

# M2 Imagina – Rendu temps réel avancé

---

David Vanderhaeghe  
IRIT - VORTEX - AGGA  
Université Fédérale de Toulouse

# Introduction

---

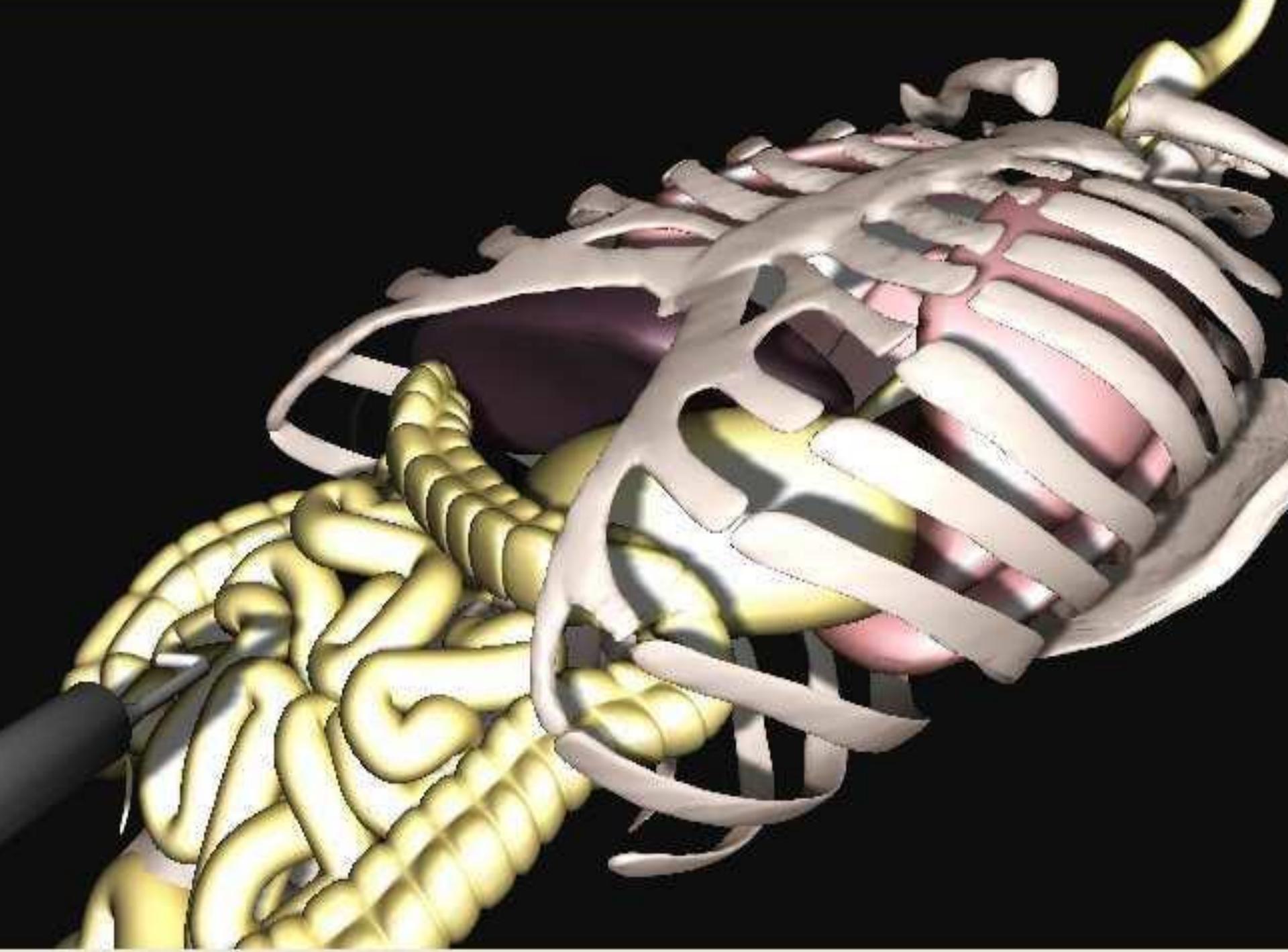
... mathématiques – **informatique** – physique – biologie ...

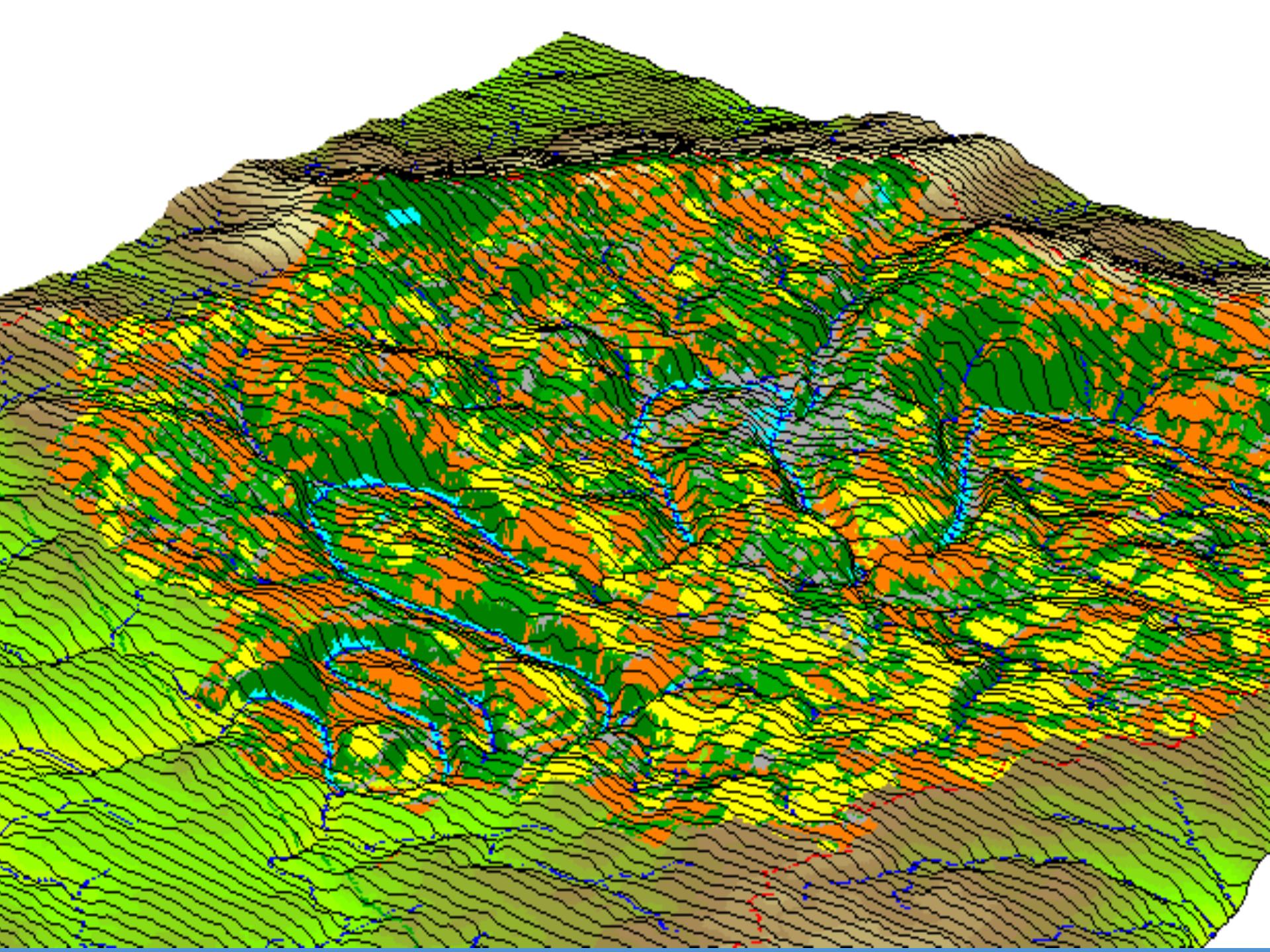
... réseau – informatique **graphique** – logique ...

... simulation – **synthèse d'images** – modélisation géométrique ...

... rendu expressif – rendu temps réel – rendu hors ligne ...







# Warm up ...

---

- Est-ce que vous pouvez définir, rapidement
  - Un ordinateur
  - L'informatique graphique
  - Le rendu temps réel
  - Le pipeline de rendu (OpenGL)
  - Un shader
  - L'équation du rendu

De l'équation du rendu au pipeline temps réel

# Pipeline graphique

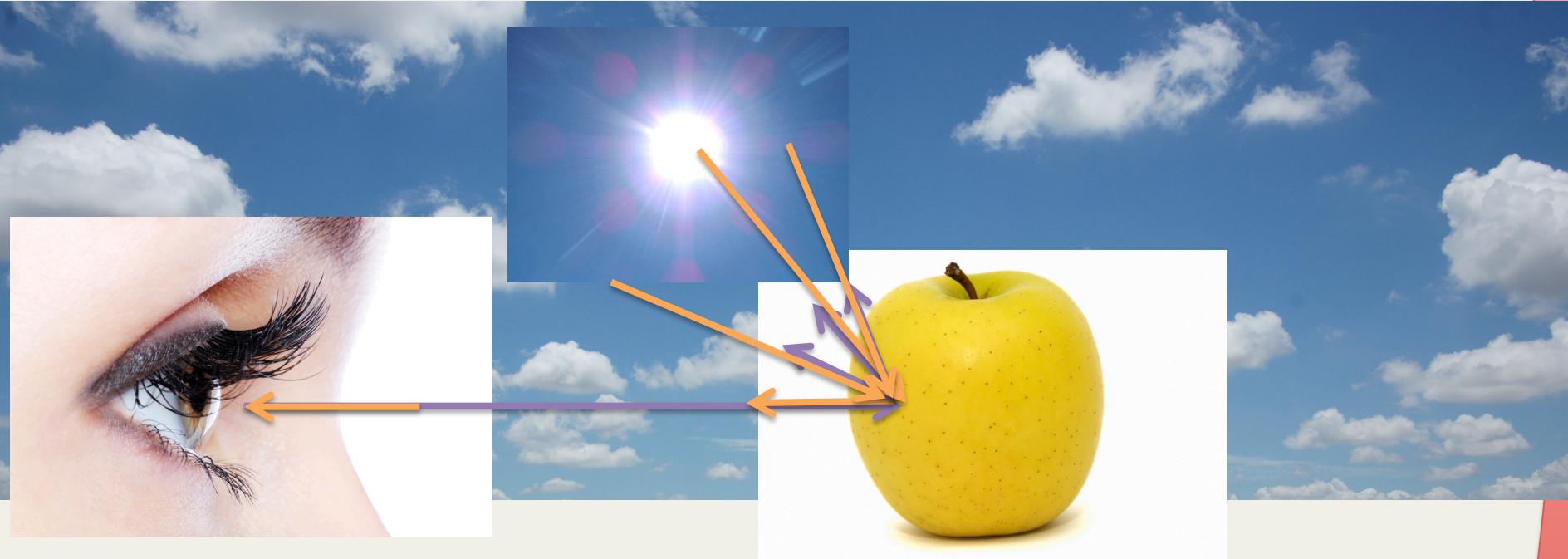
# Synthèse d'image

---

- Calculer la couleur
- Scène 3D
- Rendu réaliste

Équation du rendu

# Equation du rendu



$$L_i(x, \omega_o) = L_o(\text{vis}(x, w_o), -\omega_o)$$

$$L_o(x, \omega_o) = E(x, \omega_o) + \int_{\Omega} L_i(x, \omega_i) \rho(x, \omega_o, \omega_i) \overline{\cos}\theta_i d\omega_i$$

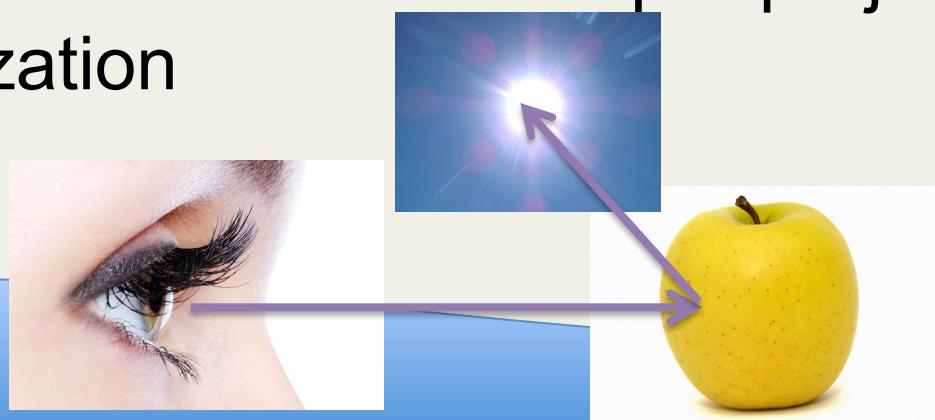
# Rendu temps réel

---

- Deux grandes familles en informatique graphique
  - Rendu hors ligne → pour les films
  - Rendu temps réel → pour l'interaction
- 
- Objectif commun : calculer des images
  - Pas le même budget temps
  - Approches qui peuvent être très différentes

# Simplification de l'équation du rendu

- Intégrale vers somme
- Prendre en compte uniquement l'éclairage direct
  - Echantillonner les directions de lumière et non pas tout l'environnement
- Détermine surface visible par projection/rasterization



# Pipeline graphique 3D

---

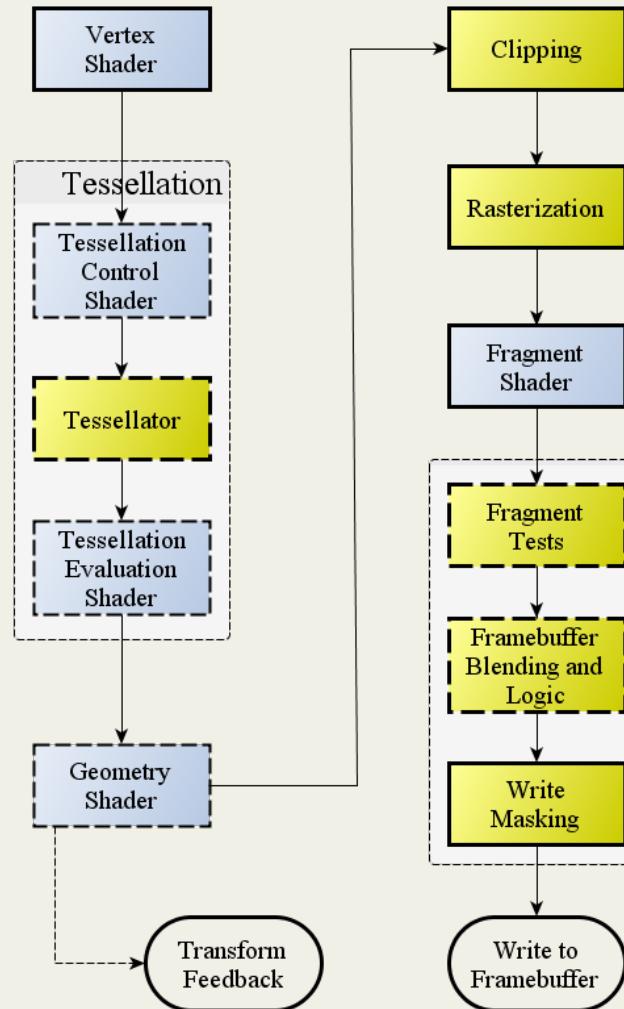
- Objectif : calculer une image
- Entrée, une scène 3D
  - Donnée 3D : définitions d'objets avec leur matériau
  - Lumières, caméras, environnement
  - Animations
  - Graphe de scène : lien entre les différents éléments de la scène
- Traitement de ces données pour l'affichage
  - Calcul de la couleur de chaque pixel
- Sortie
  - Une image ou une suite d'images (film, animation)

# Pipeline graphique 3D

---

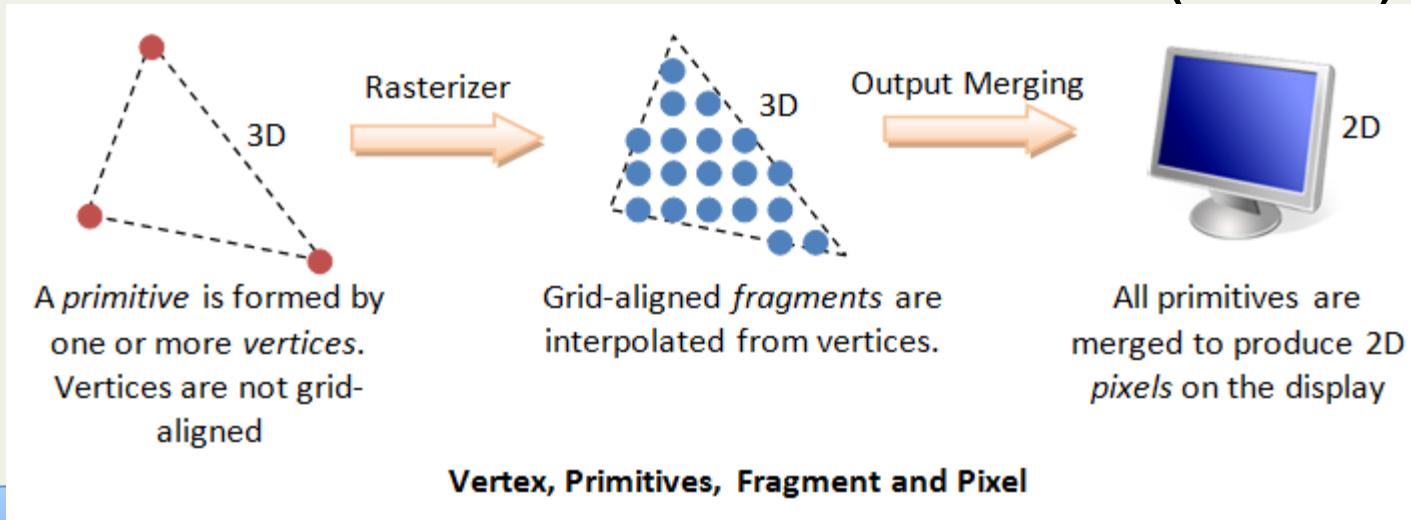
- Coté application (CPU)
  - la gestion des entrées sorties
  - certains calculs complexes, difficilement parallélisables
    - IA, collision, simu physique
- Côté GPU
  - transformations de la géométrie
    - tessellation, transformation 3D, skinning
  - Rendu
    - Rasterisation, éclairage, post FX

# Pipeline OpenGL 3+



# Pipeline graphique 3D

- Entrée : Un triangle
  - Vertex shader : effectue les transformations géométrique
  - Rasterisation : forme 2d vers ensemble de fragment
  - Fragment shader : calcul la couleur d'un fragment
- Sortie : écriture dans le Framebuffer (écran)



# Vertex shader

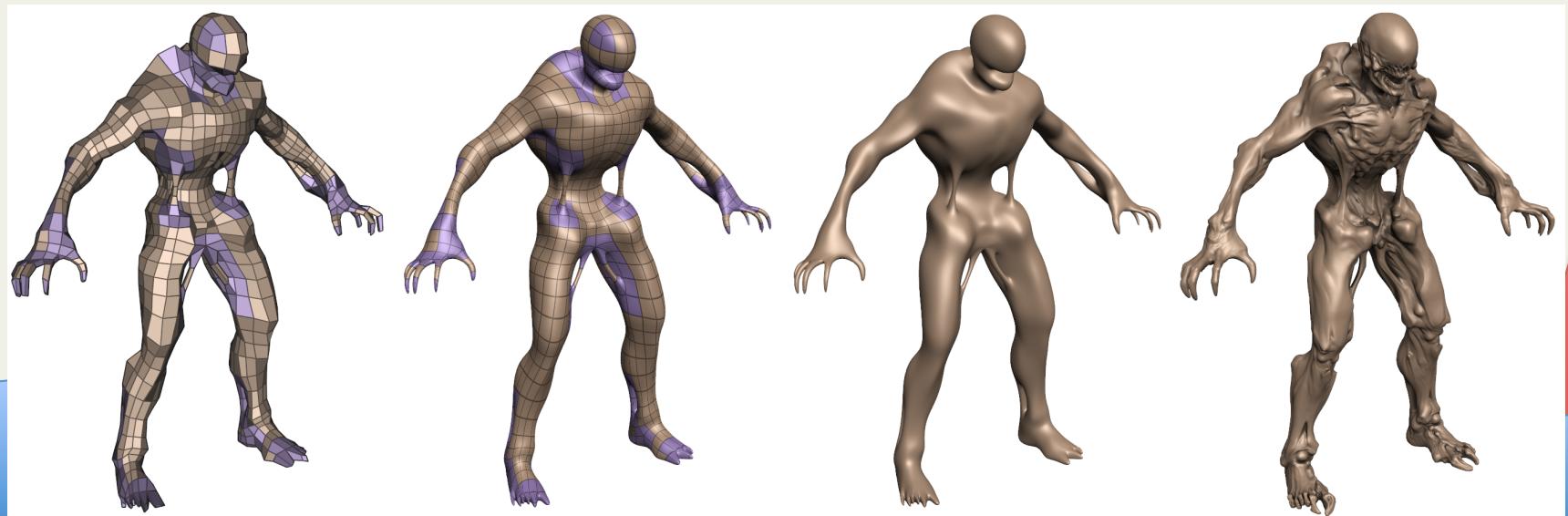
---

- Travail au niveau des vertex
  - Transformations géométriques
  - Éventuellement éclairage
  - Calcul de données au vertex
- Les vertex sont ensuite assemblés en primitives (triangles/quads)
- Pour un fragment d'un triangle, les données calculées pour chacun des trois vertex seront interpolées

# Tesselation shader

---

- Sur les cartes modernes
  - Permet de fabriquer (pleins) de nouveau triangles avec la carte graphique
  - Prend un patch (un triangle, ou autre ensemble de points)
  - Fabrique un ensemble de triangles ou de quads



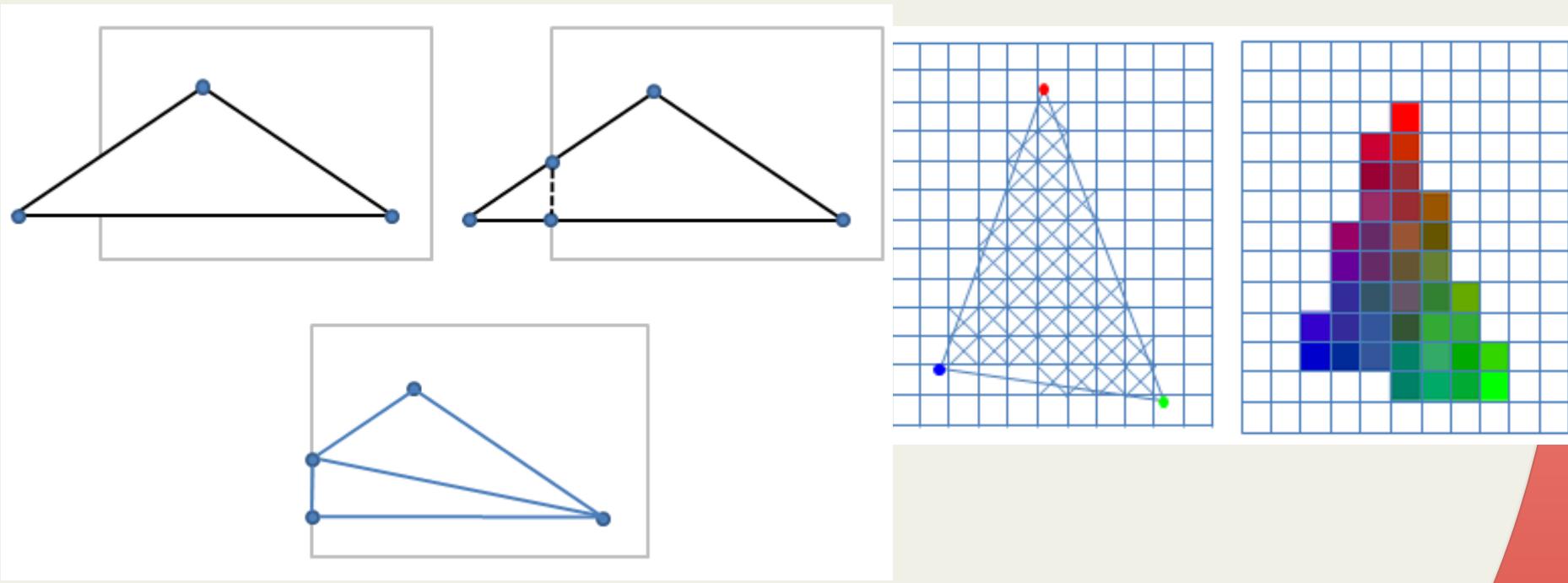
# Geometry shader

---

- Permet la transformation et la création de géométrie
  - Calcul des normales par face
  - Rendu fil de fer
  - Construction de volume d'ombre
- Moins adapté à la création massive de géométrie que le tessellation shader

# Clipping et rasterisation

- Processus hardware et non contrôlable
- En entrée une primitive à afficher
- En sortie une liste de fragment



# Fragment shader

---

- Calcule la couleur d'un fragment
  - Entrée : coordonnée écran + données interpolées + données spécifiques
  - Sortie : « couleur » du fragment

## Les dernières étapes : écriture dans le framebuffer

---

- Plusieurs fragments calculés pour un pixel
- Comment combiner ces fragments

# Opérations de base sur GPU

---

- Ping/pong
  - plusieurs passes de rendu
    - ping -> pong, puis pong->ping
- Redux
  - plusieurs passes de rendu pour calculer une valeur
  - par exemple somme, max, min

# Effets spéciaux

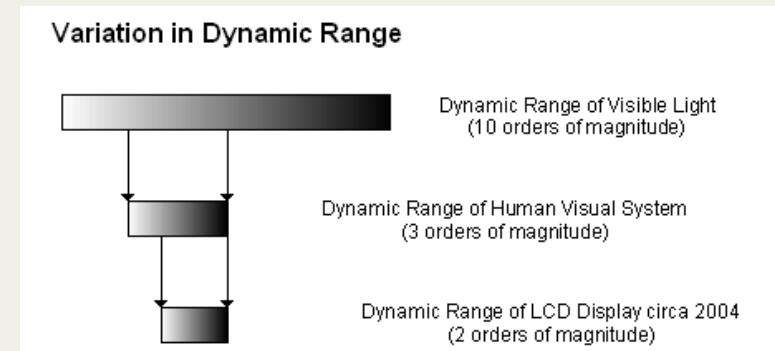
---

# HDR : tonemap et bloom

---

# Pourquoi faire un rendu HDR

- RGB codé par 256 valeur
- Différence entre blanc et noir : 255
- Adaptée au matériel graphique
- Écran = 24 bpp
- Limitations
- Entre soleil et étoile, contraste de  $10^8$
- Nombre de couleurs limité
- Perte dès l'acquisition



# Comment faire un rendu HDR

---

- Idée : capture (et rendu) adaptée à la scène
- Principe
- Limites du SVH : dynamique maximum
- Conserver l'information de la scène *sans perte* jusqu'à l'affichage



# HDR

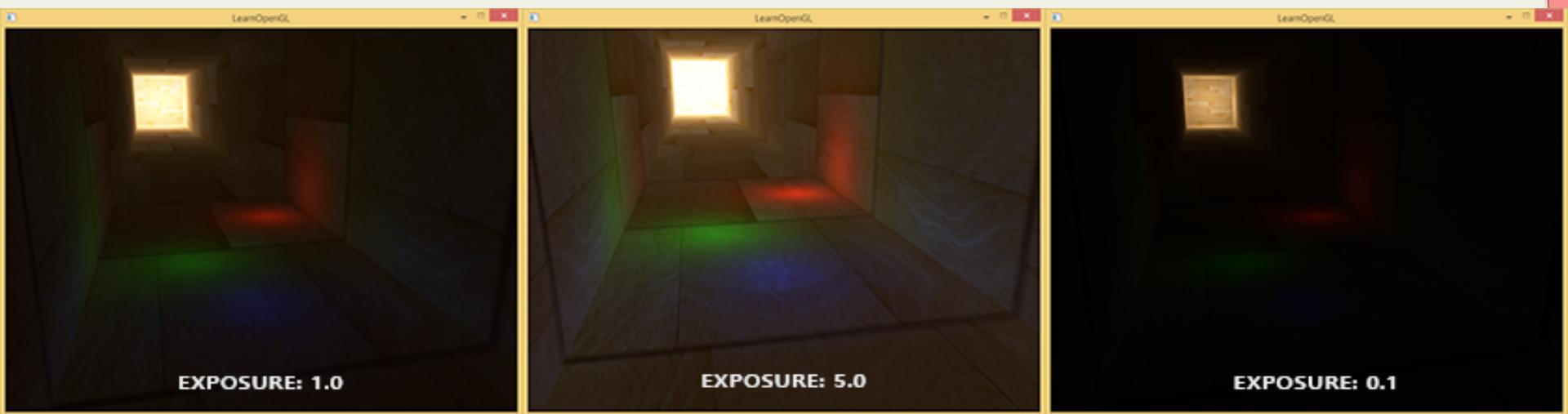
---

- Simple : clamp
  - C'est par défaut
  - Ajout de contrôle avec exposition
  - C'est global et fixe
- Statique
  - `mapped = hdr/(hdr+1);`
  - ou `mapped = vec3(1.0) - exp(-hdrColor * exposure);`
  - `corrected = pow(mapped, 1/gamma);`
- Dynamique
  - calcul de la luminance moyenne, min et max de l'image

# Dans la pratique

---

- Dépends de la scène en entrée
- Et aussi de l'application



# Dans la pratique

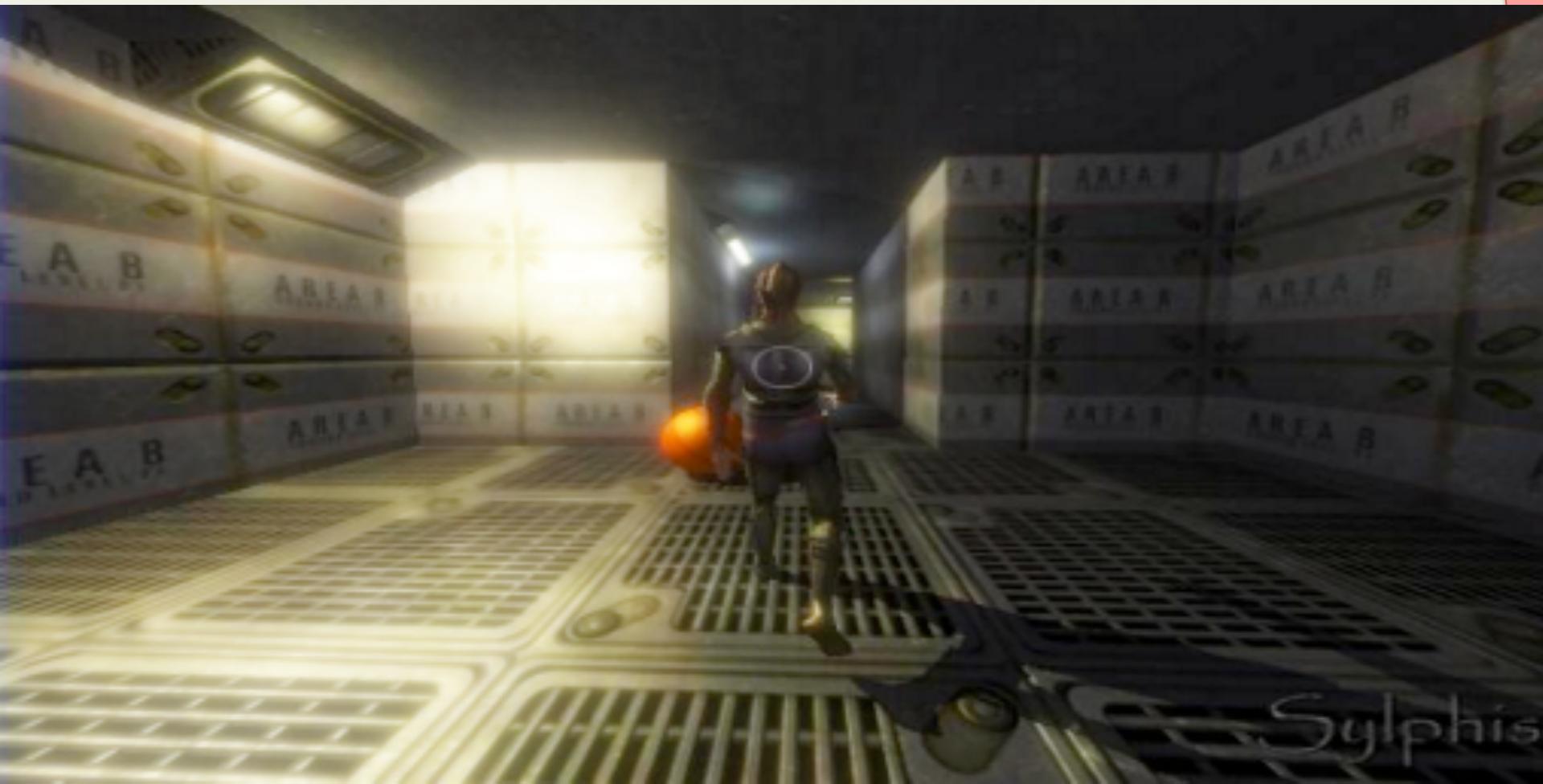
---

- Rendu de la scène en HDR
  - dans un framebuffer à valeurs flottantes, non bornées
  - descriptions des lumières et des matériaux par des valeurs avec unité physique
  - au moment de l'affichage, tonemapping

# Bloom

---





# Bloom

---

- Ajouter un halo autour des zones de luminosité intenses
- Mimes le comportement de l'œil
- Et aussi des capteurs
- Donne un aspect beaucoup moins RTR
  - Les couleurs bavent un peu

# Bloom : implémentation

---

- Sélectionner les pixels brillants
  - Appliquer un flou
  - Additionner à l'image de départ
- 
- Se fait habituellement avant le tonemap
  - En prenant en entrée l'image HDR.

# Fx : Blur (pour le bloom)

---

- Equation flou gaussien (convolution)
- (n'oubliez pas de normaliser les valeurs du noyau)

# Implémentation naïve

---

- Fragment shader
  - For( $i$  ...
    - For( $j$  ...
      - Output +=  $G(i,j)$ texture(input,  $x+i-ks,y+j-ks$ )

# Dependent texture read

---

- Calculer les coordonnées de texture dans le fragment : dependent texture read
  - Pénalité de performances
- Vertex shader :
  - Calculer les coordonnées de textures et les interpoler

# Implémentation séparable

---

- Quand on peut écrire
  - $M = \text{col} \times \text{lin}$
- Faire deux filtre 1D (ping pong)

# Viewport size et interpolation bilinéaire

---

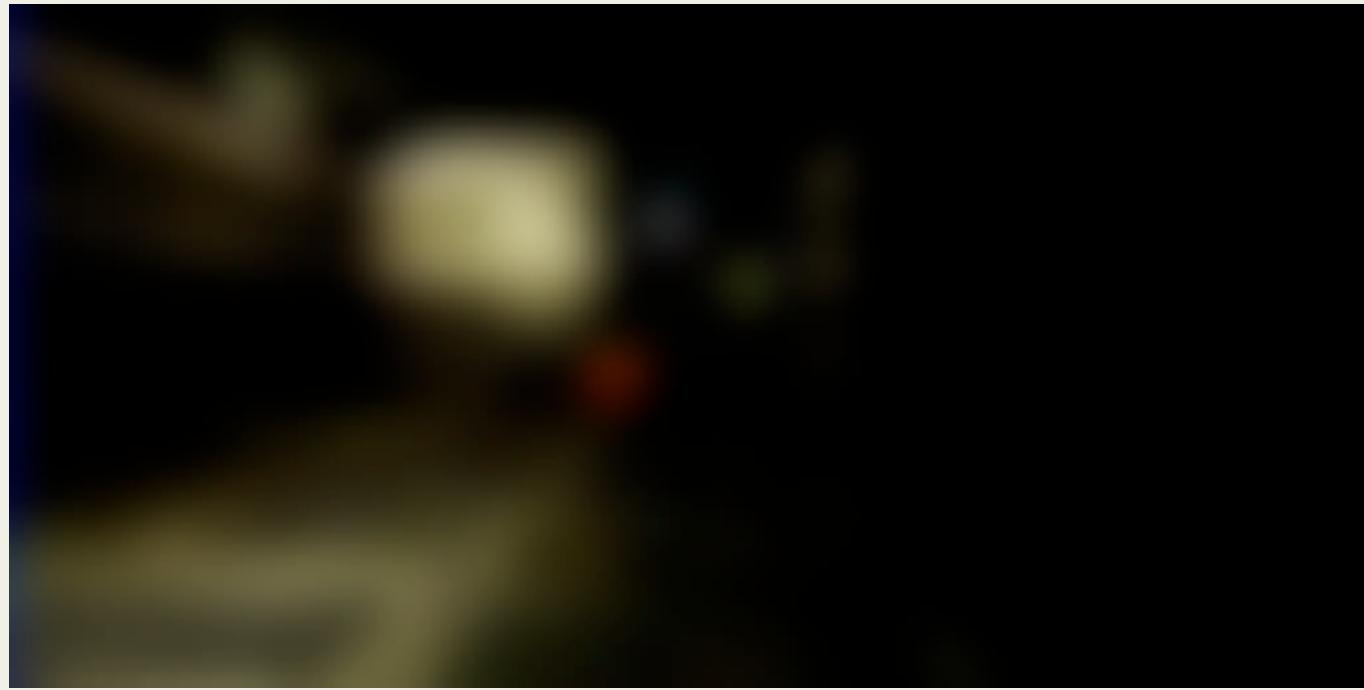
- Réduire la taille de travail
  - Lire entre les texels
- 
- Exo : calcul sigma si taille /2
  - Exo : ou lire pour s'approcher d'un flou gaussien

.006	.061	.242	.383	.242	.061	.006
------	------	------	------	------	------	------



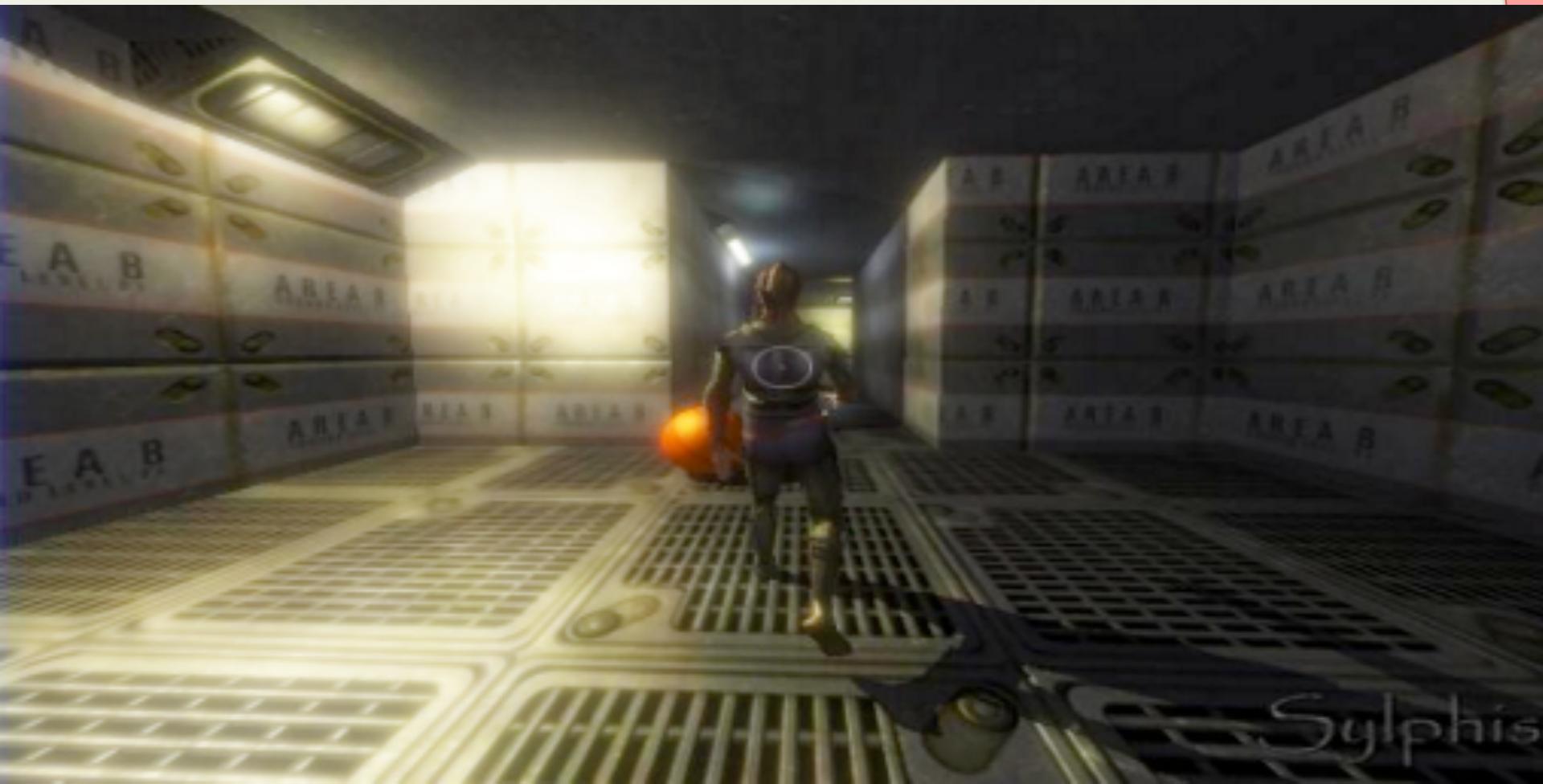


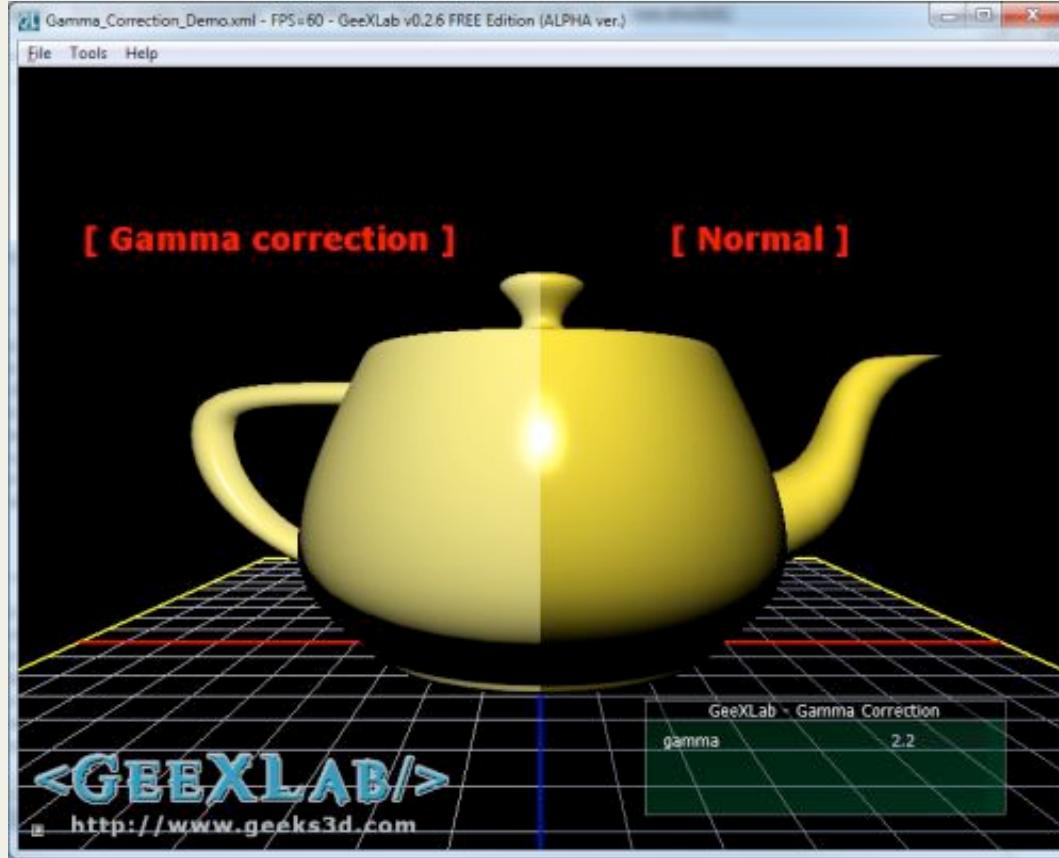








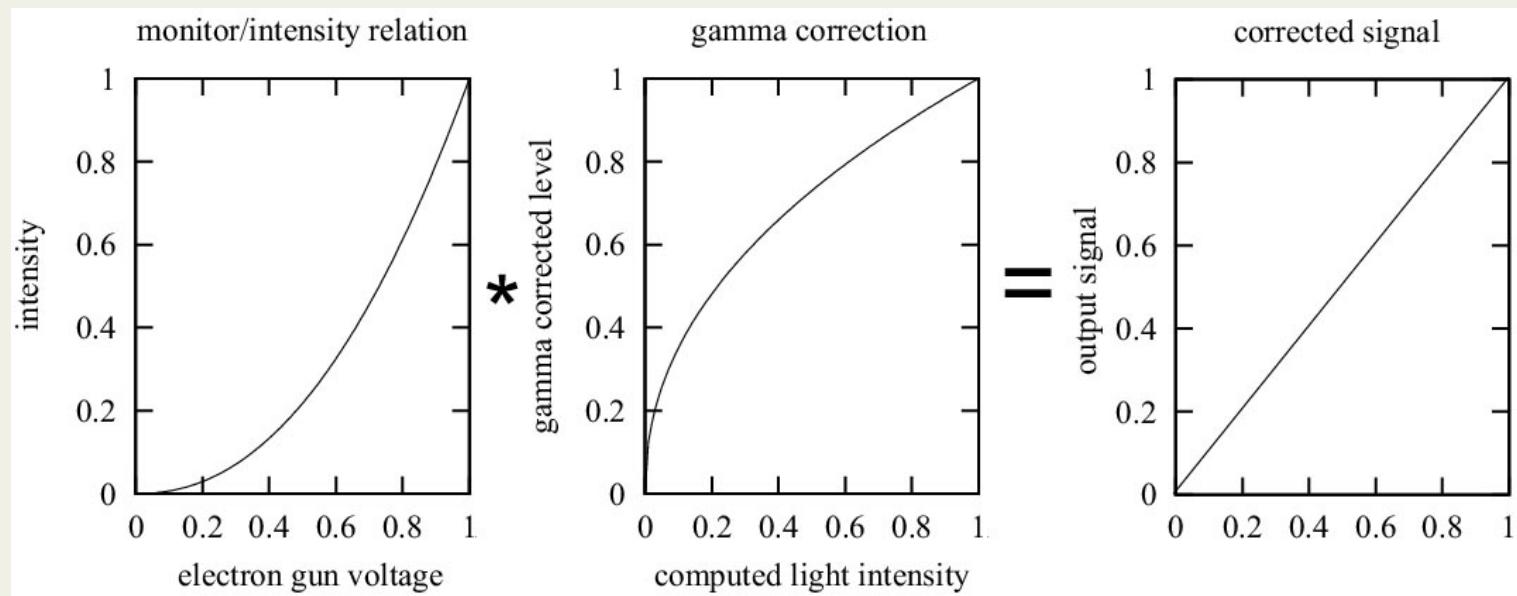




# Correction gamma

# Gamma correction

- Signal de 0.5 sur l'écran ne donne pas 0.5 d'intensité lumineuse
- Correction gamma du signal : rétabli la linéarité de la luminosité



# Gamma correction

---

- $I$  = intensité sur l'écran       $I = a(V + \varepsilon)^\gamma$
- $V$  = voltage d'entrée
- $a$ ,  $e$ , et  $g$  constante du système
- Valeur typique : 1.8-2.6
- Soit  $e=0$ , la correction gamma devient

$$c = c_i^{(1/\gamma)}$$

# Pourquoi la correction gamma est importante ?

---

- Portabilité
- Qualité de l'image
  - Texture
  - Interpolation
- Existe en hardware
  - Coté texture : GL\_SRGB8
    - > converti lors du glTexImage2d sRGB vers RGB

# Correction gamma

---

- A la lecture (texture, couleur)  $\text{pow}( \dots , \text{gamma})$
- A l'affichage  $\text{pow}( \dots , 1/\text{gamma});$

# Profondeur de champ (DoF)

---

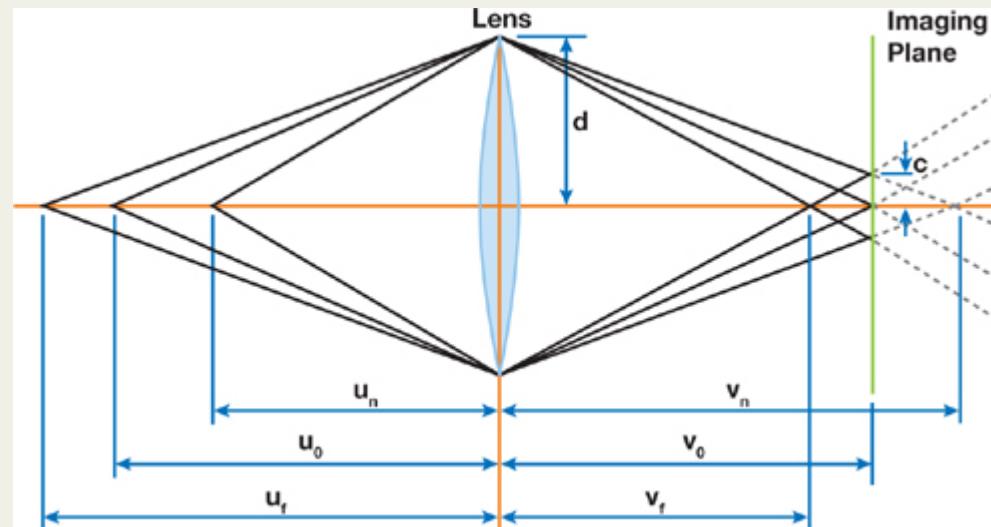
---

In this case: 24 Hz (1.7 M Tris) 



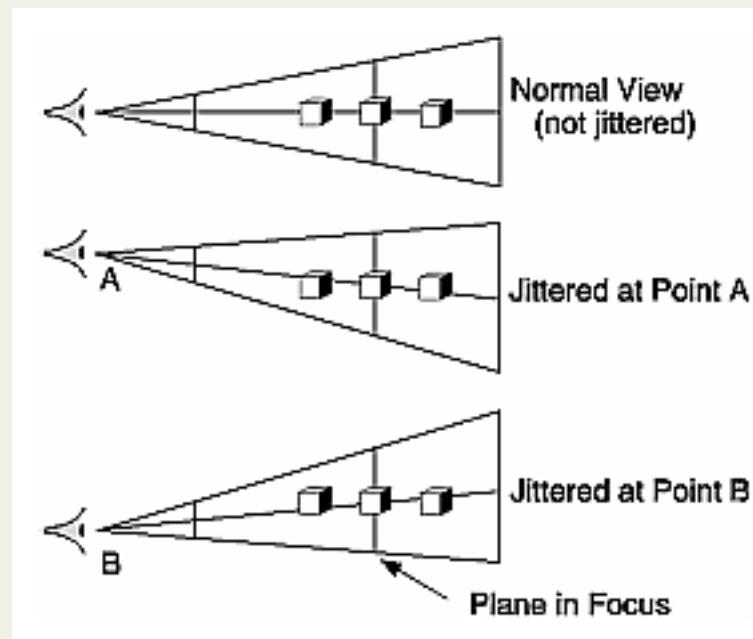
# Profondeur de champs

- Origine du phénomène



# Profondeur de champ

- Principe en rendu temps réel
  - moyenne de plusieurs rendus



- flou gaussien de largeur variable
- cone tracing sur une voxelisation de la scène

# Exercice

---

- Par groupe
  - sur papier
  - étape 1 : définir un pipeline général (étapes principales avec entrée sortie)
    - A faire valider
  - étape 2 : détailler les étapes principales
    - Vous pouvez regarder comment faire en OpenGL
- mettre en place un pipeline de rendu HDR + bloom
- écrire la boucle de rendu pour le depth of field

# Avant la pause

---

- Prendre une minute pour écrire sur un papier (à me remettre)
  - Quels sont les points les plus importants (centraux, utiles, surprenant, dérangeants) que vous avez appris durant cette partie
  - Quelles questions restent en suspend ?
  - Est-ce qu'il y a quelque chose que vous n'avez pas compris ?