

# Introduction à la synthèse d'images

Pres

## Introduction

### Séance de projet:

Peut être libre -> voir si le projet propose convient.

Si raytracer en TP:

- Ex: partir de ratracer et changer de technique de rendu ou changer aspect modélisation.
- Ex: modélisation d'arbre

2 choses

- Aspect technique
- Aspect artistique

Réfléchir à un sujet qui nous plait

On peut faire du path tracing, sur les meshs...

Modélisation d'arbre / végétaux.

Ne pas hésiter à aller voir Fabrizio

S'y prendre tôt, parce que après il y aura beaucoup de projets.

Langage libre, mais tp en c++.

Aspect performance importante, et surtout avoir un beau rendu.

Le projet :

- En **binome** ;
- Une soutenance ;
- Un rapport (quelques pages 24h ou 48h avant la soutenance).
- Une vidéo de 1/2 minutes. Ca peut être une vidéo animé ou autre.

On peut trouver des idées de projet avec des articles associés.

## Partie 1 : Rendu temps réel vs rendu photo réaliste

La synthèse d'image est le fait de modeliser une forme , apparence du monde réel ou d'objets imaginaires avec un ordi.

Pour faire un rendu il faut connaître l'optique pour rendre un côté réaliste et s'y connaître en

physique

dans la réalité virtuelle, il n'y a pas que l'info visuelle  
mais le cours porte surtout la dessus.

### **Photo réaliste:**

Prendre la photo la plus belle possible : assez difficile -> temps importe peu mais qualité avant tout

### **Rendu temps réel:**

Le temps est hyper important, et il faudra avec ce temps cours, la plus belle image possible,  
on va tricher pour faire un rendu qui sera joli.

Un jour les techniques de rendu temps réel vont disparaître, quand le matériel le permettra,  
Nvidia est en train de le faire.

Des algo photoréaliste sont en train de passer en temps réel.

Photoréaliste image la plus jolie, ce qui correspond une photo, photo = rendu  
Prend du temps, la qualité soit le plus vrai possible

Temps réel belle image mais ce qui compte le plus le temps, ex jeu vidéo  
Image peut être très belle mais pas forcément photoréaliste  
Tricher pour faire un rendu plus rapide. Les techniques de rendu temps réel vont être jetées à la poubelle car les outils permettent de faire les calculs

## **Partie 2 :**

Les domaines de l'image:

- **Géométrie:** Algorithmique discrète
  - Euclidienne
  - Projective

On peut paralléliser en bas niveau.

On va avoir besoin de bonne connaissance physique à un moment.

Il faut respecter les habitudes du cerveau.

Heureusement beaucoup de logiciels de modélisation le font très bien. Pour mes objets, j'ai besoin d'une équation mathématique (ex: faire un maillage)

La frontière intérieure / extérieure d'un objet importante.

Usuellement on prend des triangles :

- Éviter erreur d'arrondi
- Avantage les 3 points sont toujours coplanaires

On peut augmenter / diminuer résolution mesh. Mesh estimation: réduire le mesh sans trop

diminuer la qualité, ne pas les supprimer partout, choisir les zones plat, lorsqu'un objet est trop loin, on enlève des meshs par exemple

- **Multimédia:** Image son vidéo
- **Physique:**
- **Informatique**
- **Rendu**

ex: rendu de bande dessinée

(Si intéressé pour faire rendu spécifique, aller voir le prof pour en parler pour voir si le sujet, n'est pas trop compliqué ni trop simple.)

Domaine lié programmation GPU.

**Image HDR** *high dynamic range*, essaye de prendre la même scène avec != expositions, capable de repérer les sombres claires et sombres

- **Animation**

La modélisation représente et traite la forme des objets 3D.

On prend souvent des triangles en mesh car c'est coplanaire et permet d'éviter les erreurs d'arondis.

Exemple de projet:

Essayer de supprimer des meshs d'un objet 3D compliqué sans trop détériorer le modèle.

Il y a des domaines liés :

- Programmation GPU
- Traitement d'images (image Hight Dynamic Range HDR)

## Historique

Base : invention de l'écran CRT

Dans les années 60 tout a commencé avec algo(en temps réel) de *Sutherland*.

Dans les années mis 70, on a des images générées sur ordinateurs pour les films.

Début des années 80 on a de la synthèse sur les films.

Les premiers rendus réel, sont des rendu *fil de fer*.

La réalité virtuelle permet de faire des simulateurs, mais ce ne sont pas les premiers.

Les simulateurs avant realite virtuelle marchaient avec des modèles miniature.

- **Domaine d'assistance chirurgicale**

- éviter les faux mouvement.

- la réalité augmentée va permettre d'aider le chirurgien
- Les anciens chirurgiens étaient assez réticents, mais cela a changé. Nous en sommes cependant encore au balbutiement.

- Domaine de l'architecture:

- écoulement des fluides
- force des vents appliqués aux bâtiments.

**CAO** (confection assistée par ordinateur)

- Modélisation des pièces
- Imprimante 3D font couche par couche, dans quel sens il faut mettre les pièces.

On a les premiers jeux et films 3D dans les années 80.

**Films:**

- TRON (1982)
- The last star fighter (1984)
- Terminator 2 (1991)
- Jurassic Park (1993)
- Toy story (1995) (Premier long métrage)
- Mille et une patte (1998)
- Monstre et Co (2001) La fourrure du monstre est cool
- Age de glace (2001)
- Le monde de Nemo (2003)
- Monster Academy (2013) (Full raytracing)

**Jeux Vidéo:**

- Star wars 83-89
- Alone in the dark Resident Evil
- Wolfenstein 3D 96
- Myst (1993) / Riven (1997) / Exil (2002)
- Descent (1995) / Terminal velocity (1995) / Flight Gear Simulator

## Applications

- Réalité augmentée
- Cinéma
- JV
- Simulation (civil, militaire)
- Assistance chirurgicale
  - Il y a beaucoup de travaux pour faire des opérations assister par ordinateur. Mais il a

besoin de la vision.

- Architecture
- Conception Assisté par Ordinateur (CAO)
- ...

Il existait des simulateurs avant la VR. Mais ça aide au réalisme...

## Communauté scientifique

- ACM SIGGRAPH (surveiller leurs articles)
- Synthèse image à jour
- Blender rendu réaliste (compromis entre complexité et professionnalité)
- OpenGL Direct3D bas niveau
- Vulkan risque de remplacer OpenGL.
- POVRAY: intéressant pour regarder les algorithmes, (open source)(moteur de rendu)

*Second depth shadow mapping* (algo utilisé dans monstre & compagnie)

## Outils

- Modéleur
- Outils d'animation
- Moteurs de rendu
- Rendu réaliste
  - POVRAY
    - Code libre en C++
  - Blender
  - Maya
  - ...
- Rendu temps réel (API bas niveau)
  - OpenGL / Vulcan
  - Direct 3D
  - Java 3D
  - ...
- Moteurs (API haut niveau)
  - Unity
  - Ogre

## Rappels d'optique et de géométrie euclidienne

Pour modéliser une scène et faire son rendu, on a une source lumineuse primaire, les objets vont recevoir et absorber cette lumière, ils renvoient la lumière ce sont des sources lumineuses secondaires.

Couleurs primaires:

- **Additive:** Rouge / Vert / Bleu
- **Soustractive:** Cyan / Magenta / Jaune

L'objet renvoie la lumière en fonction de sa texture / couleur.

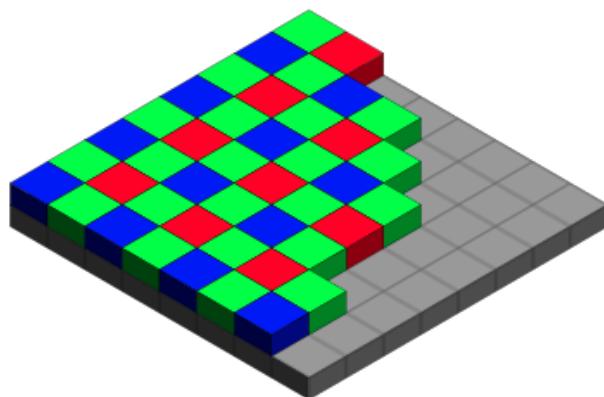
Les objets filtrent la lumière sur la lumière qu'elle diffuse. Cependant, le rayon réfléchi lui n'est pas filtré dans la plus part des cas.

Distance focale permet le zoom (plus on l'augmente plus on zoom), mais réduit l'ouverture de champs, si on augmente l'ouverture de champs, les objets deviennent plus petits.

On a une zone, avant et après cette zone, c'est flou.

Comment l'image est capturée ?

Le capteur est une matrice, chaque fois qu'un rayon rencontre un pixel, il y a une charge électrique, à chaque tic d'horloge on compte le nombre de charge pour savoir si la région est bien lumineuse, mais on n'a pas les longueurs d'ondes, utile pour les images noirs et blancs.



## Rappels de mathématique

### Géométrie euclidienne

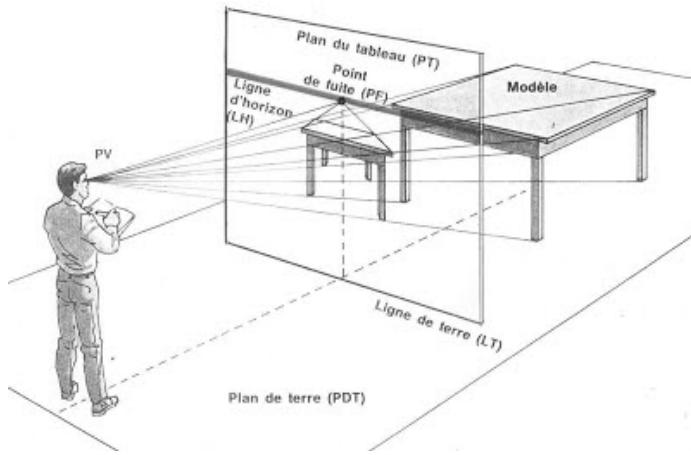
- Produit scalaire : forme bilinéaire, symétrique définie positive
- Espace pré-hilbertien  $(E, |\cdot|)$  réel
- Produit mixte utilisé pour avoir les angles
- Produit vectoriel
- Équation de droites

- Intersection de deux plans
- Équation paramétrique (un point et un vecteur)
- Équation d'un plan
  - Si on a l'équation Cartésienne on peut obtenir l'équation implicite
  - Paramétrique (un point deux vecteurs)
- Équation d'un cercle / d'une sphère
  - Équation (<https://ggbm.at/y3MJCadK>)
  - Cartésienne :  $(x - x_0)^2 + (y - y_0)^2 = r^2$
  - Paramétrique :  $R. \cos(\theta) + R. \sin(\theta)$
- Intersection droite / plan
  - Calcul de l'intersection dans le **repère** local ou global ?
- Intersection droite / sphere
  - Pas forcément utile car on peut toujours discréteriser la sphère.
- Intersection droite / triangle
  - On fait l'intersection avec le plan
  - On vérifie que le point d'intersection est dans le triangle
  - Déterminer les équations de chaque côté du triangle
    - déterminer la position de l'intersection vis à vis de chaque côté
    - ...

Pour plus d'information voir les slides. ICI (<http://jo.fabrizio.free.fr/teaching/synt/>)

## Geometrie projective

- Étude des objets tel qu'ils sont vus.
- Perception des angles, des distances, du parallélisme distordu.
- Un ensemble de droites parallèles convergent vers un point de fuite sur le plan image.
- Projection sur le plan image
- L'horizon c'est le plan qui passe par le foyer et qui s'intersecte avec le plan image
- Le point de fuite est sur l'horizon
- Les points qui sont sur l'horizon n'ont pas d'antécédant dans la géométrie euclidienne



Dans le plan :

- $RP^2$  est l'ensemble des triplets  $[p] = [p_1, p_2, p_3]$  avec  $(p_1, p_2, p_3)$  dans  $\mathbb{R}^3$  privé de  $(0, 0, 0)$
- Deux points p et q sont égaux ssi il existe un  $k$  dans  $\mathbb{R}^*$  tel que:
  - $p_1 = kq_1$  et  $p_2 = kq_2$  et  $p_3 = kq_3$

Finalement il y a deux cas :

- $p_3 = 0 [p_1, p_2, p_3] = [p_1, p_2, 0] \in RP^2$
- $p_3 \neq 0 [p_1, p_2, p_3] = [p_1/p_3, p_2/p_3, 1] \in RP^2$

Représentation des transformations usuelles dans l'espace projectif :

- Translation :

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

- Echelle :

$$\begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} dx & 0 \\ 0 & dy \end{pmatrix}$$

- Rotation

$$\begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

- Projection :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}^t$$

Exemple de combinaison **projection et translation et rotation et scale** :

$$\underbrace{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}_t \underbrace{\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}}_t \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_t \underbrace{\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_t$$

Ca nous donne une matrice que l'on peut multiplier pour n'importe quel point de mon plan.

L'ordre des transformation est important. Ce n'est pas commutatif.

### Quaternion

You don't want to click me (<https://fr.wikipedia.org/wiki/Quaternion>)

/!\ DANGER ZONE /!\ You might feel headache after reading...

## Algorithme photorealiste

- Objectif:
  - Generation d'image realiste
  - Contrainte de temps faible
- Strategies:
  - Object based rendering algorithm
    - Illumination globale calculee independemment du point de vue
  - Image-based rendering algo
  - Deterministic rendering algo
  - Monte carlo rendering algo
- Types
  - Raytracing
  - path tracing
  - PBGI (Point based global Illumination)
  - radiosity
  - photo map

### Raytracer

On part de la camera à travers le plan jusqu'à l'objet, et voir l'angle d'incidence des diffetentes sources lumineuses.

Il faut déterminer la partie **diffuse** et la partie **spéculaire** de la source de lumière.

- Calcul de l'illumination locale :
  - Composante diffuse
  - Composante spéculaire
  - Apport des sources primaires
  - Apport des sources secondaires
- Source primaires :
  - Lumières ponctuelles
  - Spots
  - Lumies directionnelles
  - Objets lumineux



- Modèle local :
- La composante diffuse :  $I_d = \underbrace{K_d \times C}_{\text{Propriété du matériaux}} \times (N \cdot L) \times I_{Li}$
- La composante spéculaire :  $I_s = K_s \times I_{Li} \times (S \cdot L)^{ns}$

L'intensité totale est la somme de la composante spéculaire et diffuse.

Il faut ensuite améliorer cette algo car on ne prend pas en compte les obstacles (ombres). On ne tient pas compte des sources secondaires.

### Sources secondaires :

Pour gérer les sources secondaires on peut utiliser le rayon réfléchi.

Celui-ci peut intersecter un autre objet qui va être source de lumière secondaire.

On peut continuer récursivement.  
Soit on fixe un niveau de récursion. Soit on fixe un palier d'intensité sous lequel on arrête la récursion.

### Gestion de sombres :

Il faut envoyer un nouveau rayon vers les sources lumineuses pour vérifier qu'aucun objet est entre le point et la source de lumière.

## Recap

- Avantages
  - L'algorithme est simple
  - Génère des images honorables
- Inconvénients
  - Temps de calcul lent
  - Aliasing élevé
  - Ombre trop brute

## Problème de l'*aliasing*

Solutions :

- Sur-échantillonnage
  - lancer plusieurs rayons pour chaque pixel
    - De manière organisée
    - Au hasard
  - Lancer plusieurs rayons pour chaque pixel où le gradient est élevé
    - Bon résultats mais peut être très lent
- Post filtrage
  - Résultat moyen mais très rapide

## Problème du temps de calcul

Solution :

- Partitionner l'espace pour trier les objets selon leur position
- Utiliser des volumes englobants
- Pre-trier les objets
- Calcul parallèle
- Utilisation d'OpenGL
- ...

## Problème des objets transparents

Solution :

- Utiliser la loi de la réfraction :  $n_1 \sin(i_1) = n_2 \sin(i_2)$ 
  - Il faut connaître en tout point l'indice de réfraction
- Distribution probabiliste

Voir Cours ([http://jo.fabrizio.free.fr/teaching/synt/isim\\_rendu\\_photorealiste.pdf](http://jo.fabrizio.free.fr/teaching/synt/isim_rendu_photorealiste.pdf)) pour schema + explication

L'autre problème est de calculer l'ombre des objets transparents.

Une solution simple est de diminuer l'intensité et de filtre la longueur d'onde sans calculer la déviation.

Ca donne un résultat faux d'un point de vue réalisme.

### Problème de l'éclairage indirect

Dans les zones d'ombre il ne fait pas complètement noir. Or c'est le cas sur notre rendu. De même les ombres sont délimitées très clairement.

Une solution lazy est de rajouter une heuristic de lumière ambiante.

$$I = k_a \times I_a + I_d + I_s + I_r + I_t$$

On peut considérer les sources lumineuses comme des zones au lieu d'un point.

Démonstration des pré-ombres

Une solution pour gérer l'ombre est de lancer plus de rayons.

### Problème de profondeur de champs

Notre image est nette même pour les objets qui sont loin.

Une solution simple est de considérer que le foyer est une zone et pas un point. Cela permet d'imiter le diaphragme de l'oeil.

### Bilan

- Avantages
  - Algorithme simple
  - Donne des images honorables
- Problèmes
  - Les sources secondaires ne sont pas suffisamment bien gérées
  - Les objets transparents non plus

## La radiosité

C'est la partie émise par l'objet  $i$ . Pour pouvoir dire la quantité d'énergie reçue par l'objet, on va se servir des sources lumineuses primaires et secondaires.

- $B_i$  la **radiosité** de la surface  $i$
- $E_i$  la quantité de **lumière émise** par la surface  $i$  (source primaire)
- $P_i$  la fraction de **lumière incidente** qui est réfléchie par la surface  $i$
- $F_{ji}$  la fraction de **lumière quittant** la surface  $j$  et atteignant la surface  $i$

$$B_i = E_i + P_i \sum_j (F_{ji} B_j)$$

Ceci **n'est pas** une technique pour obtenir une image mais juste la *radiosité* d'une scène.  
Il faut donc la **combiner** avec une technique de rendu (Ex: Raytracing 😊)

simplement l'illumination globale

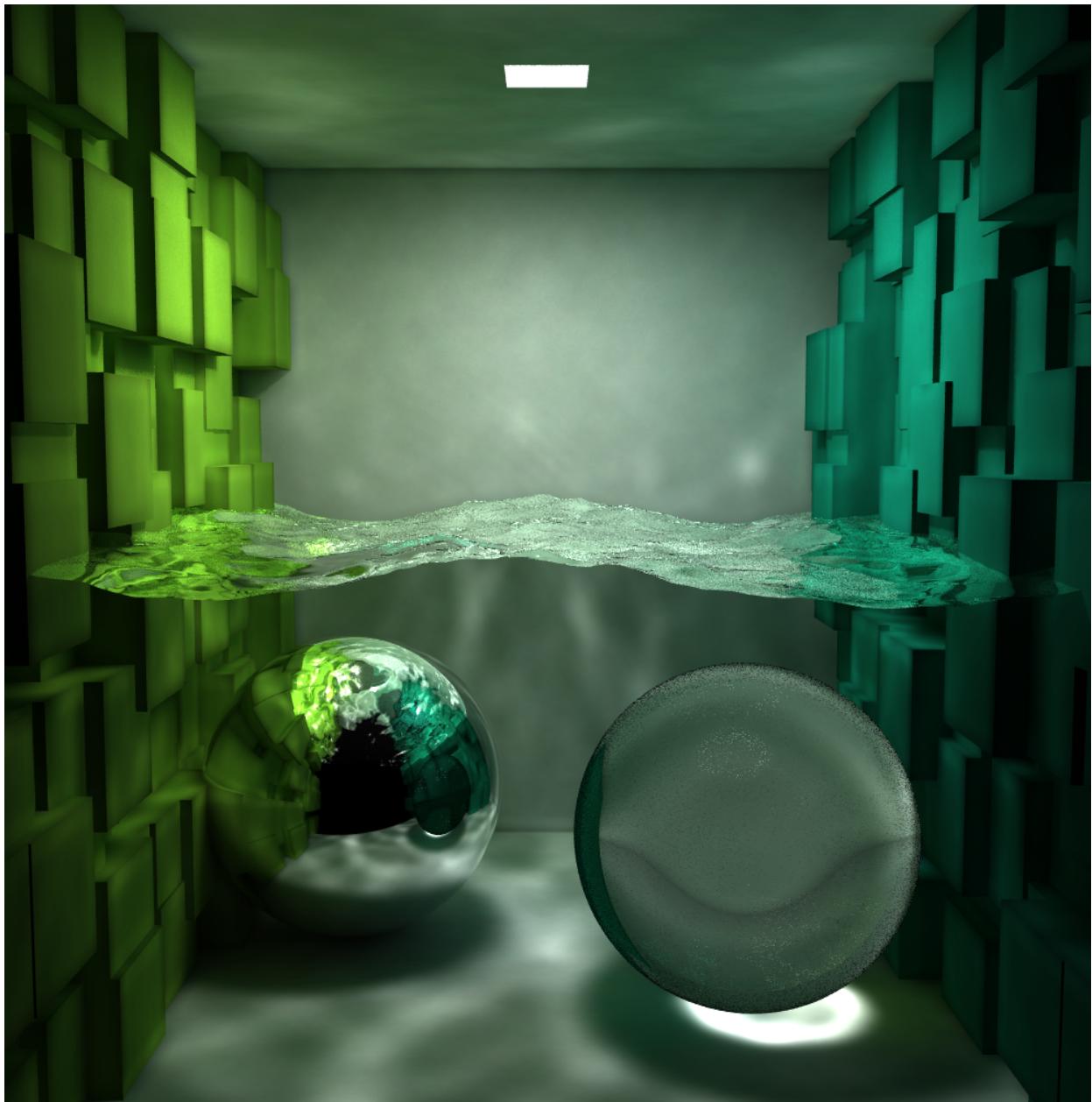
- Avantages :
  - Prend mieux en compte les sources secondaires
  - Calculée une fois pour toutes
- Inconvénients
  - Tient compte que de la diffusion
  - Assez lourd
  - Obligation d'avoir un maillage (il faut discréteriser les surfaces)
  - Objets transparents ?

## Photon Mapping

- Photon Map
  - Pré calcul de l'illumination de la scène.
  - Lancement de rayons lumineux depuis les sources et calcul des accumulations des photons.
- Avantages
  - Permet de modéliser plus proprement les sources secondaires, les ombres portées (...) et surtout les objets transparents (caustiques).
- Inconvénients
  - Calculs
  - Complexité

Ceci **n'est toujours pas** une technique pour obtenir une image mais juste la *radiosité* d'une scène.

Il faut donc la **combiner** avec une technique de rendu (Ex: Raytracing 😊)



Améliorations :

- Projection maps
- visual importance map (3-pass Technique)
- Shadow photons
- ...

## **Path tracing / Bidirectional Path Tracing**

- Résoudre l'illumination
- Modélisation des propriétés de réflexion des surfaces

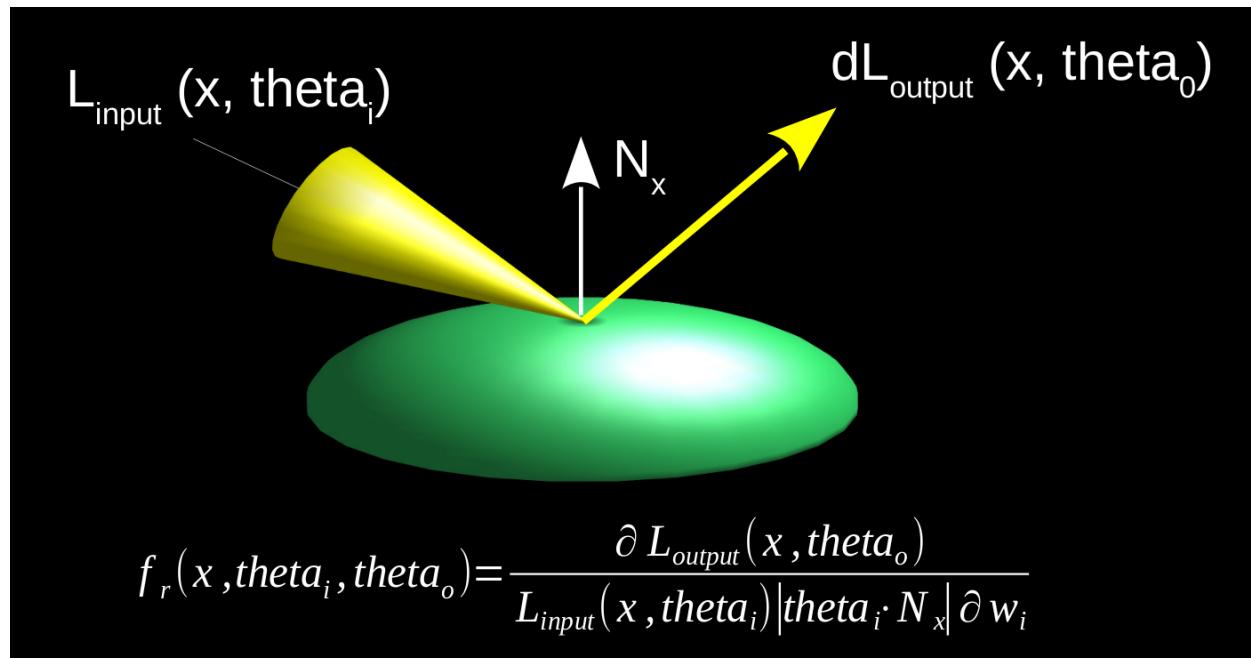
**BRDF** : Bidirectional reflectance distribution function (Réflectivité bidirectionnelle)  
 C'est un algo qui pour un angle donné indique la probabilité qu'un rayon soit renvoyé dans cette direction.

BRDF:

- Conservative
- Réciprocité de Melmholtz
- Positivité

L'idée c'est contrairement au ray tracing ou on relance un rayon en direction des sources lumineuses et de l'angle de réflexion. Ici on relance le rayon de manière aléatoire suivant le **BRDF**.

On espère tomber sur une source lumineuse à un moment. Si ce n'est pas le cas alors notre point est dans l'ombre...



- Avantages :
  - Rendu réaliste
  - Convient bien aux scènes d'extérieurs
  - Prend bien en compte l'apport des autres objets
  - Rend les caustiques
  - Possibilité de modéliser les effets (profondeur de champ...)
- Inconvénients :
  - Lent
  - Bruité (Il faut beaucoup d'itérations pour converger)
  - Difficile pour scènes avec des petites sources lumineuses (ou des sources cachées)

## Bidirectional Path tracing

L'idée c'est de lancer aussi des rayons de la source lumineuse et calculer les interactions entre les deux.

- Amélioration du calcul du rendu.
- Lancement des rayons depuis l'observateur et depuis les sources.

## PBGI: Point-Based Global Illumination

- Méthode pour estimer l'illumination globale
- Beaucoup utilisée pour le cinéma
- Avantages
  - Rapide
  - Image non bruitée (pas d'artefacts temporel)
- Inconvénients
  - Pas aussi précis que leraytracing
  - Difficile de gérer les effets miroir

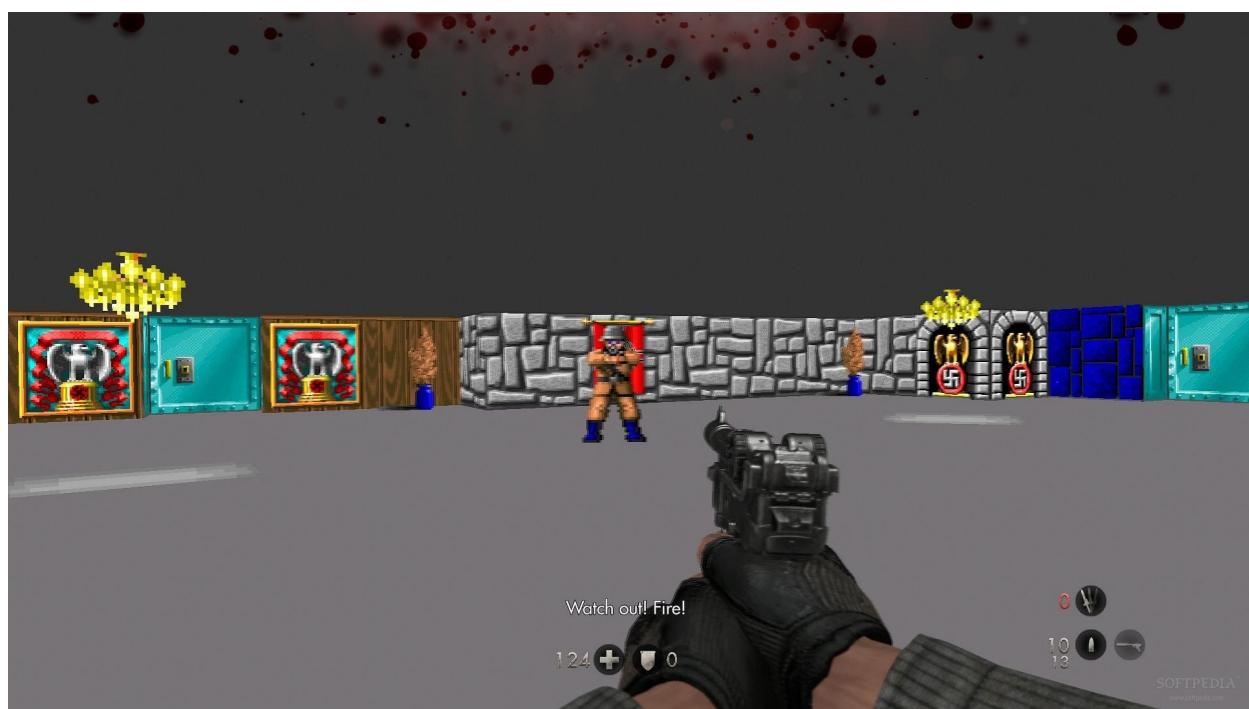
## Raycasting

Principe :

On ne lance que les rayons depuis l'observateur et on ne calcule pas les rebonds...

C'est du faux 3D car on calcule juste la distance avec un rayon puis on affiche un sprite de l'objet.

Ca permet d'avoir du 3D en temps réel sur des machines pauvres.

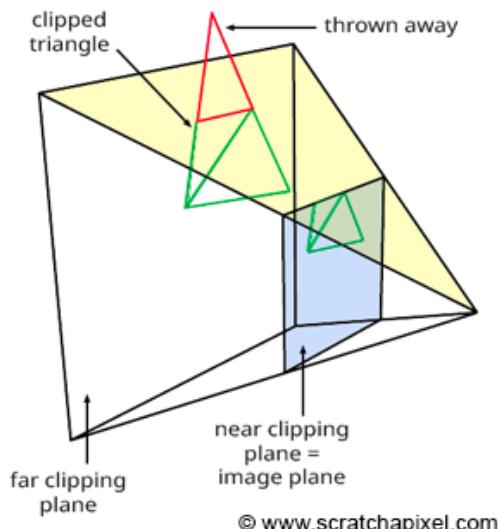


# Rendu temps réel

- Principe général
  - Modélisation des objets dans un repère local
  - Modélisation de la scène dans un repère global
  - Projection de la scène sur le plan image
- Algorithmes 3D fondamentaux
  - Clipping 3D
  - Projection
  - Clipping 2D (Si clipping 3D non faite)
  - Paint
- Algorithmes 2D fondamentaux

## Algorithmes 3D : Clipping 3D

On utilise un cone de vision avec un Z min (le plan image) et un Z max.



© www.scratchapixel.com

## Algorithmes 3D : Backface culling

Il suffit de déterminer l'orientation de la face par rapport à l'axe optique.

Il suffit de faire un produit scalaire entre la normal ede la face et l'axe optique.

## Algorithmes 3D : BSP

Comment déterminer les objets cachés (ou partiellement cachés) ?

- Utilisation d'un arbre **Binary Space Partitionning Tree**
  - Chaque noeud représente un hyperplan (déduit d'une face F)

- Le premier fils contient les faces du demi-espace derrière F et le second fils contient les faces du demi-espace devant F.
- Lorsque l'hyperplan intersecte une face, la face est coupée en deux

## Algorithmes 3D: Z-buffer

Comment déterminer les objets cachés (ou partiellement cachées) ?

- Sauvegarder la profondeur pour chaque pixel dessiné
- Simple à programmer
- Inconvénients:
  - Oblige à projeter l'ensemble des polygones
  - Problème de résolution lors de l'encodage du Z

## Modélisation

---

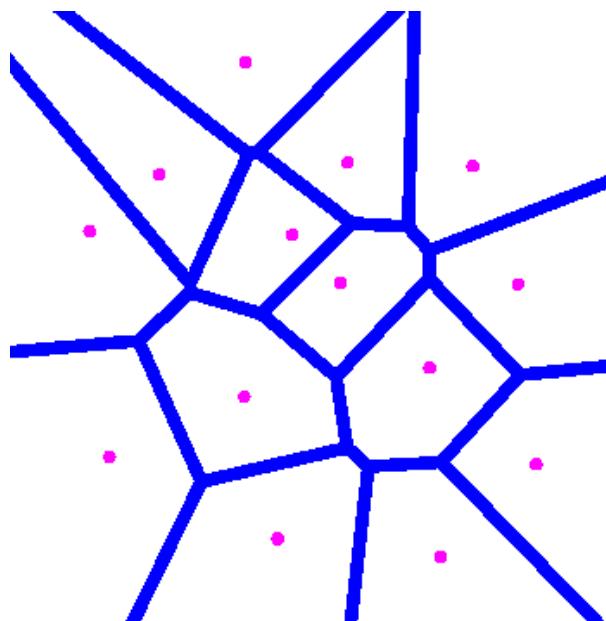
### Représentation d'un maillage

On peut stocker la liste des sommets de chaque triangle:

- C'est simple
- C'est lourd en mémoire

On peut partager des sommets qui sont commun a plusieurs triangle. Il faut donc stocker séparément les sommets des faces.

**Triangulation de Delaunay**, cette représentation repose sur le diagramme de Voronoï.  
 $\{x \in E; \forall qd(x, p) \leq d(x, q)\}$



## Misc

---

- Cartes d'altitude

- On peut les construire par des stratégies aléatoires ou itératives.

Chaque altitude est liée à un type de terrain.



- Blobs/Metaballs

- On peut s'en servir pour la modélisation.

(livre de référence (<https://developer.nvidia.com/gpugems/GPUGems/>))

- L systems

- Inspiré de grammaire de THL.

- scan 3D

- Pour certains films, les gens préfèrent faire leur modèle 3D à la main, puis les numériser.

- Sculpture 3D

- On peut retoucher les objets en 3D avec des logiciels de sculpture.

- Codage des Formes / Maillages

- Arêtes alignées
  - B-Rep
  - Array of vertex
  - Array of index

- Vitesse de mouvement

- Elle n'est pas forcément linéaire.

- Animation tissu vêtement
  - beaucoup de calcul de collision
- Textures
  - plaquées
    - consommation de mémoire élevée
  - Projection
    - triplanar
    - cylindrique
    - sphérique
  - Peinte en 3D
  - Environnement au loin
    - On utilise une skybox

Cela va permettre de faire des rendus de scène extérieure, plus simplement