Visibilité

Cours de master Images et Vie Artificielle. M1

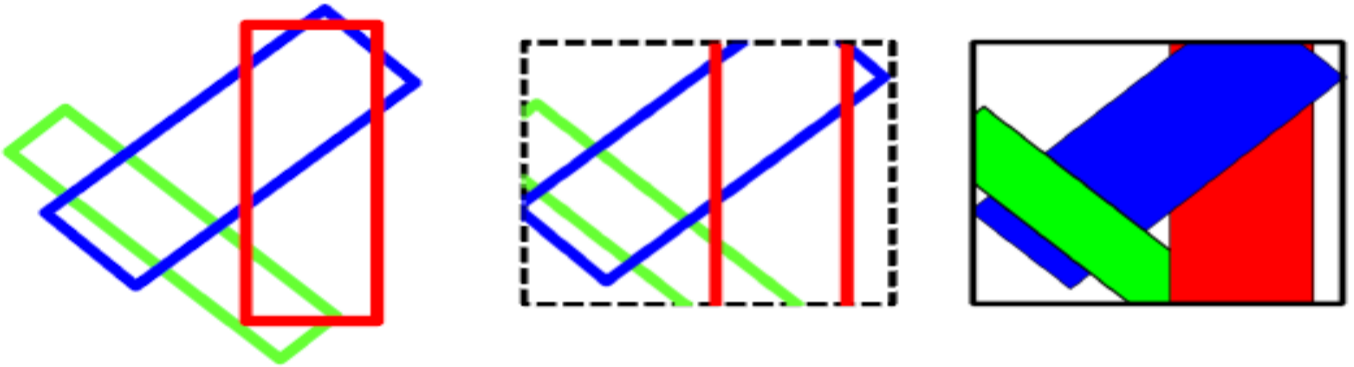
Université Mohamed Khider Biskra

2019- 2020

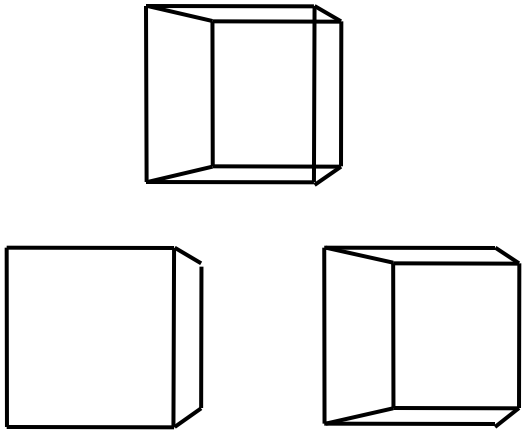
Dr: Zerari Abd El Mouméne

Visibilité

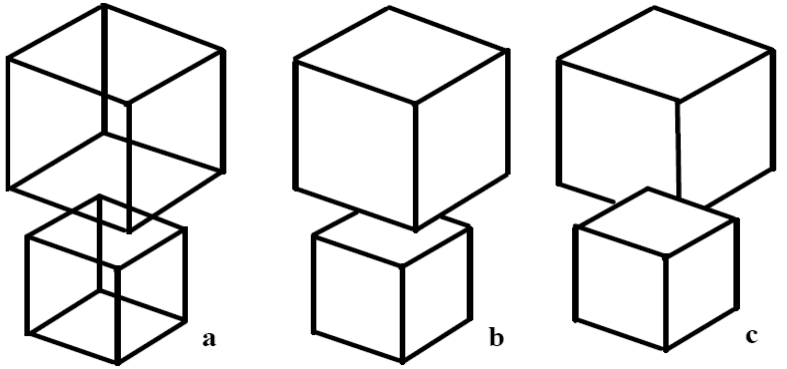
1. Déterminer les surfaces visibles ou éliminer les surfaces cachées
2. Pour créer une image, on doit premièrement effectuer la projection et le clippage
3. Pour une scène constituée d'objets opaques et non-réfléchissants, on doit ensuite déterminer ce qui est à l'avant pour chaque élément de l'image.
4. Finalement, on doit afficher l'élément de l'image avec la bonne couleur



Élimination des parties cachées
Pourquoi éliminer les parties cachées ?



Quel vue de l'objet
veut-on donner ?



- Déterminer les lignes, arêtes, surfaces, volumes, visibles par un observateur

3

Élimination des parties cachées
Position du problème

- Les méthodes d'élimination des parties cachées visent à établir un ordre (appelé ordre de priorités) entre les différents éléments (face, arrête , sommet, etc.) de l'espace à visualiser en fonction de leur profondeurs et, cela pour chaque pixel.
- La difficulté de la résolution des problèmes d'élimination des parties cachées est liée à la complexité de la scène .
- Les méthodes d'élimination des parties cachées existantes se divisent en deux catégories selon le système de coordonnées dans lequel les calculs sont effectués :
  - Algorithme dans l'espace objet.
  - Algorithme dans l'espace image.

4

Algorithmes en espace image

- Le traitement est effectué sur les coordonnées de l'écran,
  - c à d que le calcul des parties cachées n'est effectué qu'après la transformation de visualisation (projection de la scène de l'espace 3D au plan 2D).
- Détermine pour chaque élément d'image (de ***p*** pixels), l'objet (de ***n*** objets) qui est le plus près du centre de projection tout en étant dans le volume de vision. Le temps de calcul est assez élevé.
- O(***n p***) : **n*p** opérations

Ex: 100 x (640x480) = 30,720,000 opérations

- Il vaut mieux choisir un algorithme travaillant dans l'espace objet, mais la résolution du problème dans l'espace image est plus efficace .

5

Algorithmes en espace image

- Précision dépend de la résolution de l'image
- La visibilité doit être recalculée à chaque changement de résolution et/ou de positionnement de la fenêtre
+ Les algorithmes et leurs structures sont habituellement plus simples

Ex: tampon de profondeur, balayage de lignes, Subdivision récursive de l'image, lancer de rayons

Algorithmes en espace objet

- Dans cette catégorie, le problème est résolu de manière géométrique dans le système de coordonnées tridimensionnelles sans prendre en considération l'affichage sur un dispositif à résolution finie.
- Dans ce cas le temps de calcul croît avec le carré du nombre de facettes.
- Principe: Détermine entre chaque objet lequel est devant l'autre étant donnée la direction de projection
- Si on a **n** objets, les algorithmes sont en O($\mathbf{n^2}$)

Ex: 100 x 100 = 10,000 opérations

Algorithmes en espace objet

```
for ( chaque primitive graphique P dans la scène )
    {
    if ( aucune primitive ne cache P )
      {
      afficher P de la bonne couleur;
      }
    }
```

- La précision dépend de la résolution des objets.
- Le calcul de visibilité est indépendant de la résolution de l'image.
- Les algorithmes et leurs structures sont habituellement assez complexes.

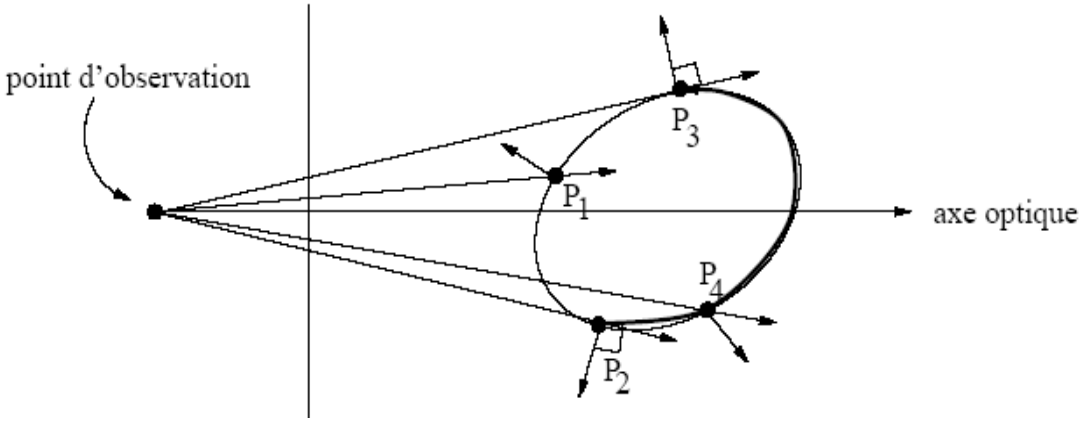
Ex: algorithme du peintre, octree, arbres BSP

Algorithmes en espace objet
Backface culling

- Le backface culling (élimination des faces arrières) repose sur une idée simple.
  - Si l'on connaît les normales à la surface de l'objet observé,
  - il est alors possible d'éliminer les parties de la surface pour lesquelles la normale pointe dans la direction opposée au point d'observation.
  - Ces parties sont en effet occultées (cachées) par l'objet lui-même.

9

Algorithmes en espace objet
Backface culling



- P1 *est visible,* P2 *et* P3 *sont à la limite de visibilité,* P4 *n'est pas visible.*
- Pour les points de la figure ci-dessus, on vérifie aisément que :
  - $N1 \cdot V1 < 0$; $N2 \cdot V2 = N3 \cdot V3 = 0$; $N4 \cdot V4 > 0$;
  - où Vi est la direction de la ligne de vue au point Pi.
- On en déduit que les parties pour lesquelles $N \cdot V > 0$ ne sont pas visibles et peuvent être éliminées.
- pour un seul objet convexe, l'élimination des parties cachées se résume au backface culling.

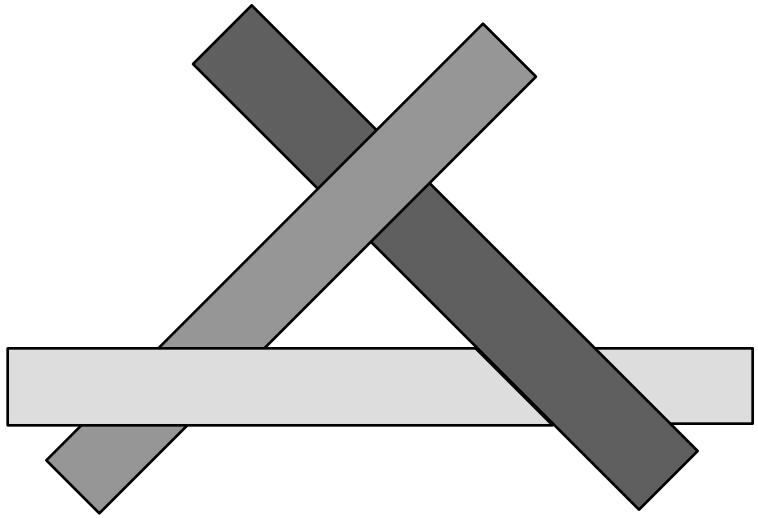
10

Algorithmes en espace objet
L'algorithme du peintre

- Dans l'algorithme du peintre, les objets sont ordonnés en fonction de leur distance à la fenêtre, et ils sont affichés du plus éloigné au plus rapproché

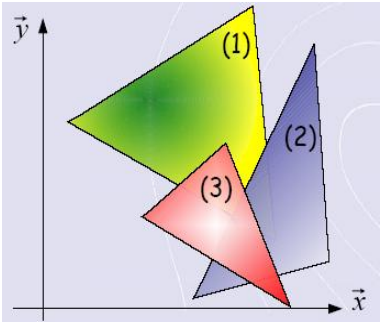
  Ex: celluloïdes

- Dans certains cas, cet ordre est impossible sans découpage au préalable

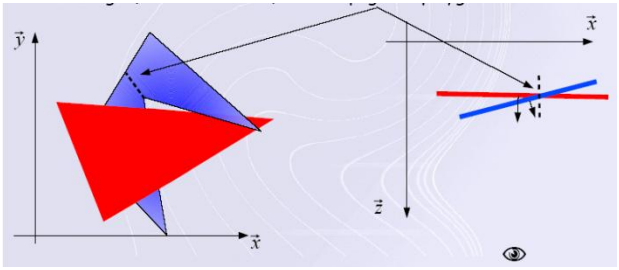


Algorithmes en espace objet
L'algorithme du peintre

- Principe :
  - Tracé des polygones **bien orientés**, du plus éloigné au plus proche
  - Tri des polygones
- Cas trivial (non chevauchement)



- Cas ambigu (chevauchement) : Découpage de polygones



12

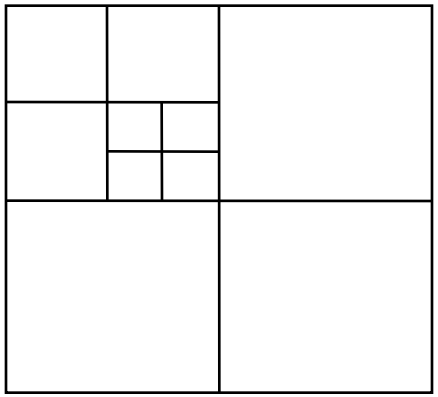
Algorithmes en espace objet

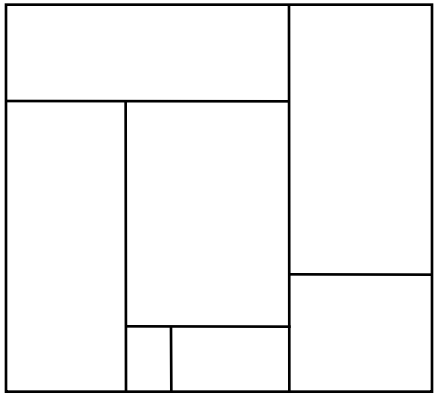
Octree

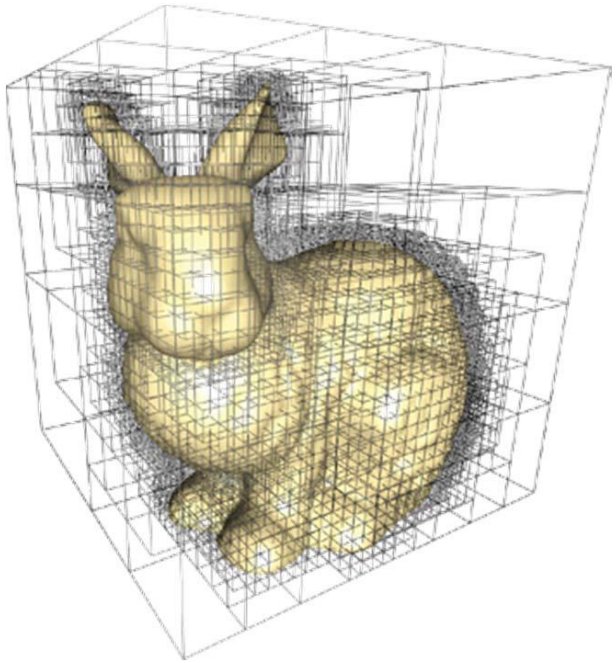
- Un octree est une manière de subdiviser l'espace 3D.
- Le premier objectif des octrees est de réduire le nombre de comparaisons recquis afin de déterminer les polygones de notre monde que l'on va traiter, qu'il s'agisse d'affichage, de tests de visibilité, de tests de collision ou bien de niveau de détails.
- Les octrees engendrent une réduction significative du temps requis pour trier les polygones dans un monde et les afficher.

Algorithmes en espace objet
Octree

- Pour n'importe quel point de vue, on peut énumérer les faces d'un cube de la plus éloignée à la plus rapprochée.
- Cette approche se généralise à l'*octree*







- En alternant selon *X, Y, Z* et glissant le plan de subdivision, on obtient un *kd-tree*

Voir: http://deptinfo.unice.fr/twiki/bin/view/Minfo03/AlgosJeux3DRapportOctrees

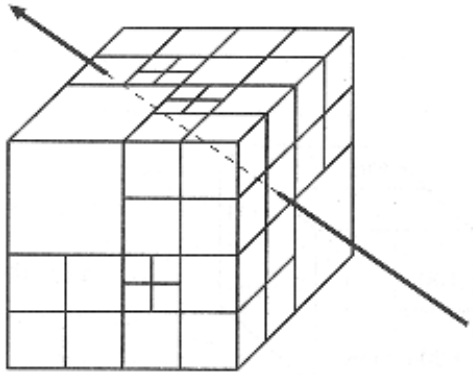
Algorithmes en espace objet

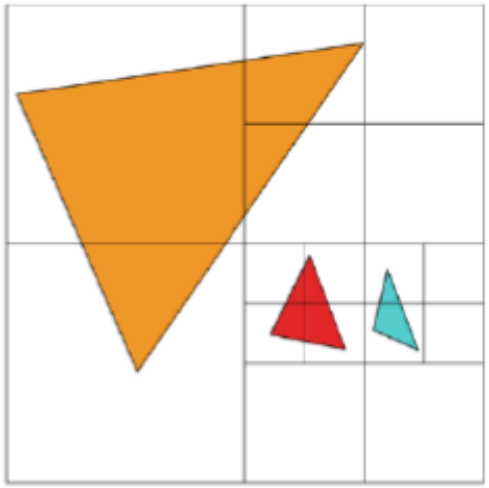
Octree

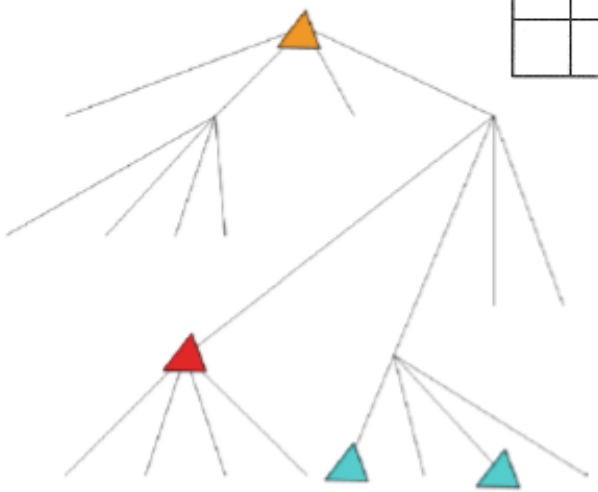
Grille adaptative : Octree

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 »







Algorithmes en espace objet
Partition binaire de l'espace (BSP tree)

- BSP-tree : arbre binaire utilisé pour trier des primitives dans l'espace
- Principe (Schumaker 1969, Fluch 1980) :
  - Choix d'une arête de référence
    - Partage l'espace en 2 demi-espaces (avant et arrière)
    - Répartition des arêtes selon le demi-espace occupé
    - Les arêtes à cheval sont découpées selon le plan de référence
  - Réitération dans les deux demi-espaces

  - \+ Très utile lorsque la scène est statique mais que le point de vue peut être n'importe où dans la scène

16

Création d'une partition binaire

BSPTree * BSPMakeTree(liste de polygones)

choisit un polygone comme **racine**

Pour(tous les autres polygones **p**)

si (**p** est devant)

ajoute **p** à la liste de devant

si (**p** est derrière)

ajouter **p** à la liste de derrière

sinon

découpe **p** en deux par rapport au plan de la racine et ajoute chaque moitié à sa liste respective

BSPTree * node = **new** BSPTree()

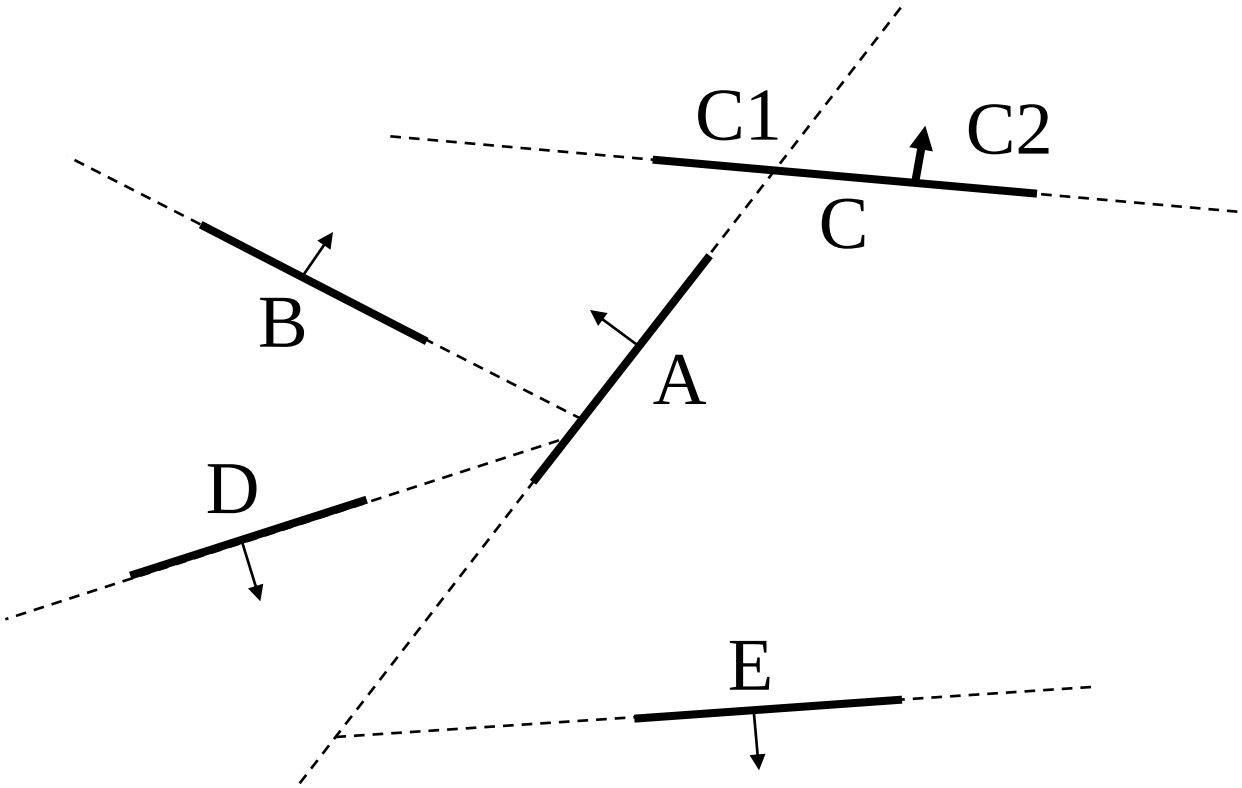
node->element = **racine**

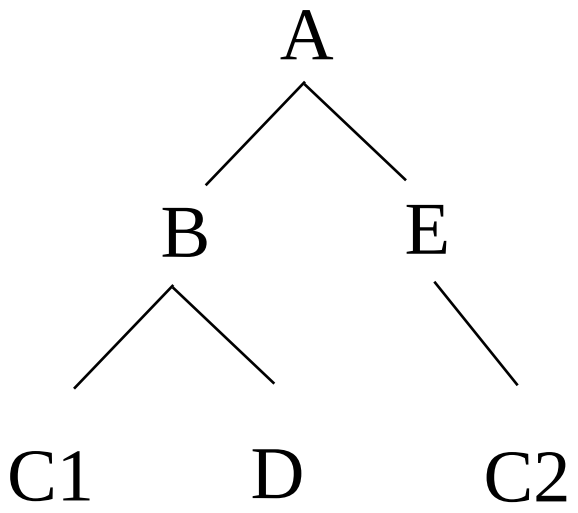
node->devant = **BSPMakeTree**(liste de devant)

node->derrière = **BSPMakeTree**(liste de derrière)

return node

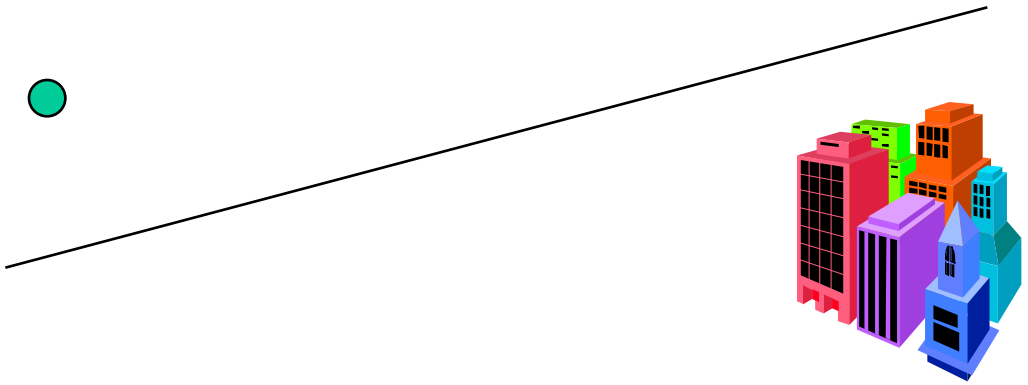
Exemple de construction d'arbre *BSP*





Ordonnancement dans un arbre *BSP*

- Si l'oeil est d'un côté d'un plan, les objets de l'autre côté ne pourront jamais être devant un objet du même côté du plan que l'oeil
- On peut donc traverser l'arbre de derrière à devant pour tout point de vue en affichant simplement avec un algorithme du peintre



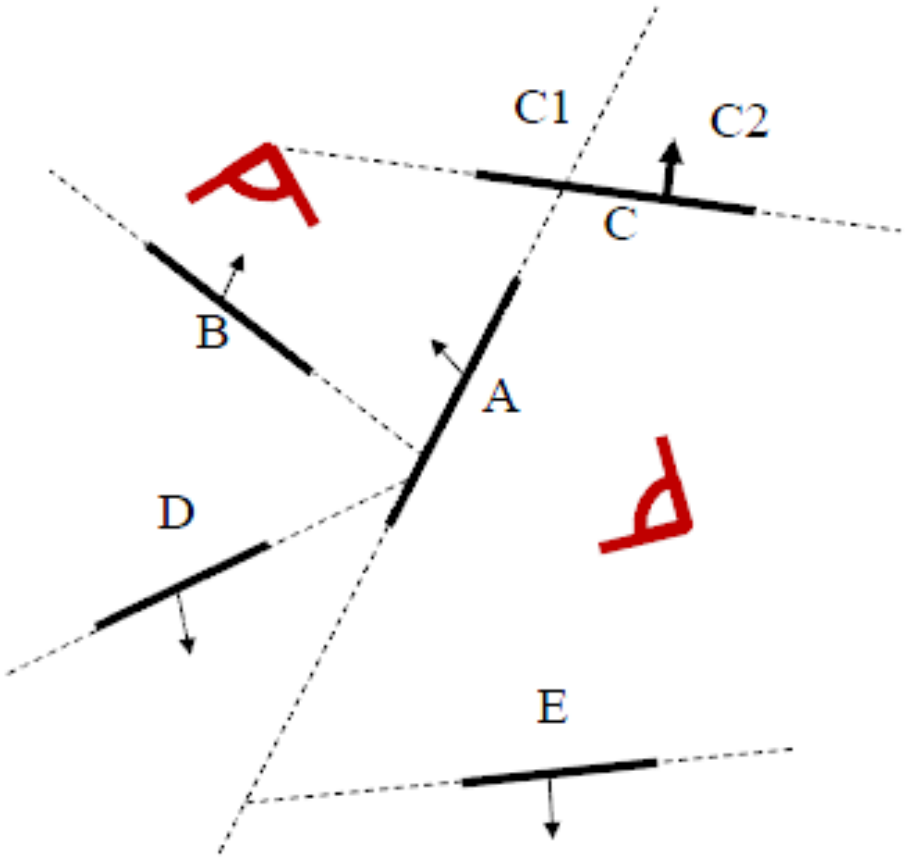
Algorithme d'affichage d'un arbre *BSP*

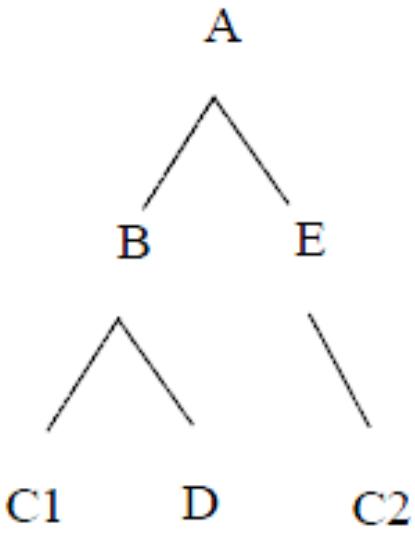
AfficheArbre (sousArbre)

- si l'oeil est devant la racine
  - AfficheArbre (sousArbre->arrière)
  - AffichePolygone (sousArbre->racine)
  - AfficheArbre (sousArbre->devant)
- sinon
  - AfficheArbre (sousArbre->devant)
  - AffichePolygone (sousArbre->racine)
  - AfficheArbre (sousArbre->arrière)

// si aucun *backface culling*

Algorithme d'affichage d'un arbre *BSP*





Avantages et inconvénients du *BSP*

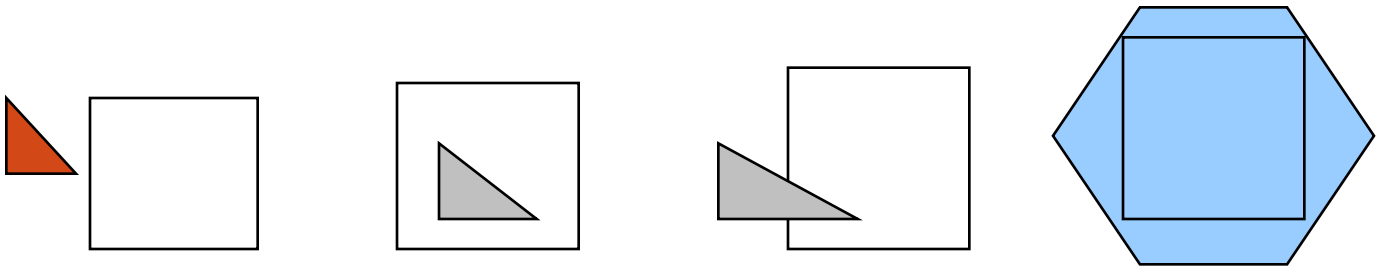
+ Les plans de la pyramide de vue peuvent être utilisés pour le clippage avec *BSP*
+ Si l'illumination (*shading*) est coûteuse, on peut aussi afficher de devant à derrière en n'affichant un pixel que s'il n'a pas déjà été affiché
+ On peut aussi traiter les ombres de sources de lumière ponctuelles
- Obtenir un arbre *BSP* avec le moins de polygones coupés est une tâche complexe
- Problèmes de robustesse en créant des polygones étroits

Algorithmes en espace image

Subdivision récursive de l'image

Principe « diviser – pour - régner » :

- diviser l'image en quadtree
  - Pour chaque quadrant :
    - Déterminer la liste des polygones visibles
  - Tracer le quadrant de l'image directement dans les cas simples:
    - Rien dans le quadrant
      - couleur du fond
    - Un seul polygone : contenu facile à dessiner avec la couleur du polygone
      - le polygone couvrant le quadrant
        - partiellement
        - totalement



23

Algorithmes en espace image

Subdivision récursive de l'image

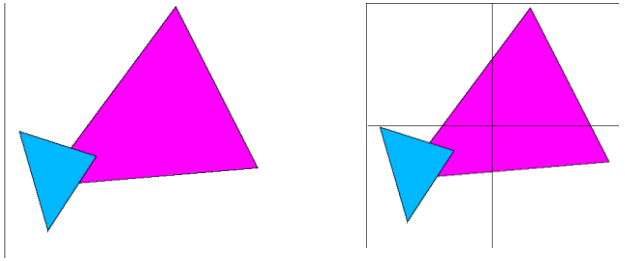
- Cas complexes : plusieurs polygones dans le quadrant
  - Trier les polygones selon la profondeur
  - Si un polygone masque tous les autres :
    - couvre tout le cadrant et est en avant des autres (sans test complexe)
    - tracer avec la couleur du polygone
  - Sinon subdiviser et itérer le processus
- Arrêt quand le quadrant est plus petit qu'un pixel
  - Tracer avec la couleur du polygone en avant des autres

24

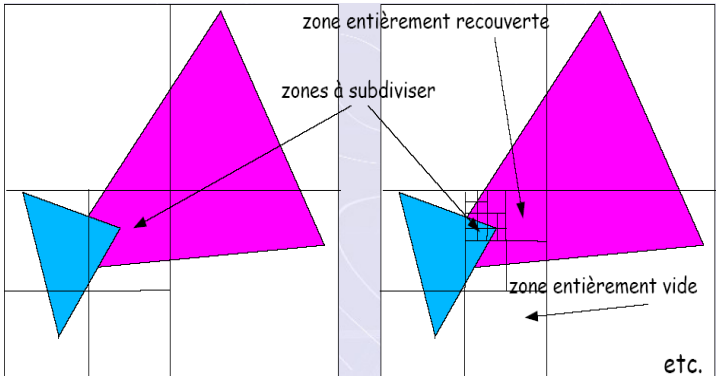
Algorithmes en espace image

Subdivision récursive de l'image

- Première étape
  - Subdivision et test
    - Contenu trop difficile à tracer : réitérer le traitement dans chaque zone



- Deuxième étape ...jusqu'à la dernière étape
  - Découpage récursif dans les quadrants contenant plusieurs polygones
    - Le plus fin le long des arêtes



25

Algorithmes en espace image
Tampon de profondeur (z-buffer)

Initialise tous les pixels

rgb (x,y) = couleur arrière-plan

$z\ (x,y)$ = distance arrière-plan

Pour chaque objet de la scène à représenter

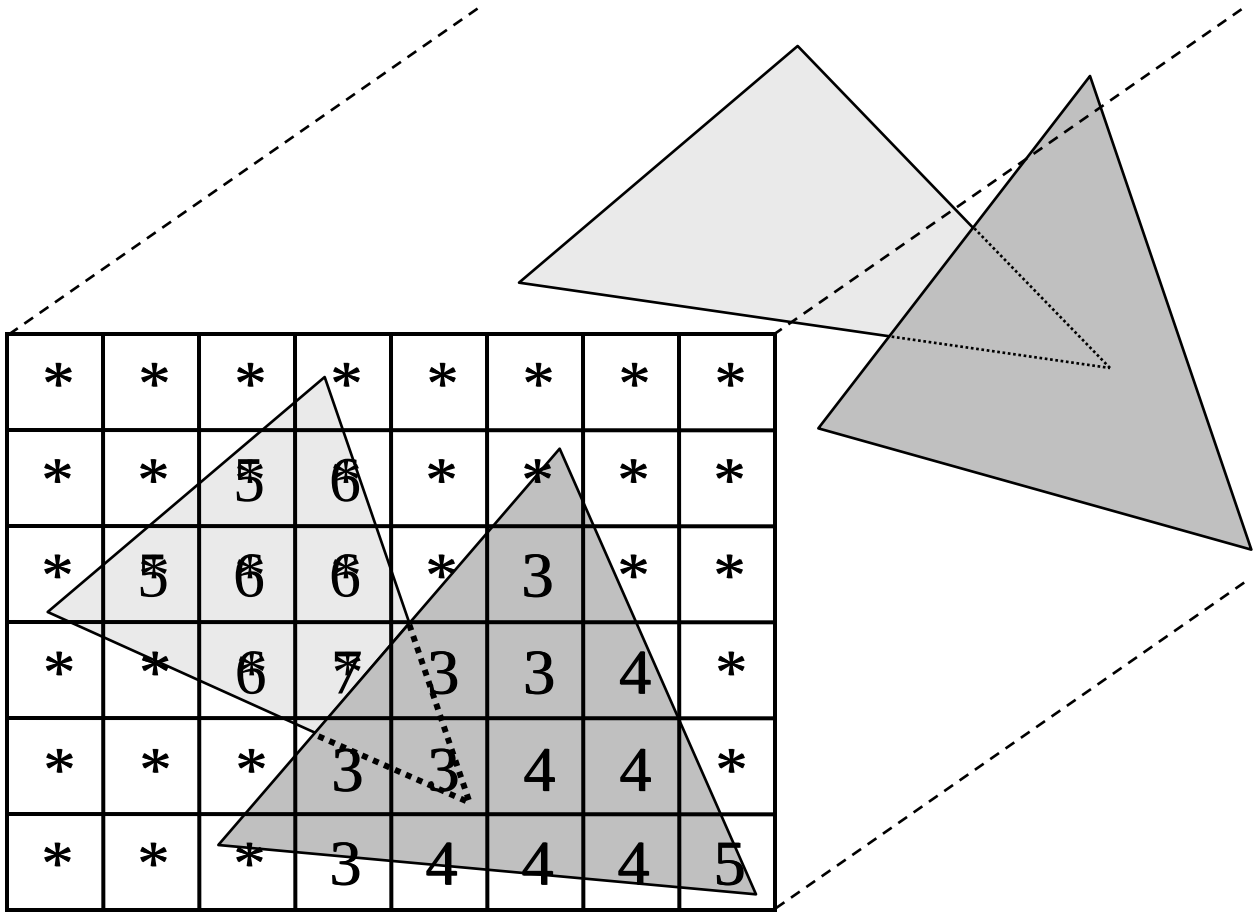
Pour chaque pixel (x,y) couvert par l'objet

Si distance-objet (x,y) $<$ $z\ (x,y)$

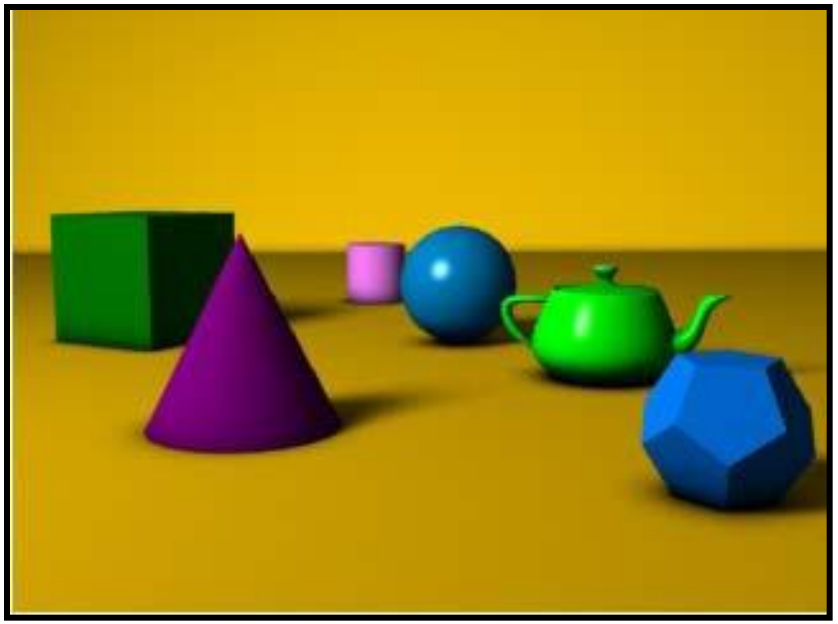
$z\ (x,y)$ = distance-objet (x,y)

rgb (x,y) = couleur objet

Algorithmes en espace image
Tampon de profondeur (z-buffer)



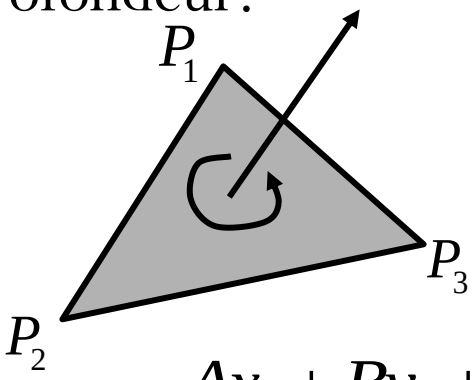
Algorithmes en espace image
Tampon de profondeur (z-buffer)



Algorithmes en espace image
Tampon de profondeur (z-buffer)

Scan-conversion: remplissage par balayage

Profondeur:



$$\vec{N} = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\left\| (P_2 - P_1) \times (P_3 - P_1) \right\|} = (A, B, C)$$

$$Ax_1 + By_1 + Cz_1 + D = 0 \quad \Rightarrow \quad D = -Ax_1 - By_1 - Cz_1$$

$$z(x, y) = \frac{-D - Ax - By}{C}$$

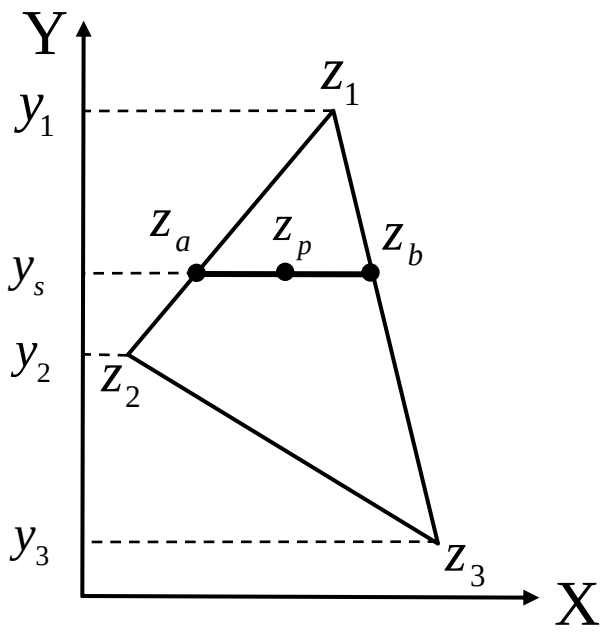
$$z(x + \Delta x, y) = \frac{-D - A(x + \Delta x) - By}{C} = z(x, y) - \frac{A\Delta x}{C} \left.\begin{array}{l} \Delta x = 1 \\ A/C = const \end{array}\right\}$$

$$z(x, y + \Delta y) = \frac{-D - Ax - B(y + \Delta y)}{C} = z(x, y) - \frac{B\Delta y}{C} \left.\begin{array}{l} \Delta y = 1 \\ B/C = const \end{array}\right\}$$

http://csis.pace.edu/~marchese/CG/Lect9New/cg_l9new.htm

Algorithmes en espace image
Tampon de profondeur (z-buffer)

Interpolation bilinéaire (alternative)



$$y_s = y_1 + t(y_2 - y_1)$$

$$z_a = z_1 + t(z_2 - z_1)$$

$$\frac{z_a - z_1}{z_2 - z_1} = \frac{y_s - y_1}{y_2 - y_1}$$

$$z_a = z_1 + (z_2 - z_1)\frac{(y_s - y_1)}{(y_2 - y_1)}$$

$$z_b = z_1 + (z_3 - z_1)\frac{(y_s - y_1)}{(y_3 - y_1)}$$

Interpole aussi:
couleur, texture, normale, etc.

$$z_p = z_a + (z_b - z_a)\frac{(x_p - x_a)}{(x_b - x_a)}$$

Algorithmes en espace image
Tampon de profondeur (z-buffer)

Avantages/désavantages:

\+ simplicité, généralité

\+ hardware, parallélisme

\- mémoire additionnelle pour les tampons

\+ aucune mémoire pour conserver la scène

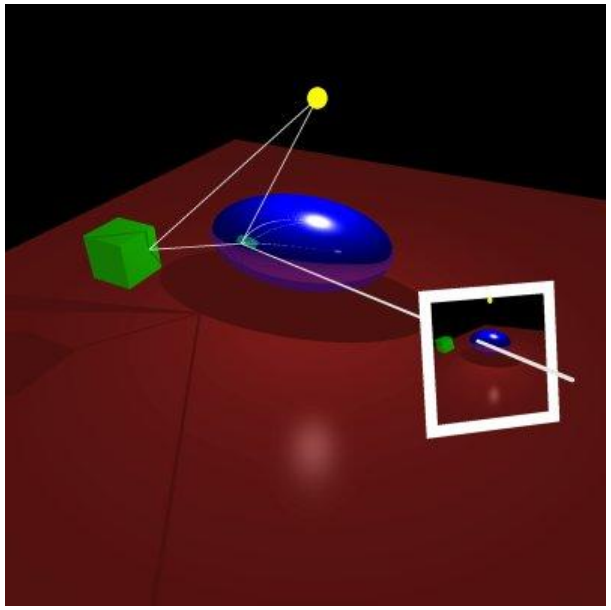
\- aliassage (en XY et en Z)

Algorithmes en espace image
Lancer de rayons *(ray tracing/ray casting)*

- Les objets ne sont pas éclairés seulement par la lumière émise directement par les sources de lumières .
- Ils sont éclairés par celle qui atteint l'objet après réflexion et transmission à travers les surfaces .
- Le lancer de rayon, ou Ray Tracing en anglais :
  - C'est une méthode de calcul d'image sur ordinateur qui permet de tenir compte de la plupart des phénomènes optiques :
    - Ombrage propre, ombres portées, réflexions spéculaires, réfraction des rayons au travers de certains matériaux tels que le verre ou le cristal.
  - Les images ainsi obtenues sont souvent qualifiées de *photo-réalistes* car elles simulent assez bien la réalité.
  - C'est l'une des techniques les plus avancées pour un rendu correcte.
  - Il permet une résolution dans l'espace image et l'élimination des parties cachées en même temps.

Algorithmes en espace image
Lancer de rayons *(ray tracing/ray casting)*

- Généralement, les techniques précédentes en espace image projettent les objets sur une fenêtre. Il faut alors ***projeter*** tous les ***n*** objets: *O(**n p**)*
- Le lancer de rayons (ou *ray casting* pour ne traiter que la visibilité) consiste à lancer
  - un rayon au travers de la fenêtre
  - pour identifier la première surface
  - entre la fenêtre et la scène 3D:
  - *O(n res)* pour *res* pixels (résolution)
- Traitement pixel par pixel plutôt que objet par objet

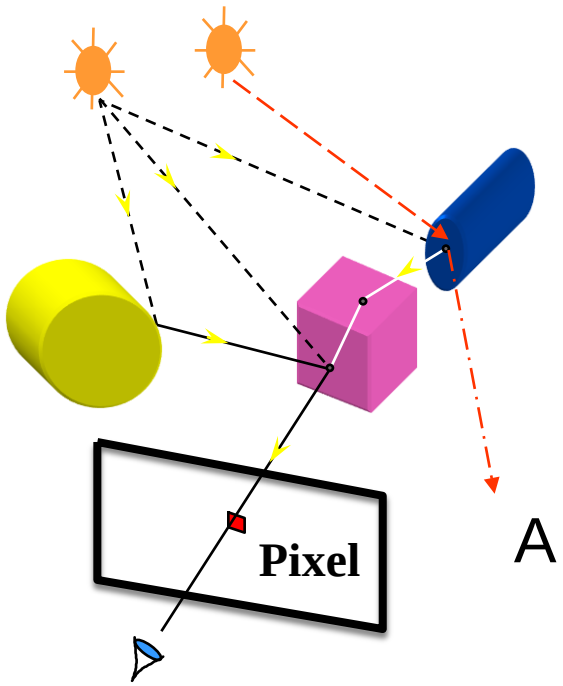


Lancer de rayons
Principe géométrique

- La lumière part des sources lumineuses pour aboutir, après diverses réfraction et réflexion, à l'observateur.
- parmi les rayons issus d'une source lumineuse, nombreux sont ceux qui, n'aboutissent jamais à l'observateur (A) .

❖ Impossible de suivre tous les rayons lumineux .

➢ Trajet inverse des rayons lumineux .

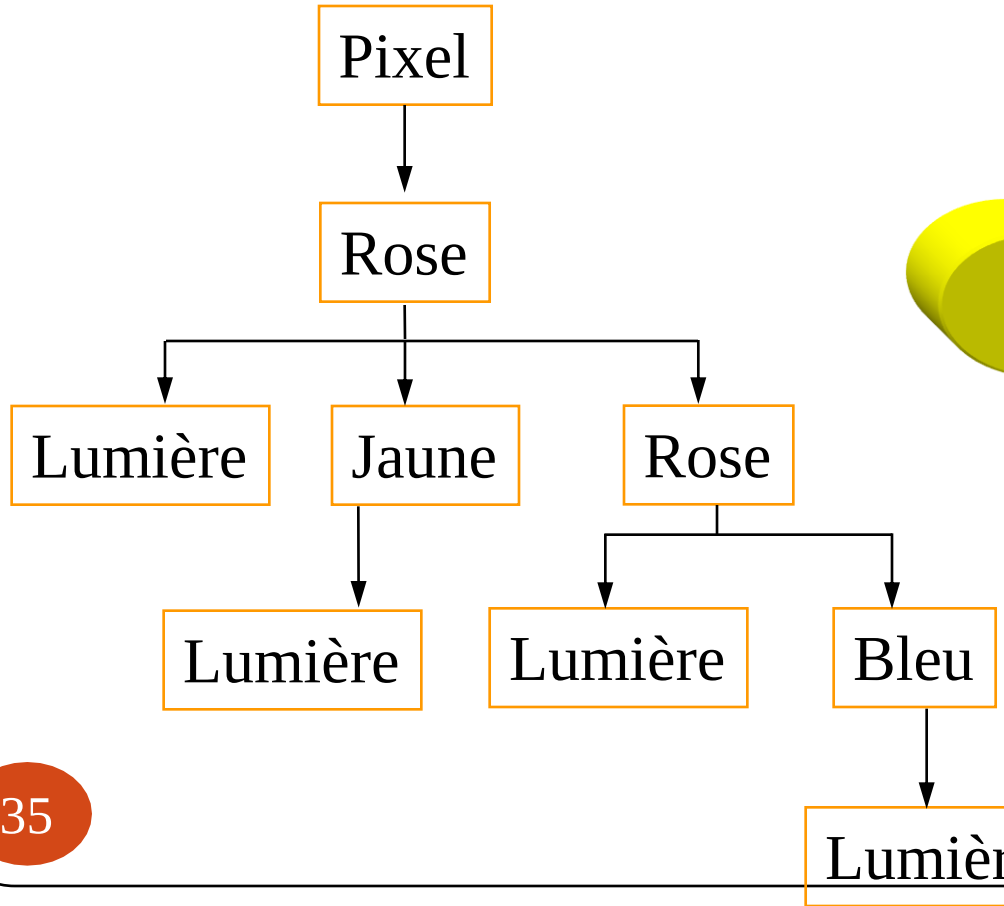


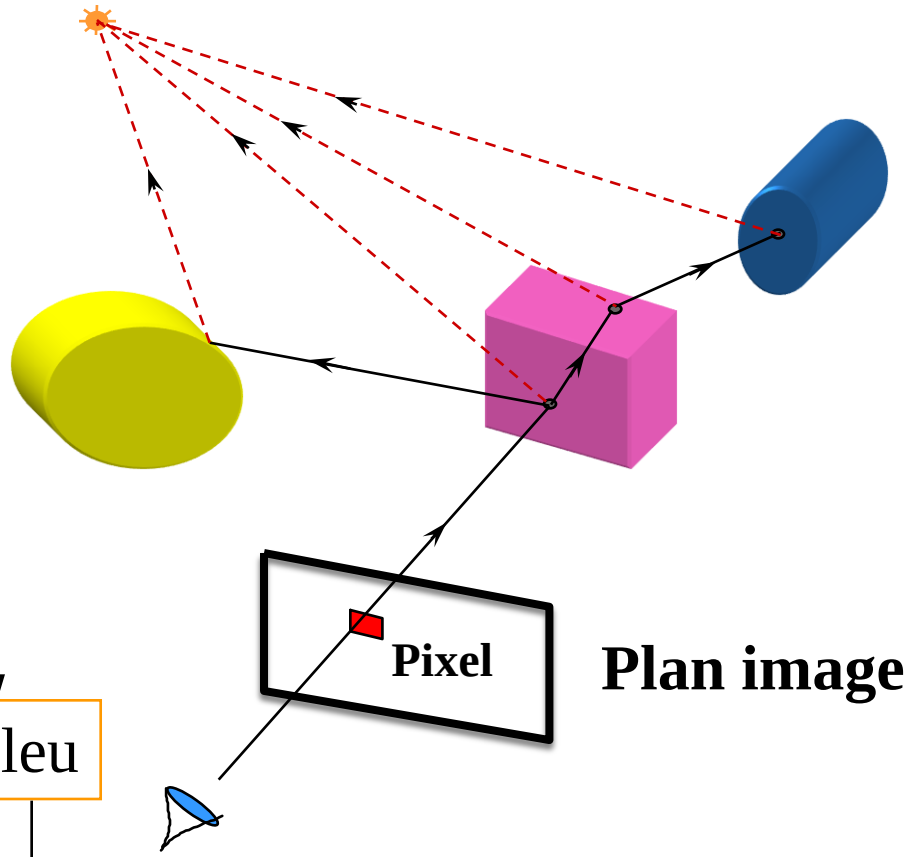
Plan image

Lancer de rayons
Modèle d'éclairage

- C'est une technique qui consiste à suivre le trajet de la lumière en sens inverse, de l'observateur vers les sources lumineuses,
- En intégrant des propriétés de transparence et de réflexion.

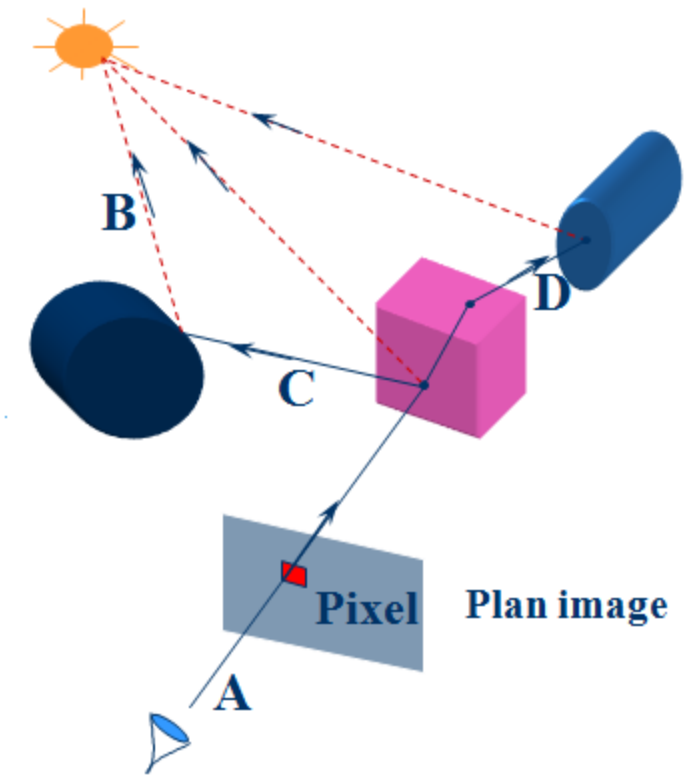
- Arbre de rayons





Lancer de rayons
Les classes des rayons lumineux

- **Les rayons primaires (A) :**
  - *issus directement de l'observateur .*
- **Les rayons d'éclairement (B) :**
  - *transmettent la lumière directement d'une source lumineuse vers la surface d'un objet .*
- **Les rayons réfléchis (C) :**
  - *transmettent la lumière réfléchie par un objet .*
- **Les rayons de transparence (D) :**
  - *transmettent les rayons réfractés à travers un objet transparent ou partiellement transparent .*

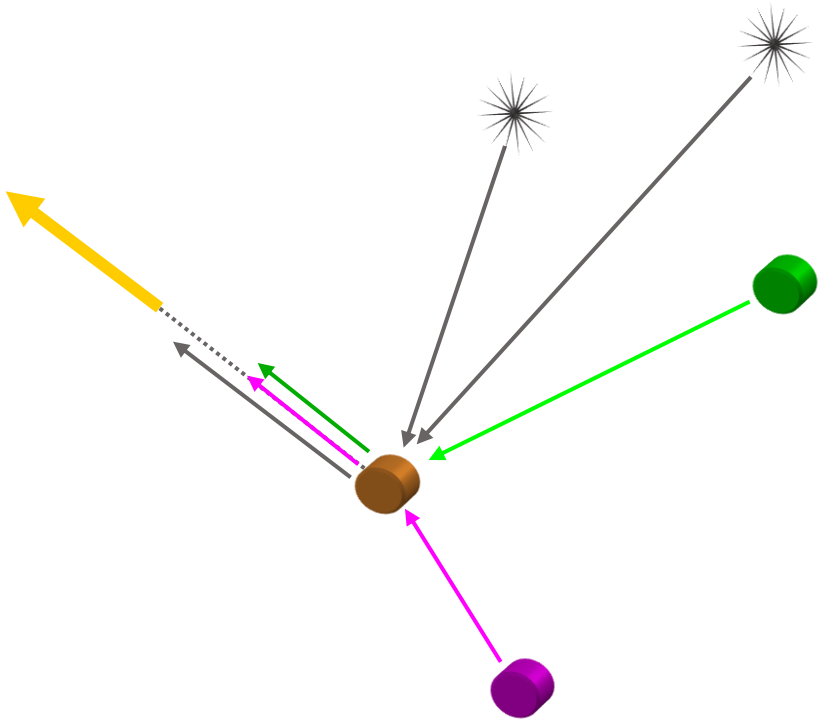


Lancer de rayons
Modèle d'éclairage

Intensité lumineuse sur un rayon

- Modèle local
- Reflets spéculaires

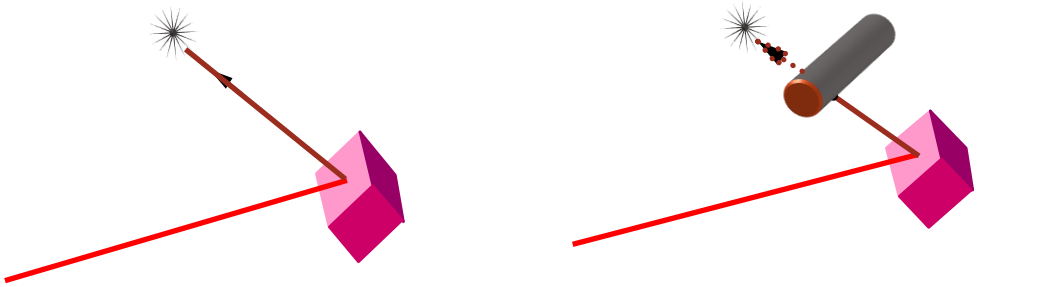
$$I_R = I_{Local} + k_s I_{Refl} + k_t I_{Trans}$$



Lancer de rayons
Modèle d'éclairage

- Modèle Local :

  Calcul des ombres portées



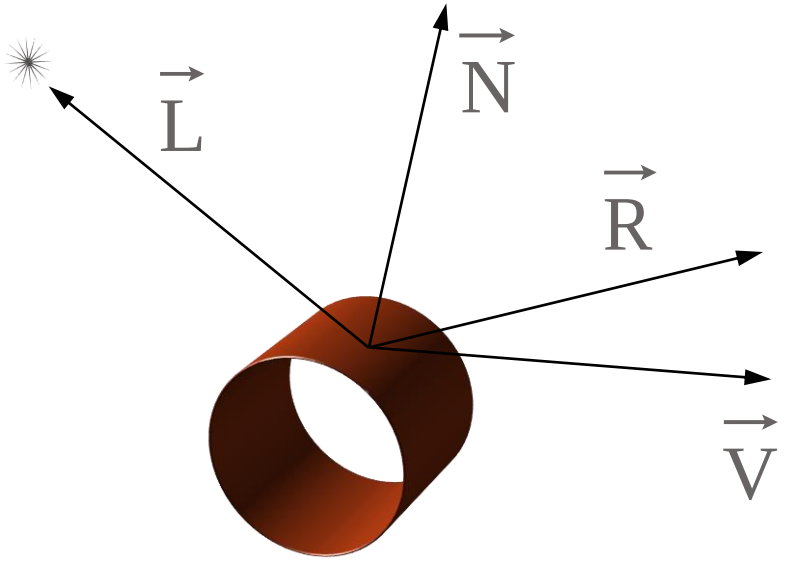
Point éclairé **Point dans l'ombre**

Lancer de rayons
Modèle d'éclairage

- Modèle Local :

Modèle de Phong [Phong75]

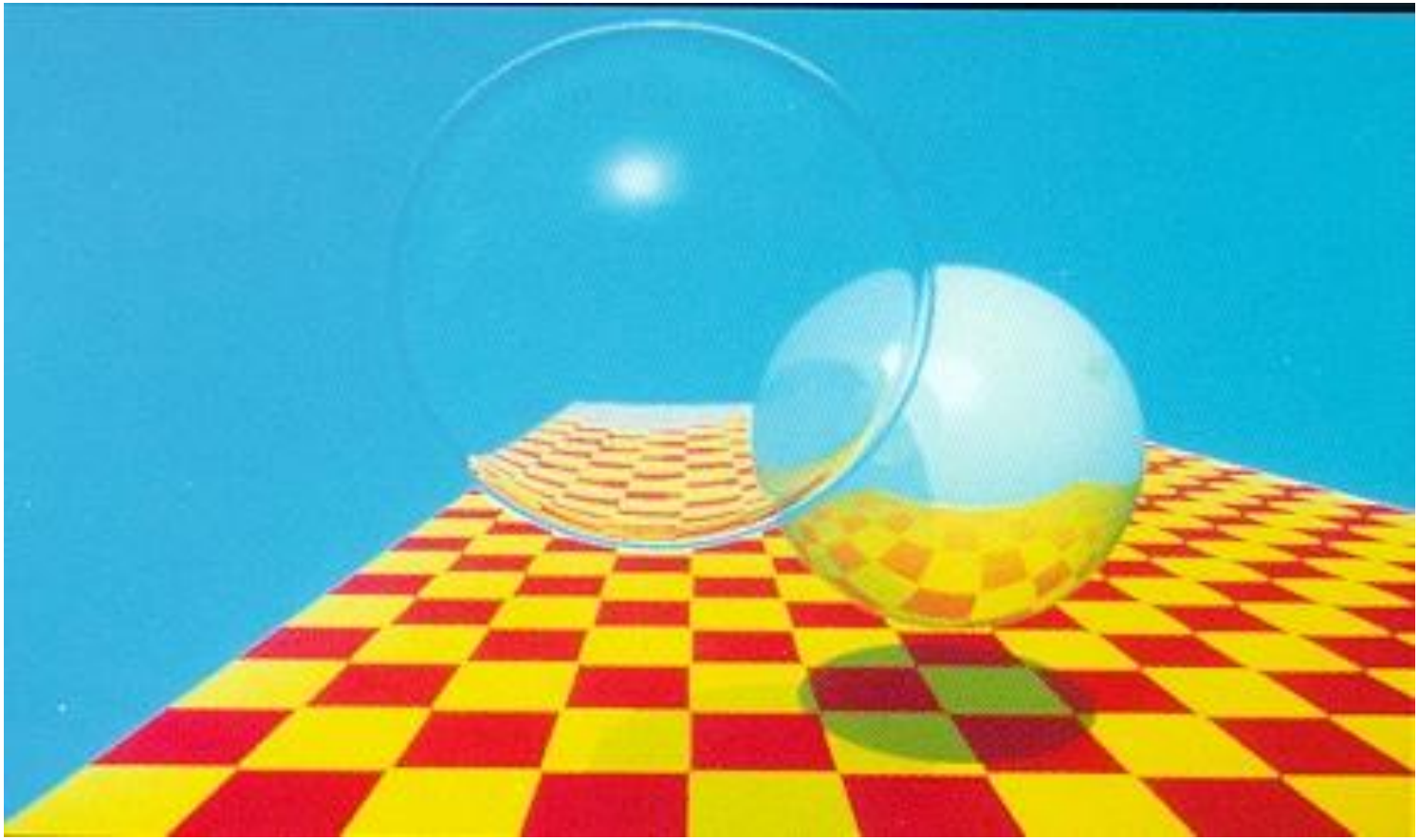
$$I_{local} = \sum_{i=0}^{nbLum} I_i \times vis(i) \times \left(k_d \left(\vec{N} \cdot \vec{L} \right) + k_s \left(\vec{R} \cdot \vec{V} \right)^n \right)$$



Lancer de rayons
Algorithme

```
Couleur LR(origine, direction, profondeur)
// origine et direction sont des vecteurs de R3
   Si profondeur>Max_Prof Alors
      couleur=Noir;    // choix discutable mais que mettre ?
   Sinon
       //Calcul et tri des intersections
       Si intersection Alors
      Calcul du I_local (C_locale);
      Calcul du rayon réfléchi (D_réfléchi);
      Calcul du rayon réfracté (D_réfracté);
      C_réfléchi = LR(Pt-inter, D-réfléchi,profondeur+1);
      C_ réfracté = LR(Pt-inter, D-réfracté,profondeur+1);
      couleur = Somme des 3 composantes couleurs;
       Sinon
      couleur = couleur de fond;
```

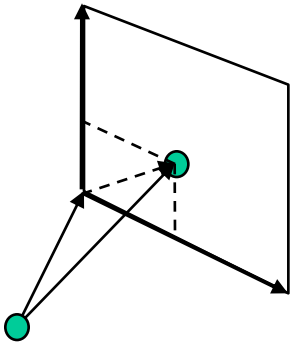
Résultat



Intersection rayon - surface

- Au cœur de calcul de lancer de rayon réside la tâche de détermination d'intersection d'un rayon avec un objet
- Le problème de l'intersection rayon-surface dépend de la complexité de l'objet
- L'utilisation d'une méthode analytique pour trouver l'intersection d'un rayon avec une surface, consiste à analyser les équations de la surface et le rayon, et à essayer de trouver une solution commune (ou plusieurs) s'il en existe.

Lancer de rayons - Le rayons



- Un rayon est défini par

  une origine (x_0, y_0, z_0)

  une direction $(\Delta x, \Delta y, \Delta z)$

  définie par un point sur la fenêtre (x_1, y_1, z_1)

- Sous forme paramétrique, un rayon s'exprime comme

$$P = P_0 + t(P_1 - P_0)$$

$$x = x_0 + t\Delta x \qquad \Delta x = x_1 - x_0$$

$$y = y_0 + t\Delta y \qquad \Delta y = y_1 - y_0$$

$$z = z_0 + t\Delta z \qquad \Delta z = z_1 - z_0$$

Lancer de rayons - Intersection avec un plan

- L'intersection d'un rayon avec un plan (A,B,C,D) consiste à exprimer le plan en fonction de la forme paramétrique du rayon

$$Ax + By + Cz + D = 0$$

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$$

$$t = \frac{-(Ax_0 + By_0 + Cz_0 + D)}{A\Delta x + B\Delta y + C\Delta z}$$

- Pour l'intersection avec un polygone, il faudra déterminer si le point d'intersection est à l'intérieur du polygone

Lancer de rayons - Intersection avec une sphère

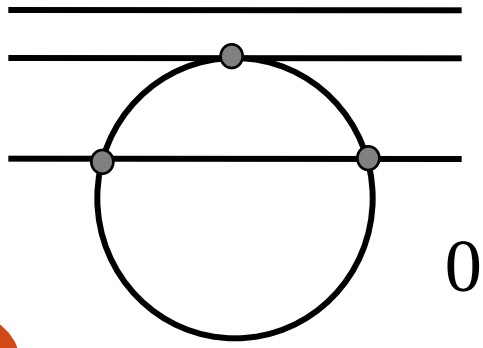
- Similairement pour l'intersection d'un rayon avec une sphère

$$x^2 + y^2 + z^2 = 1$$

$$(x_0 + t\Delta x)^2 + (y_0 + t\Delta y)^2 + (z_0 + t\Delta z)^2 = 1$$

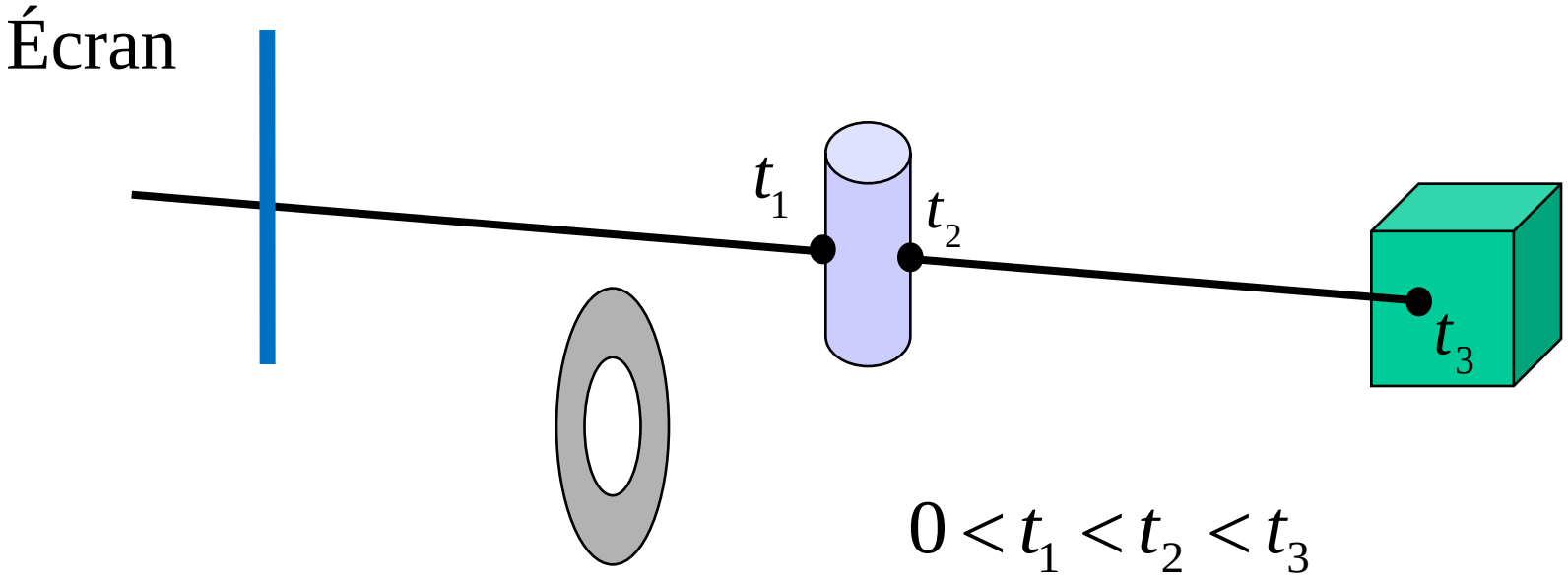
$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2(x_0\Delta x + y_0\Delta y + z_0\Delta z)t + (x_0^2 + y_0^2 + z_0^2 - 1) = 0$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



0, 1 ou 2 intersections réelles

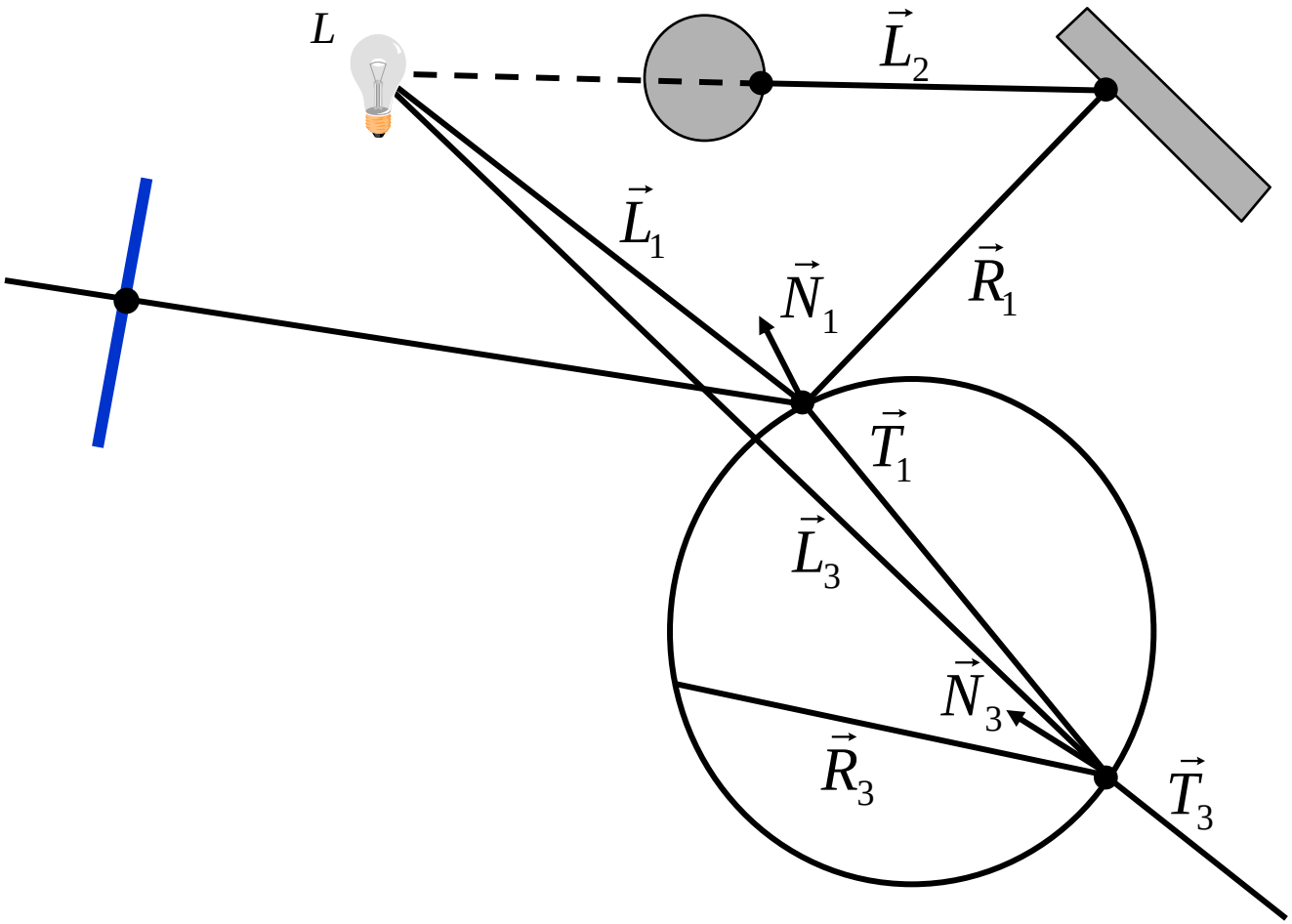
Lancer de rayons - Visibilité au pixel



Lancer de rayons récursif *(ray tracing)*

- Le premier rayon identifie la visibilité à partir de l'œil
- Du point d'intersection, d'autres rayons peuvent aussi être tracés pour déterminer les contributions de la réflexion et de la transmission/réfraction
- On doit s'assurer de ne pas ré-intersecter l'objet au nouveau point de départ (*surface acné*)
  - ajouter une distance epsilon au rayon (trouver la bonne valeur par rapport aux objets et la scène)
  - ne pas intersecter la surface elle-même (plus limité: surface convexe et test par rapport à la normale)

Lancer de rayons récursif *(ray tracing)*



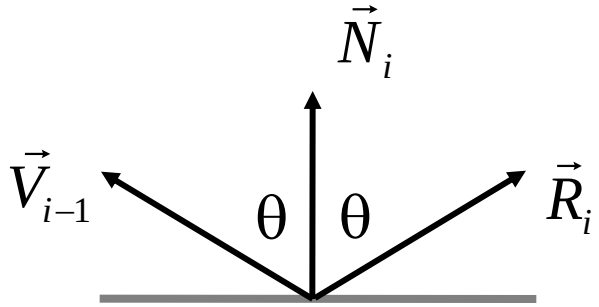
Lancer de rayons: types de rayons

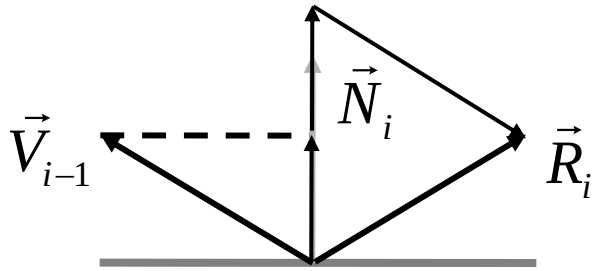
- Rayons d'ombre:

$$\vec{L}_i = \frac{(L - P_i)}{\|(L - P_i)\|}$$

- Rayons réfléchis:

$$\vec{R}_i = 2(\vec{V}_{i-1} \cdot \vec{N}_i)\vec{N}_i - \vec{V}_{i-1}$$

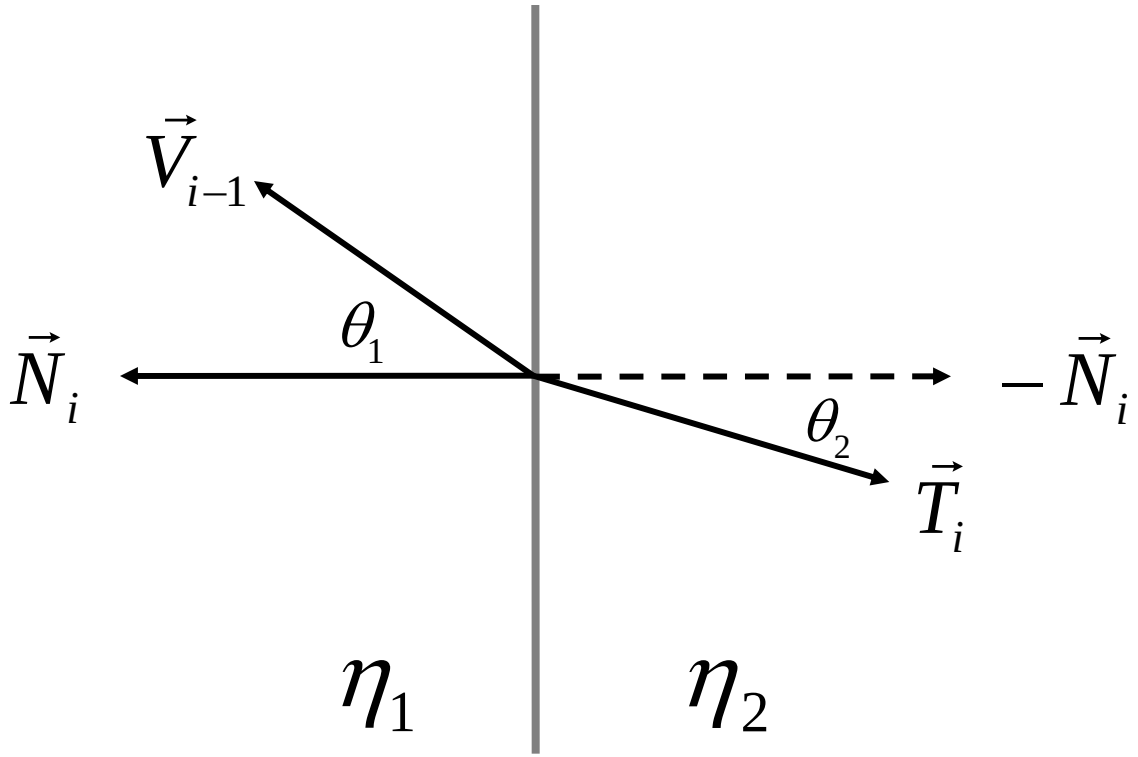




Lancer de rayons: types de rayons

- Rayons transmis / réfractés:
  (la loi de Snell)

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{\eta_2}{\eta_1}$$



Lancer de rayon: réfraction

$$\vec{T}_i = \vec{M}\sin\theta_2 - \vec{N}_i\cos\theta_2$$

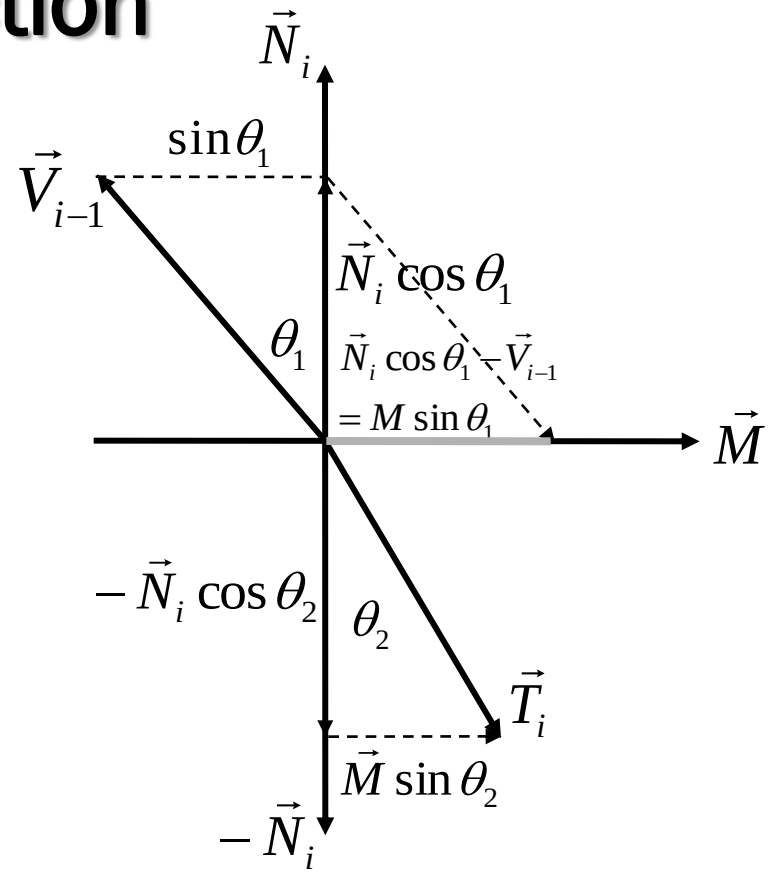
$$\vec{M} = \frac{(\vec{N}_i\cos\theta_1 - \vec{V}_{i-1})}{\sin\theta_1}$$

$$\vec{T}_i = \frac{(\vec{N}_i\cos\theta_1 - \vec{V}_{i-1})}{\sin\theta_1}\sin\theta_2 - \vec{N}_i\cos\theta_2$$

$$\vec{T}_i = \frac{\eta_1}{\eta_2}(\vec{N}_i\cos\theta_1 - \vec{V}_{i-1}) - \vec{N}_i\cos\theta_2$$

$$\eta = \eta_1/\eta_2$$

$$\vec{T}_i = \vec{N}_i(\eta\cos\theta_1 - \cos\theta_2) - \eta\vec{V}_{i-1}$$

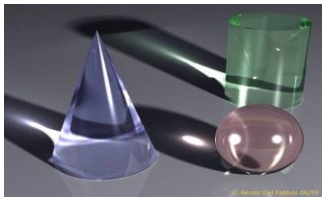


$$\sqrt{1-\sin^2\theta_2} = \sqrt{1-\eta^2\sin^2\theta_1} = \sqrt{1-\eta^2(1-\cos^2\theta_1)}$$

$$\vec{T}_i = \vec{N}_i\left(\eta(\vec{N}_i\cdot\vec{V}_{i-1}) - \sqrt{1-\eta^2(1-(\vec{N}_i\cdot\vec{V}_{i-1})^2)}\right) - \eta\vec{V}_{i-1}$$

Avantages du lancer de rayons

- Réalisme
  - Des effets tels que les réflexions et les transparences sont restitués intrinsèquement. Aucun autre algorithme n'est capable de modéliser ces effets de manière aussi satisfaisante.
  - Les parties cachées sont éliminées de manière intrinsèque.
  - Au niveau 1 de la récursivité, on est en présence d'un Z-Buffer (Ray Casting).
  - Les ombres portées sont elles aussi intrinsèques au modèle.



Avantages du lancer de rayons

- Objets de base évolués
  - Pas de décomposition de la scène en facettes ou surfaces élémentaires
    - les objets lisses apparaissent réellement lisses.
  - Les opérations de base utilisées sont les tests d'intersection objet-rayon.
  - Classiquement les objets modélisant une scène affichée en lancer de rayons sont des sphères, des parallélépipèdes rectangles, des cônes,… (affectés de rotations, translations, zooms, affinités).
  - On peut aussi utiliser des facettes

Inconvénients du lancer de rayons

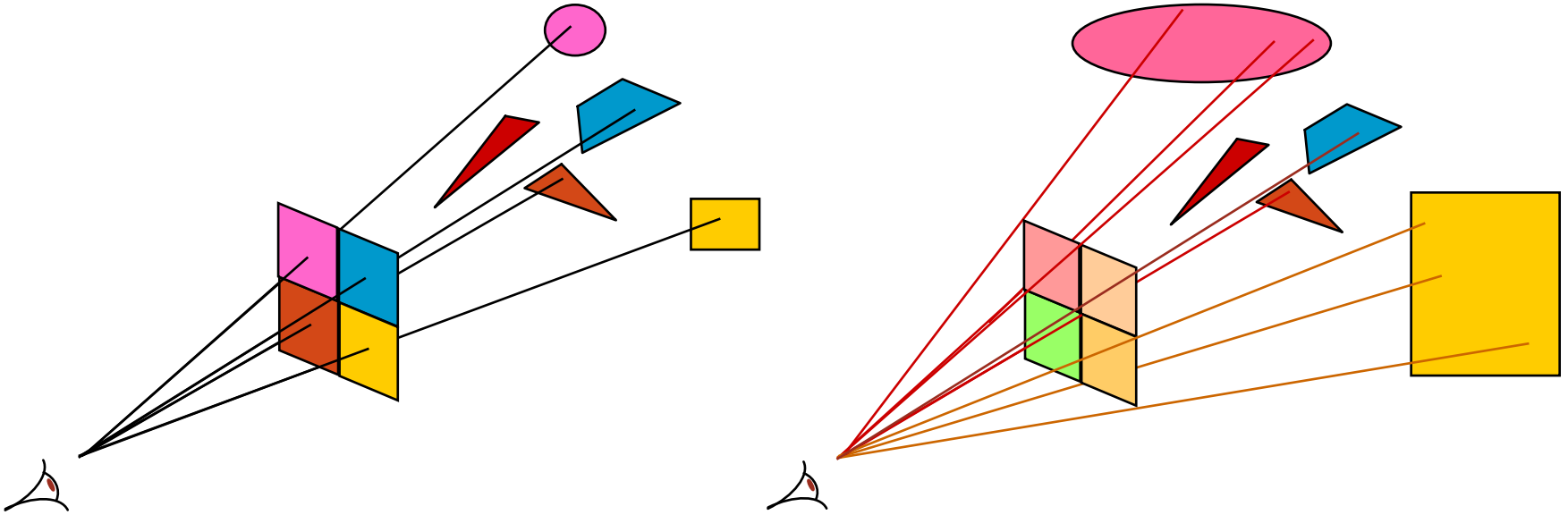
- Récursivité
  - A chaque interception d'un objet par un rayon, on lance potentiellement deux rayons:
    - un rayon pour la gestion de la transparence et
    - un rayon pour la gestion de la réflexion spéculaire.
  - ➢ On assiste possiblement à une explosion exponentielle du nombre total de rayons tracés à l'intérieur de la scène
    - car si n est le niveau de profondeur de la récursivité, il y a au maximum de l'ordre de 2^n rayons tracés pour chaque pixel.
  - Pour obtenir un certain réalisme le niveau de récursivité doit être poussé assez loin (au moins 8, habituellement 10 à 12 voire 14) et donc les temps de calcul peuvent être très longs (plusieurs heures pour des scènes complexes).

Inconvénients du lancer de rayons

- Problèmes de calcul d'illumination
  - Le lancer de rayons ne peut pas restituer un effet tel que la propagation de la lumière à l'intérieur d'une loupe (obtention d'une concentration ou d'une atténuation énergétique en certaines zones) car il ne modélise que très imparfaitement les transferts d'énergie.
  - On ne peut pas restituer la diffusion de la lumière dans les matériaux (en particulier l'air)
    - le non réalisme de certaines ombres.
  - L'illumination est "moyennement" locale c'est à dire qu'elle provient toujours quasiment directement d'une source de lumière répertoriée comme telle, tandis que dans la réalité toutes les surfaces et tous les milieux sont récepteurs et émetteurs de lumière
  - Ce problème est partiellement résolu en attribuant une composante de lumière ambiante à chacun des éléments de la scène

Critiques du Lancer de Rayons

- Aliassage et disparition de petits objets



Critiques du Lancer de Rayons

- Temps de calcul important
  - Intersections
  - Nombre de rayons

- Illumination calculée dans l'espace image
  - Stockage au niveau du pixel
  - Recalculée pour chaque image

Problème

- Le lancer de rayon produise de plus belles images
- Il est très coûteux en temps de calcul (test d'intersection)

➢ réduire le temps de calcul d'intersection

➢ améliorer l'efficacité de lancer de rayon

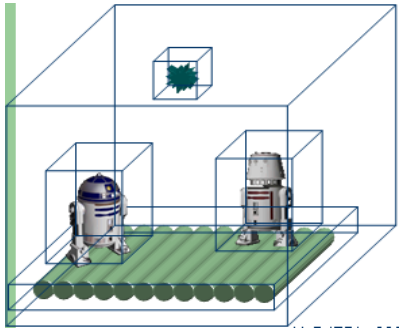
  - Optimisation
  - => Accélération du Lancer de Rayons

Accélérations du lancer de rayons

- Les optimisations classiques mises en œuvre autour de l'algorithme du lancer de rayons portent sur l'amélioration de trois facteurs clés:
  - le calcul des intersections → amélioration des tests d'intersection,
  - le nombre de calculs d'intersection → diminution de ce nombre,
  - le nombre de rayons tracés → diminution du nombre de rayons primaires et diminution du nombre rayons secondaires.

Le calcul des intersections

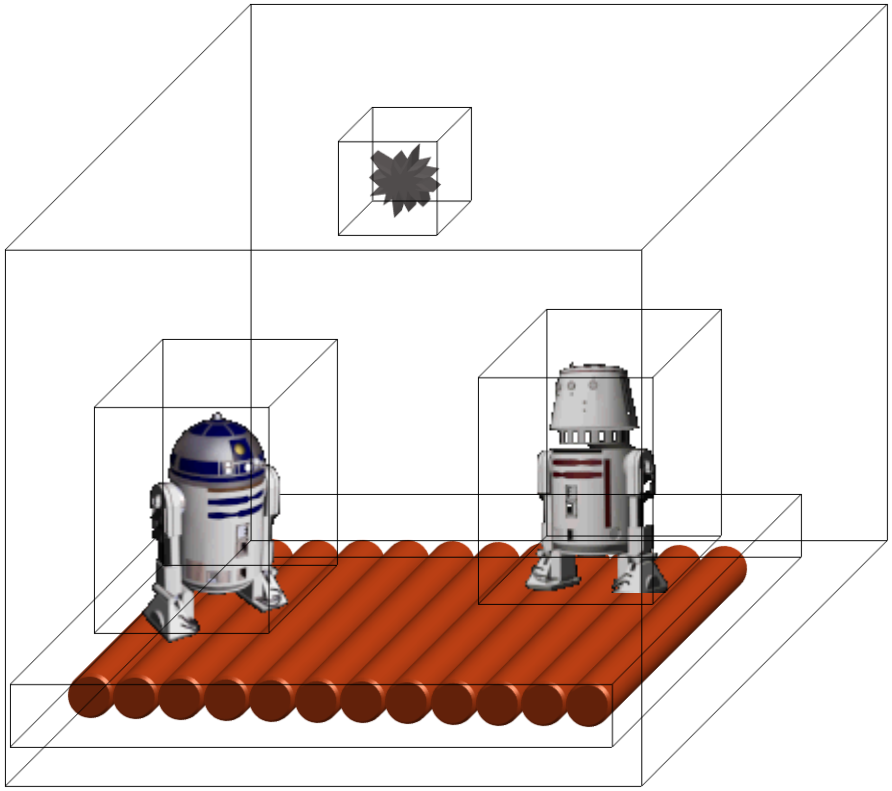
- Le test d'intersection est l'opération de base de tout algorithme de lancer de rayon.
- Son optimisation pour les différents objets pris en compte permet des gains de temps.
- *Les volumes englobants*
  - Objectif :
    - Diminuer le nombre d'intersections rayon-objet
    - Accélérer la détection de non-intersection

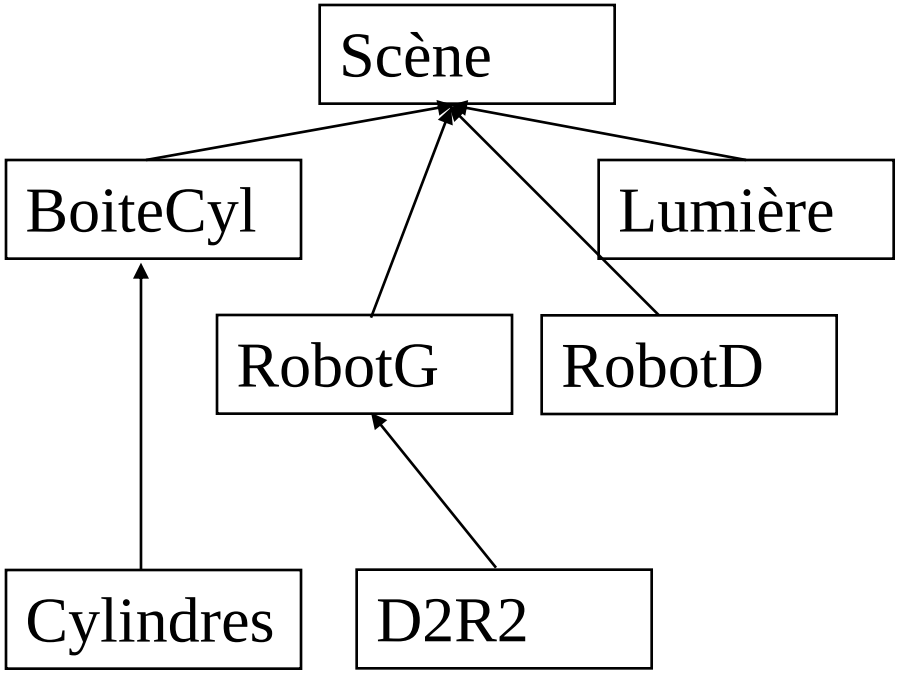


  - Principe :
    - Définir un volume englobant l'objet.
    - Lorsqu'un rayon est lancé, on calcule l'intersection entre le volume englobant et le rayon.
    - S'il y a une intersection, on calcule l'intersection rayon-objet.
    - Construction d'une hiérarchie de boites englobantes
    - Parcours de l'arbre pour le calcul d'intersection

Volumes englobant

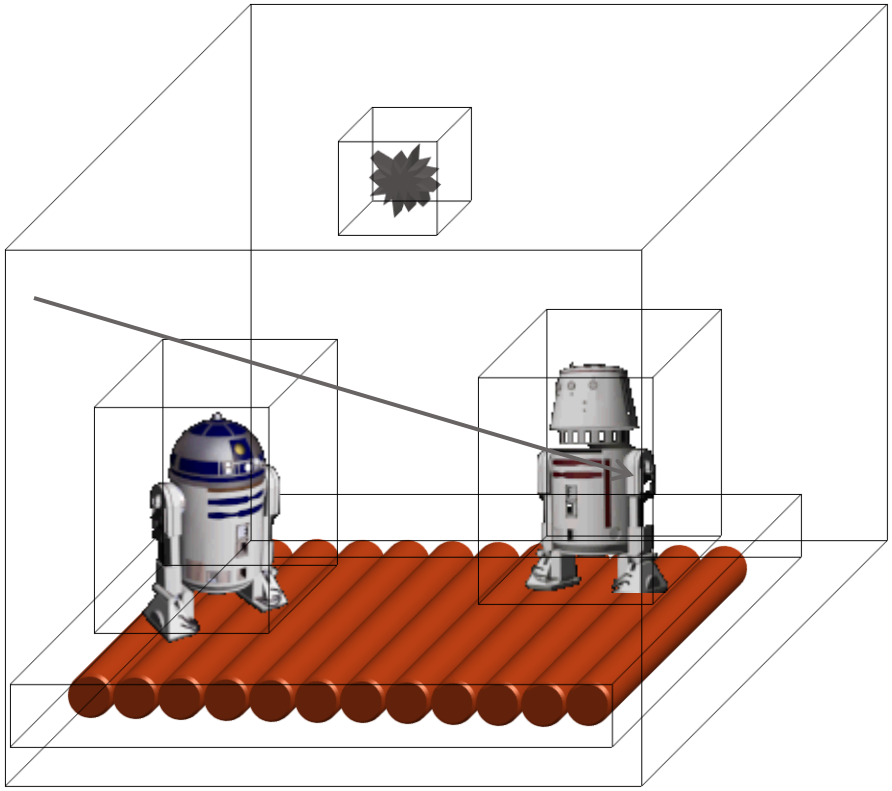
- Construction de la hiérarchie :
  - En général, construction montante

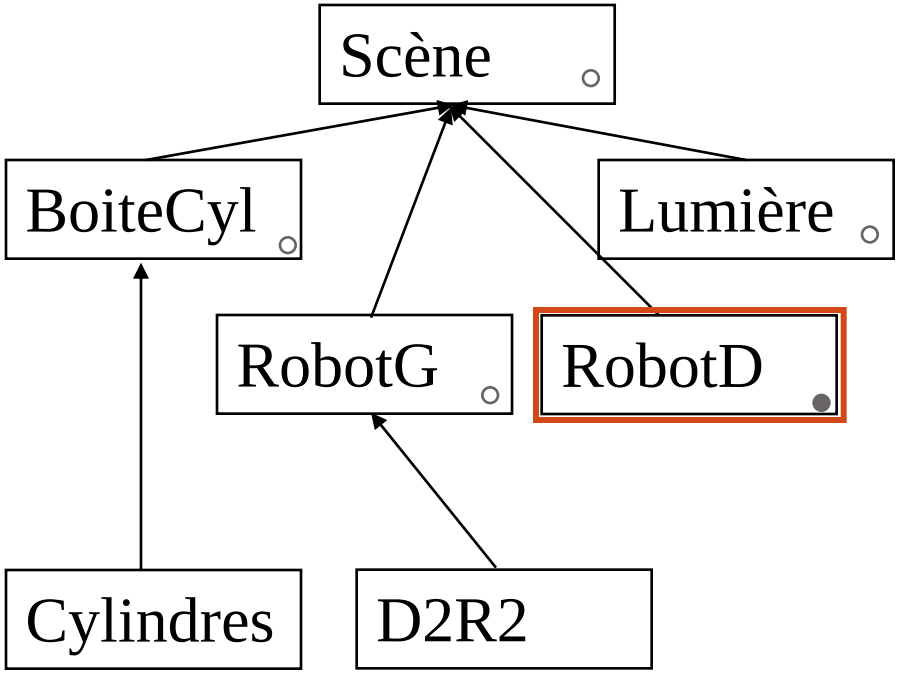




Volumes englobant

- Utilisation de la hiérarchie :
  - En général, en profondeur d'abord



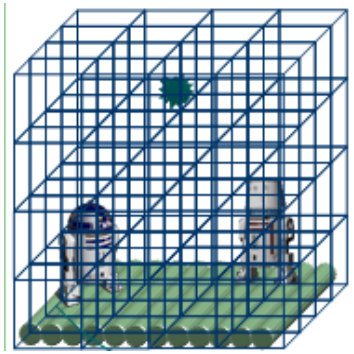


Volumes englobant

- Avantages :
  - Simplicité de mise en œuvre
  - Bonne efficacité
- Inconvénients :
  - Précision des volumes englobants
    - Slabs, boite min-max, sphére
  - Difficulté de construction (surtout si scène fermée)

Le nombre de calculs d'intersections

- L'acquisition de connaissances sur la position des objets à l'intérieur de la scène peut éviter la réalisation obligatoire d'une recherche systématique d'intersection.
- Objectif :
  - Diminuer le nombre de calcul d'intersections rayon-objet
  - Accélérer la détection de non-intersection
  - Accélérer la détection d 'une intersection



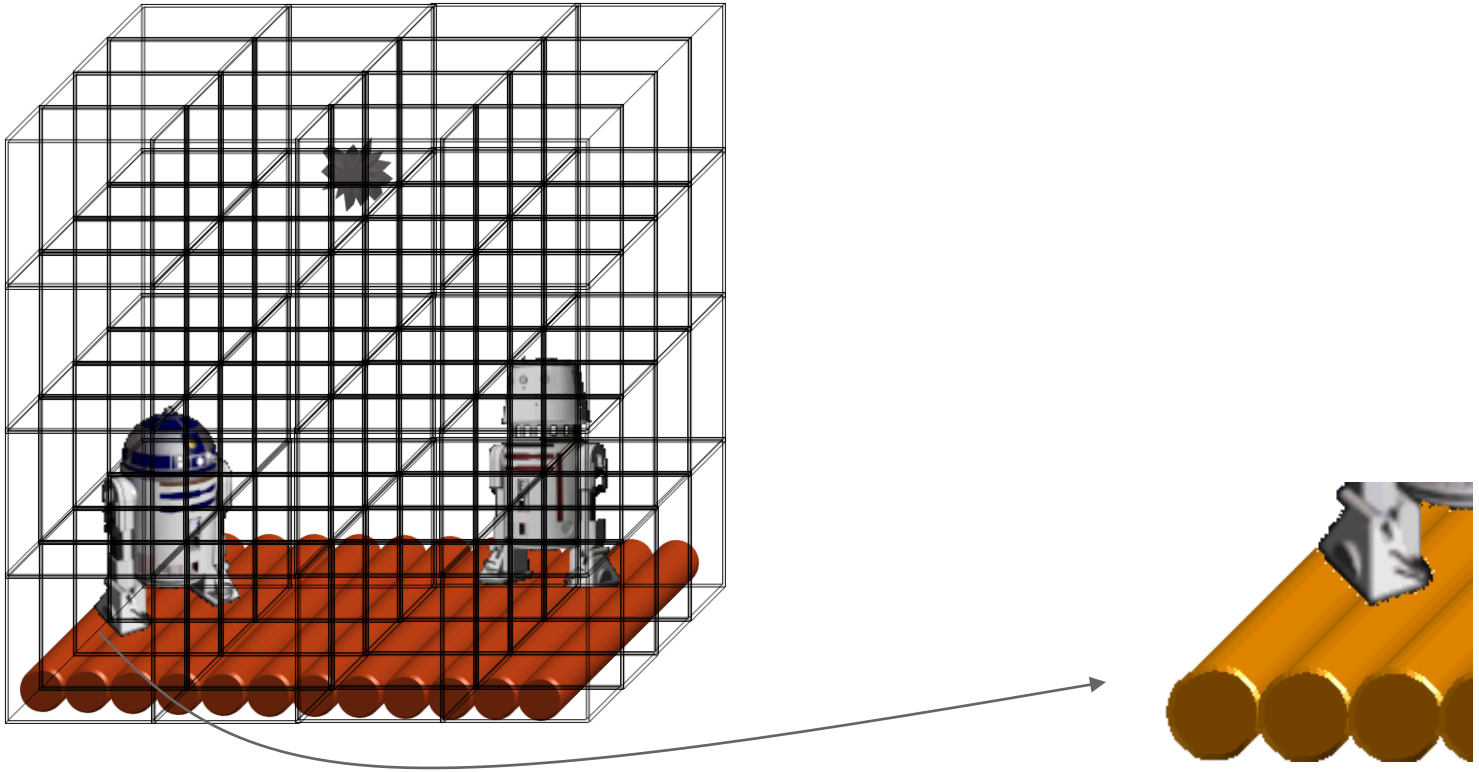
- Principe :
  - Construction d'une grille fixe ou adaptative
    - Découpage de l'espace de la scène en sous espace
    - Pour chaque sous espace (région) on connaît l'ensemble des objet qu'il y contenu.
  - Parcours incrémental de la grille

Subdivision régulière

- Principe :
  - L'espace est subdivisé en une grille de N^3
  - Les rayons sont considérés comme des droites discrètes de N^3
  - Utilisation d'algorithmes de tracé de droites discrètes pour le suivi du rayon

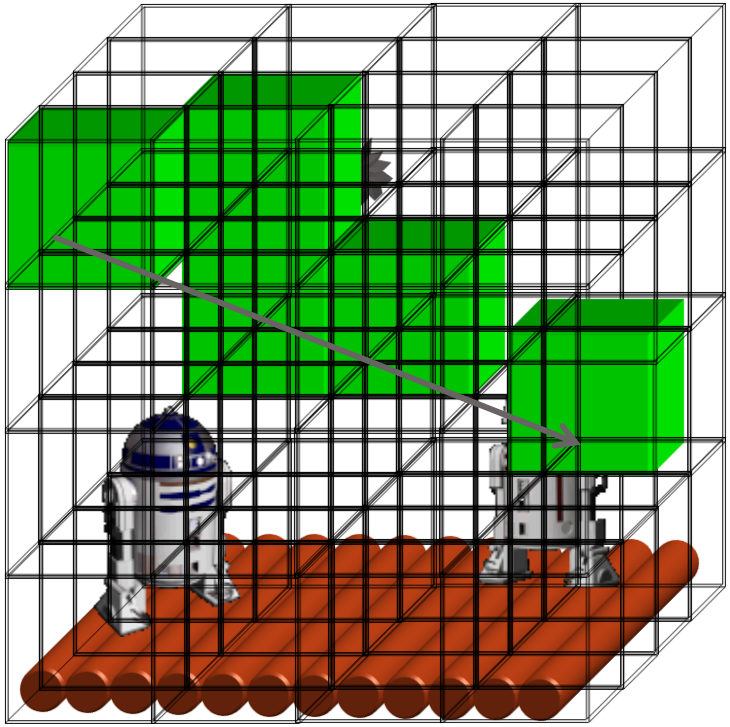
Subdivision régulière

- Découpage de la scène en voxels



Subdivision régulière

- Propagation du rayon



Dès qu'une intersection est trouvée, arrêt de la propagation

Subdivision régulière

- Avantages :
  - Construction facile de la grille
  - Utilisation de calculs entiers pour la propagation des rayons => ASIC ?

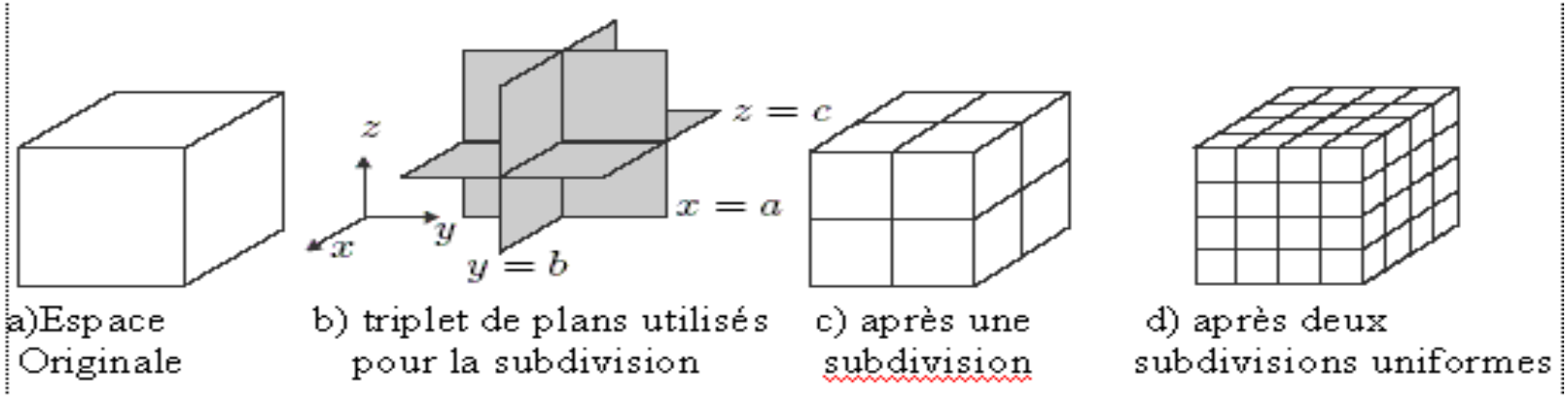
- Inconvénients :
  - Répartition hétérogène des objets dans les voxels
  - Coût de stockage important

Le nombre de calculs d'intersections Subdivision spatiale

- Avantages:
  - Les intersections ne soient testées qu'avec les objets situés à proximité du trajet du rayon.
  - la recherche d'une intersection est arrêter dès la première région contenant une intersection
  - Effectuer un lancer de rayons en temps constant, presque indépendamment de la complexité de la scène .
- Inconvénient :
  - Difficulté de l'implémentation par rapport la méthode de volume englobant

Le nombre de calculs d'intersections Subdivision spatiale

- Arbre BSP :
  - Arbre qui représente une subdivision d'espace et dans lequel chaque nœud a 8 fils.
  - La subdivision est fait à l'aide de triplets de plans généralement orthogonaux.



a)Espace Originale | b) triplet de plans utilisés pour la subdivision | c) après une subdivision | d) après deux subdivisions uniformes

Le nombre de rayons tracés

- Une solution simple pour diminuer le nombre de rayons primaires consiste à
  - effectuer préalablement au lancer de rayon une projection des objets de la scène sur l'écran de visualisation.
- On connaîtra ainsi quels sont les pixels en lesquels aucun objet n'apparaît.
- Le problème de la diminution du nombre de rayons secondaires est complexe.

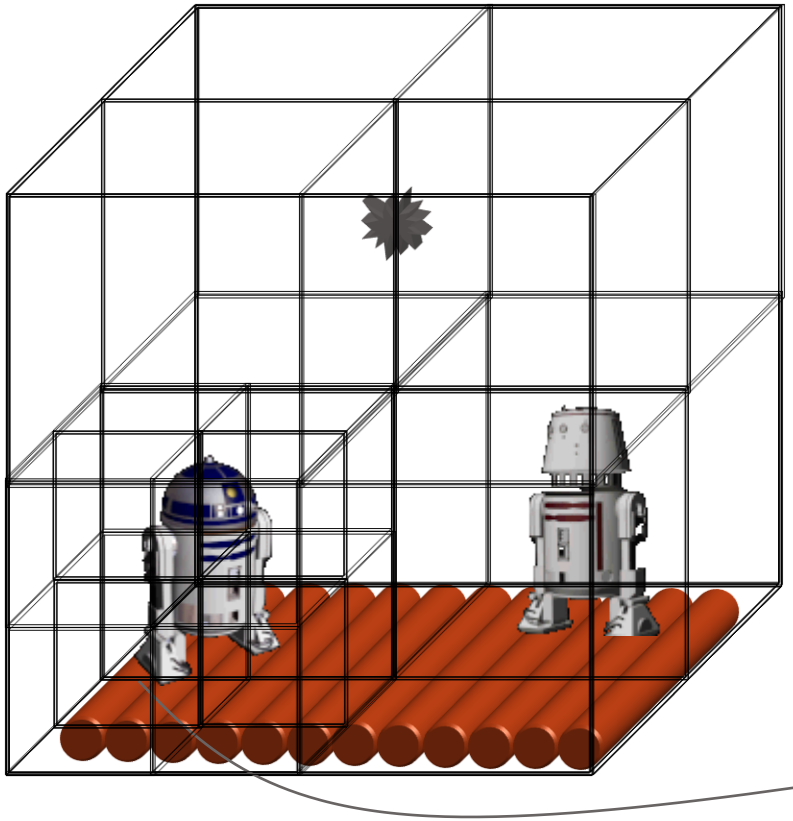
Octree

- Objectif :
  - Diminuer le coût de propagation d ’un rayon dans un espace discret.

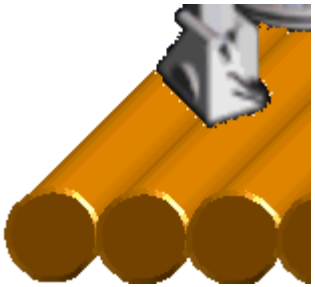
- Principe :
  - Subdivision irrégulière de la scène.
  - Construction descendante des voxels.
  - Organisation en arbre octal indexé

Octree

- Construction descendante



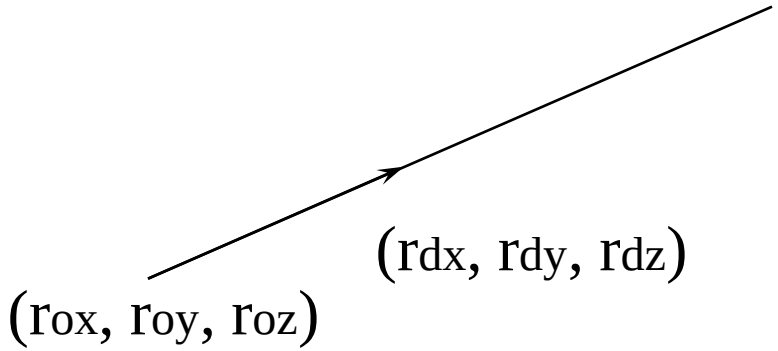
Subdivision du Boite englobante



Octree

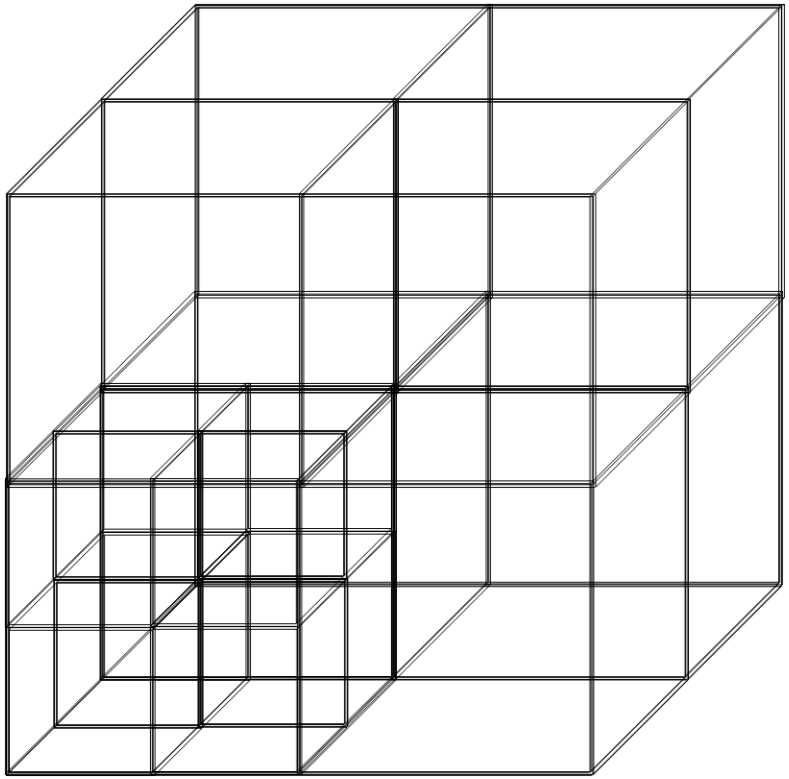
- Représentation du rayon
  - Demi-droite paramétrique orientée

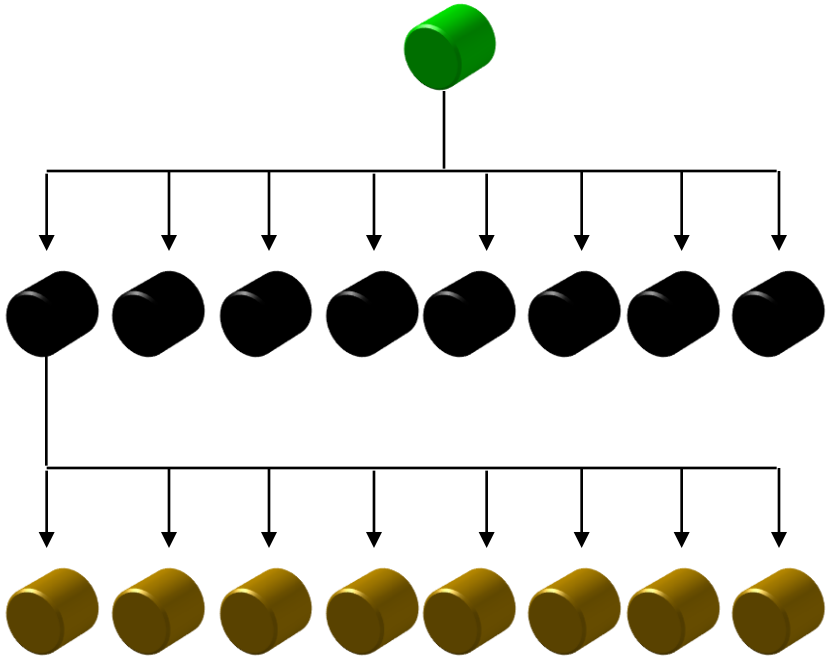
$$x(t) = r_{ox} + t * r_{dx}$$
$$y(t) = r_{oy} + t * r_{dy}$$
$$z(t) = r_{oz} + t * r_{dz}$$



Octree

- Représentation et stockage de l'arbre





Octree

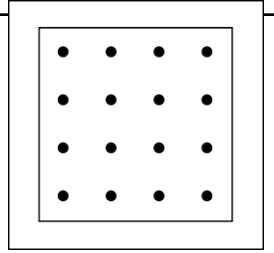
- Avantages :
  - Meilleure adaptation de la subdivision à la scène.
  - Détermination efficace de la suite de voxels.

- Inconvénient :
  - Efficacité inversement proportionnelle à la profondeur.
  - Temps d'initialisation.

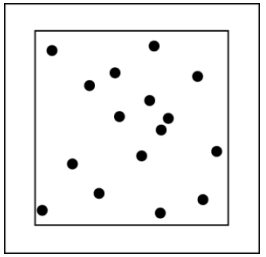
Lancer de rayons distribués

- Anti-aliassage
  - échantillonnage régulier
  - échantillonnage stratifié (*jitter*)
- Pénombres
  - échantillonner la lumière
  - échantillonner l'angle solide formé par la lumière
- Profondeur de champ
  - échantillonner les lentilles (simulées) d'une caméra
- Réflexion *glossy*
  - échantillonner le cone de réflexion
- Flou de mouvement
  - échantillonner (le mouvement dans) le temps

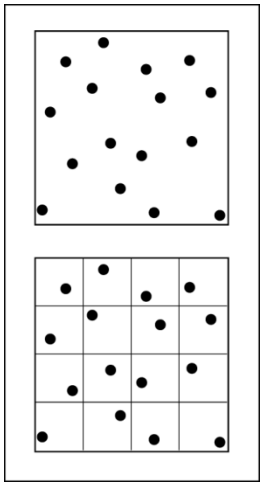
Lancer de rayons distribués



- Anti-aliassage
  - échantillonnage régulier
  - échantillonnage stratifié (*jitter*)

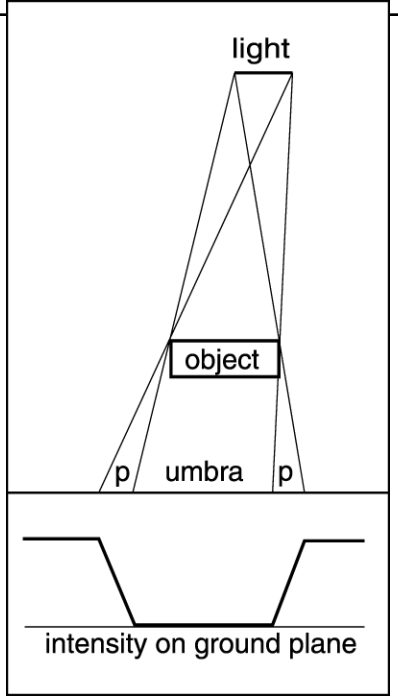


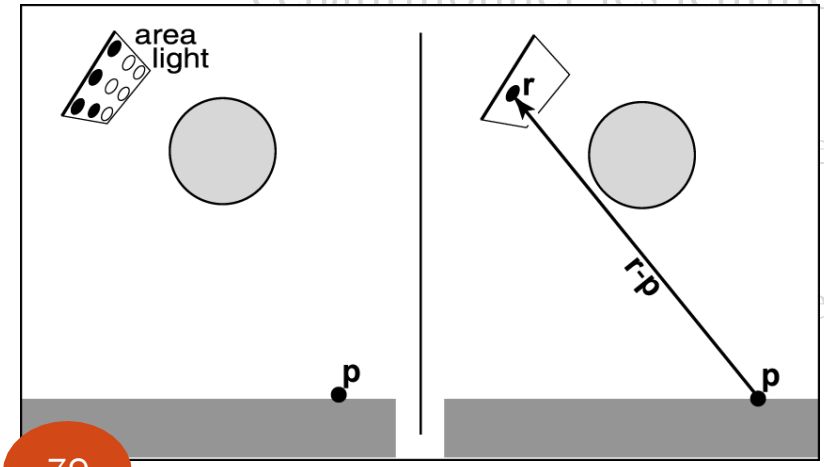
- Pénombres
  - échantillonner la lumière
  - échantillonner l'angle solide formé par la lumière
- Profondeur de champ
  - échantillonner les lentilles (simulées) d'une caméra
- Réflexion *glossy*
  - échantillonner le cone de réflexion
- Flou de mouvement
  - échantillonner (le mouvement dans) le temps

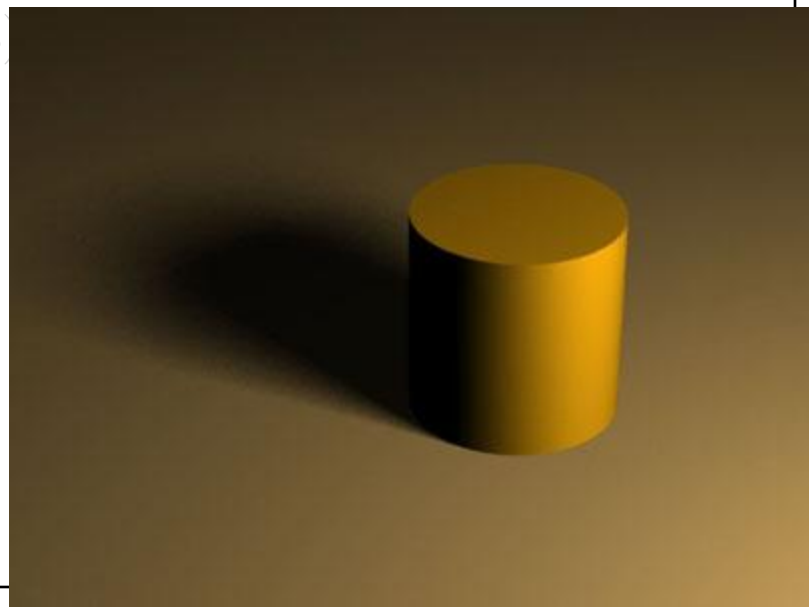


Lancer de rayons distribués

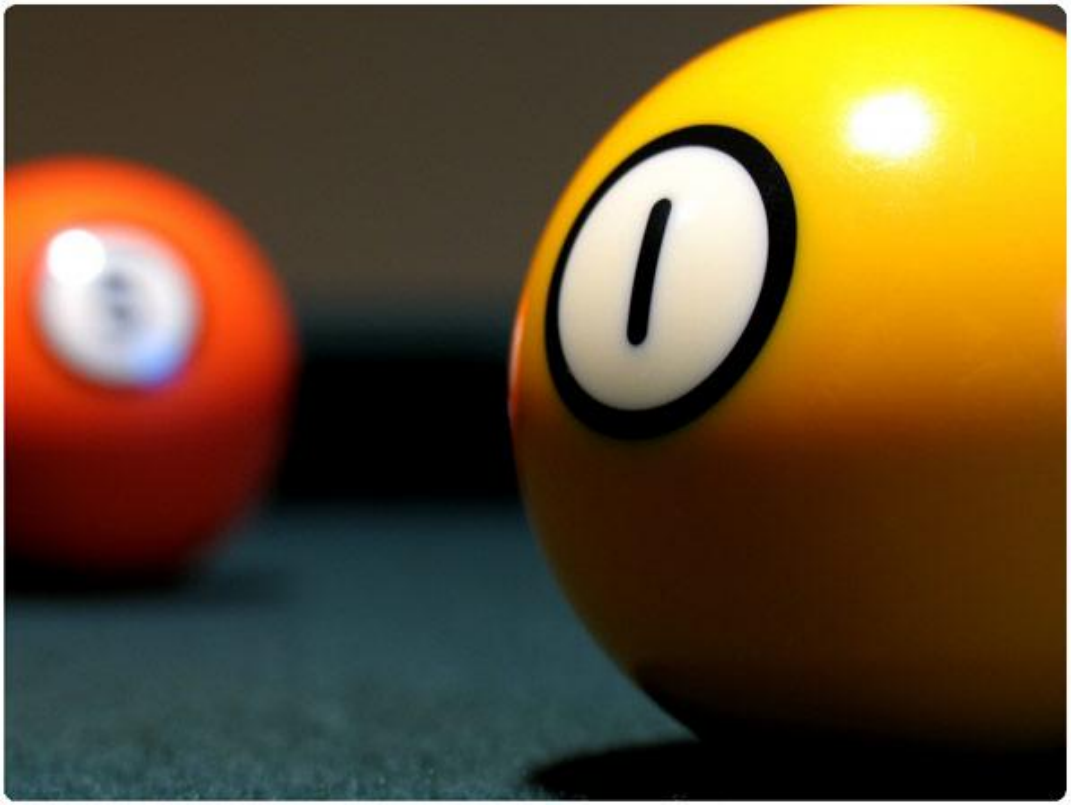
- Anti-aliassage
  - échantillonnage régulier
  - échantillonnage stratifié (*jitter*)
- Pénombres
  - échantillonner la lumière
  - échantillonner l'angle solide formé par la lumière
- Profondeur de champ
  - échantillonner les lentilles (simulées)



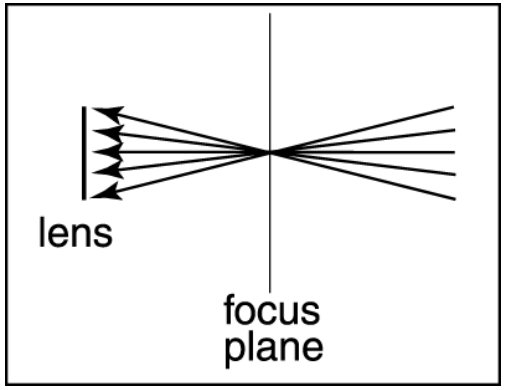


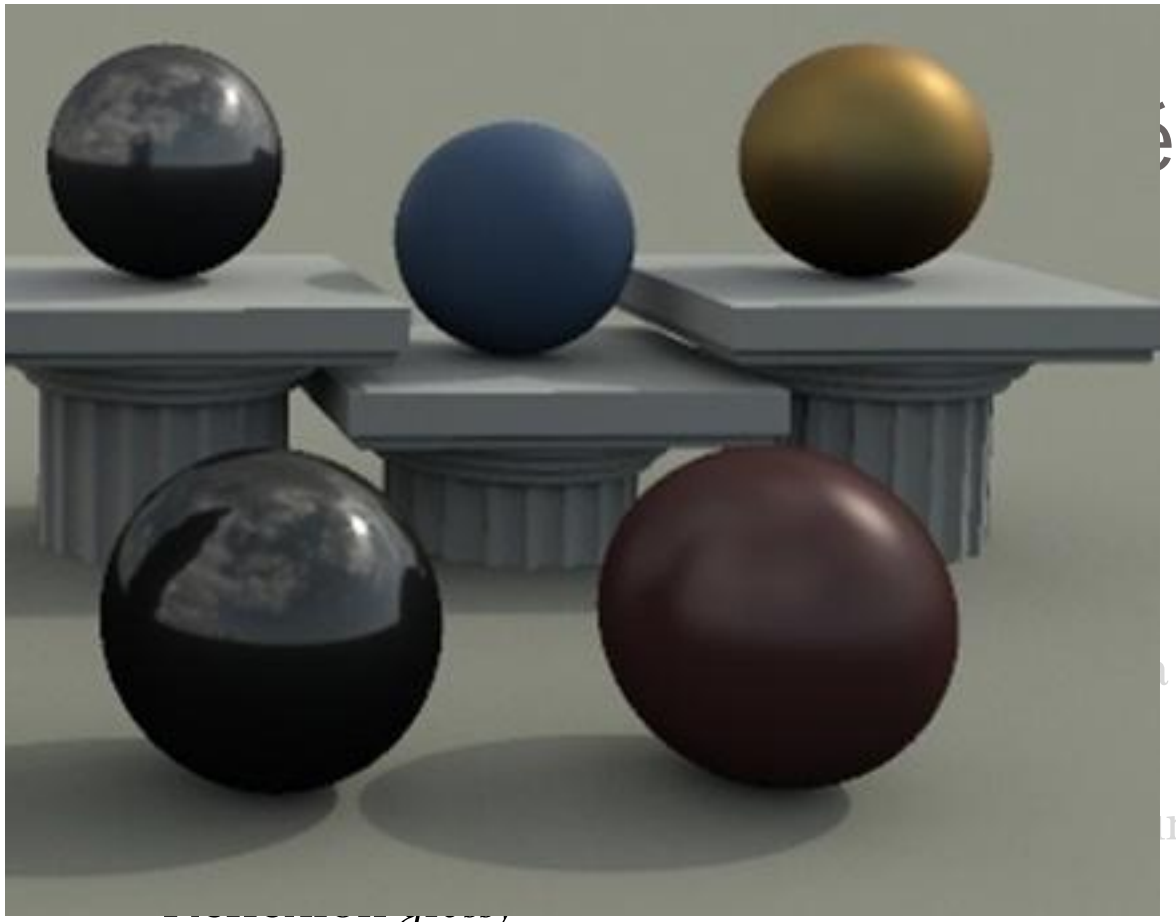


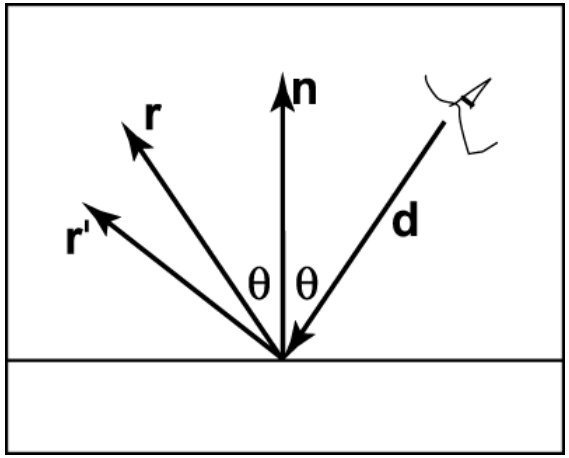
ibués



- Profondeur de champ
  - échantillonner les lentilles (simulées) d'une c
- Réflexion *glossy*
  - échantillonner le cone de réflexion
- Flou de mouvement
  - échantillonner (le mouvement dans) le temps



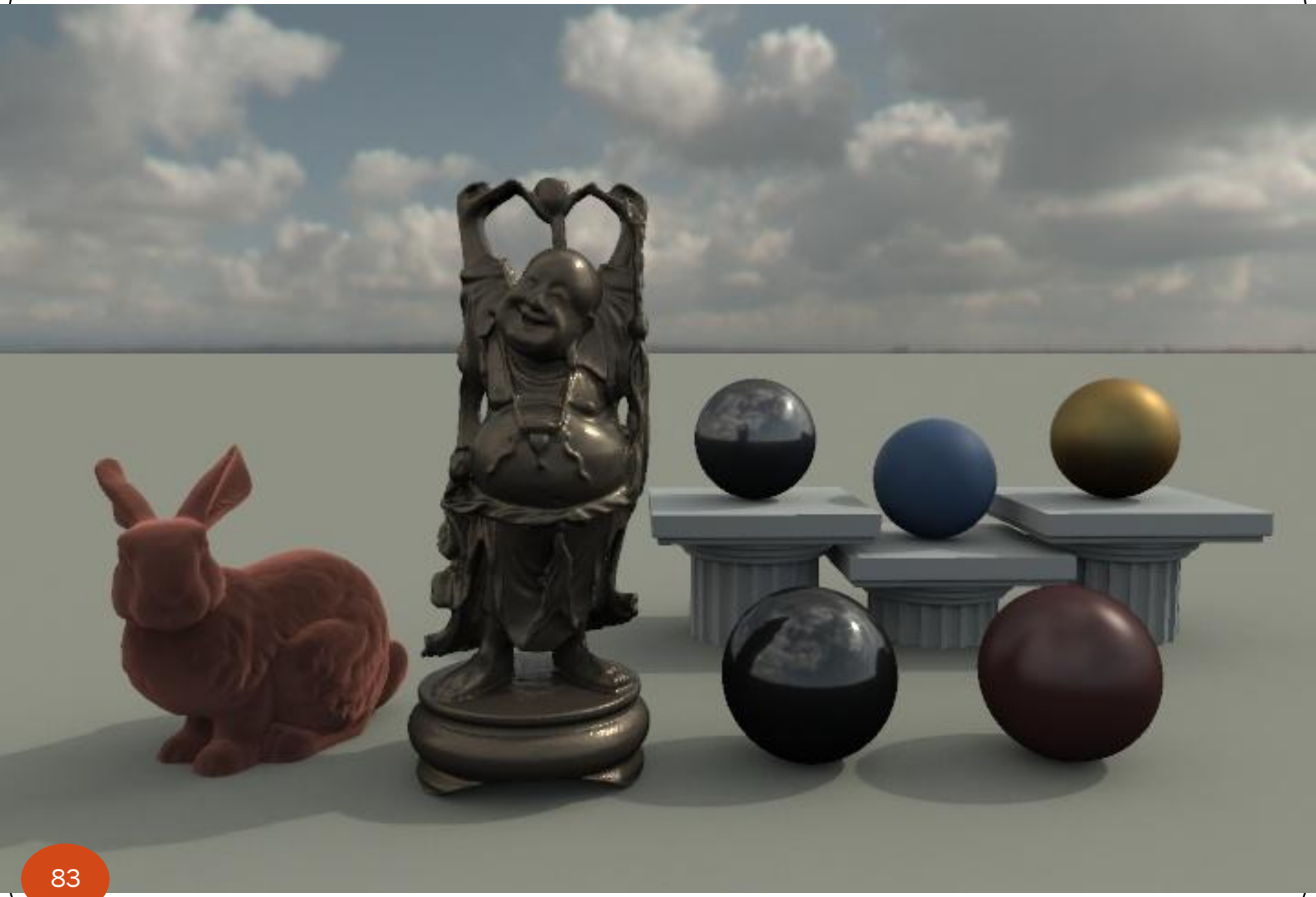




- Flou de mouvement
  - échantillonner (le mouvement dans) le temps

ués

- échantillonner les lentilles (simulées) d'un
- Réflexion *glossy*
  - échantillonner le cone de réflexion
- Flou de mouvement
  - échantillonner (le mouvement dans) le temps



Weathering