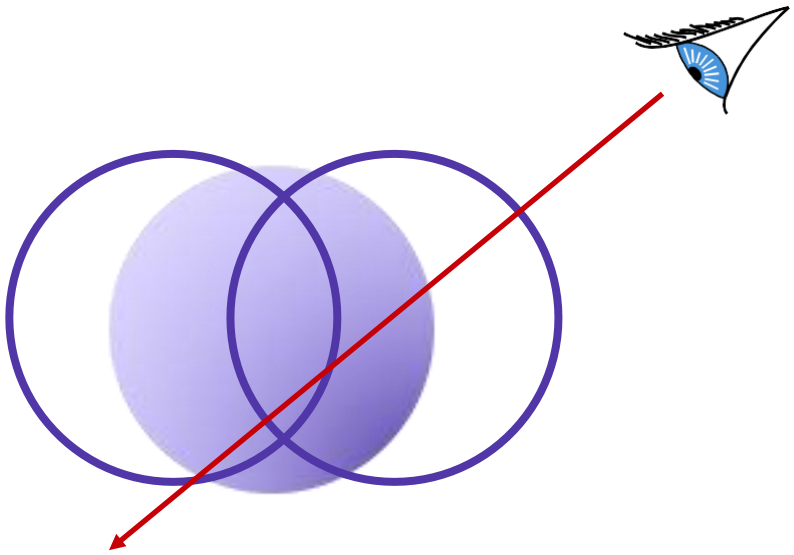
3 rayons

Encore plus de rayons

[Cook et al.1984]

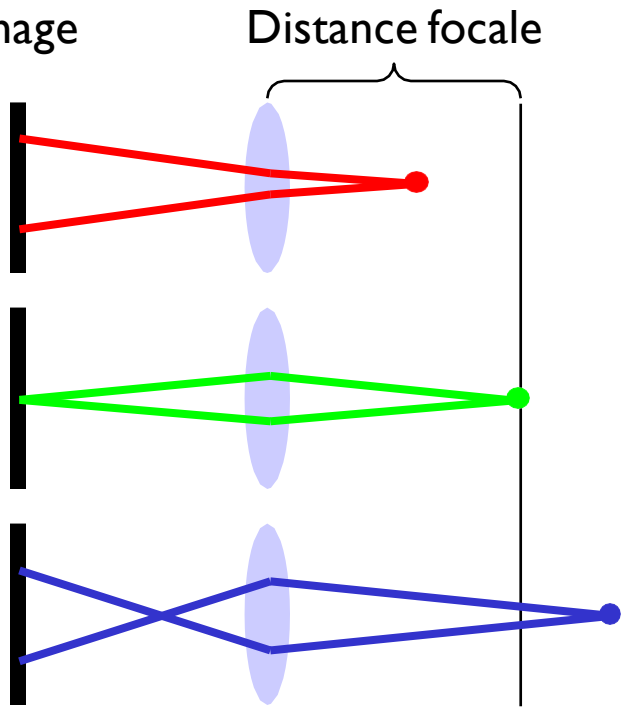
Flou cinétique

plusieurs rayon au cours du temps



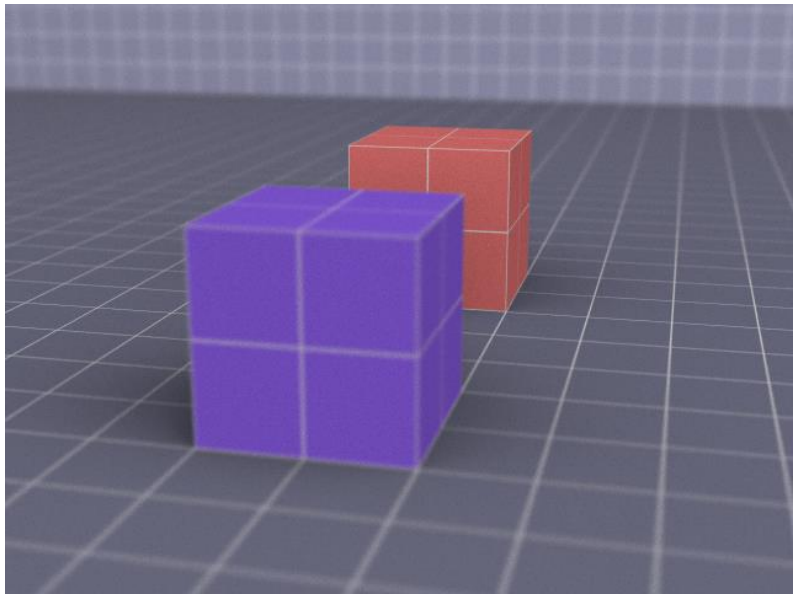
Encore plus de rayons

[Cook et al. 1984]

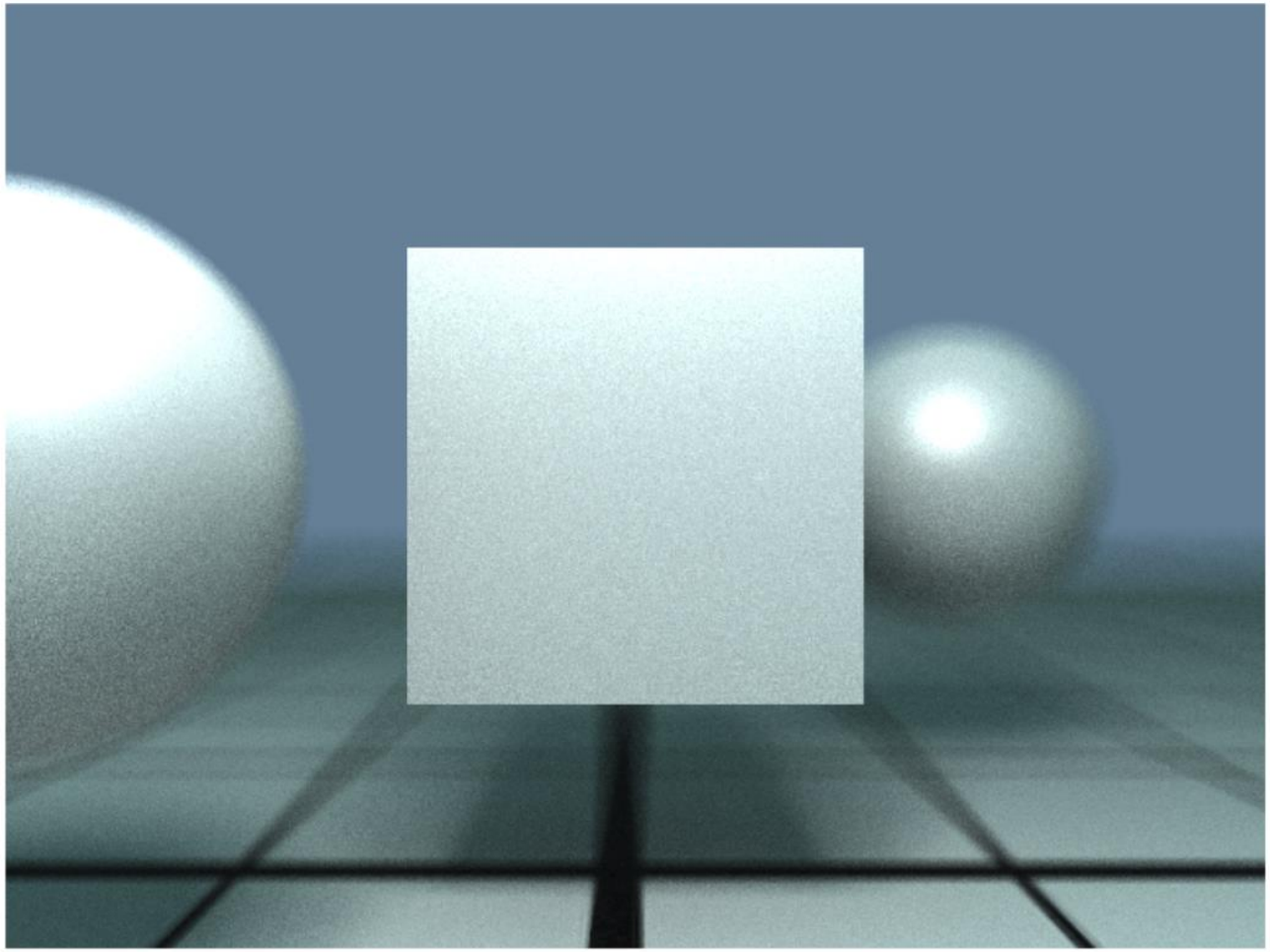


Profondeur de champ

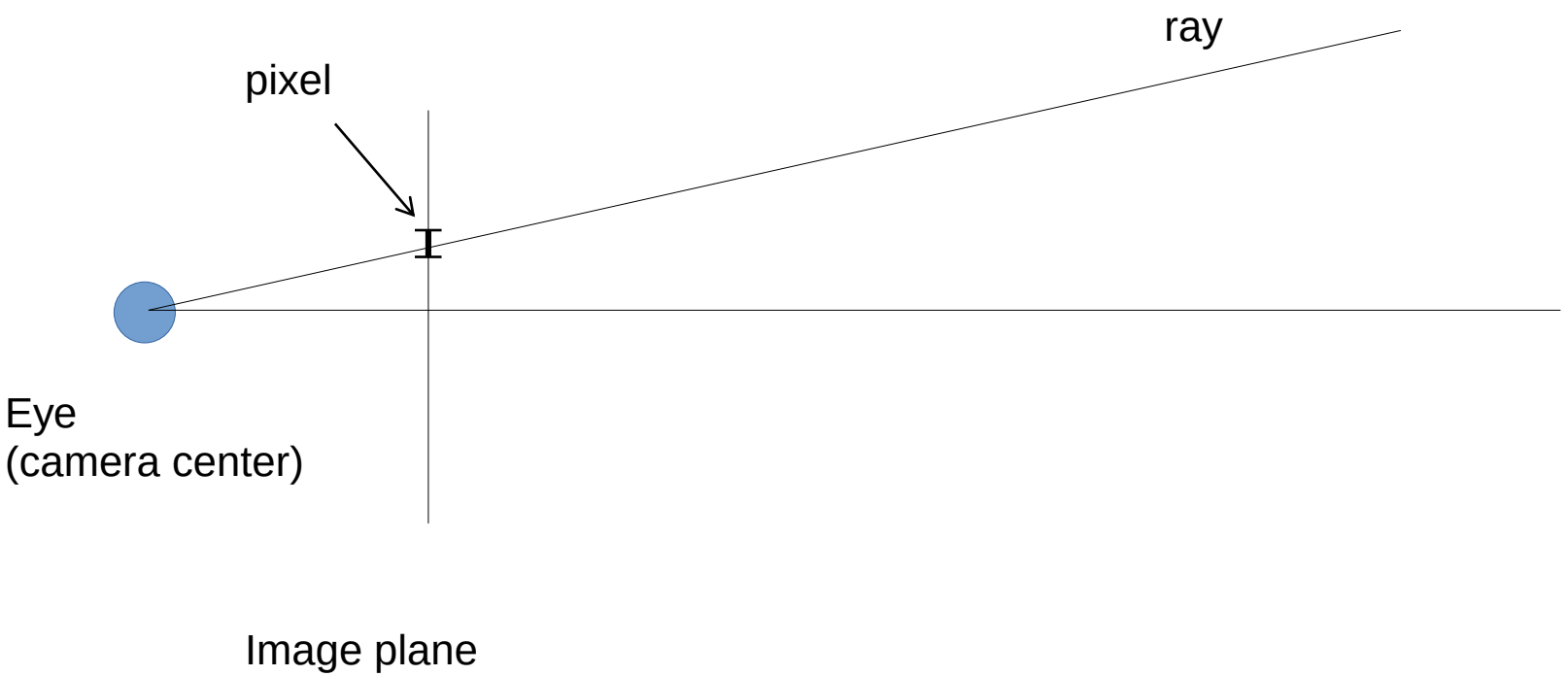
- plusieurs rayons par pixel en considérant une lentille
- changements de direction dû à l'objectif



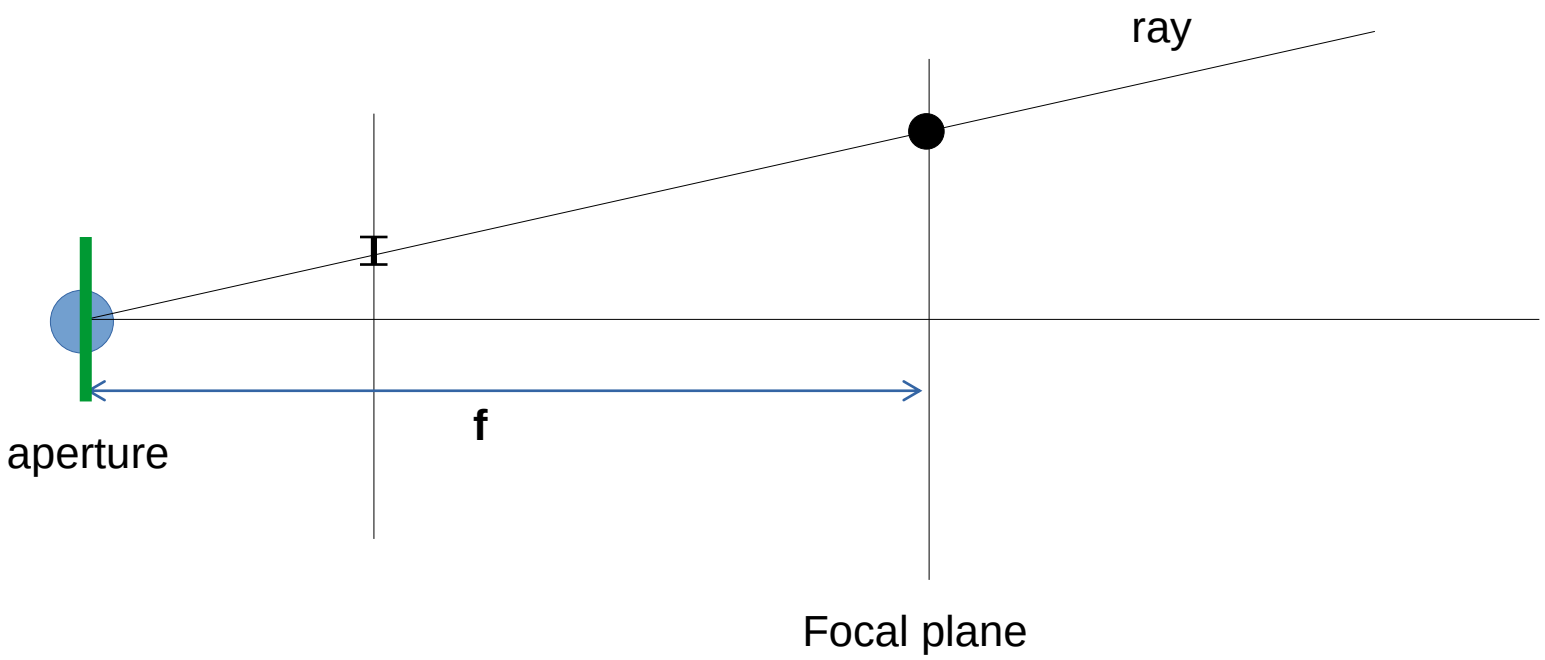
Depth of field



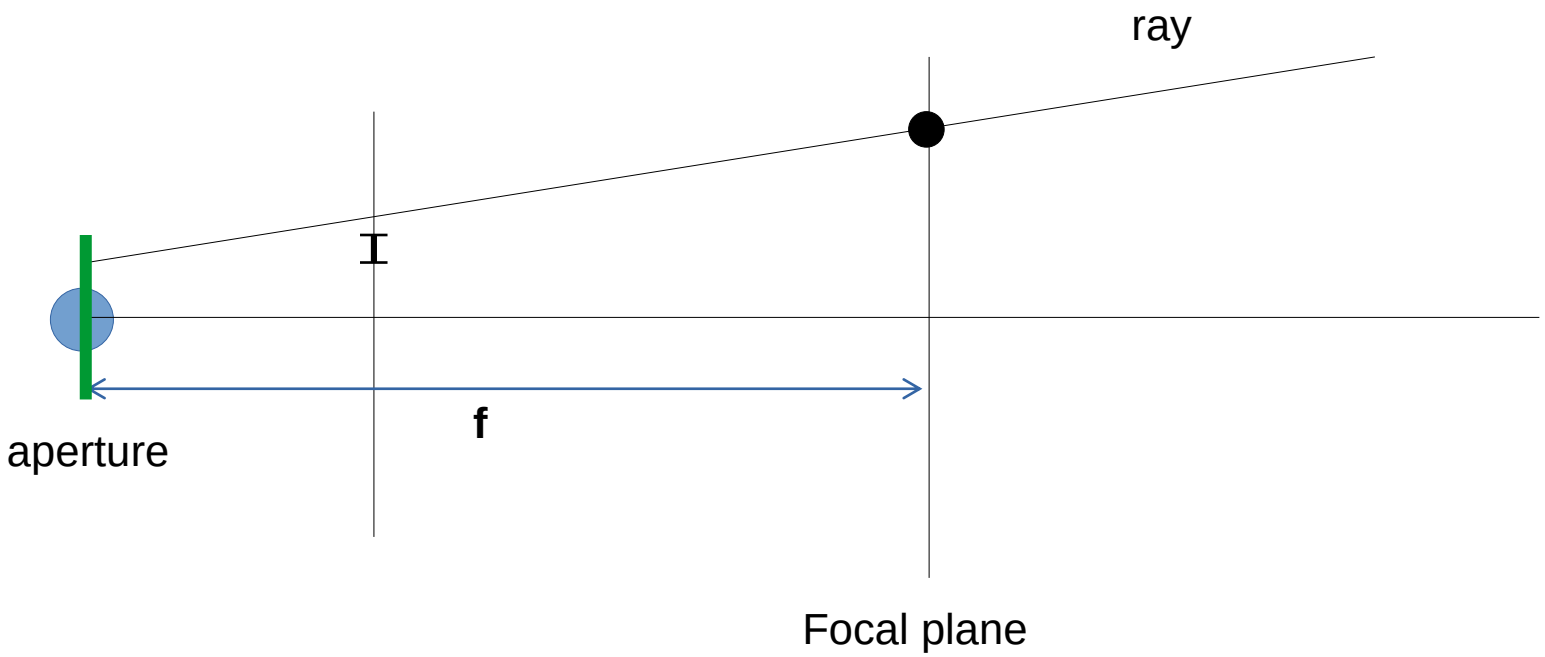
What to change ?



What to change ?



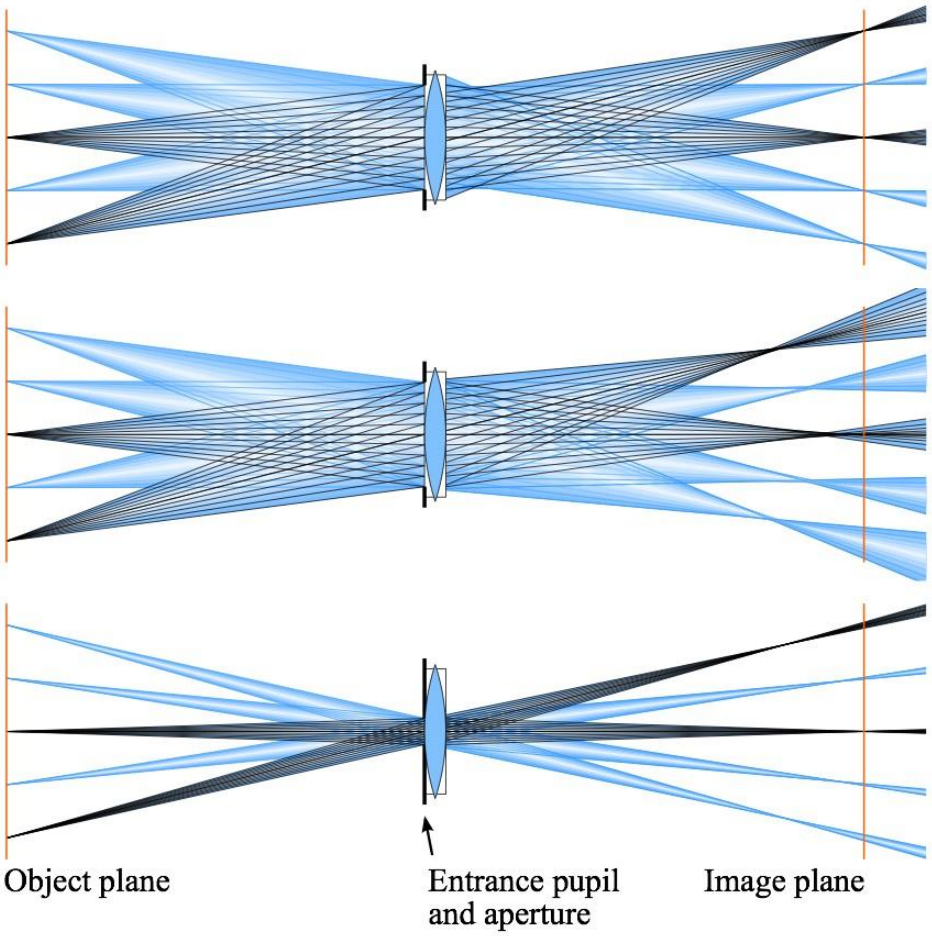
New ray from random point in aperture

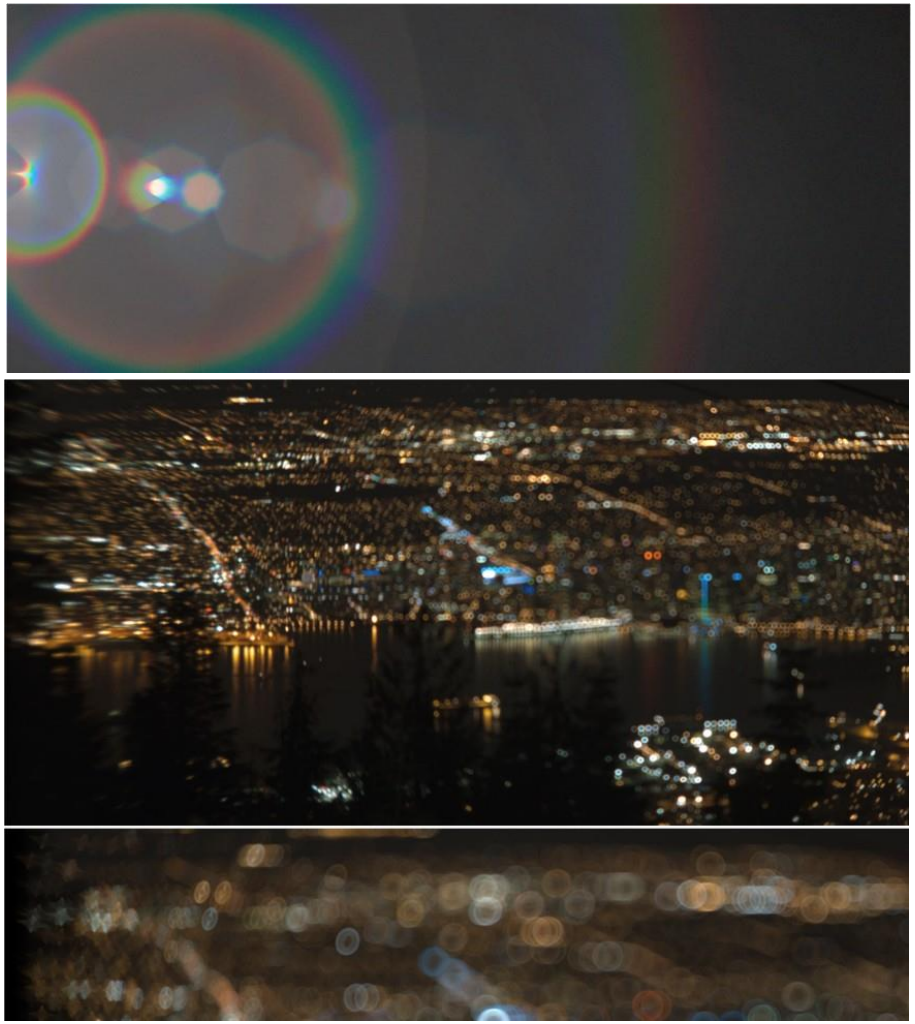


Careful ! You still add the contribution of the ray to the original pixel !

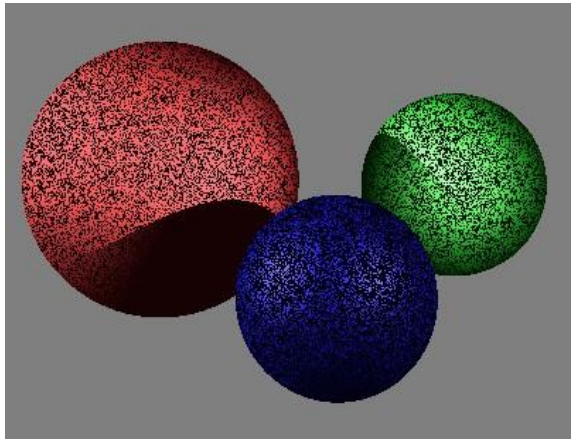
Simulation avancée

En pratique les optiques ne sont pas parfaites



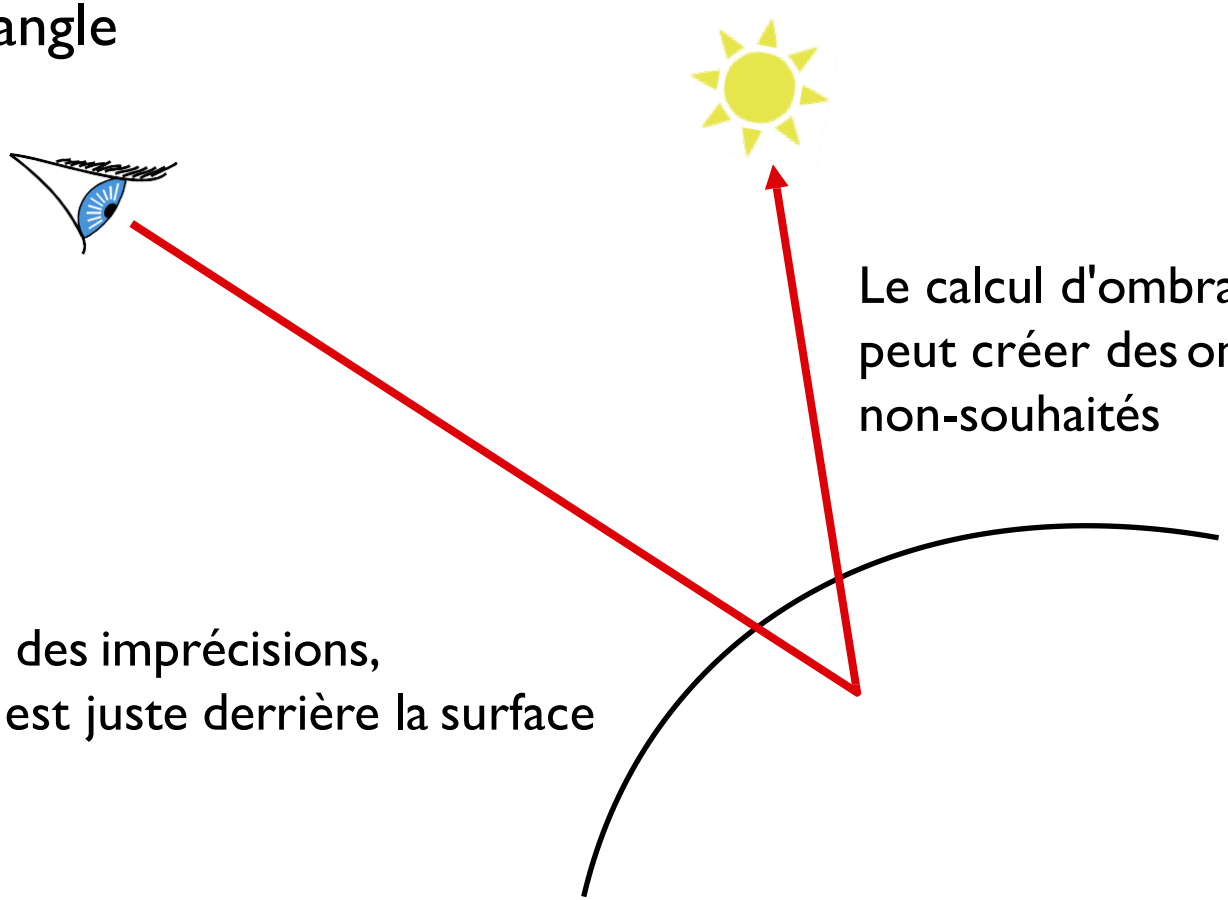


Problème de précision



Un point n'est jamais **exactement**

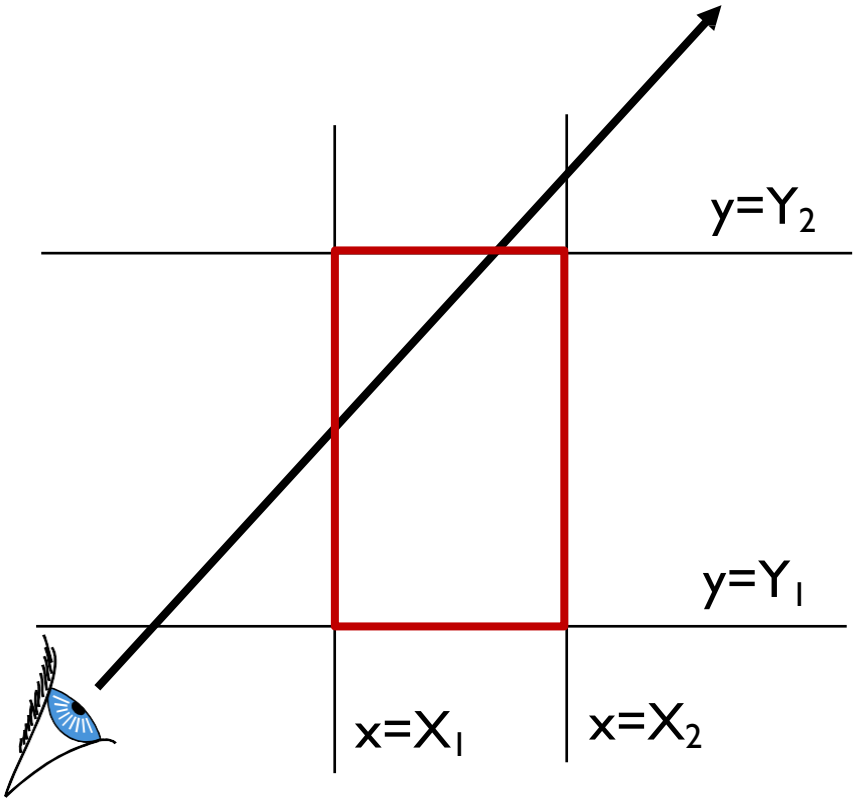
- sur le plan ou la sphère
- dans le triangle



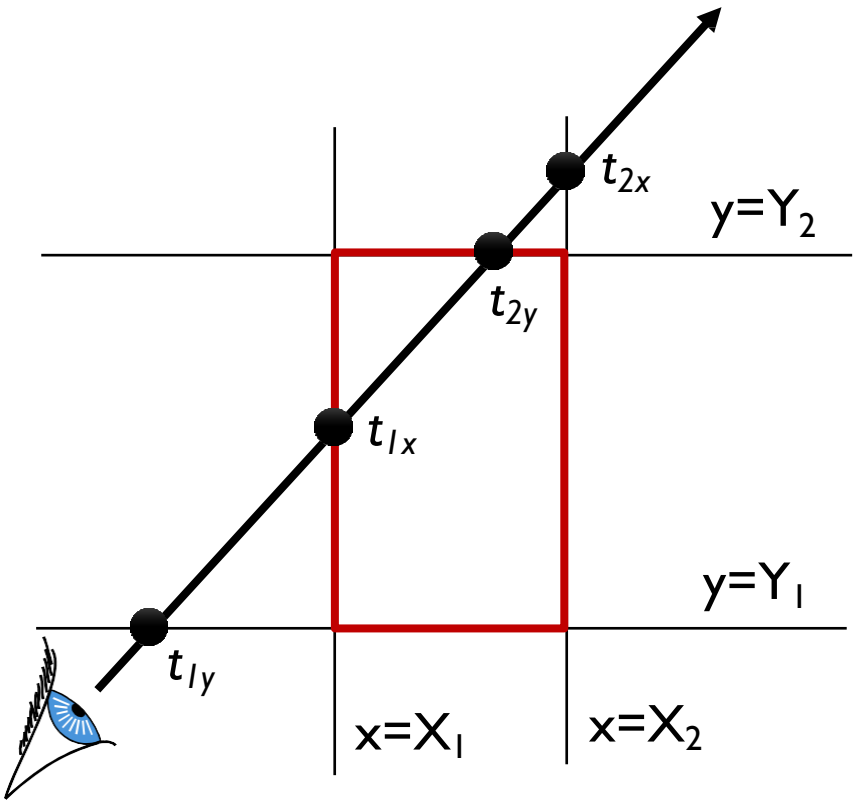
Le calcul d'ombrage peut créer des ombres non-souhaités

À cause des imprécisions,
le point est juste derrière la surface

Boîte englobante alignée sur les axes

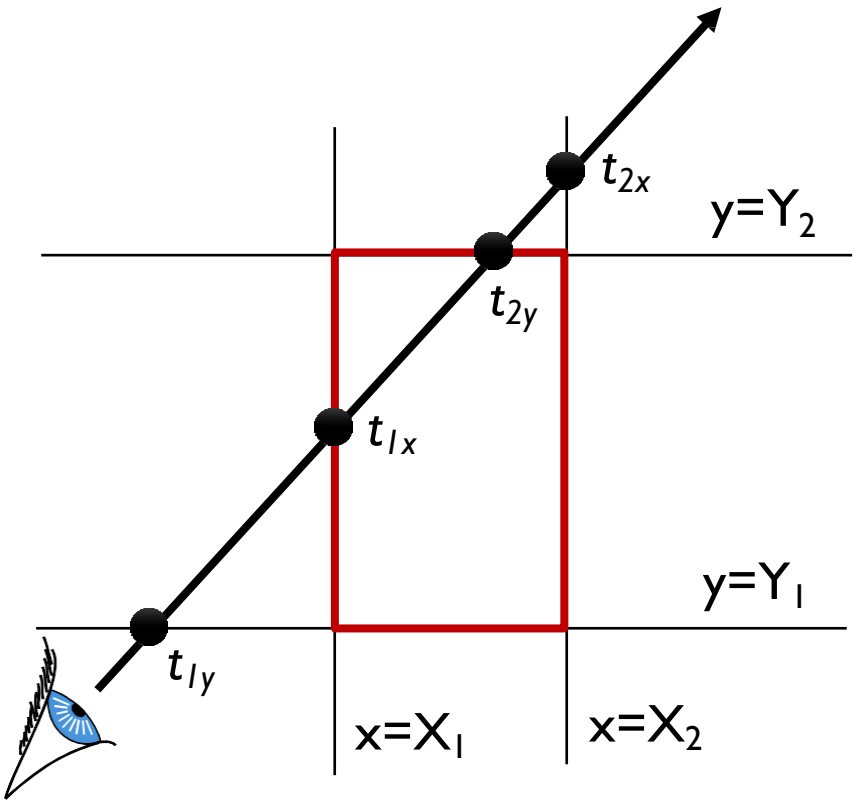


Boîte englobante alignée sur les axes



- Boîte = 6 plans
- Calculer toutes les intersections
- Conserver la plus proche **à l'intérieur** de la boîte

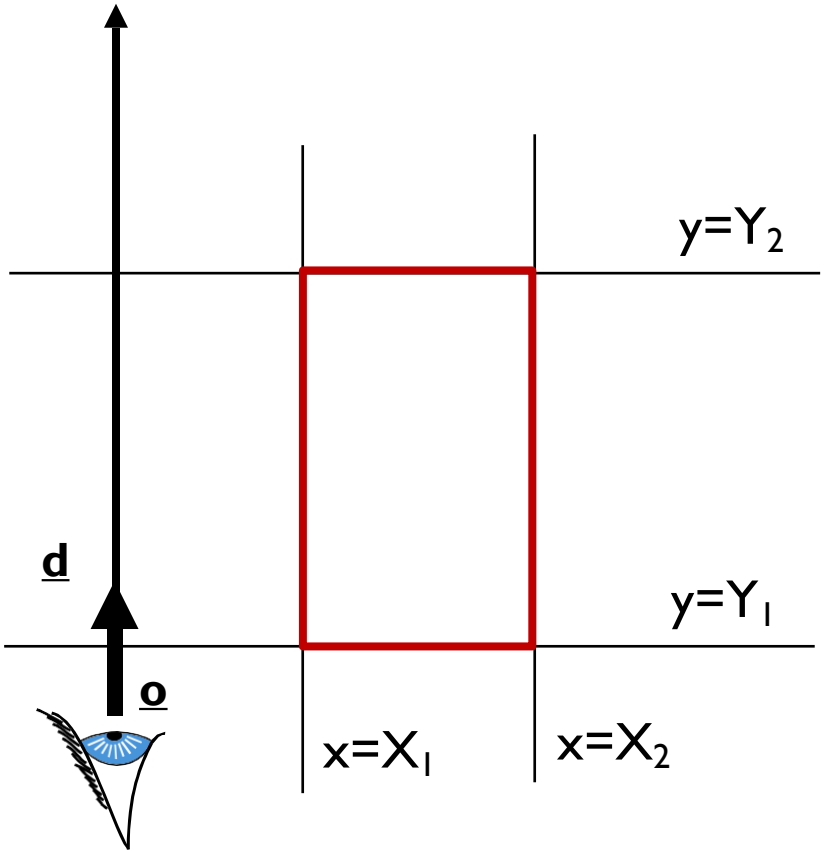
Simplifier les calculs



- Chaquepaire de plan a la **mêmes normale**
- Une **seule** composante de la normale est **non-nulle**

$\Longrightarrow$ considérer une dimension à la fois

Tester si le rayon est parallèle



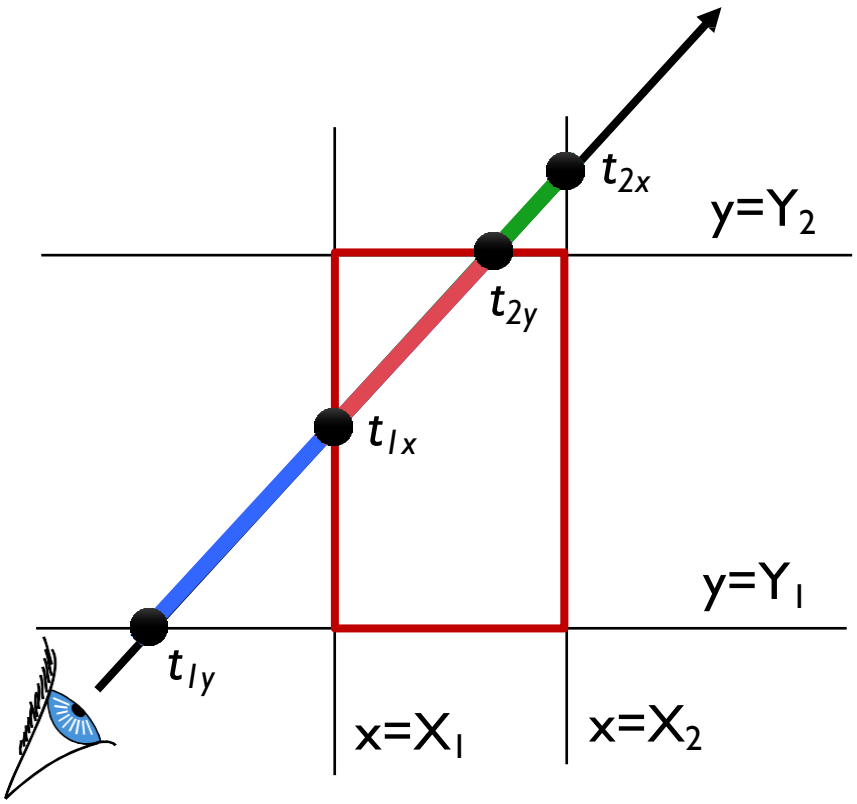
$\underline{d}_x = 0$

…et s'il n'y a pas d'intersection :

$\underline{o}_x < X_1$ ou $\underline{o}_x > X_2$

(Idem enY et Z, bien-sûr)

Trouver les intersections par axe



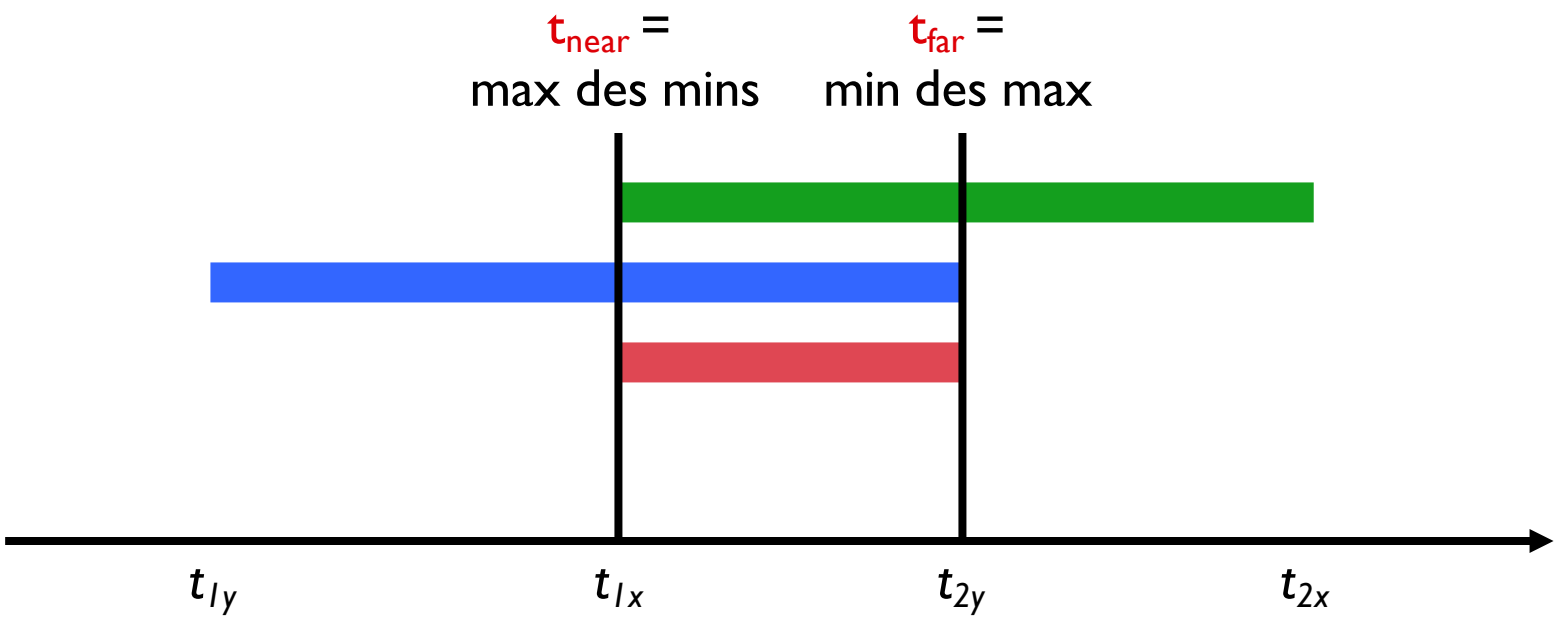
- Déterminer un intervalle par dimension
- Calculer l'intersection de ces intervalles 1D

Intervalle entre X_1 et X_2

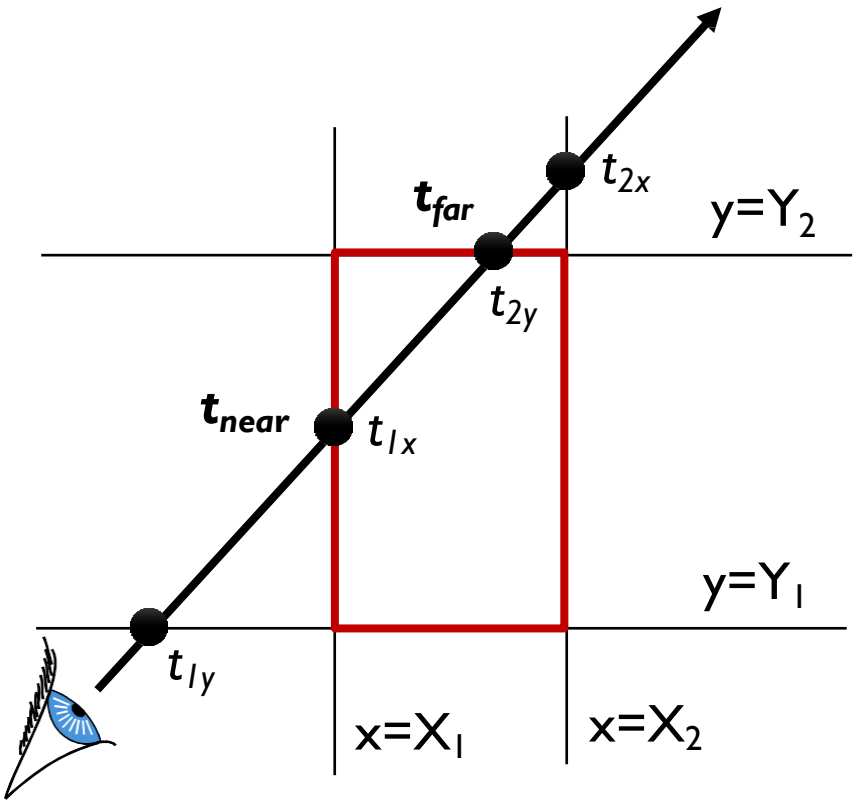
Intervalle entre Y_1 et Y_2

Intersection

Intersection d'intervalles 1D



Boîte englobante alignée sur les axes



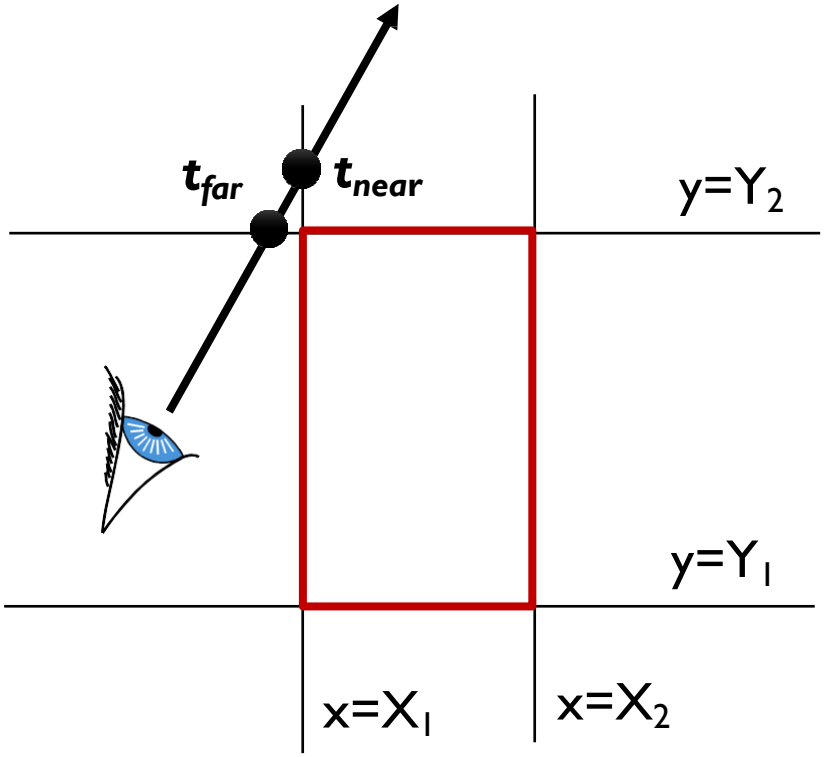
Calculer les distances d'intersection t_1 et t_2 pour chaque axe :

- $t_{1x} = (X_1 - o_x) / \mathbf{d}_x$
- $t_{2x} = (X_2 - o_x) / \mathbf{d}_x$

puis :

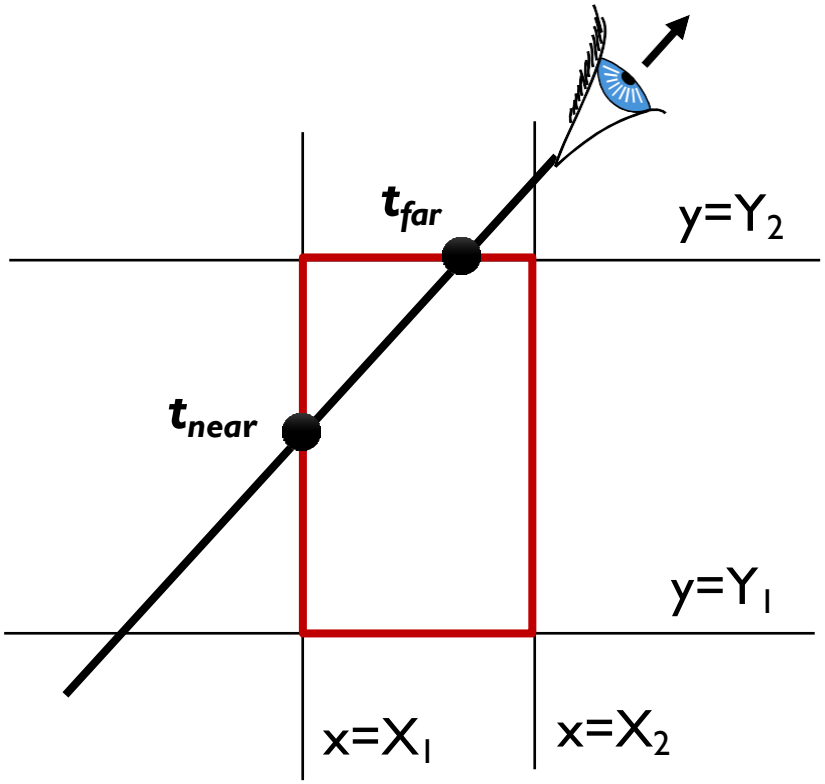
- $t_{near} = \max(t_{1x}, t_{1y})$
- $t_{far} = \min(t_{2x}, t_{2y})$

Boîte englobante alignée sur les axes



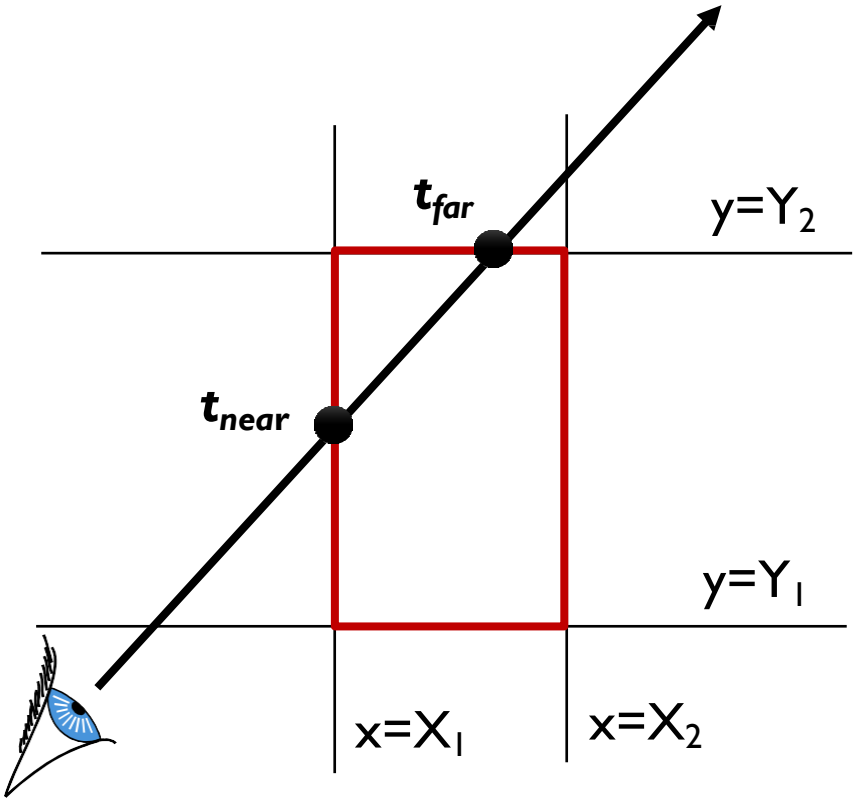
Si $t_{near} > t_{far}$, pas d'intersection

Boîte englobante alignée sur les axes



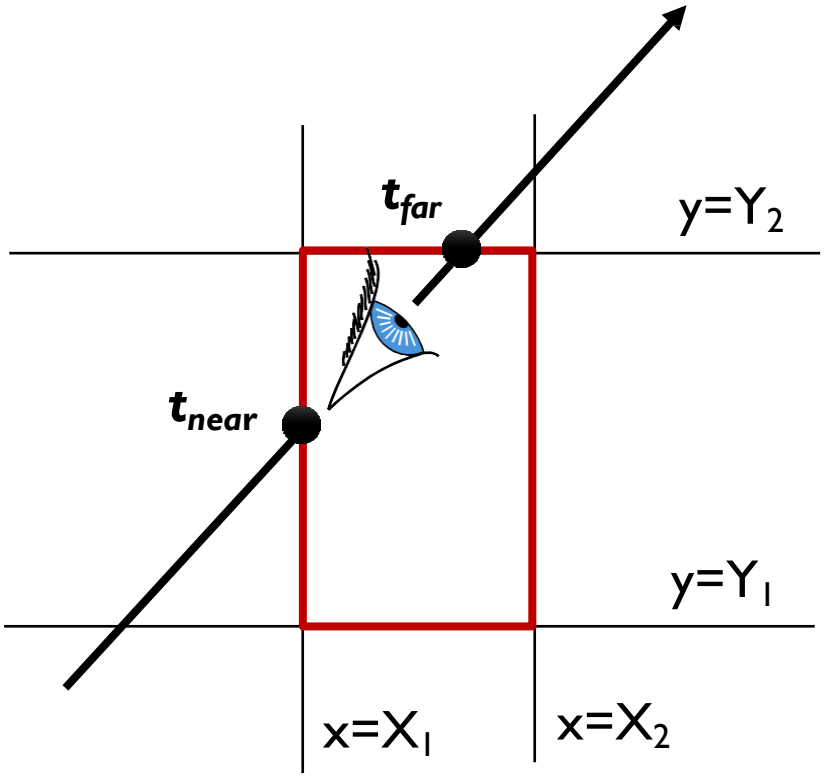
Si $t_{far} < t_{min}$, la boîte est derrière

Boîte englobante alignée sur les axes



Si $t_{near} > t_{min}$, l'intersection se fait à la distance t_{near}

Boîte englobante alignée sur les axes



Sinon, elle se fait à t_{far}

Intersection rayon-scène

- Polygones : [Appel '68]
- Quadriques, CSG : [Goldstein & Nagel '71]
- Torres : [Roth '82]
- Patches bi-cubiques : [Whitted '80, Kajiya '82, Benthin '04]
- Surfaces algébriques : [Hanrahan '82]
- Swept surfaces : [Kajiya '83, van Wijk '84]
- Fractales : [Kajiya '83]
- NURBS : [Stürzlinger '98]
- Surfaces de subdivision : [Kobbelt et al. '98, Benthin '04]
- Points : [Schaufler et al. '00, Wald '05]

Structures d'accélération

Trouver l'intersection la plus proche

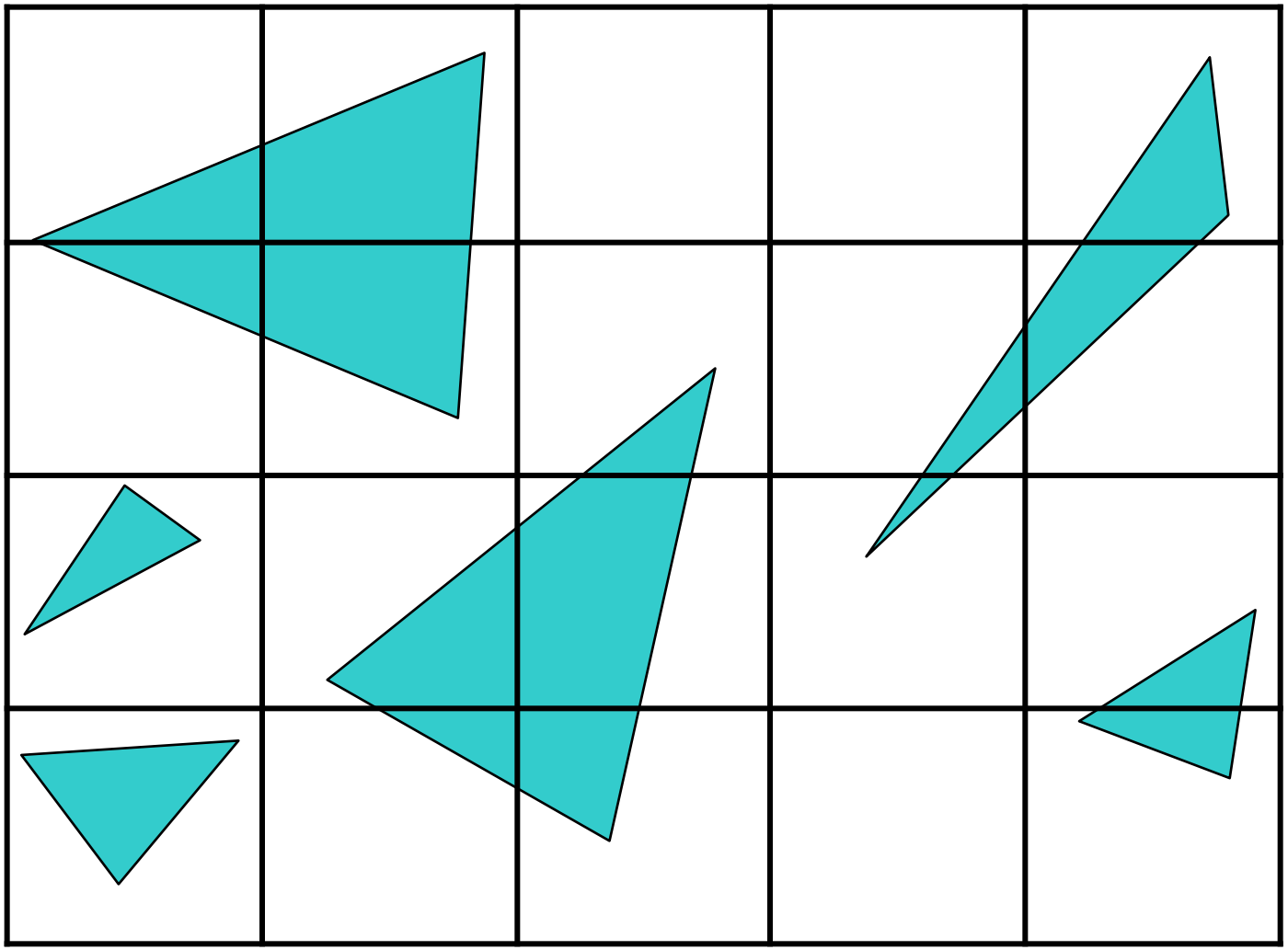
- Tester l'intersection du rayon avec **tous** les objets
- N objets, M rayons $\Longrightarrow$ O(NM)
- Trop coûteux !

Objectif

- Faire en sorte que la 1ère intersection calculée soit la bonne
- En pratique :compromis

Solution : partitionnement de l'espace
(souvent hiérarchique)

Grille régulière



Grille régulière

Construction

- Subdivision de la boîte englobante
- Résolution : souvent $\sim \sqrt[3]{n}$
- Une cellule : liste des objets l'intersectant

Parcours

- De proche en proche
- De l'origine vers l'arrière
- Arrêt si une intersection est trouvée

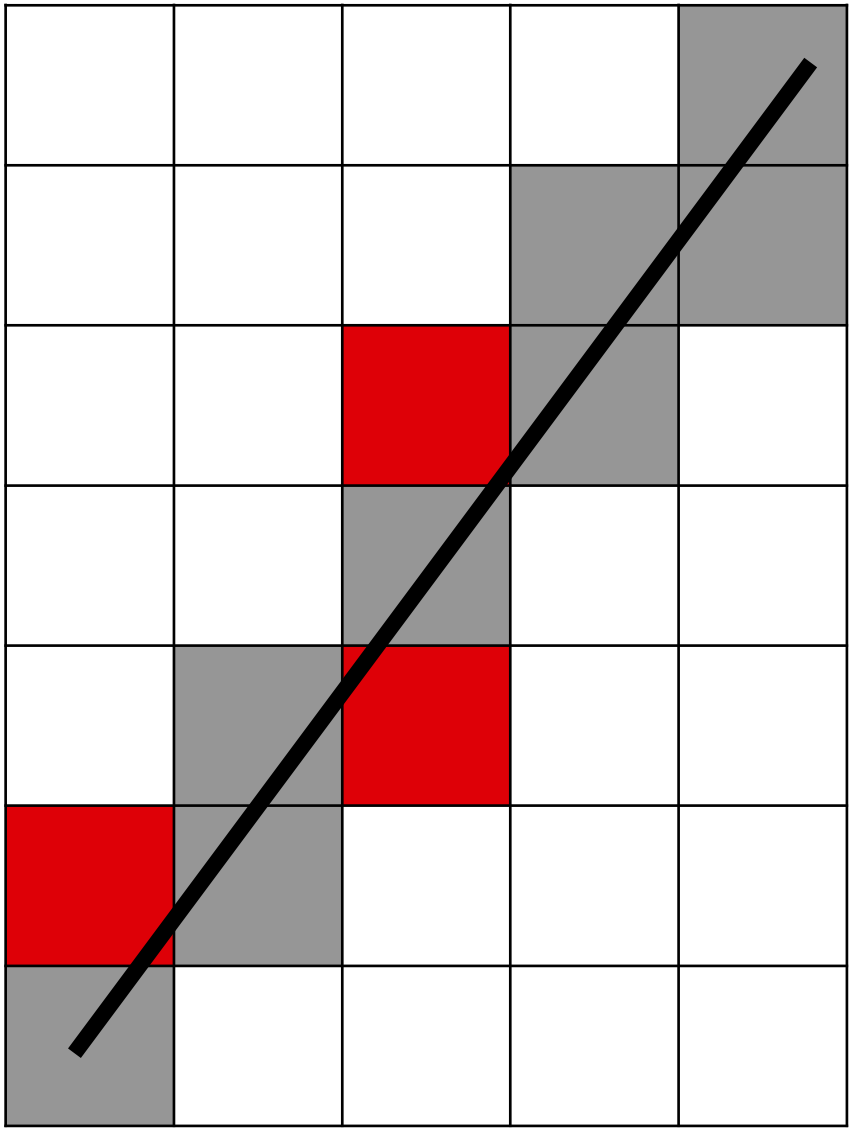
Parcours dans la grille

3DDDA

Three Dimensional Digital Difference Analyzer

Similaire à la rastérisation

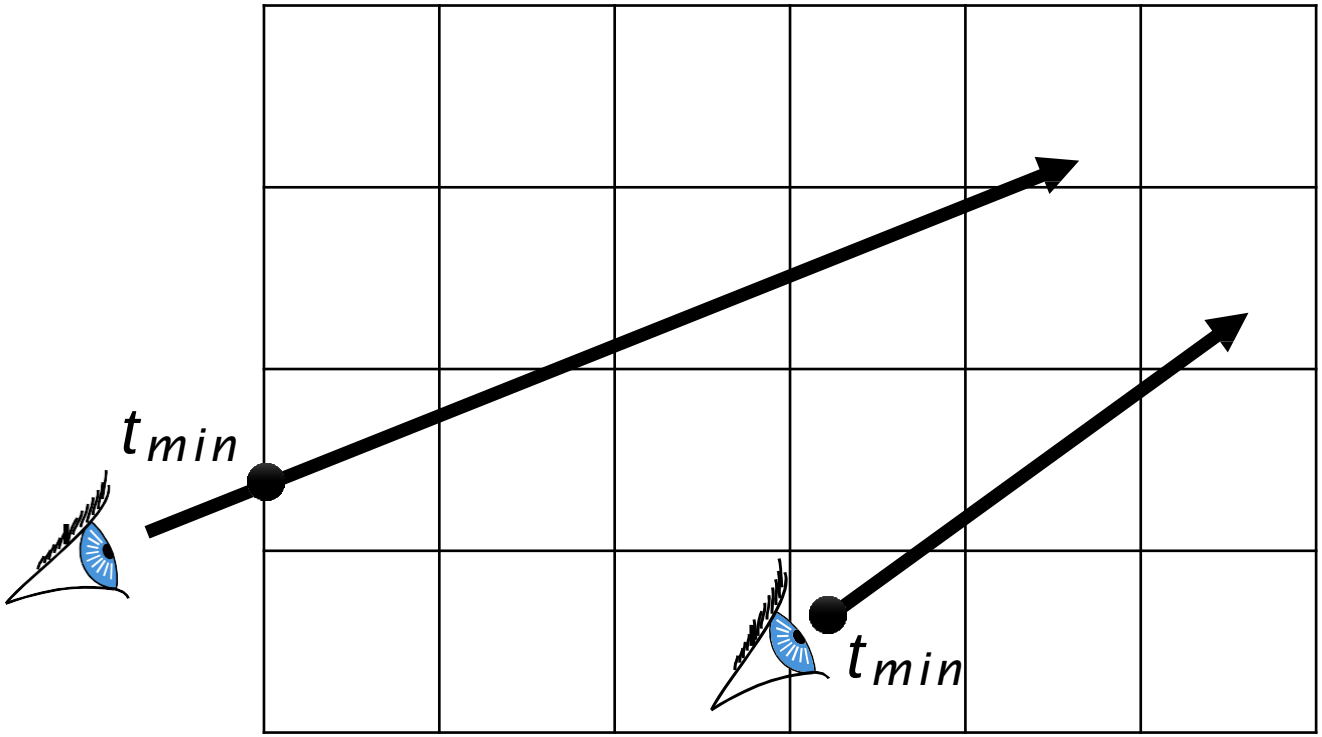
…mais on veut **toutes** **les cellules intersectant le rayon**



Initialisation

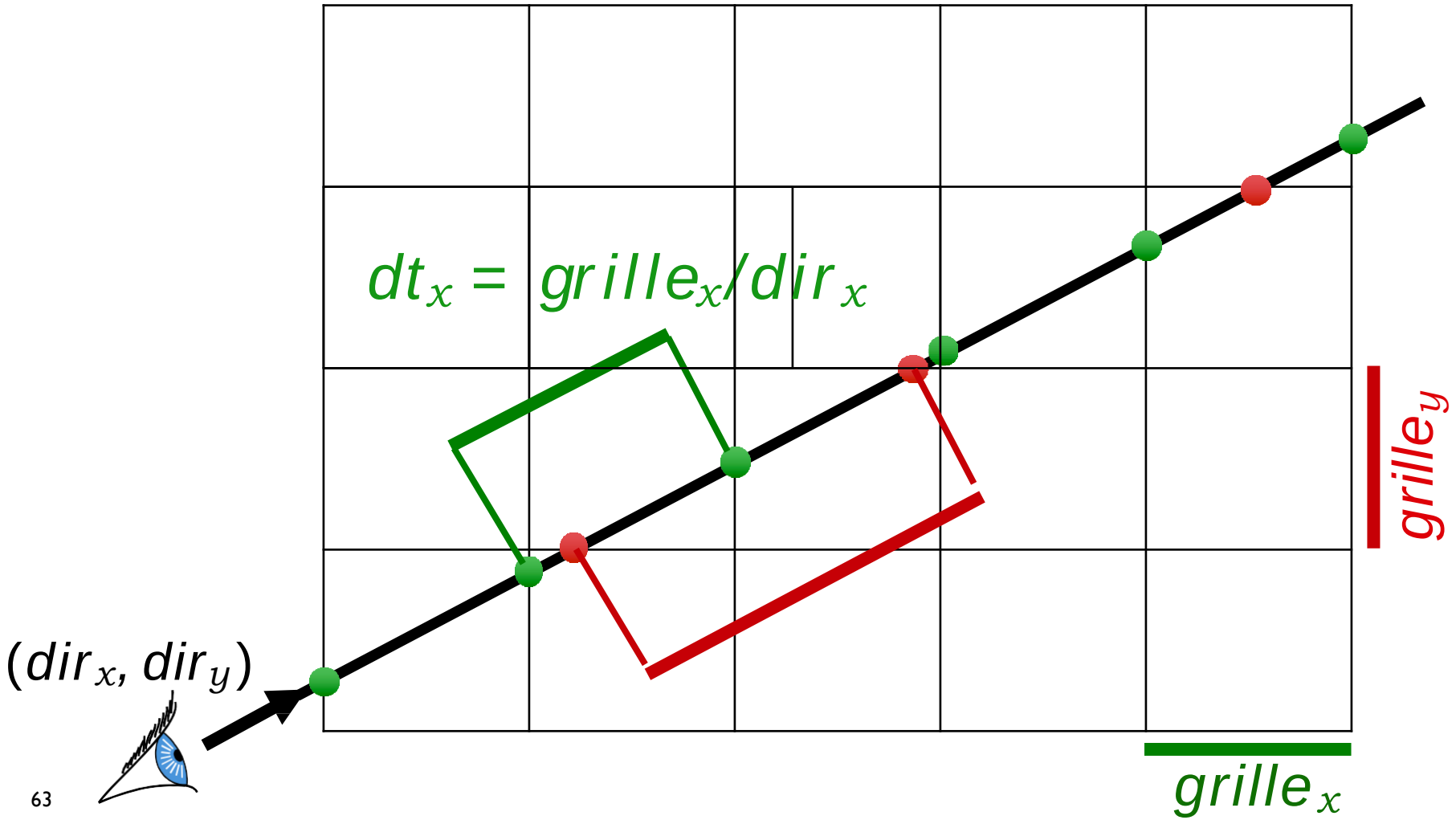
Calculer l'intersection avec la boite englobante t_{min}

(Attention, l'origine du rayon peut être dans la boîte)



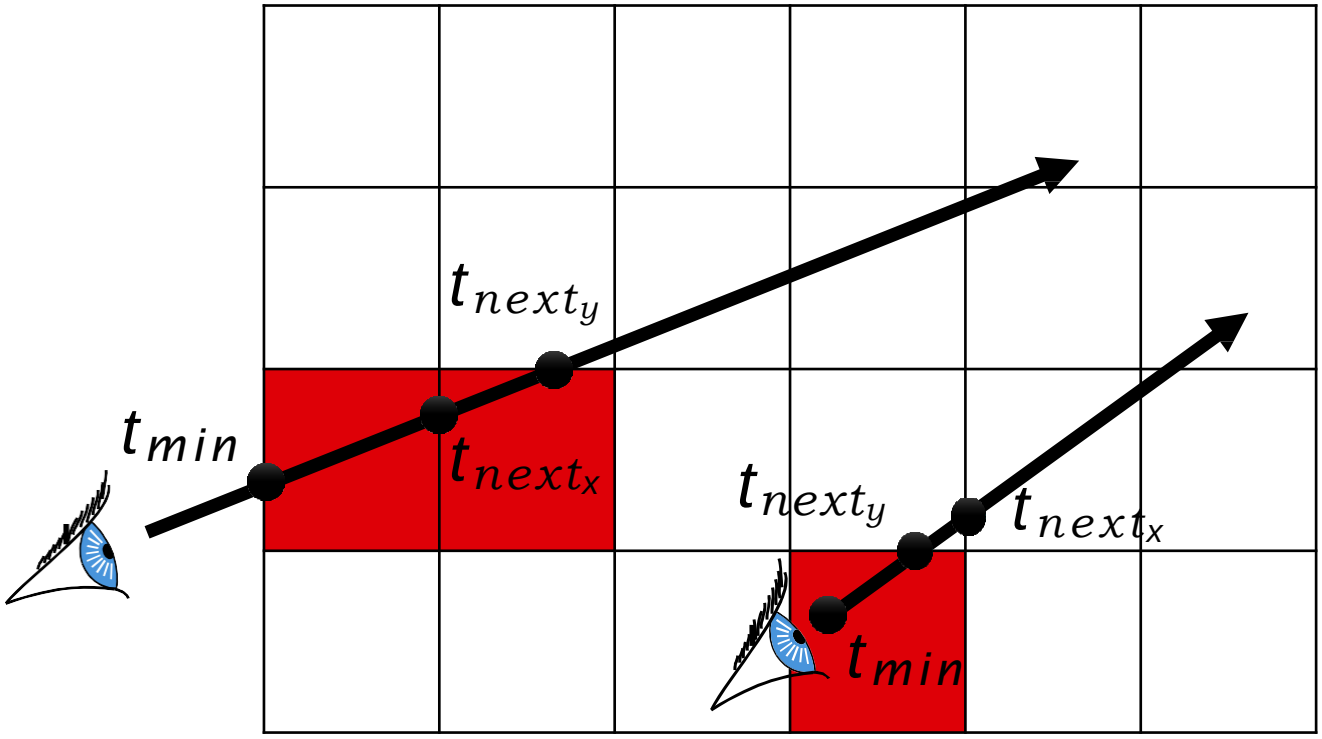
L'intersection est répétitive

Les intersections sur les axes sont équidistantes



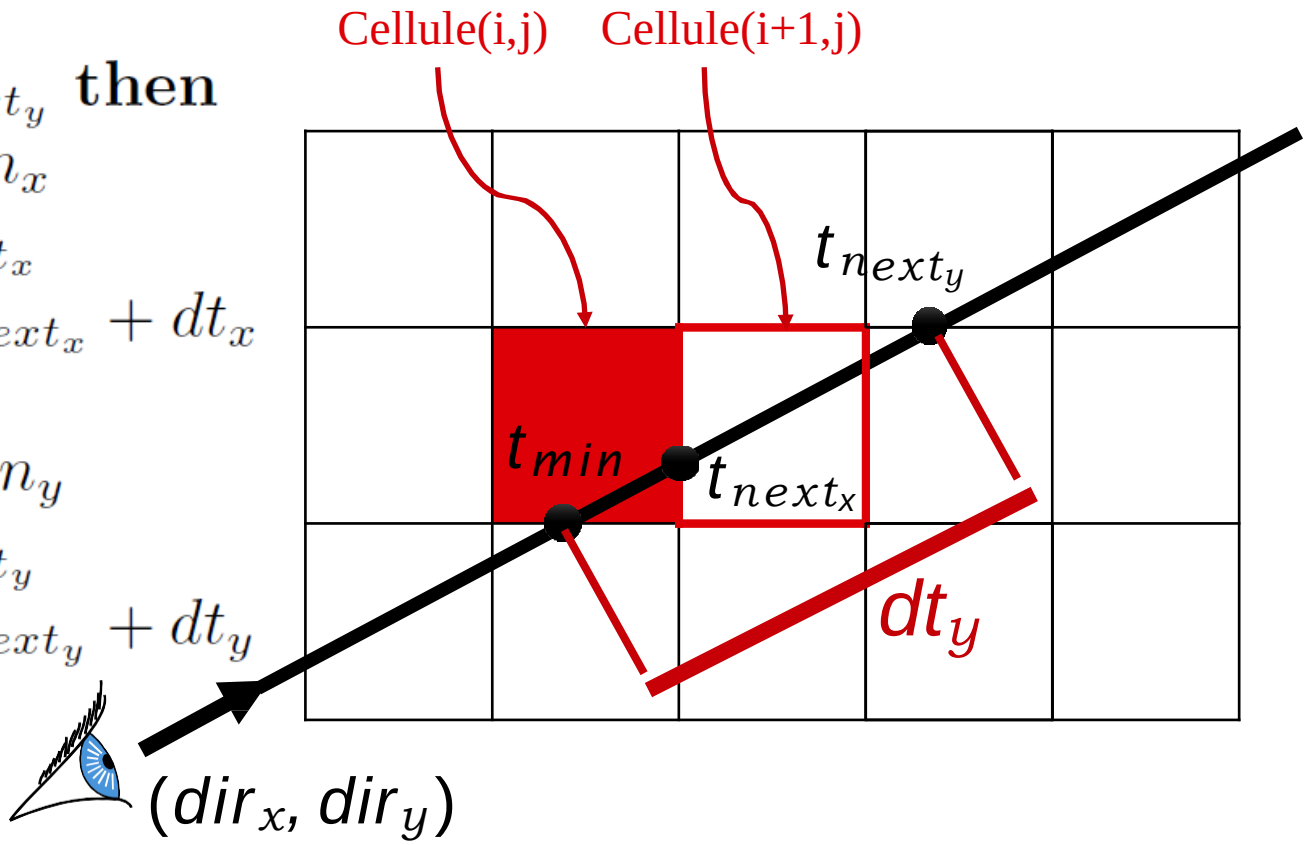
Choisir la cellule suivante

Calculer les 2 intersections suivantes avec les axes $\begin{cases} t_{next_x} \\ t_{next_y} \end{cases}$



Choisir la cellule suivante

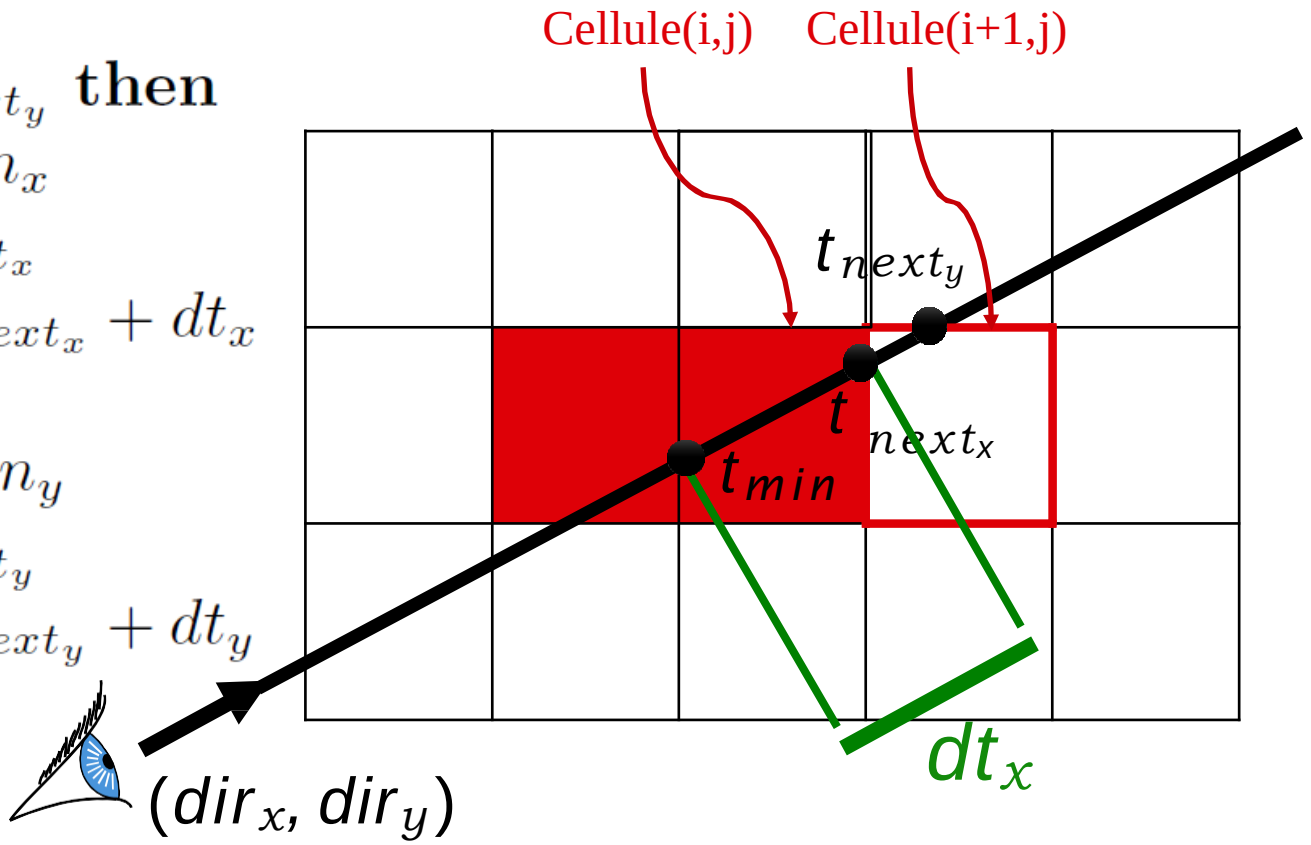
if $t_{next_x} < t_{next_y}$ **then**
  $i \leftarrow i + sign_x$
  $t_{min} = t_{next_x}$
  $t_{next_x} \leftarrow t_{next_x} + dt_x$
else
  $j \leftarrow j + sign_y$
  $t_{min} = t_{next_y}$
  $t_{next_y} \leftarrow t_{next_y} + dt_y$
end if



if $dir_n > 0$ **then** $sign_n \leftarrow 1$ **else** $sign_n \leftarrow -1$ **end if**

Choisir la cellule suivante

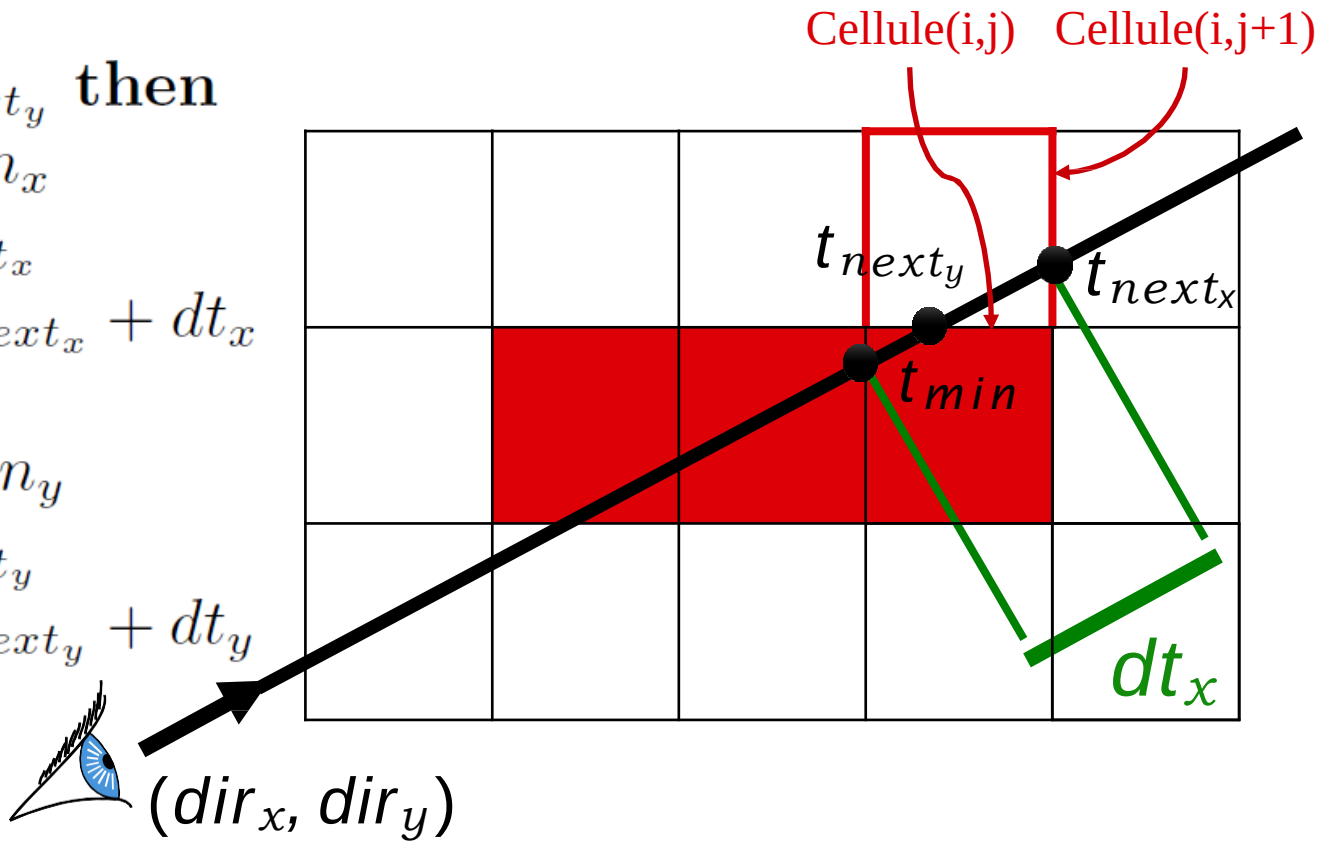
if $t_{next_x} < t_{next_y}$ **then**
$i \leftarrow i + sign_x$
$t_{min} = t_{next_x}$
$t_{next_x} \leftarrow t_{next_x} + dt_x$
else
$j \leftarrow j + sign_y$
$t_{min} = t_{next_y}$
$t_{next_y} \leftarrow t_{next_y} + dt_y$
end if



if $dir_n > 0$ **then** $sign_n \leftarrow 1$ **else** $sign_n \leftarrow -1$ **end if**

Choisir la cellule suivante

if $t_{next_x} < t_{next_y}$ **then**
$i \leftarrow i + sign_x$
$t_{min} = t_{next_x}$
$t_{next_x} \leftarrow t_{next_x} + dt_x$
else
$j \leftarrow j + sign_y$
$t_{min} = t_{next_y}$
$t_{next_y} \leftarrow t_{next_y} + dt_y$
end if



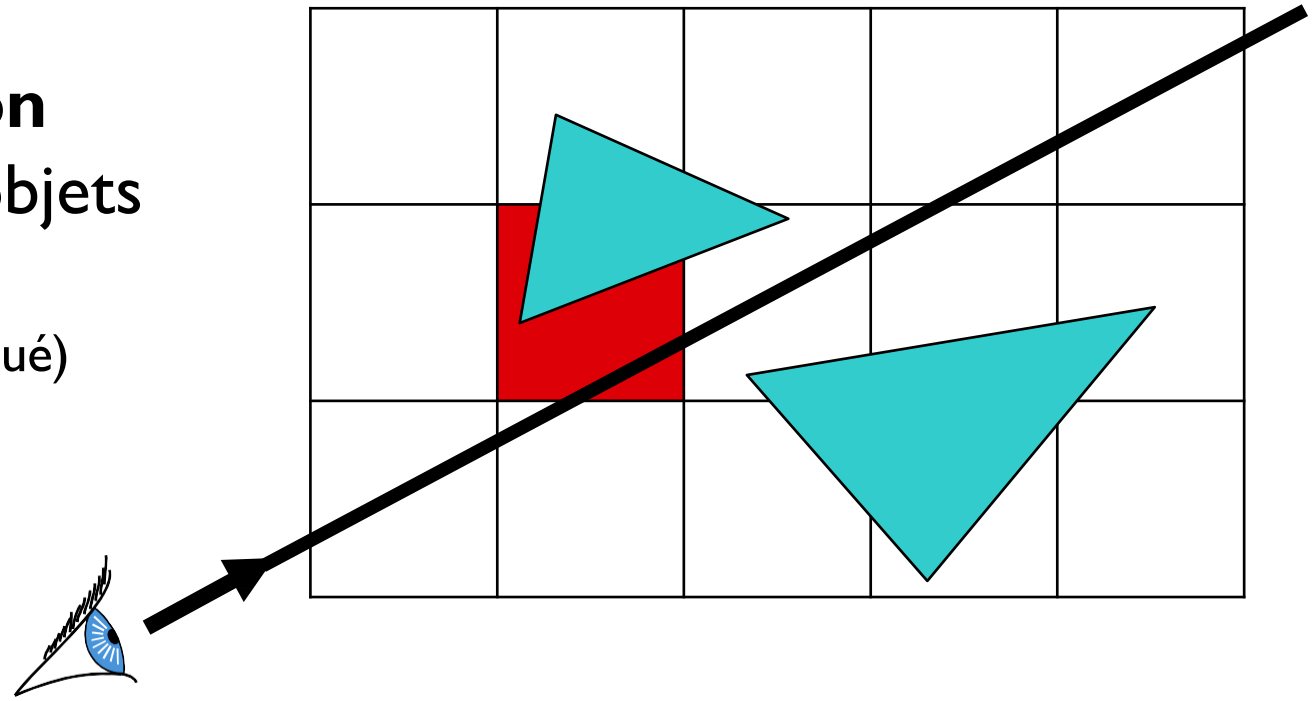
if $dir_n > 0$ **then** $sign_n \leftarrow 1$ **else** $sign_n \leftarrow -1$ **end if**

Test à faire sur chaque cellule

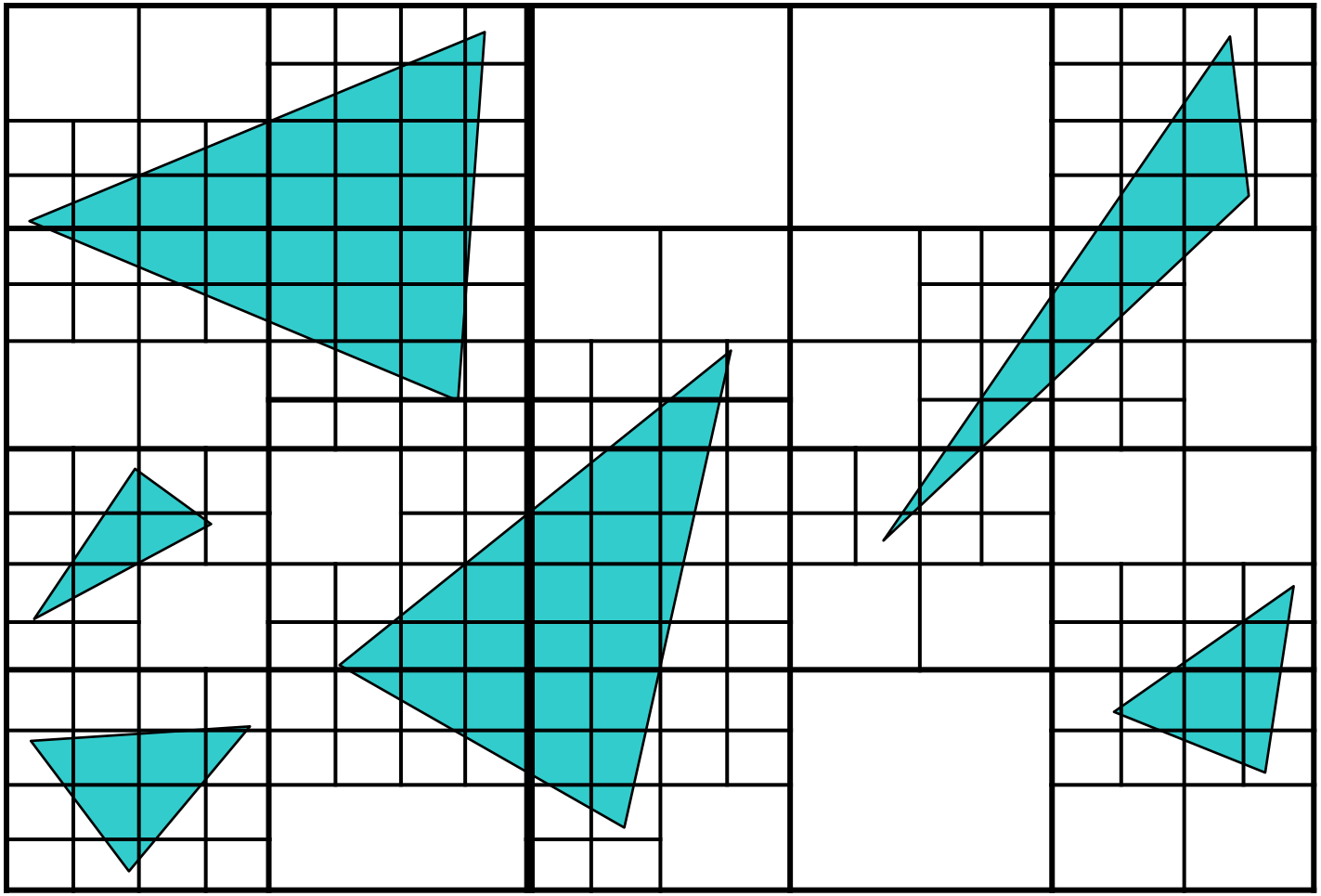
Intersection(s) dans la cellule ?

- Oui :retourner la plus proche
- Non :continuer

Optimisation
marquer les objets déjà testés
(mais plus compliqué)



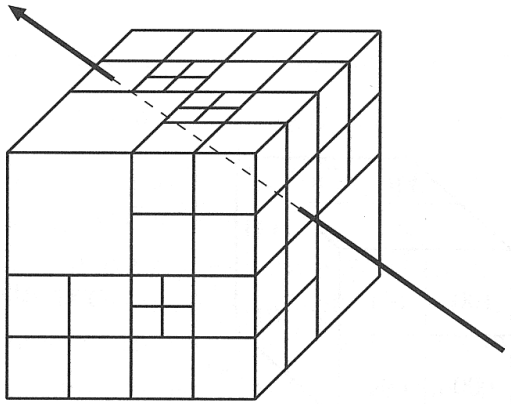
Grille adaptative : Octree [Meagher '80]

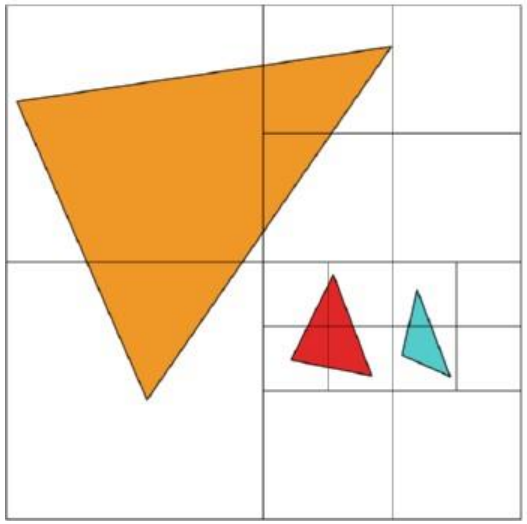


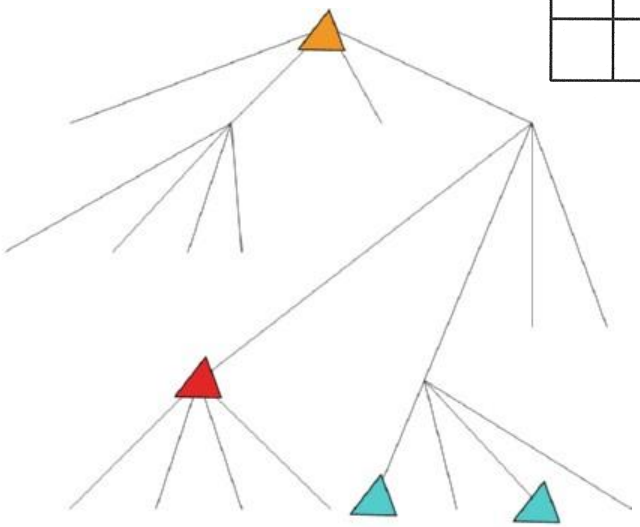
Grille adaptative : Octree

Partitionnement hiérarchique de l'espace

- Subdivise adaptivement chaque voxel en **8** sous-voxels de manière récursive
- Différents critères possibles
  - nombre de primitives par cellules
  - ratio de « vide »







Comparaison

Grille régulière

- ✓ construction facile et rapide
- ✓ parcours simple
- ✗ nombreuses cellules vides
- ✗ cellules avec beaucoup d'objets
- ✗ choix de la résolution

Octree

- ✓ initialisation rapide
- ✓ peu de cellules vides
- ✗ parcours récursif couteux
- ✗ convergence lente pour les zones complexes

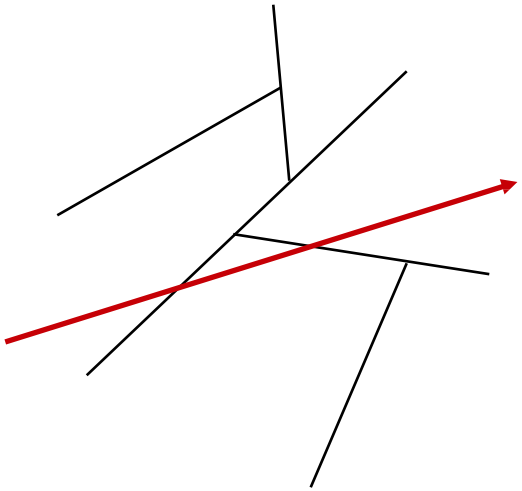
Grille adaptative : BSP- et Kd-Trees

Arbres binaires

- Nœud = plan de subdivision de l'espace

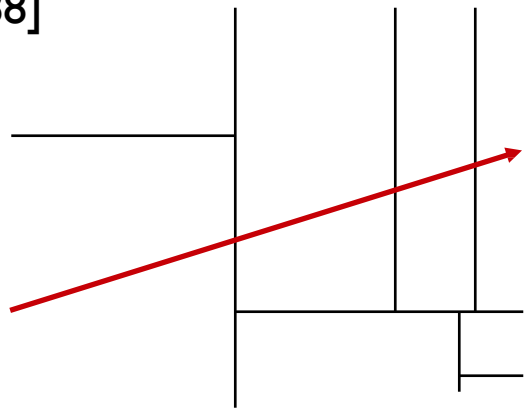
Binary Space Partition [Fuchs et al. 1980]

- Plans quelconques « judicieusement » placés
- Couteux à construire,stocker et utiliser
-



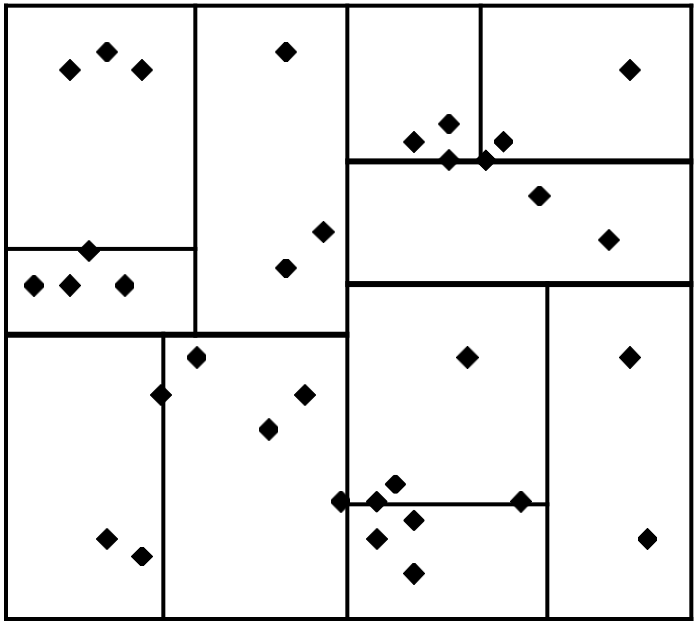
Kd-Tree [Bentley 1975, Fussell and Subramanian 1988]

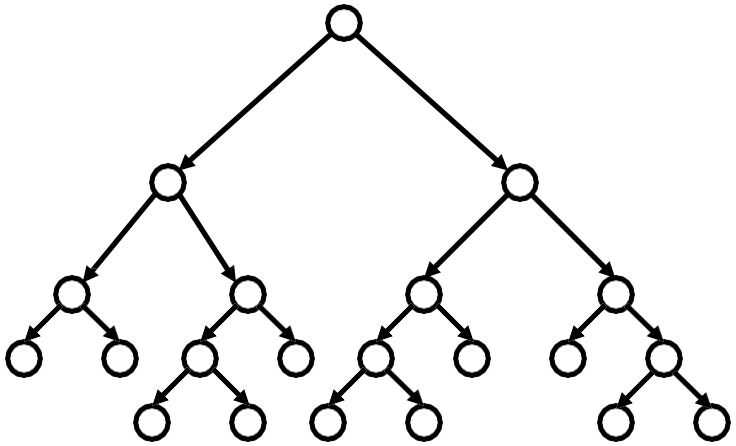
- Plans alignés sur les axes
- Simple, léger et très efficace



KD-tree : définition

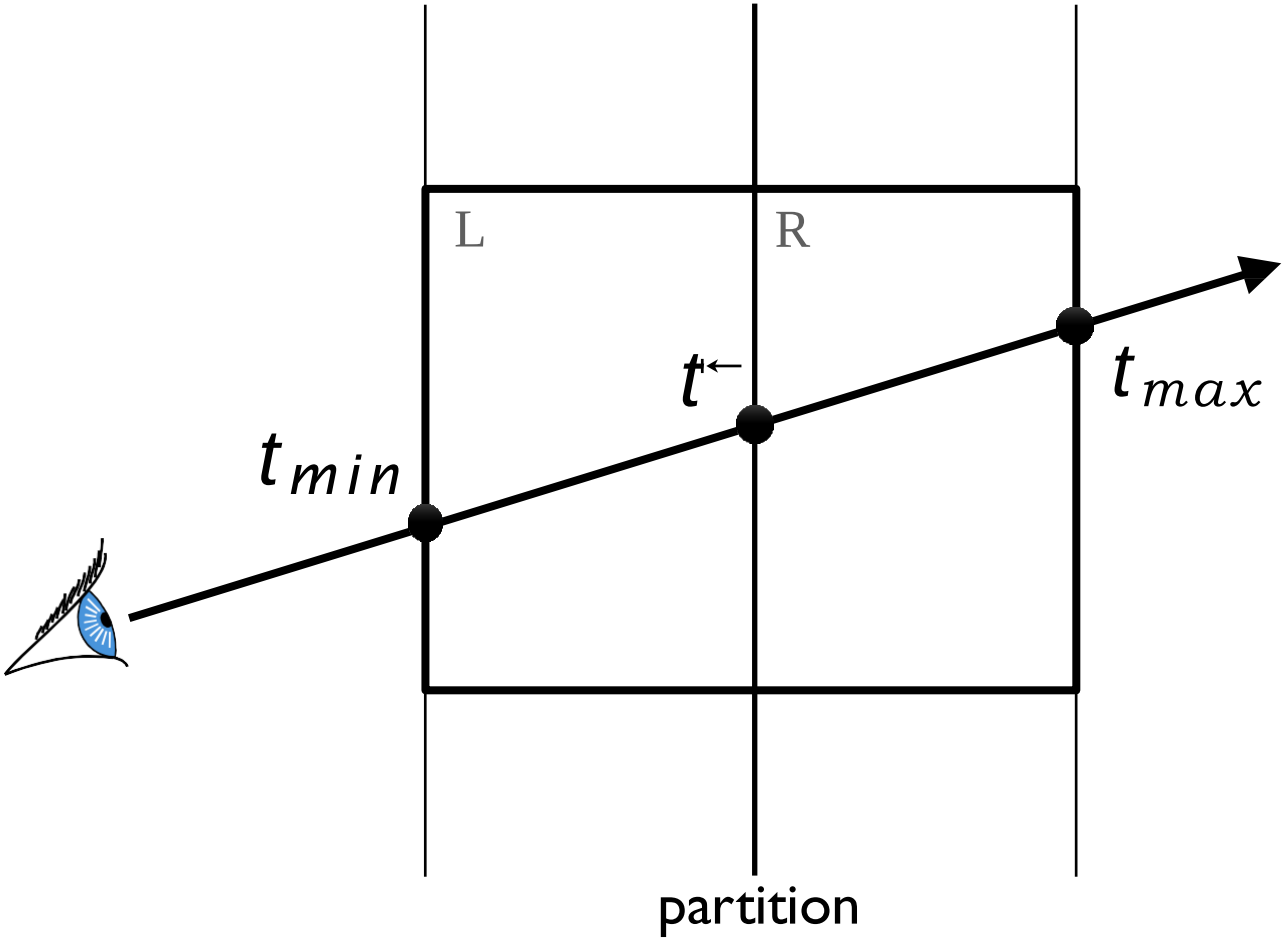
Subdivision récursive





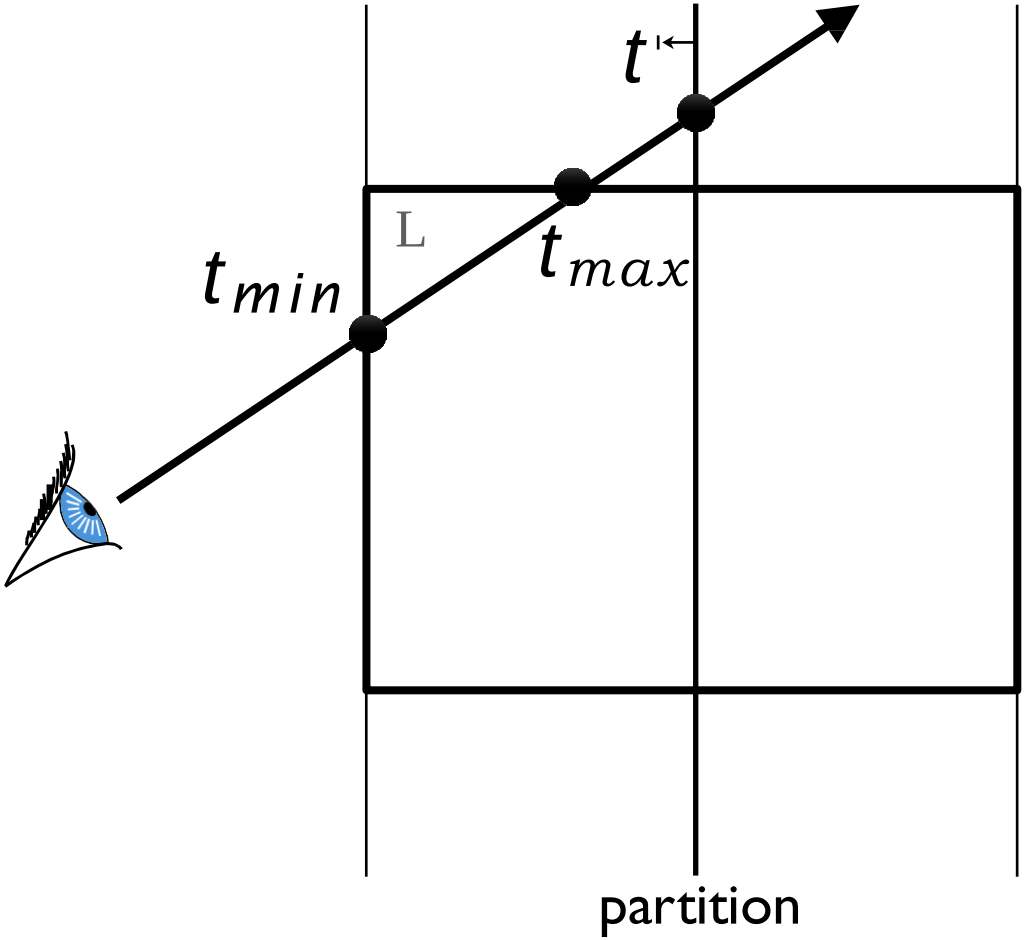
KD-tree : parcours récursif

$t_{min} < t_{\leftarrow} < t_{max}$) Intersect(L,t_{min},$t^{\leftarrow}$) et Intersect(R,$t^{\leftarrow}$,t_{max})



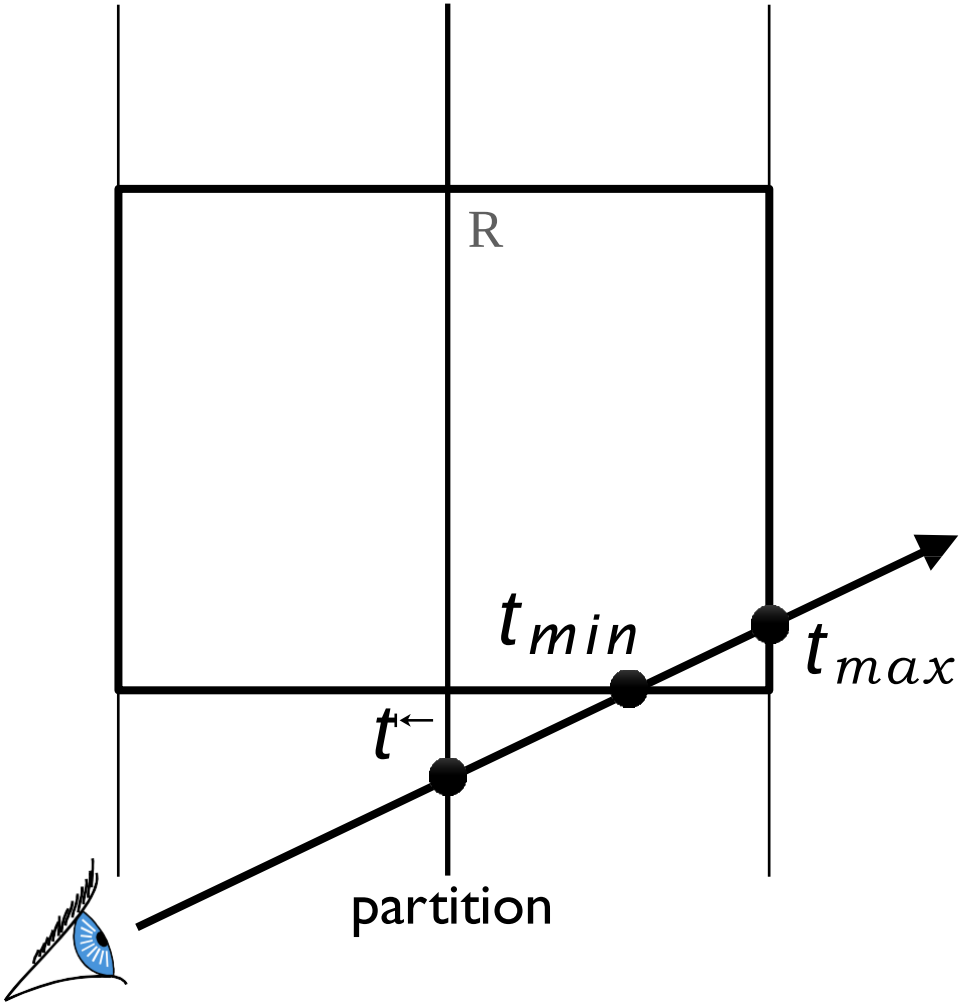
KD-tree : parcours récursif

$t_{max} < t^{\leftarrow}$) Intersect(L,t_{min},t_{max})



KD-tree : parcours récursif

$t_{\leftarrow} < t^{min}$) Intersect(R,t_{min},t_{max})



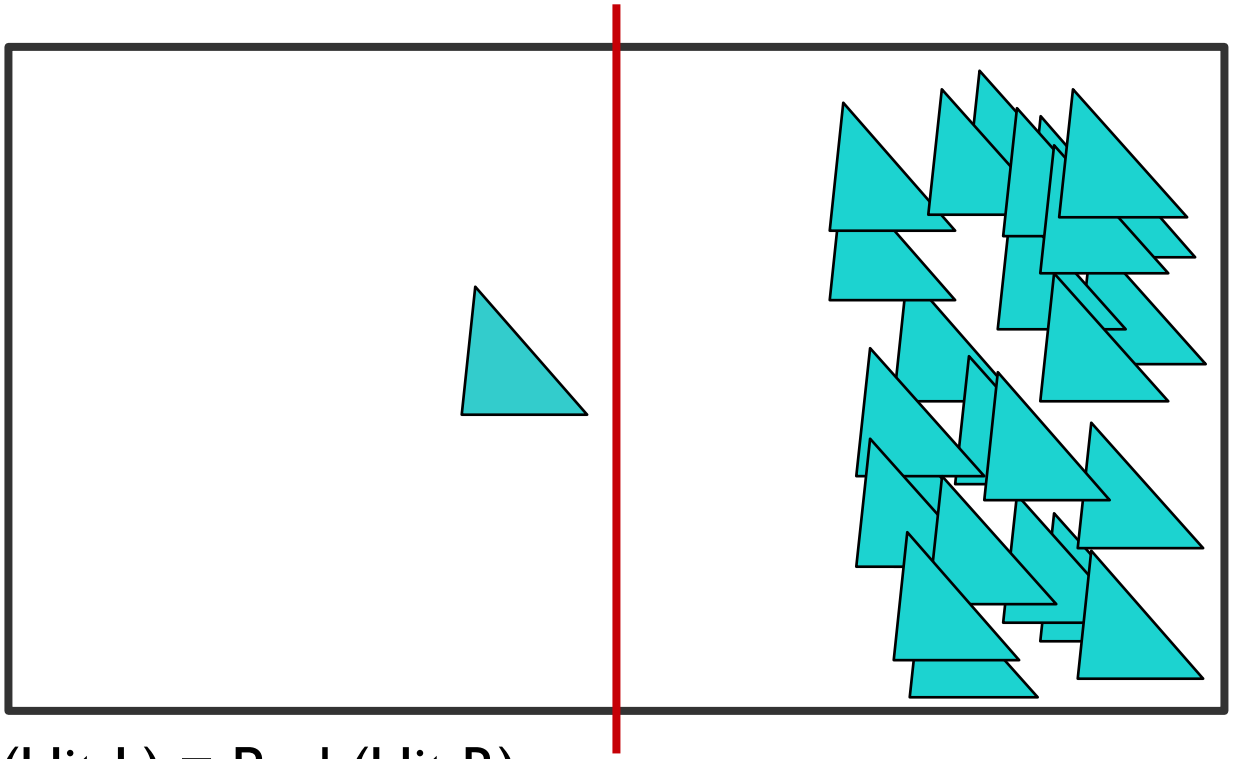
KD-tree : construction

Algo « naïf »

- Axe de coupe : le long de la plus grande dimension
- Position de coupe : au centre ou médian de la géométries (arbre équilibré)
- Critère d'arrêt : nombre de primitives, profondeur max.

KD-tree : construction

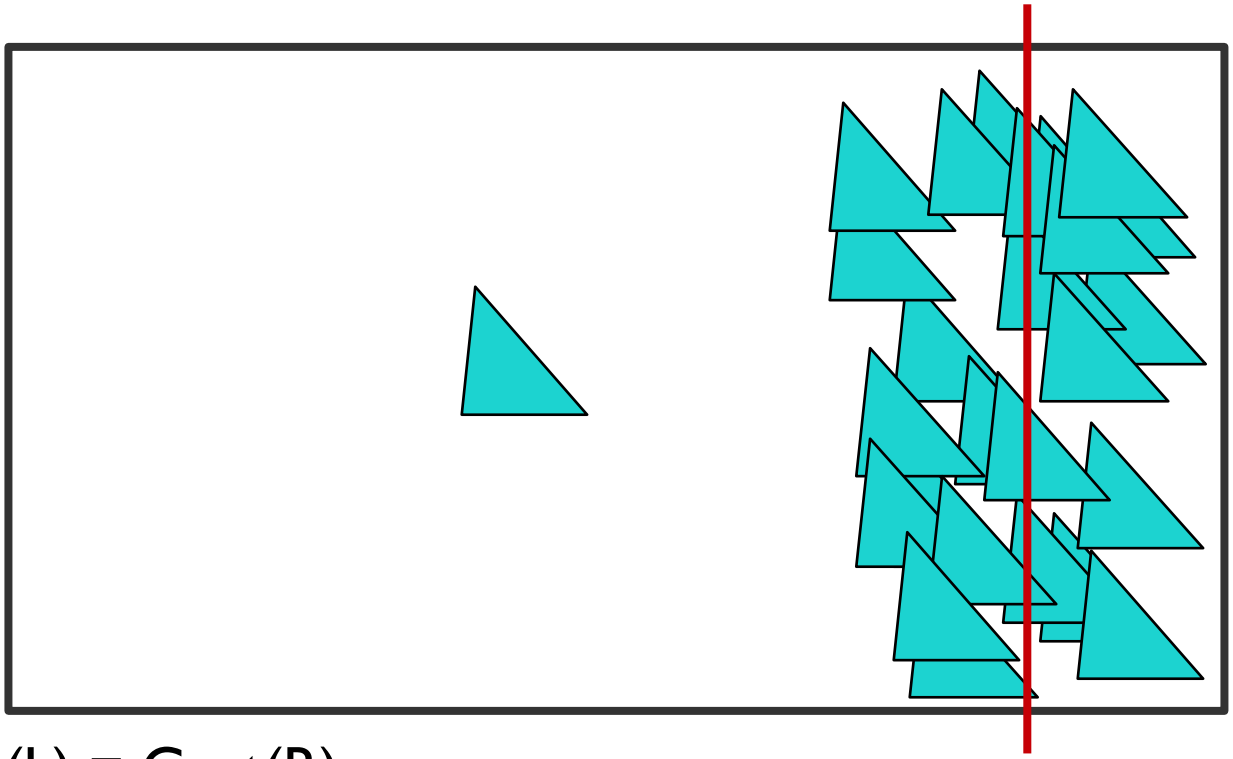
Couper au milieu



- Prob(Hit L) = Prob(Hit R)
- Ne prend pas en compte le coût de L & R

KD-tree : construction

Couper à la médiane



- Cost(L) = Cost(R)
- Ne prend pas en compte les probabilités d'entrer dans L et R

KD-tree : construction

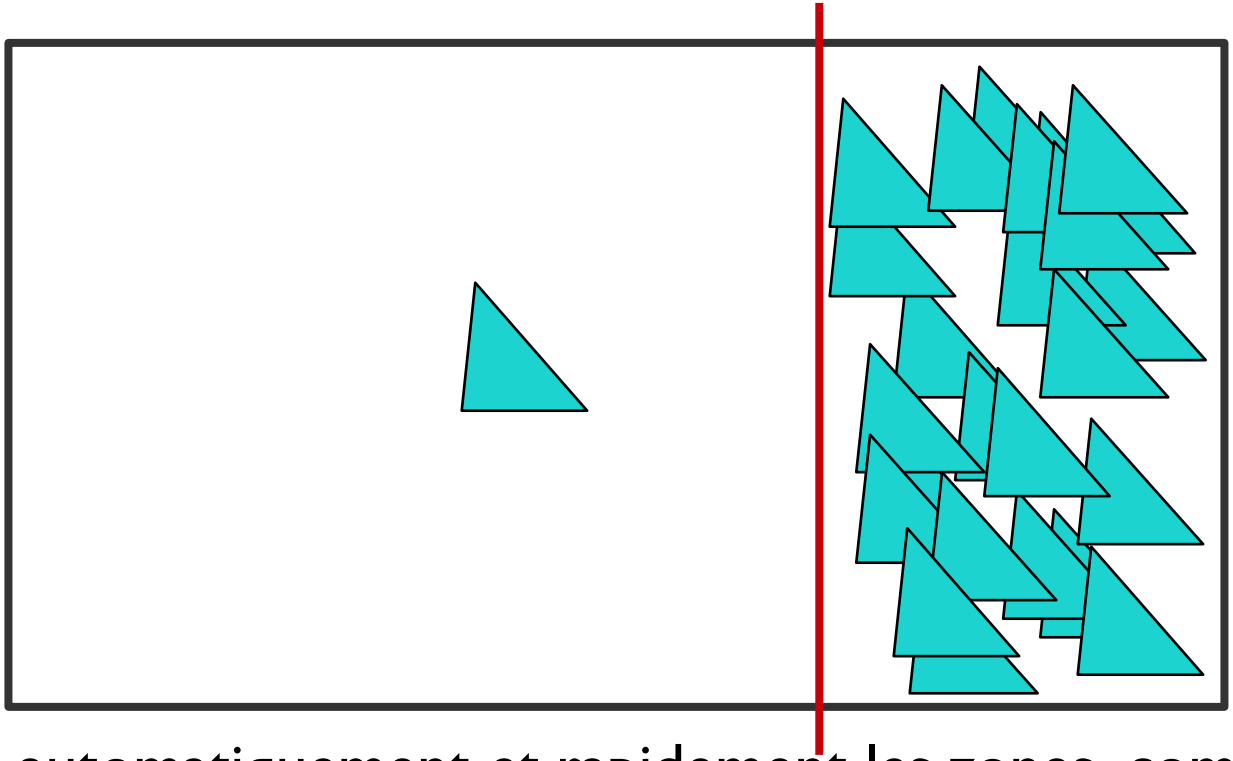
Algo « intelligent »

- Objectif : choisir le plan de coupe qui rend le tracé de rayon le moins couteux possible
- Définir un modèle de coût et le minimiser
- Quel est le coût de tracer un rayon au travers d'une cellule ?

Cost(cell) = C_trav + Prob(hit L) x Cost(L) + Prob(hit R) x Cost(R)

KD-tree : construction

Optimisation de la fonction de coût



- Isole automatiquement et rapidement les zones complexes
- Génère de grands espaces vides / concentre les primitives dans de petits nœuds

KD-tree : construction [MacDonald and Booth 1990]

Probabilité de rentrer dans une cellule

$\Longrightarrow$ proportionnel à l'aire de la surface de la cellule (SA)

Coût de parcours d'une cellule

$\Longrightarrow$ nombre de triangles (TriCount)

Cost(cell) = C_trav + Prob(hit L) x Cost(L) + Prob(hit R) x Cost(R)

= C_trav + SA(L) xTriCount(L) + SA(R) xTriCount(R)

Critère d'arrêt

$\Longrightarrow$ quand subdiviser ne réduit plus le modèle de coût (seuillage)

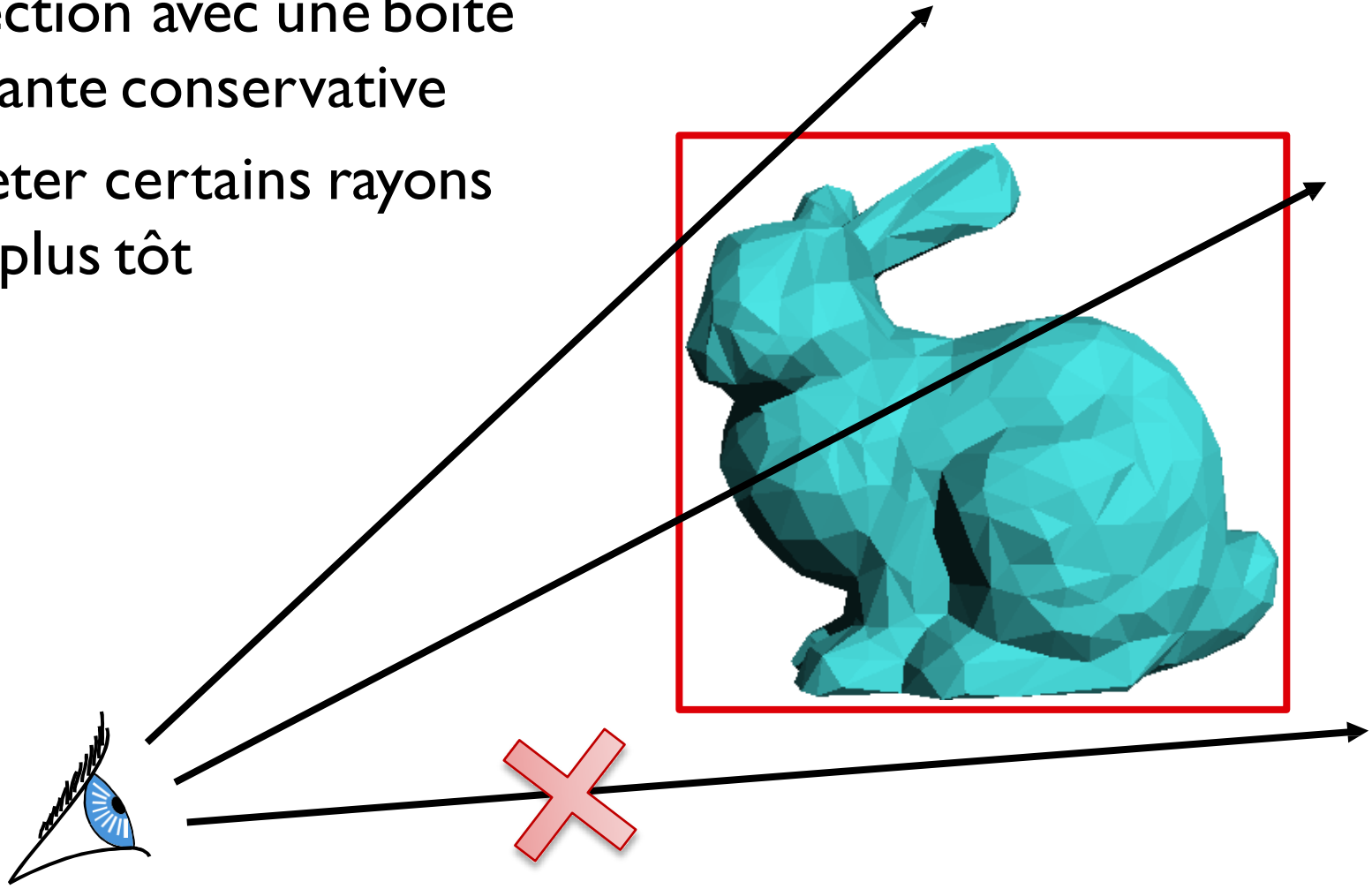
Résultats

$\Longrightarrow$ un « bon » KD-tree est de 2 à 5 fois plus rapide

Boites englobantes

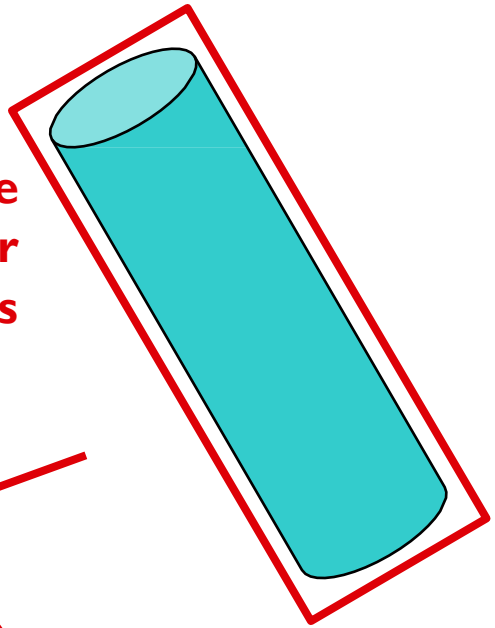
Intersection avec une boite englobante conservative

$\Longrightarrow$ rejeter certains rayons au plus tôt

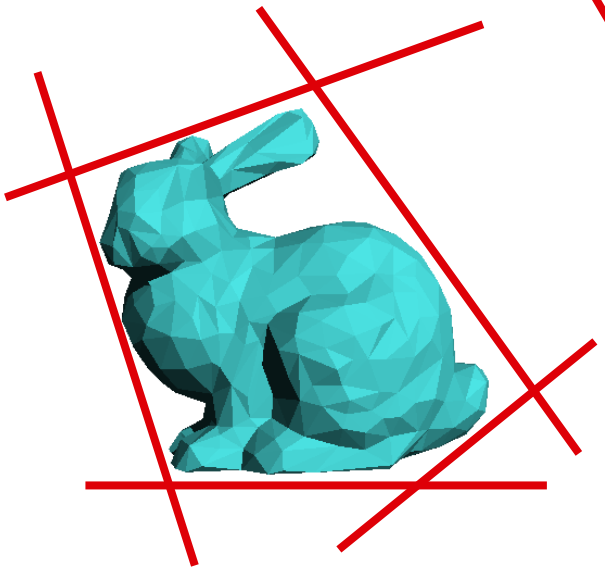


Boites englobantes

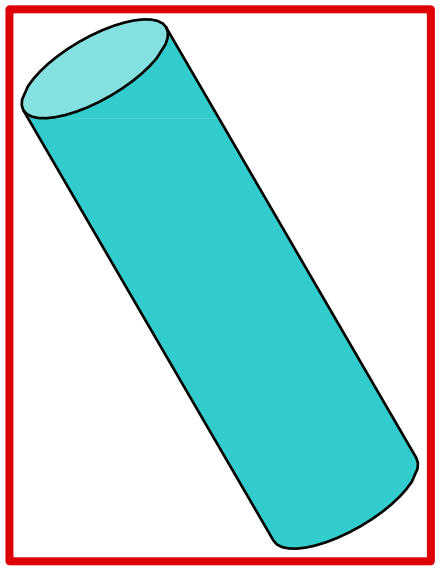
Boite englobante non-alignée sur les axes



Sphère englobante



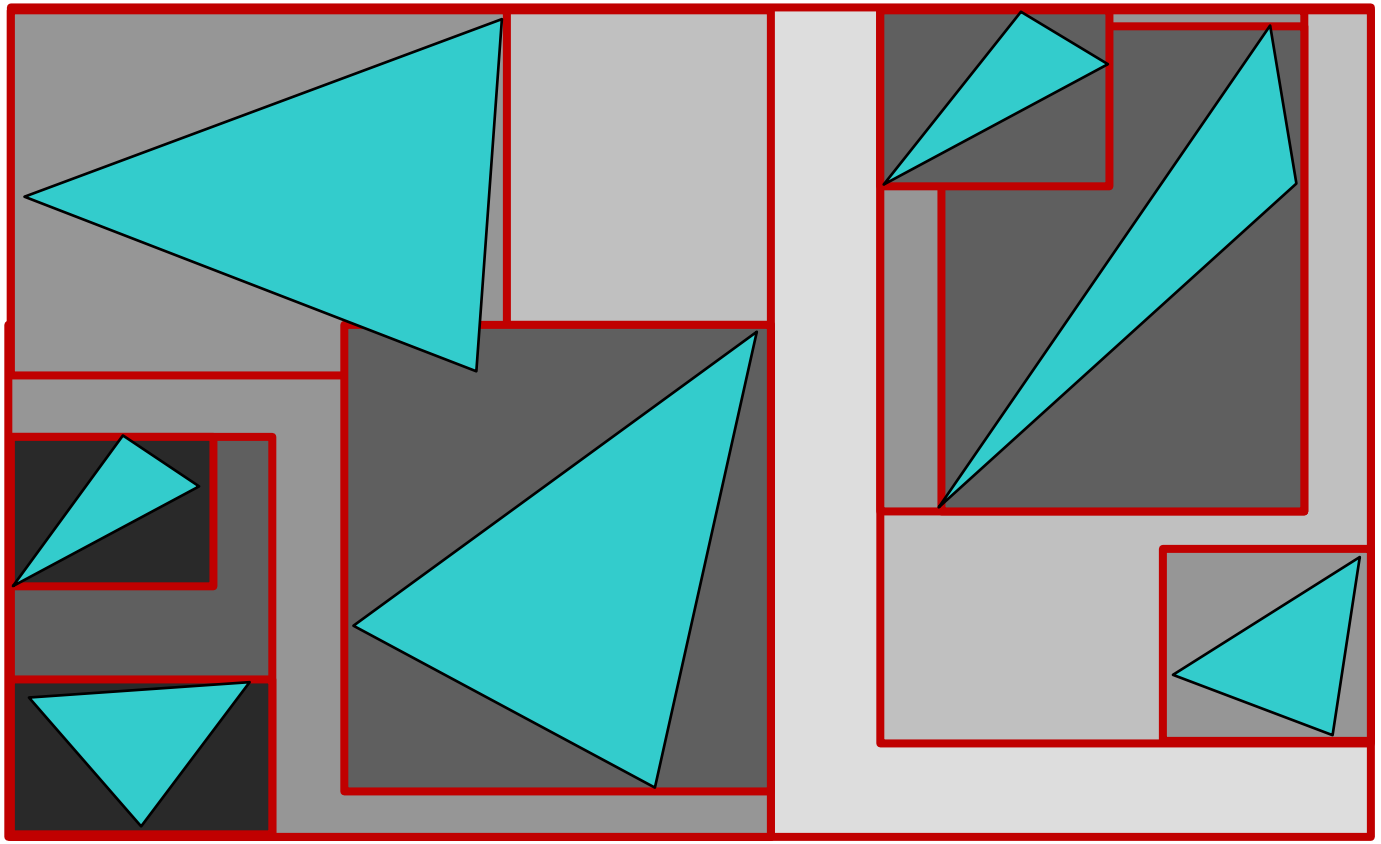
Région convexe arbitraire



Boite englobante alignée sur les axes

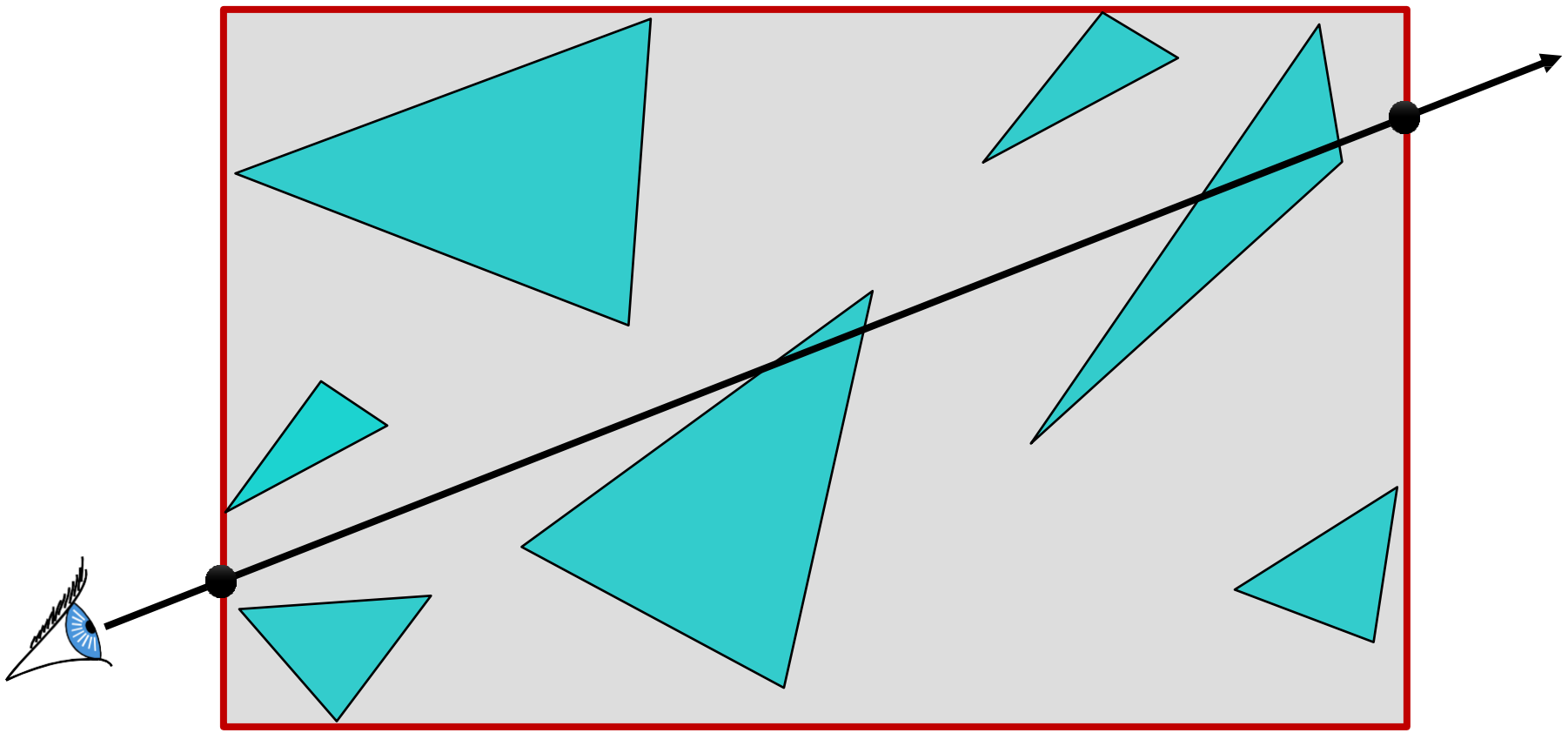
Hiérarchie de boites englobantes

BVH = KD-tree avec 1 boîte par nœud



BVH : intersection

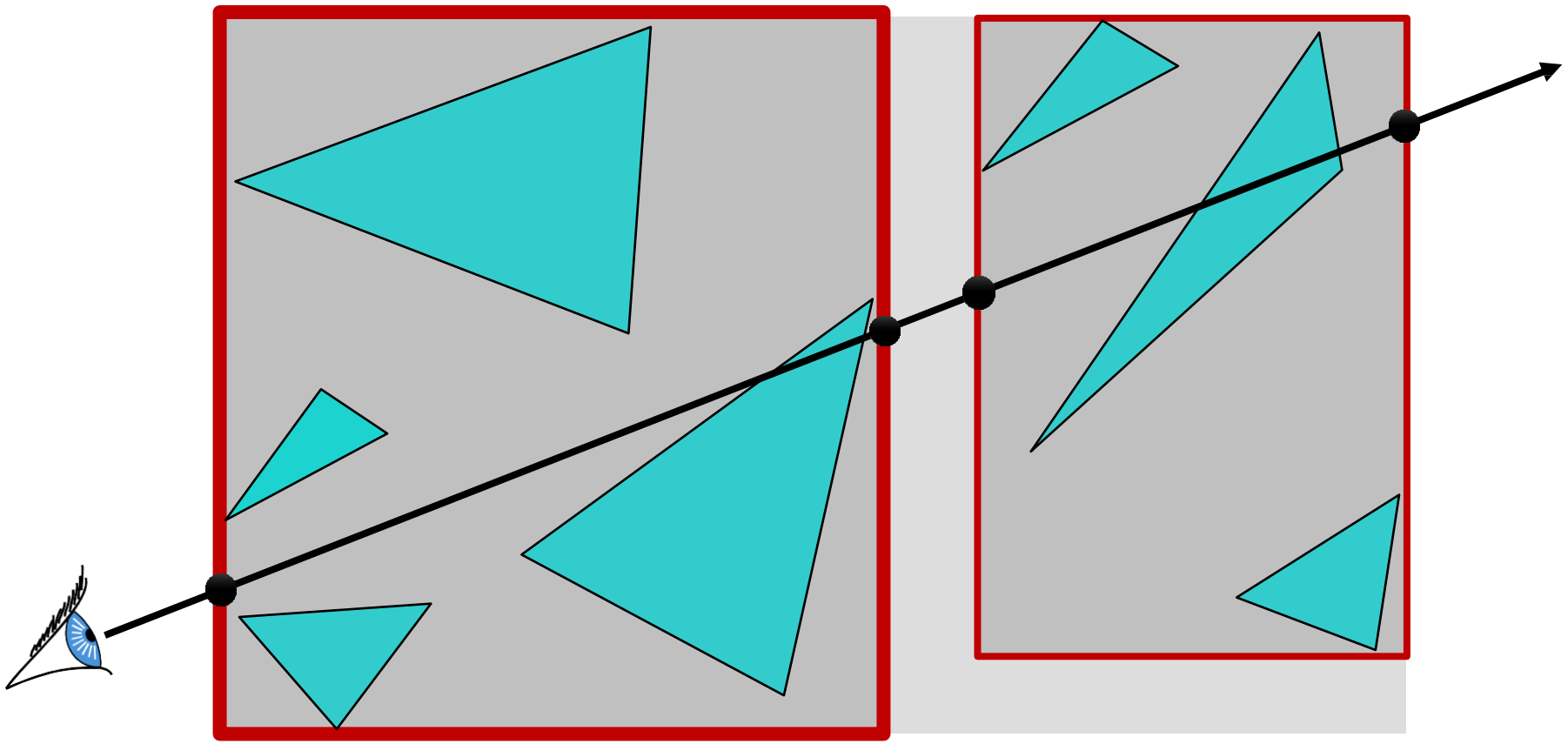
Tester la boîte parente



Hiérarchie de boites englobantes

Si Intersection,descendre sur les fils

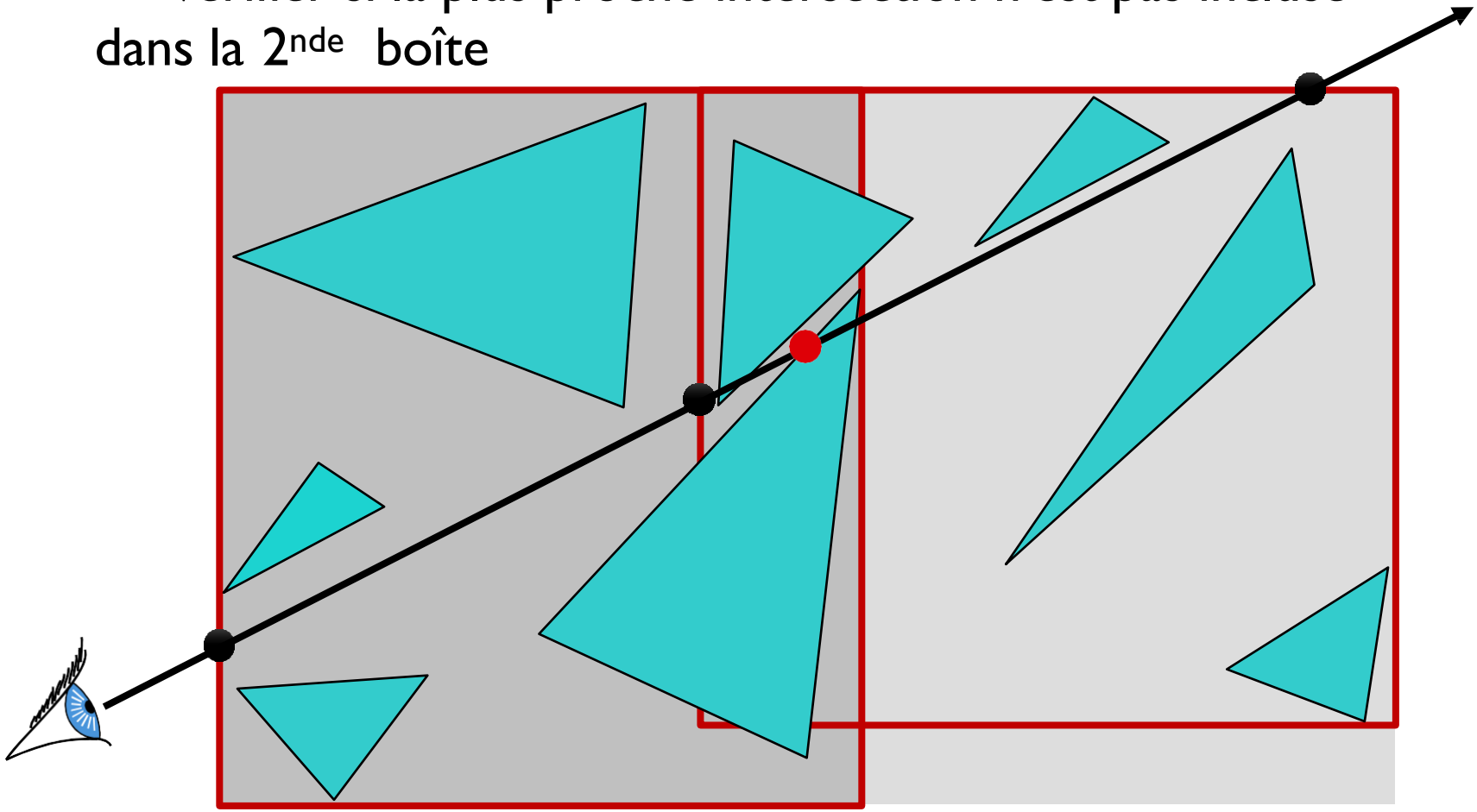
Tester la boîte avec l'intersection la plus proche



Hiérarchie de boites englobantes

Attention, les boîtes peuvent se superposer !

$\Longrightarrow$ vérifier si la plus proche intersection n'est pas incluse dans la 2nde boîte



Comparaison

KD-tree

- léger en mémoire (si bien codé)
- parcours simple et rapide
- construction optimale plus facile

BVH

- arbre moins profond
- permet de décaler légèrement un objet sans avoir à reconstruire entièrement l'arbre

Conclusion

- Scènes statiques $\Longrightarrow$ KD-tree
- Scènes dynamiques $\Longrightarrow$ BVH

  (les préférences semblent converger vers le tout BVH)

Lancer de rayon cohérent

Idée : 2 rayons proches ont de fortes chances d'intersecter la même primitive

- Tracer des paquets de KxK rayons en même temps (e.g., 8x8)
- Amortissement du coût de traversée de l'arbre
- Permet d'exploiter au mieux les instructions SIMD (e.g., SSE)
- Réduit les défauts de cache

Gros facteur d'accélération (~ x10)

Attention aux rayons secondaires !
(vectorisation par rayon préférable)

[Wald et al. EG 2001]

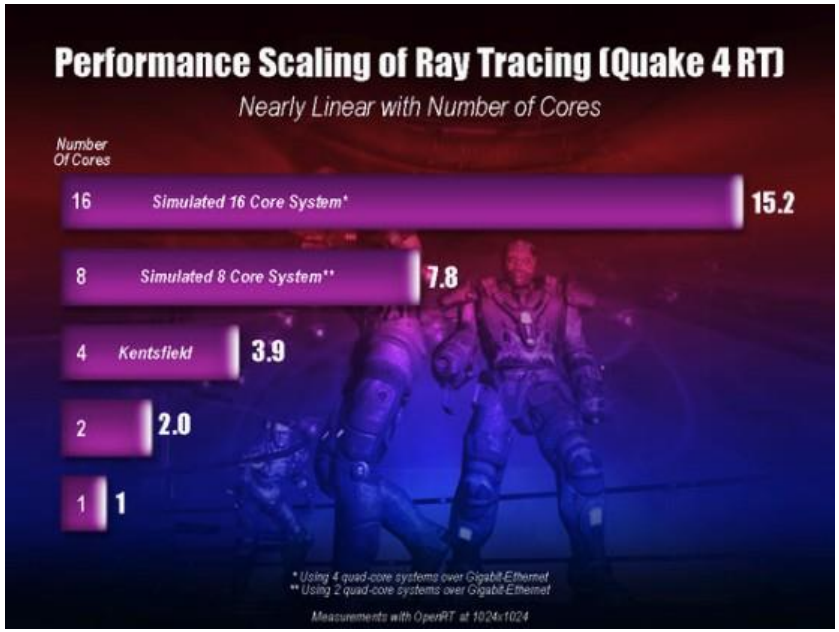
Parallélisation

Plusieurs possibilités :

- distribution image
- distribution scène
- distribution rayon / fonctionnelle

Dépend de l'architecture de la machine, souvent combinaison des 3 !

Exemple
(quake4 RT, distribution image)



Cohérence + Parallélisation

Exemple :

Intel Embree [Wald et al. Siggraph2014]

Kernels bas-niveau optimisés +API

Lancer de rayon temps réel

Exploitation des GPU

Structures de données sophistiquées

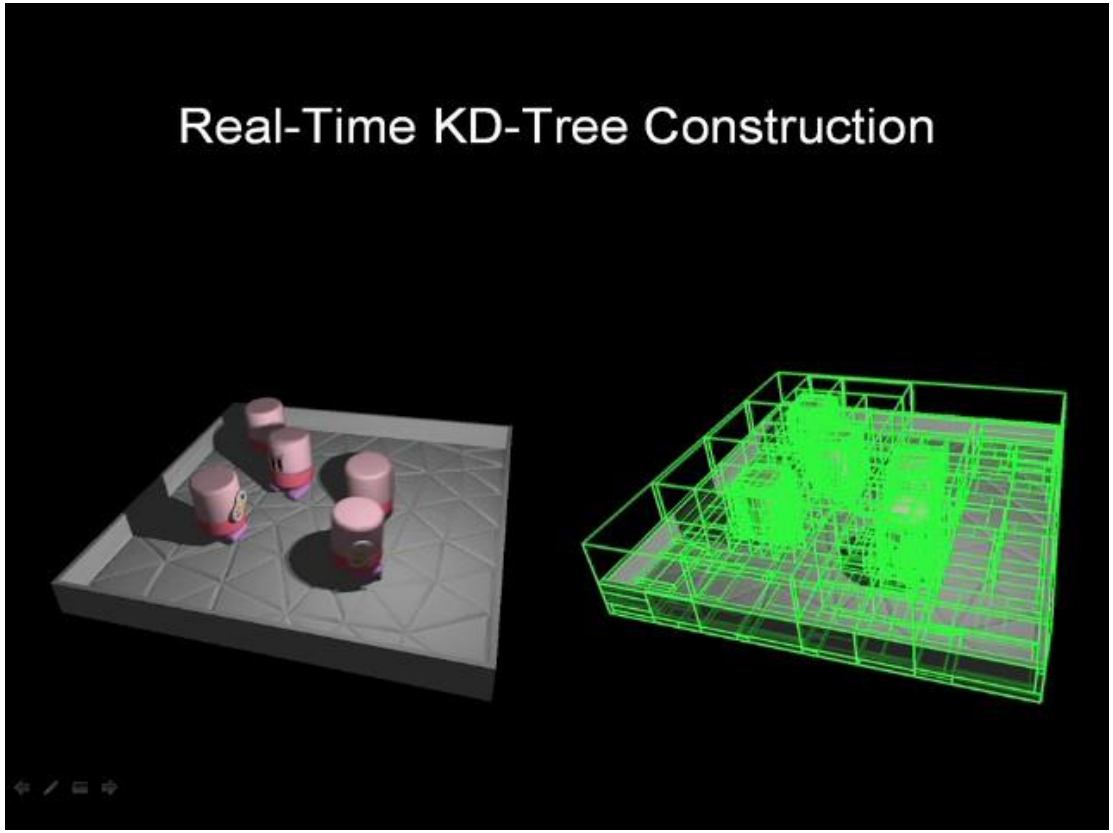
- Peu adaptées aux GPU des générations précédentes
- OK pour les GPU les plus récents, et futures

KD-tree GPU

Scènes animées :

Real-Time KD-Tree Construction on Graphics Hardware

Zhou et al. SIGGRAPH 2008



BVH GPU

Fast BVH Construction on GPUs, Lauterbach et al., Eurographics 2009

Lancer de rayon sur GPU

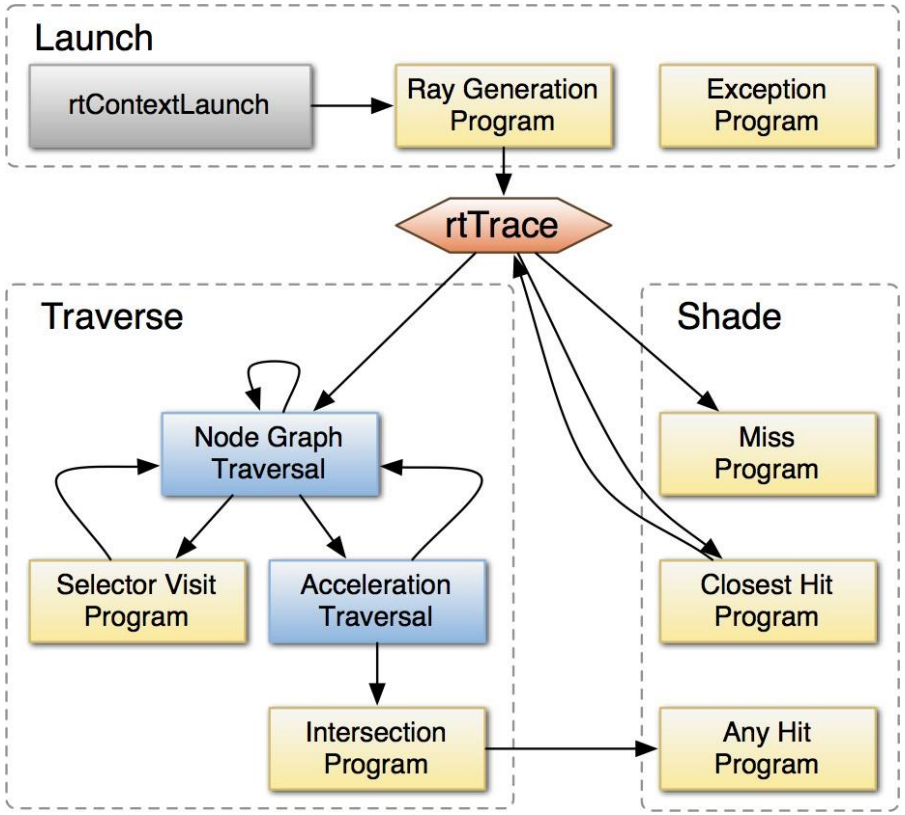
Nvidia OptiX [Parker et al. Siggraph 2010]

bibliothèque bas-niveau destinée aux développeurs

- structure de données :*BVH
- génération des rayons
- entièrement configurable

Autres

- Lightworks : basé sur OptiX
- Arion : GPU(CUDA)/CPU
- OctaneRender (CUDA)
- V-Ray RT (OpenCL)
- …



Source

Cours
Pierre Benard
Jean-Marc Thiery