Ray Tracer / Rasterizer

Graphics Programming

Eric Cannet

Avant le cours

- Moi :Eric Cannet
- But du cours: Vous faire une introduction à la programmation 3D via DirectX
- Cours non exhaustif
- Utilisation de DirectX 9 avec uniquement des Shaders

Plan des séances

- Raytracing, Rasterizer, GPGPU
- Le pipeline graphique, Transformations dans l'espace, géométrie et affichage
- Win32, les textures, les shaders, l’éclairage
- Le jeu vidéo, Des moteurs existant, les consoles, des concepts
- Premier affichage avec DirectX
- Debugger, Affichage de terrain
- Mise en pratique des textures avec DirectX
- Mise en pratique de l'éclairage avec DirectX

But de ce cours

- Introduction au Ray Tracer
- Introduction au Rasterizer
- GPGPU

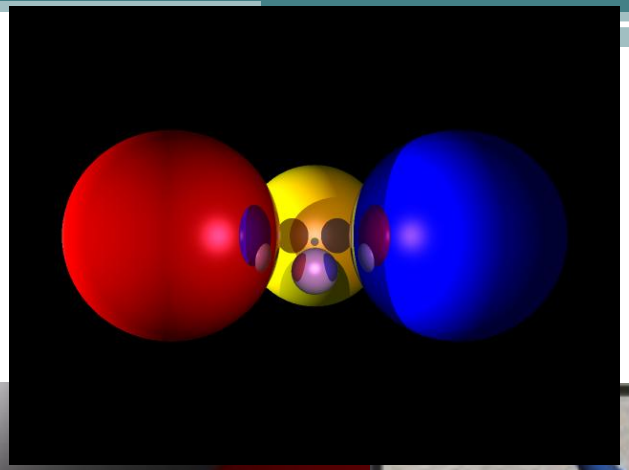
Where am I ?

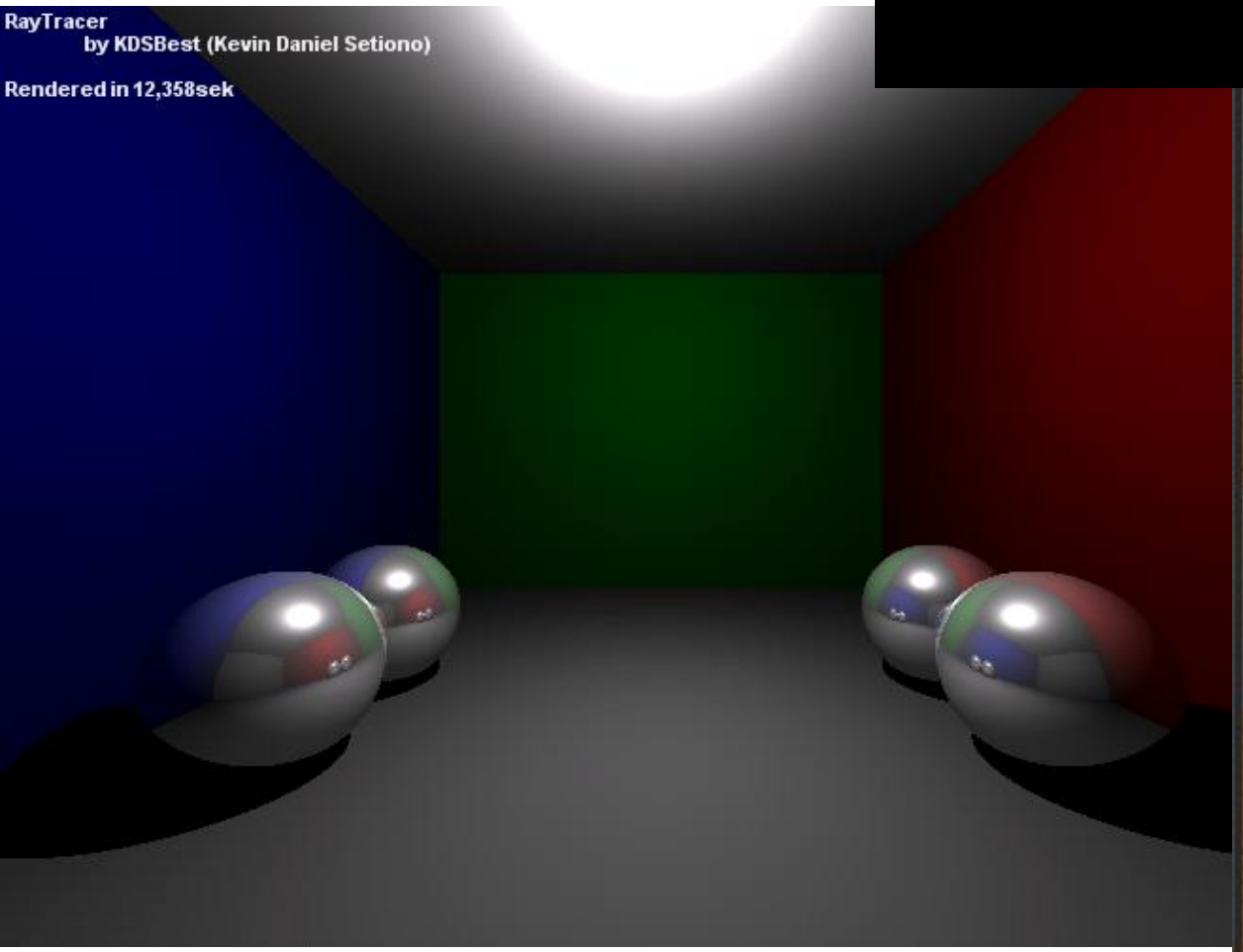
- Où est utilisé le ray tracing ?

- Où est utilisé la rasterization ?

Ray Tracer

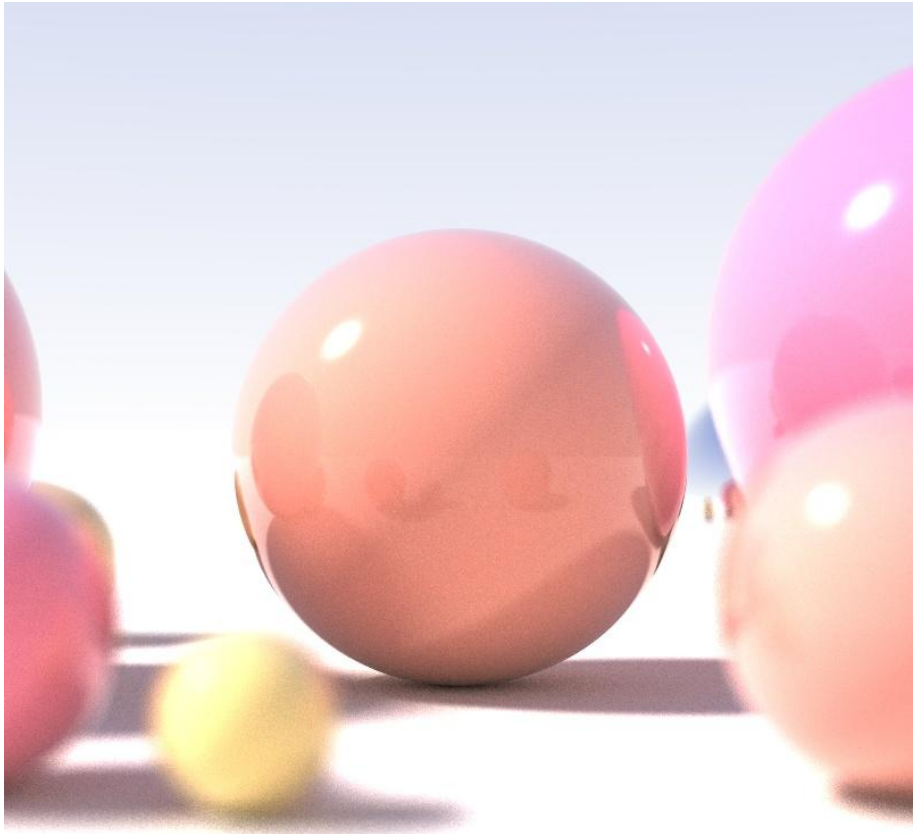
Des images

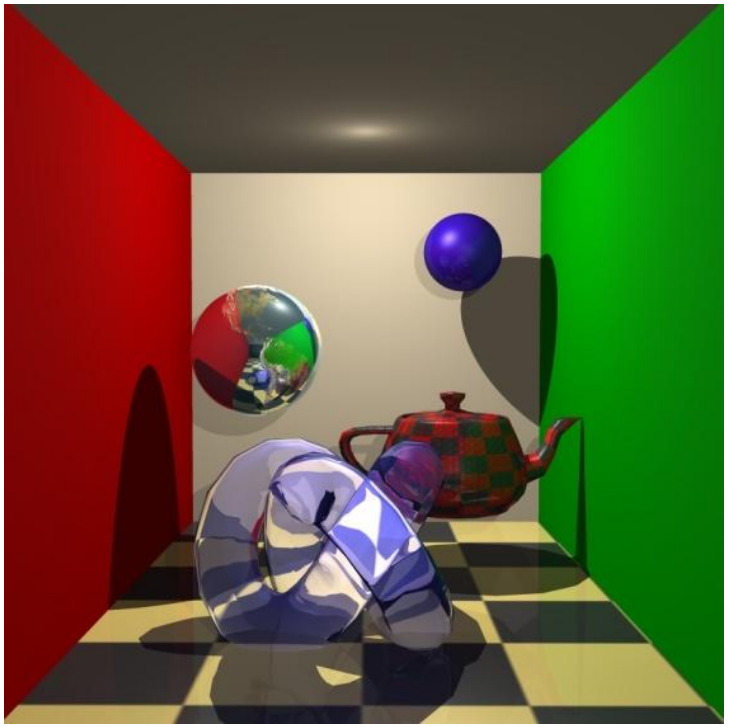



RayTracer
by KDSBest (Kevin Daniel Setiono)
Rendered in 12,358sek

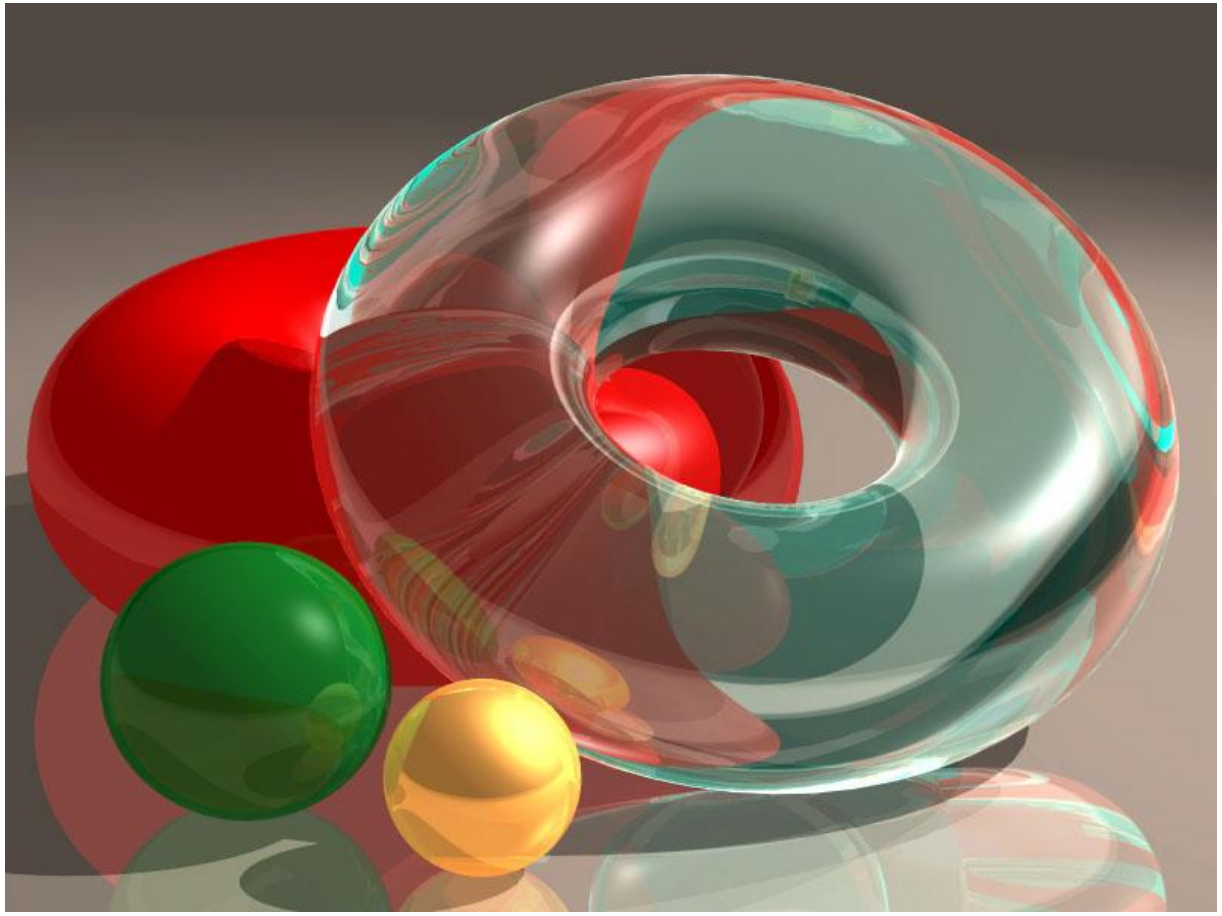

Rasterized
Ray traced

Des images !

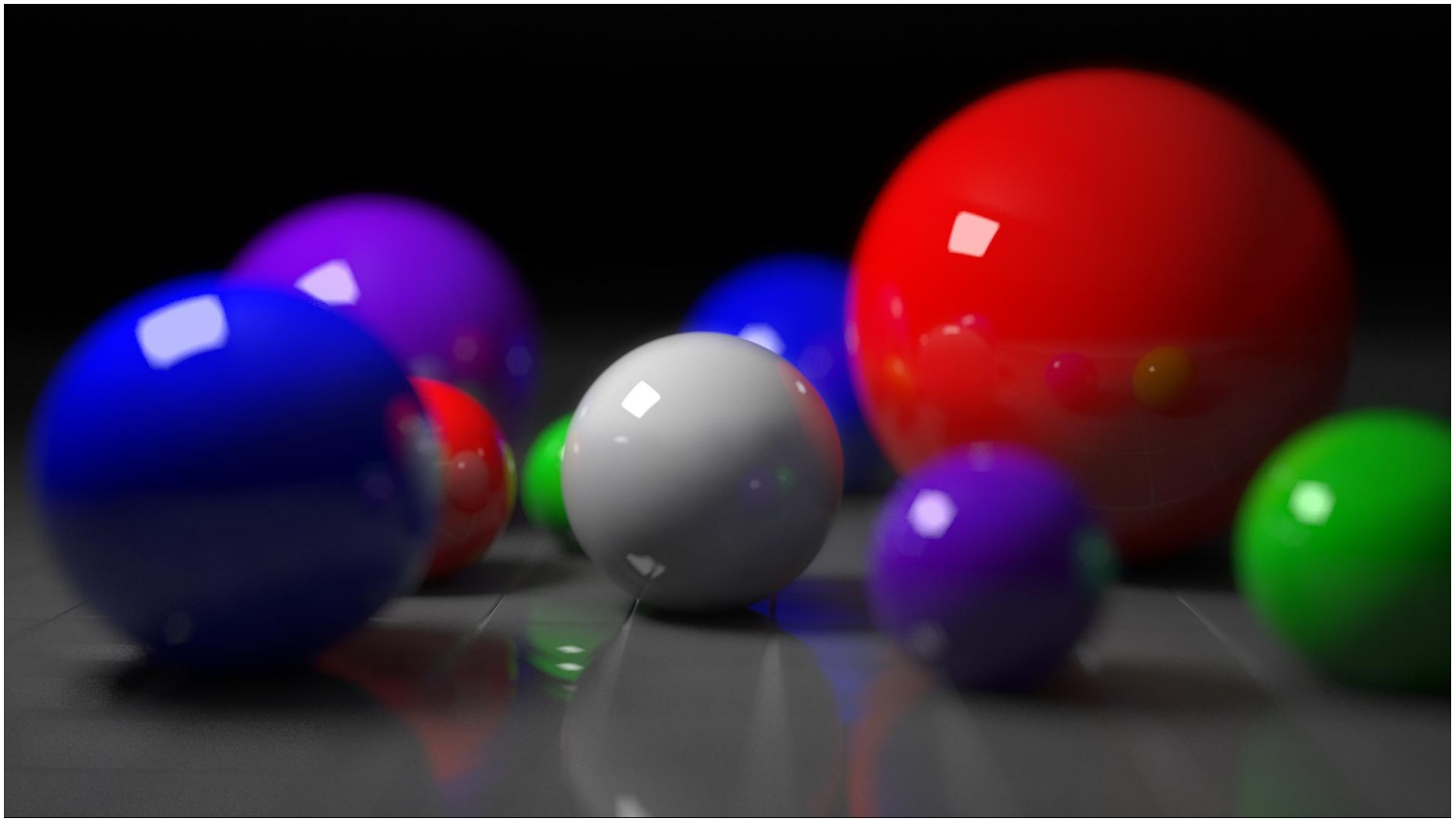




Encore des images !



On peut faire que des sphères ?



Et une petite dernière!

Les avantages

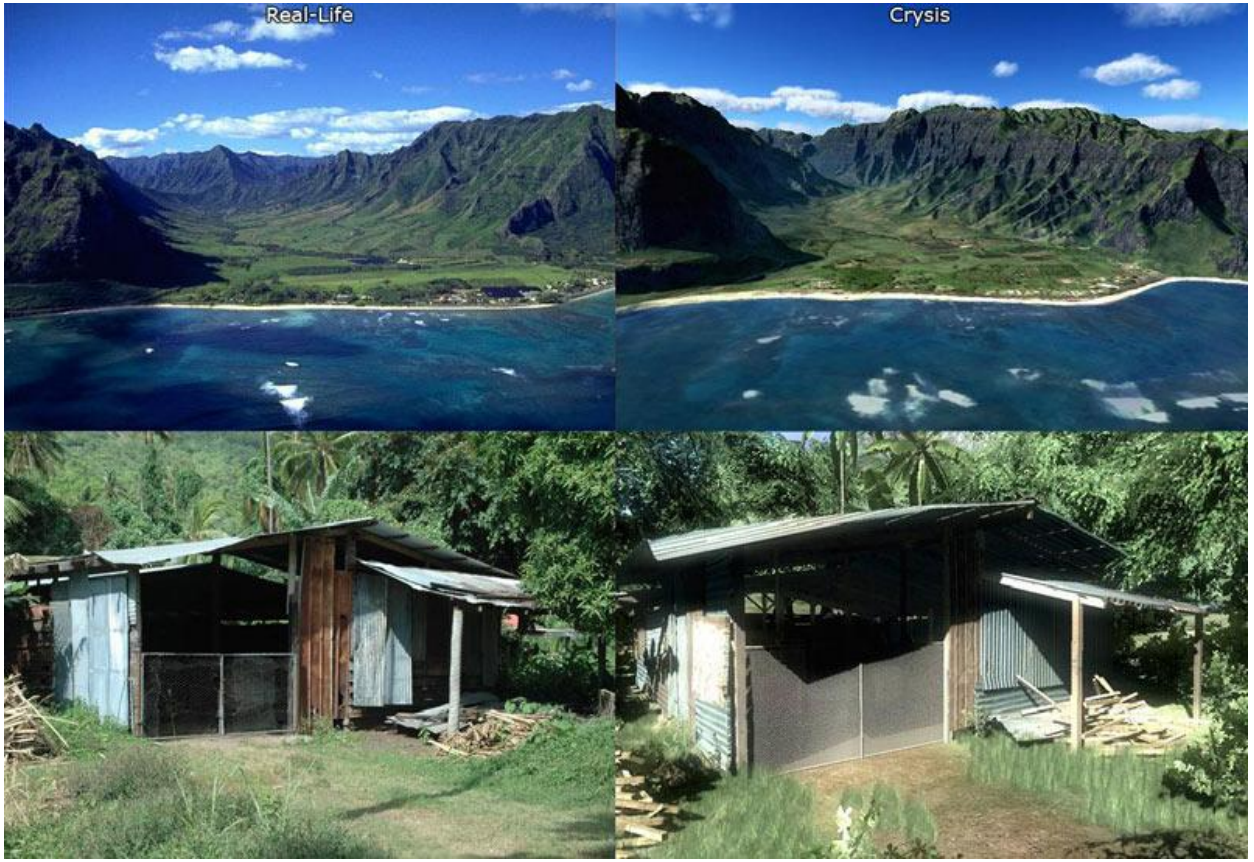
- Permet de produire des images très réalistes.
- Algorithme proche de la réalité.
- Parallélisable
- Objet définit mathématiquement.

C’est beau, pourquoi ce n’est pas dans les jeux ?

- Très couteux.
- L’habitude.
- Trop coûteux.
- Très trop couteux.

Oui, mais encore

- Avec les rasterizer on arrive à des choses très réalistes



Oui mais il y a quake wars

- Il tournait entre 14 et 29 fps en résolution HD (720p) (1280x720). (2008)

- Sur une machine :
  - 16 core !
  - (4 socket, 4 cores) Tigerton
  - 2.93 GHz

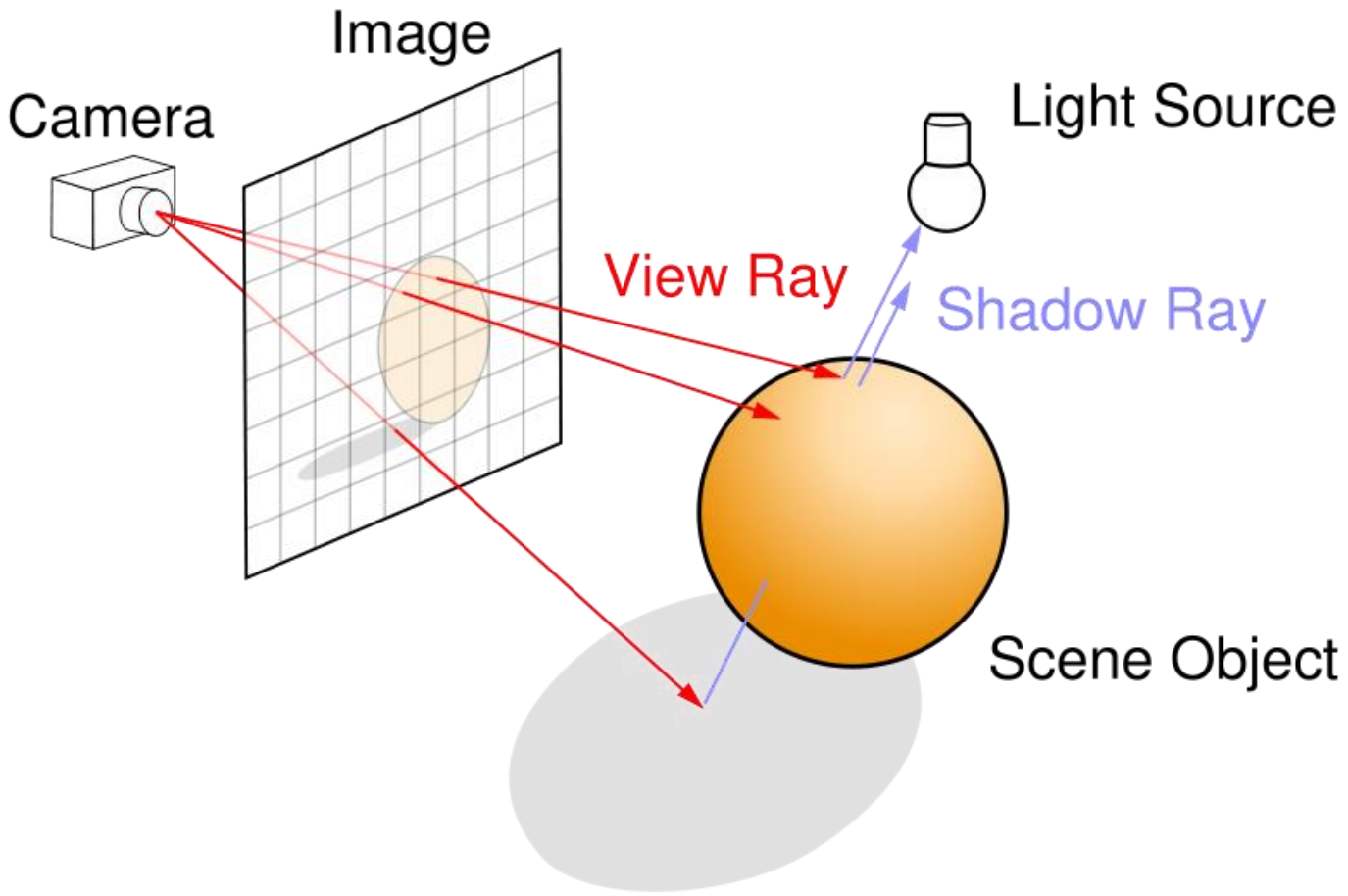
- 2008 : 20 - 35 fps sur 4 Dunnington 6 cores (2.66 GHz) soit 24 cores
- 2009 : 15 - 20 fps sur 2 Nehalem (core I5/core I7) 4 cores (3.2 GHz) soit 8 cores

Un seul échange social : le tir à vue

Le principe en code

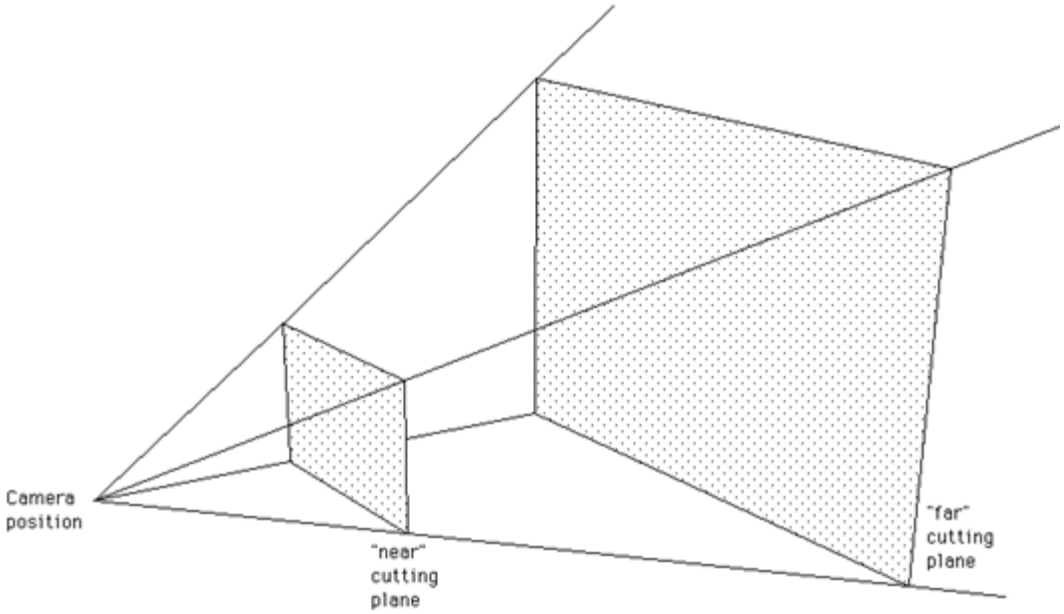
- Pour chaque pixel
  - Créer le rayon correspondant
  - Pour chaque objet de la scène
    - Si il y a une intersection
      - Regarder si c'est la plus proche.
- Calcul de la couleur :
  - Soit au moment de l'intersection
  - Soit à la fin

Le principe en image



Frustum

- C’est une pyramide avec le haut coupé.
- Représente ce que voit la caméra.

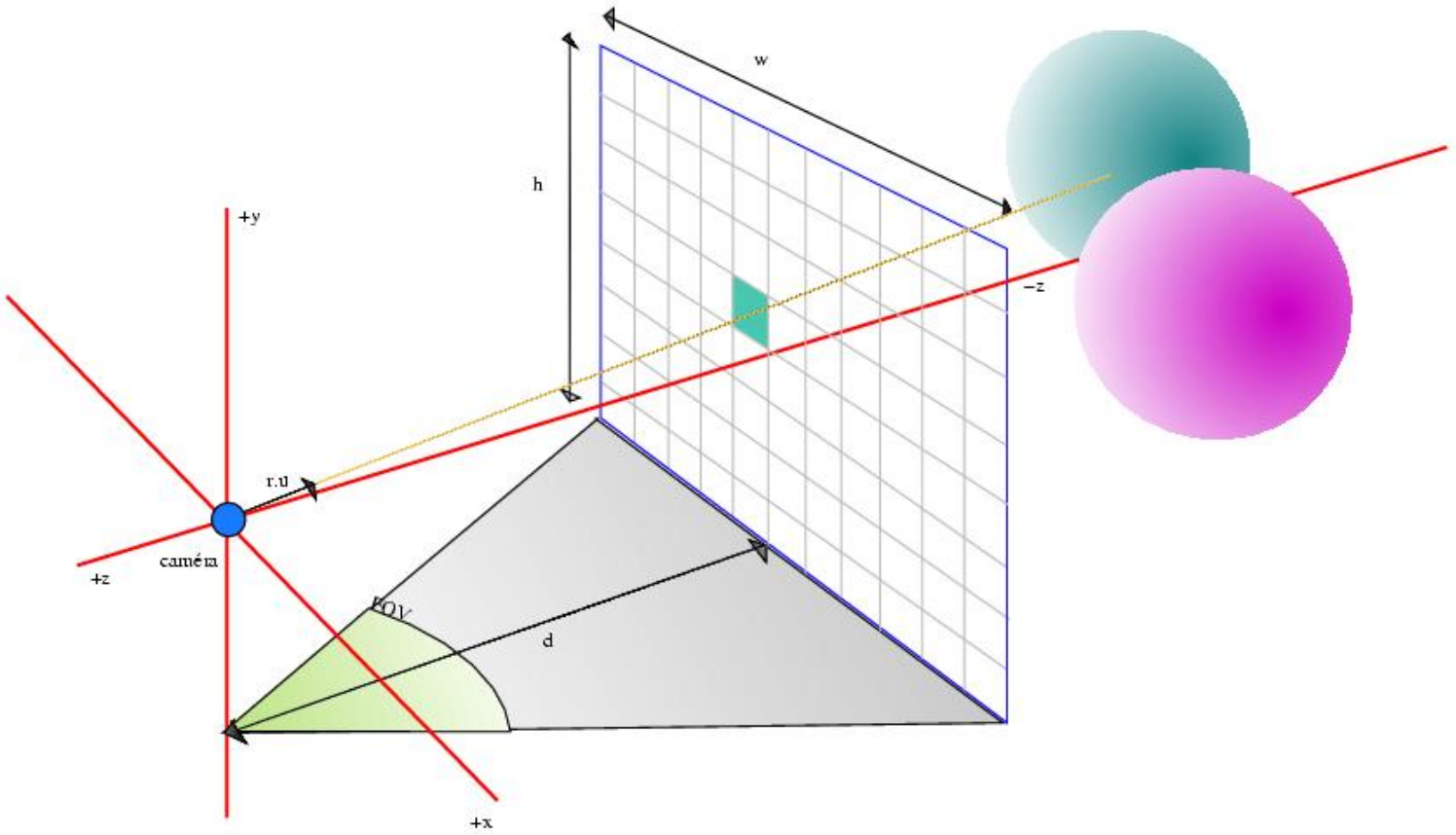


Un rayon ??

- Représenter par :
  - Un vecteur3 de position (l'origine)
  - Un vecteur3 de direction

- Qui représente la droite d'équation :
  - P(t) = origine + t * direction

Création du rayon



Création du rayon

- L'origine correspond à la position de la caméra
- La direction :
  - X = (2 * x / SreenX – 1) * xScale
  - Y = - (2 * y / SreenY – 1) * yScale
  - Z = 1
  - Avec :
    - yScale = tan(FOVy / 2.0f)
    - xScale = yScale * Ratio
  - Et on le normalise bien sûr !
  - On le transforme pour prendre en compte l'orientation de la caméra

Un exemple! un exemple !

- L'équation de la sphère est: $|| P - C ||^2 = R^2$
- Equation de la droite : $P(t) = O + t * D$
- Une collision signifie que les deux équations sont vérifiées :
  - $|| O + t * D – C ||^2 = R^2$
  - $|| t * D + A ||^2 - R^2 = 0$ avec $A = O – C$
  - $t^2 * D^2 + 2 * t * D.A + A^2 - R^2 = 0$
  - Polynôme de variable t.

L'exemple continue en roulant

- $t^2 * D^2 + 2 * t * D.A + A^2 - R^2 = 0$
- Si d est normalisé on peut simplifier ($D^2 = 1$)
  - $t^2 + 2 * t * D.A + A^2 - R^2 = 0$
- Si $\Delta < 0$ alors il y a pas de collision.
- Sinon 1 ou 2 solutions :
  - $T1 = - D.A + \sqrt{((D.A)^2 - (A^2 - R^2))}$
  - $T2 = - D.A - \sqrt{((D.A)^2 - (A^2 - R^2))}$

- Prendre le plus près (T le plus petit et positif)

</maths>

- Pour trouvé le point d'impact :
  - Pimpact = O + Tmin * D
- En terme de code ca donne quelque chose comme ca:
  - float3 dst = ray.o - sph.o; //calcul de A
  - floatB = dot(dst,ray.d); // calcul de D.A
  - float C = dot(dst,dst) - sph.r * sph.r; //(A^2 - R^2)
  - float D = B*B - C; // Δ
  - return D > 0 ? - B - sqrt(D) : std::numeric_limits<float>::infinity()

Avec un plan ?

- Equation d'un plan :
  - $Ax + By + Cz + E = 0$

- Même principe on injecte l'équation de notre rayon dans la précédente
  - $A*(Ox + t * Dx) + B*(Oy + t * Dy) + C*(Oz + t * Dz) + E = 0$

Aziz lumière !

- Couleur diffuse.

- Coefficient : Produit scalaire entre la normale du point et la direction de la lumière
- On y reviendra…

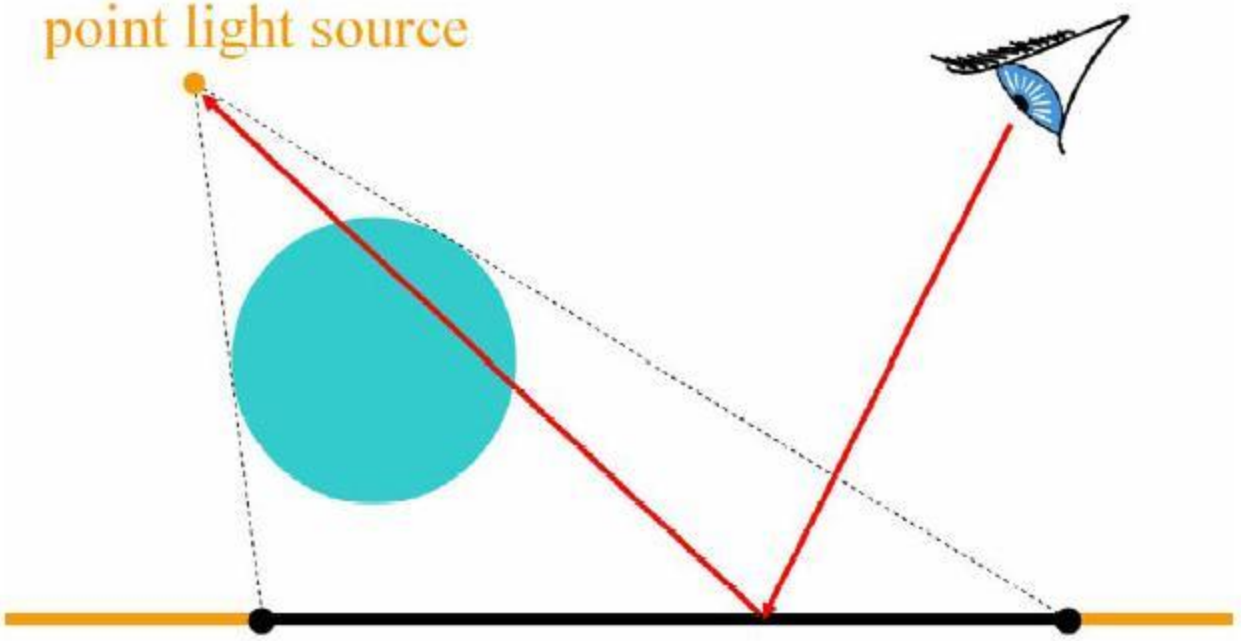
I’m a shadow in the dark,

- Qu'est-ce qui fait qu'il y ait une ombre ?
- Réponse : le fait qu'un objet se situe entre le point et la lumière.

- Il suffit donc de lancer un autre rayon vers la lumière avec comme point de départ le point d’intersection.

And in the night

- Si il y a une intersection alors c'est ombré.
- Pas besoin d'avoir l'intersection la plus proche.

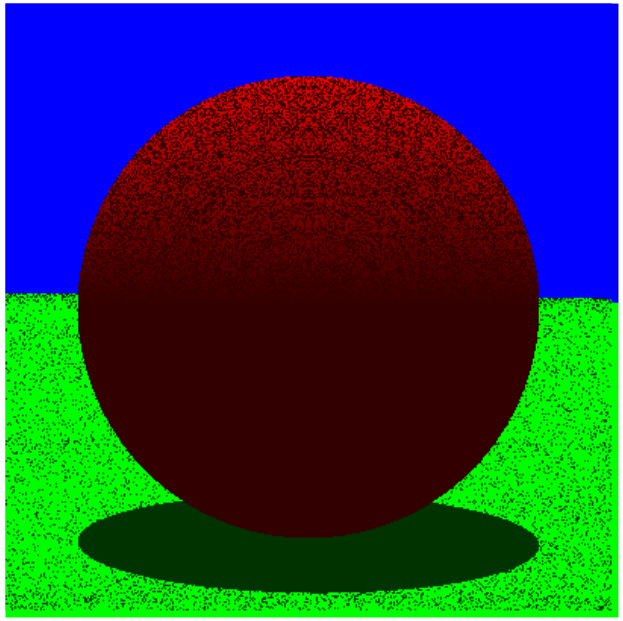


Cool, il est où le piège ?

- Il détecte une collision avec lui-même.
- On ne peut pas l'exclure du test de collision car sinon un objet ne se ferait jamais d'ombre

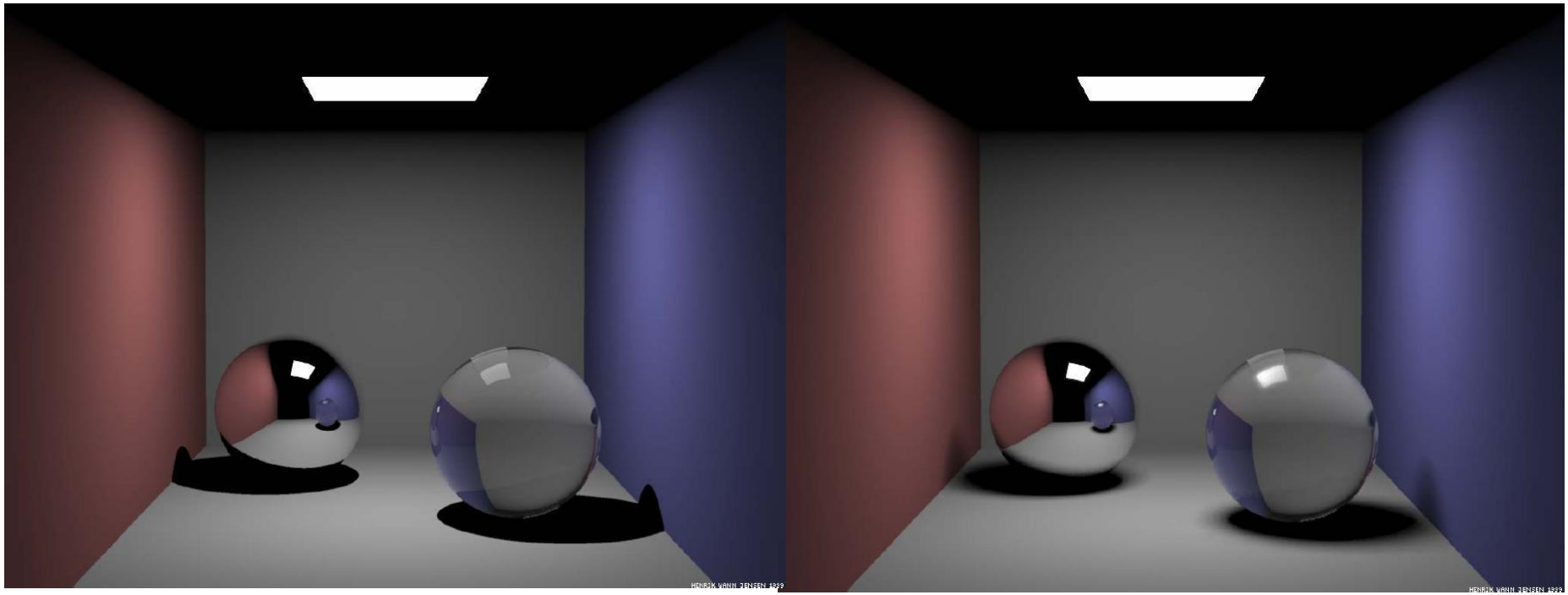
- Solution : Epsilon
- Partir de epsilon plus loin

que le point d'intersection.



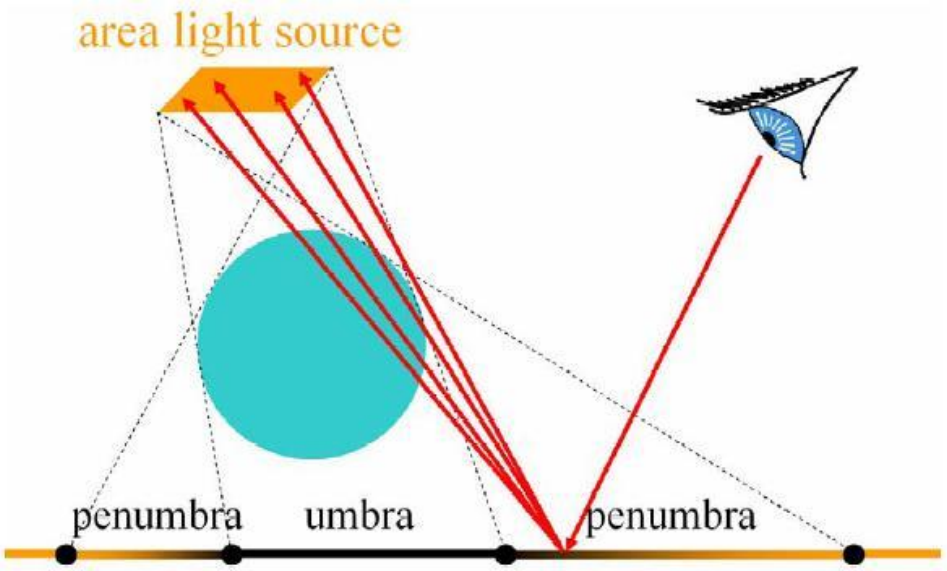
Un peu hard, non ?

- L'ombrage est net: soit c'est ombré soit ça ne l'est pas.
- Solution : Soft shadow



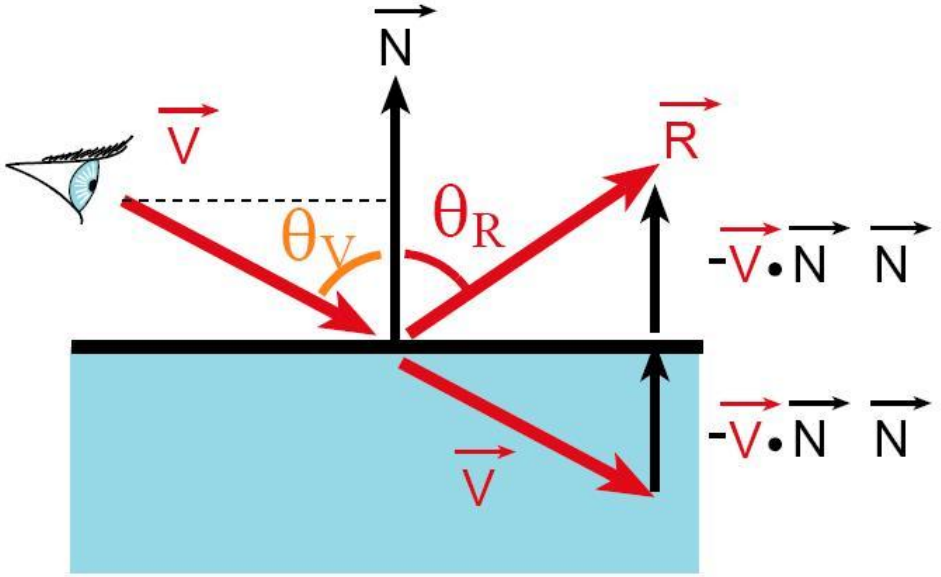
Du soft c’est mieux

- Considérer la lumière comme un quad et non comme un point.
- Lancer plusieurs rayons vers le quad et faire une moyenne.



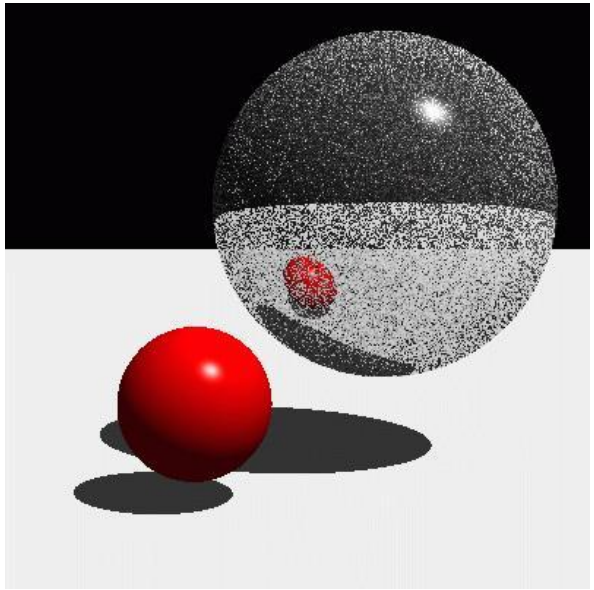
Une petite réflexion !

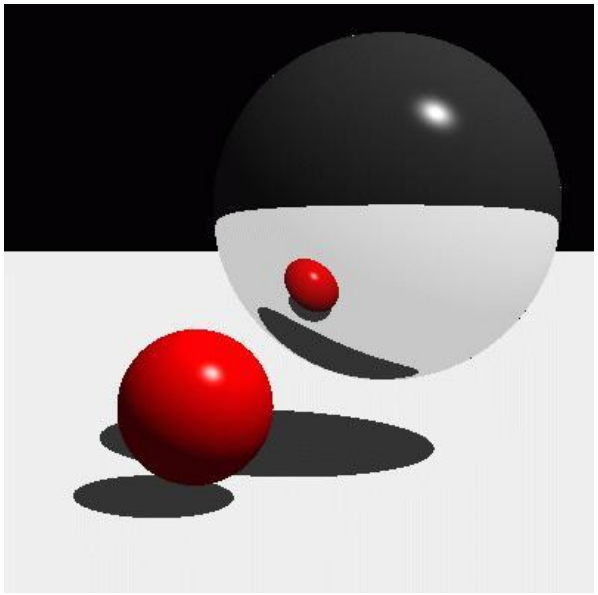
- Même principe, on relance un rayon :
  - Origine : Le point de collision
  - Direction :
    - c1 = -N.V
    - R = V + (2 * N * c1)



Mais pas trop non plus

- N’oublier pas l’epsilon !
- Multiplier par le coefficient de réflexion.
- On a besoin du plus proche et de la couleur !





Rappel du code

- Pour chaque pixel
  - Créer le rayon correspondant
  - Pour chaque objet de la scène
    - Si il y a une intersection
      - Regarder si c'est la plus proche.

- Deux voies d'optimisation :
  - Optimisation d'algorithme classique
  - Moins de rayon.
  - Optimiser le test de collision.

AA et non AAA

- L'anti-aliasing permet:
  - D'avoir des bordures plus douce
  - On va « manquer » moins d'objets

- En temps réel il existe une multitude de technique:
  - AA, MSAA, FXAA, FSAA, SSAA, Temporal AA, MLAA, SMAA, SRAA, TSSA, TXAA, SFAA….
  - Non non c'est pas une blague.

Anti-aliasing

- De façon général et naïve si on veut une image en 1280x720 (720p) en anti-aliasing 4 fois (4x):
  - La scène est rendu en 5120x2880
  - On downscale en faisant une moyenne (pour 1 pixel on fait la moyenne des 4 au alentour)

- Prend plus de mémoire et plus de temps

NoAA

4xAA

Oscar, mon squelette

- Le squelette est représenté par un arbre général
- Chaque nœud de l'arbre correspond à un os et contient une matrice.
- La matrice final de l'os est SaMatrice * LaFinalDuParent

Avec des muscles

- La matrice de chaque bone sera mis à jour par le gestionnaire d'animation.
- La matrice final des bones doit être mise à jour par vos soins. Ou en tout cas en dehors de l'animation
- Les animations sont composer de pose, position et orientation en absolue. Si c'est du relatif on parle d'animation additive

On rajoute la peau

- On le verra, un objet est composer de plein de sommets reliés entre eux.
- Chacun de ces sommet doit contenir les identifiants des bones qui l'affectent ainsi qu'un poids pour chacun. En général quatre.
- On parle de Blending.

Bouge ton corps

- Une animation est un ensemble de KeyFrame pour chaque os.
- Une KeyFrame c'est:
  - Un temps
  - Un Vecteur3 de translation
  - Un Vecteur3 de scale
  - Un Quaternion de rotation

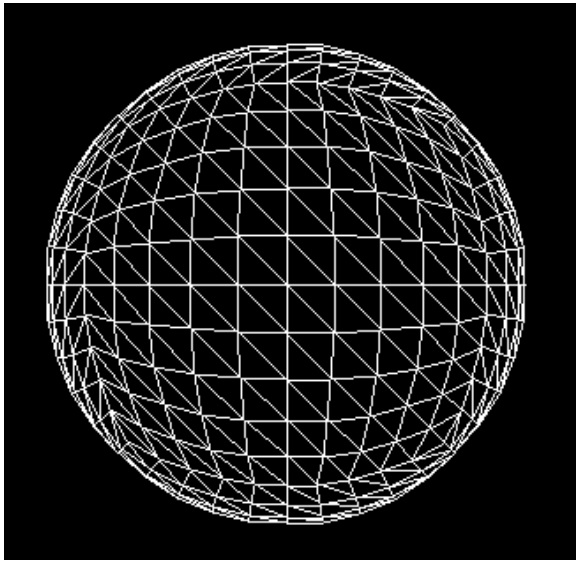
- On lerp les Vecteur3 et on Slerp le Quaternion

- La matrice final d'un os est la matrice SRT = Scale * Ratation * Translation (et dans cette ordre !)

Rasterizer

Rasterizer, c'est français ?

- Transforme une image vectorielle en image matricielle !
- Transformation des triangles en screen space vers des pixels.
- Effectué par la carte graphique.



- Objet composé de triangles.

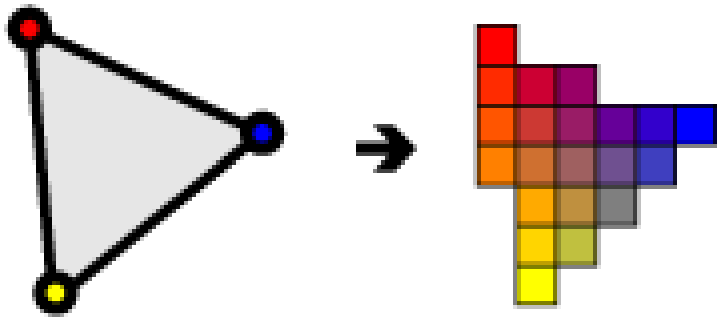
Trop fort mon GPU

- Réalisé par la carte graphique.
- C'est une des étapes du pipeline graphique. Cette étape est non programmable.

- En entrée trois points avec des coordonnées en float. On est dans du vectoriel

- En sortie la grille de pixel est remplis et les données ont été interpolé

Et si je veux l'être aussi ?



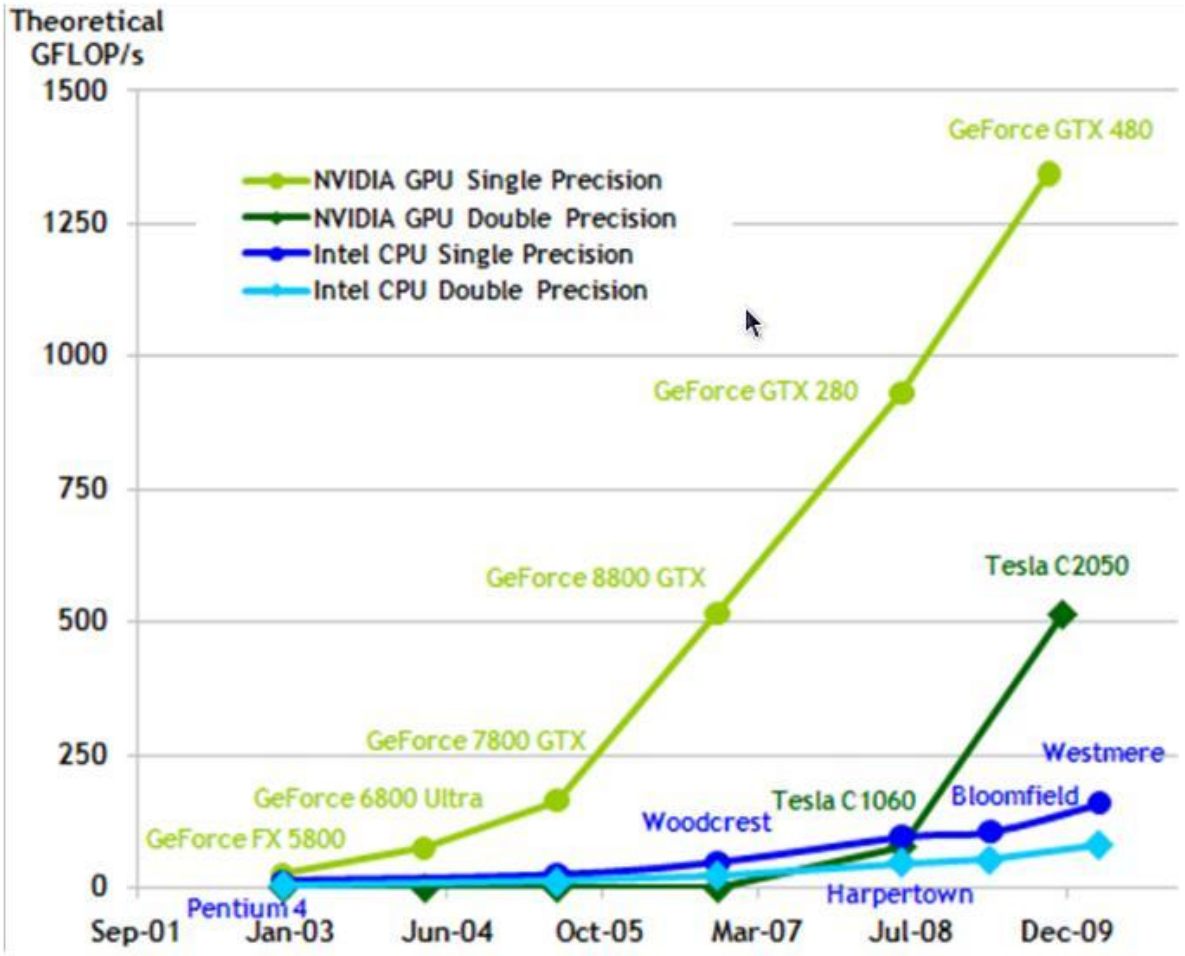
- Si vous souhaitez aller plus loin, les mots clés Google:
  - Digital Difference Analyzer
  - Bresenham's Algorithm

GPGPU

GPGPU

- General-Purpose computation on GPUs
- Faire du calcul autre que graphique sur le GPU. Dans le sens autre que d'utiliser le pipeline graphique pour afficher des triangles.
- La puissance des GPU est bien supérieure à celle de CPU pour les calculs mathématiques.
- Massivement « multi-core »

Et en image !


Theoretical GFLOP/s
1500
1250
1000
750
500
250
0
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla C2050
Tesla C1060
Westmere
Bloomfield
Woodcrest
Harpertown
Pentium 4
Sep-01
Jan-03
Jun-04
Oct-05
Mar-07
Jul-08
Dec-09

Comment on fait ?

- Les moyens :
  - Cuda
  - OpenCL
  - DirectX 11 (Direct Compute)
  - Stream

- Le GPU(x86) d'Intel : Larrabee (projet mis en pause)

Whaou on peut faire ca

- Exemples d'utilisation:
  - Du Chiffrement (Cryptographie)
  - Calcul mathématique (matrice)
  - PhysX
  - Traitement d'image
  - Traitement du signal
  - Ray tracing
  - Et tellement d'autres choses (une seule limite votre imagination)

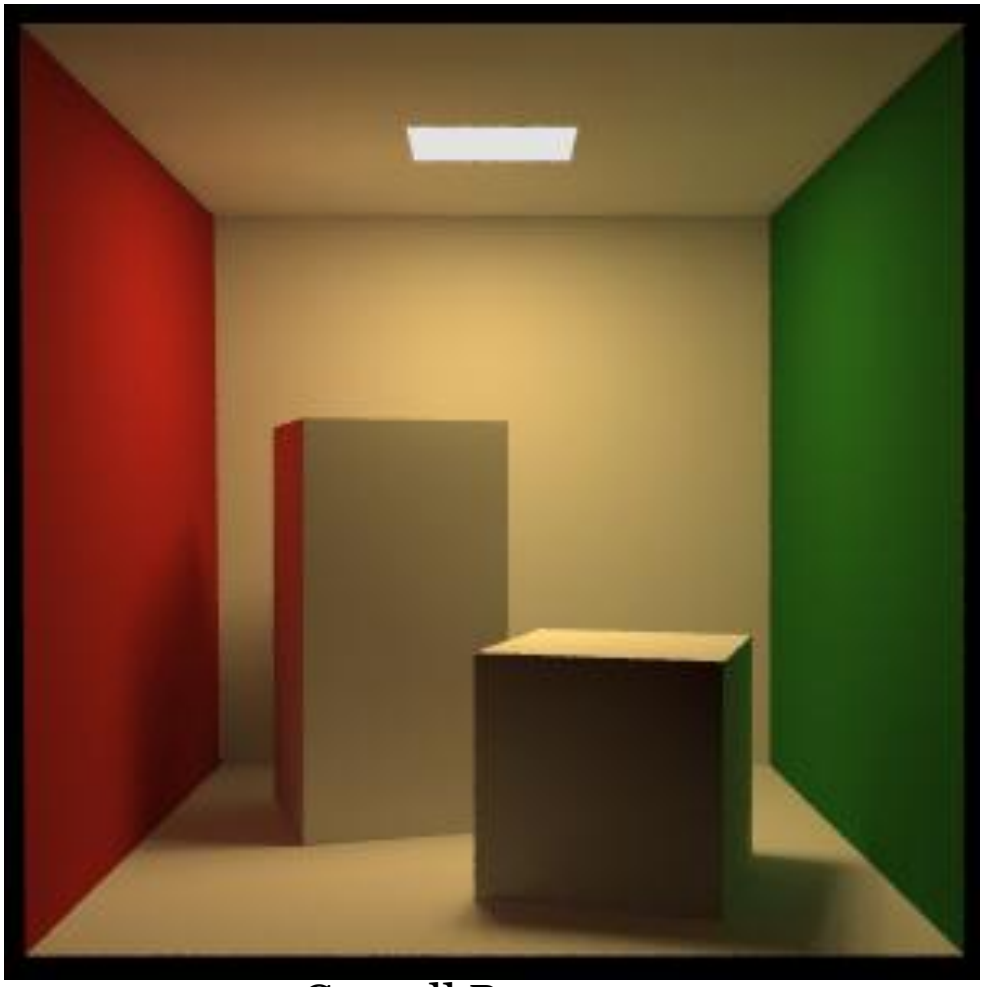
On peut revenir au graphisme

- Certains calcul graphique utilise maintenant du GPGPU
  - Le calcul de l'éclairage
  - Du culling (occlusion culling)
- Sur PS3 ces calculer sont fait sur les SPUs

Conclusion

- Commencer dès maintenant à:
  - Installer Visual Studio C++ 2012
  - Installer le dernier SDK (Software Developpent Kit) de DirectX (le dernier tant qu'a faire)
- Le prochain cours : de la plomberie !, vers l'infinie et au-delà, dessine moi un mouton
- Des questions ?

Image du ~~jour~~ cours



Cornell Box

Travail demandé

- Un projet technique, sujet et groupe libre:
  - Prendre un sujet/thème et réaliser un projet de rendu sur ce sujet/thème le plus beau possible
- Un mini rapport:
  - Décrire le but de votre projet
  - Aperçu des techniques existantes
  - Description de votre démarche et de la technique utilisée
  - Benchmark, optimisation et amélioration apportés
- Soutenance

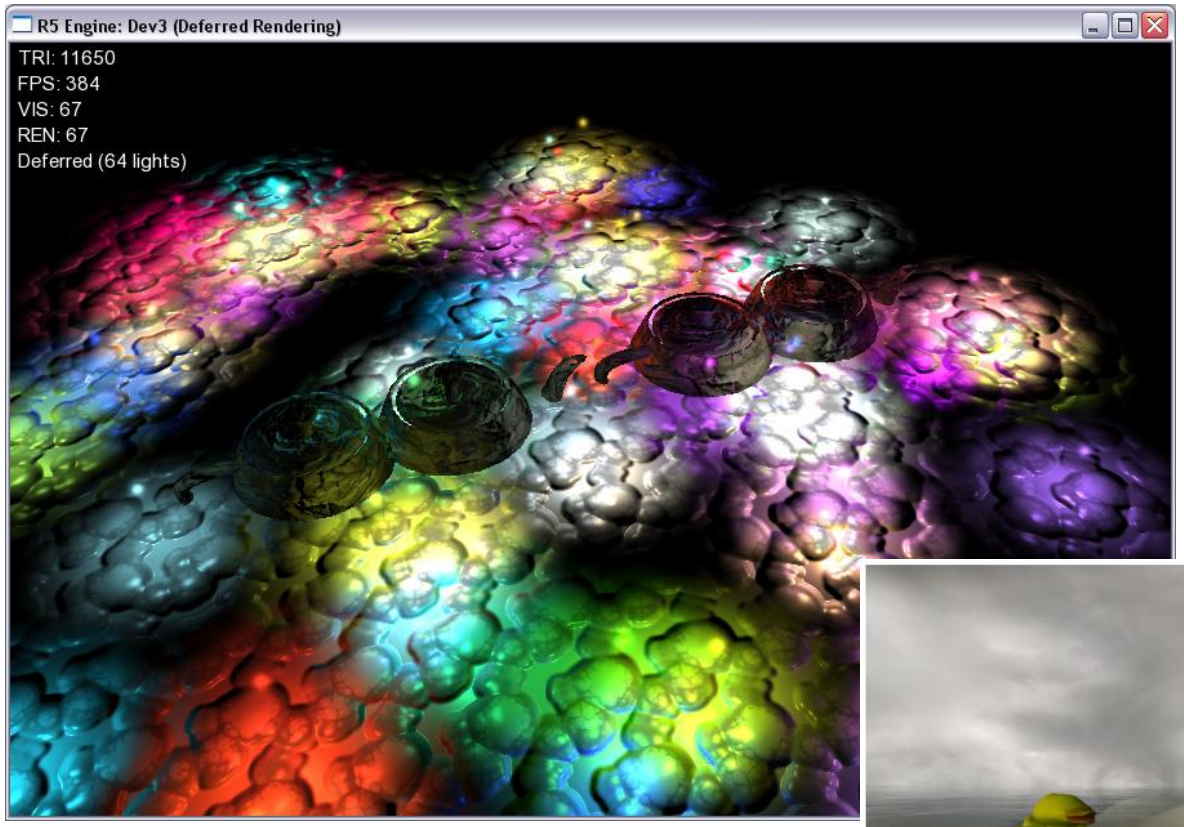
Projet technique

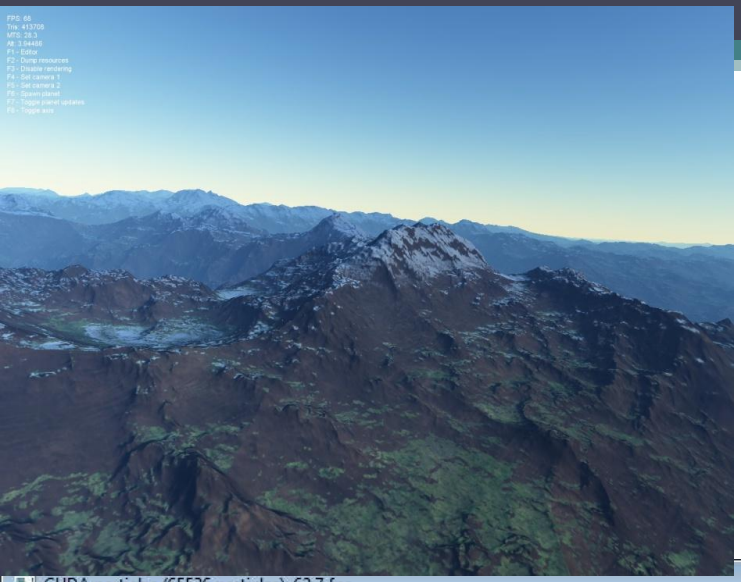
- Exemple de sujet/thème
  - Système de particule (beau, efficace et générique)
  - Affichage d'eau.
  - Algorithme de niveau de détail de terrain.
  - Affichage d'herbe.
  - Multiple effet (Motion blur, Screen Space Ambient Occlusion, Depth of Field, Glow, HDR, Light Shaft)
  - Affichage de ciel
  - Ombre
  - Rendu non réaliste
  - Deferred Rendering, Light Prepass
  - GI (Virtual Point Light, Light Propagation Volume)

La notation

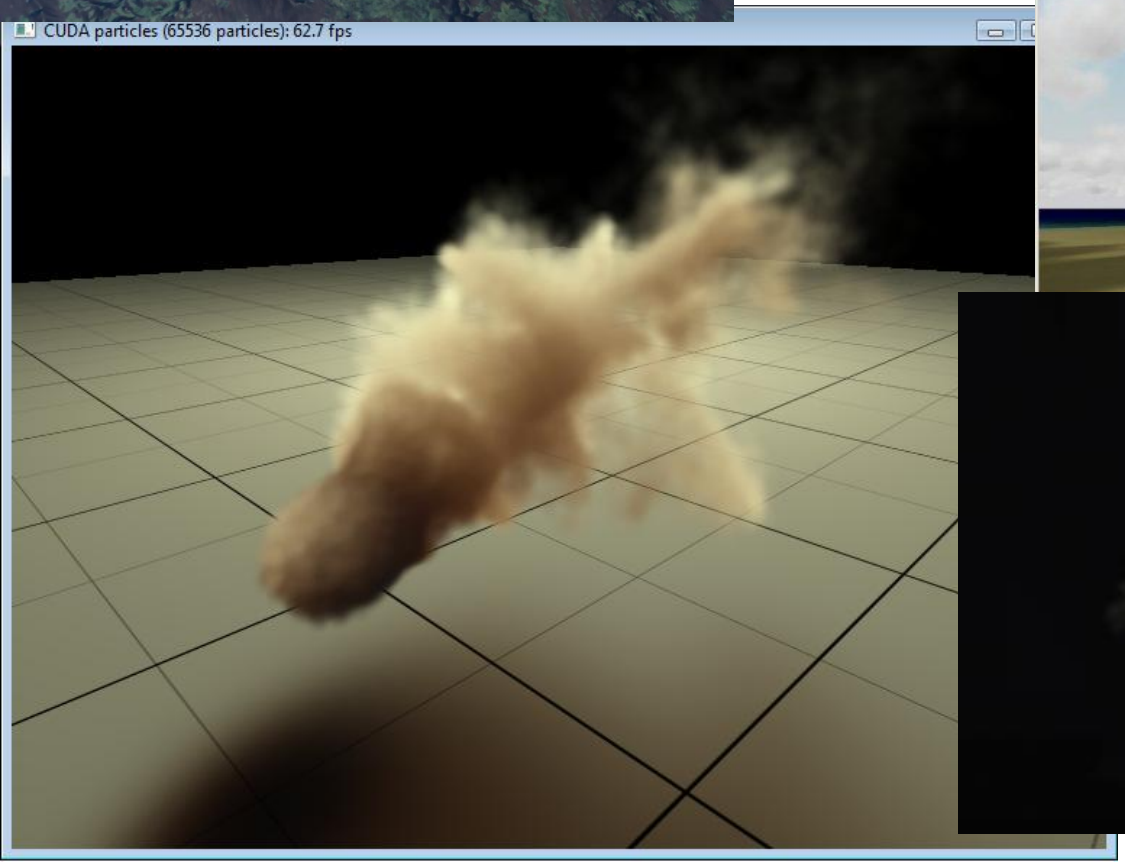
- Ce qui rentre en compte dans la notation (liste non exhaustive):
  - Le résultat visuel et technique
  - La qualité et compréhension du code

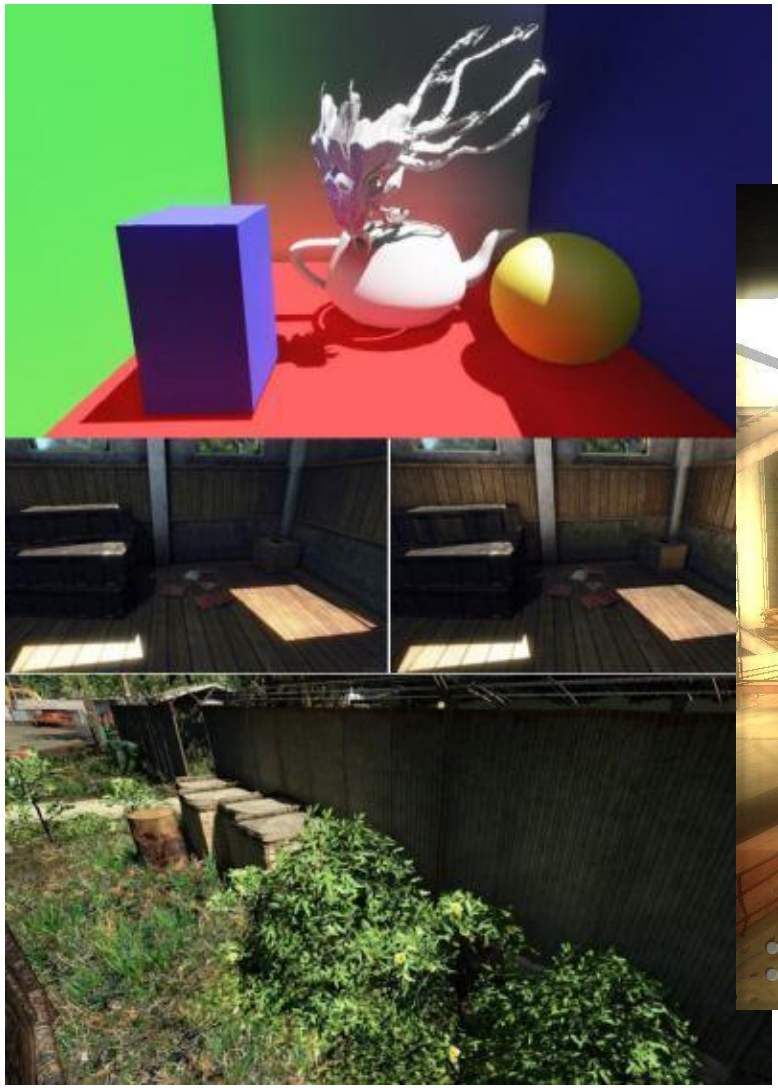
  - La somme mise sur mon compte en banque. Un RIB vous sera fourni plus tard


R5 Engine: Dev3 (Deferred Rendering)
TRI: 11650
FPS: 384
VIS: 67
REN: 67
Deferred (64 lights)




Framerate: 71.2 fps (123915 tris)


CUDA particles (65536 particles): 62.7 fps




::UNiGiNE
www.unigine.com

Points sur les TPs

- Il y en aura sans doute au moins un de noté
- Je ne veux pas vous donner des TPs « copier – coller »
- Pour bien faire un TP il vous faut:
  - Un pc sous windows, Visual Studio, le SDK DirectX
  - Le TP
  - Les slides
  - La documentation de DirectX