M2-Images

Intersections

J.C. Iehl

October 6, 2011

résumé des épisodes précédents . . .

afficher plusieurs objets :

- transformations,
- primitives planes, primitives non planes,
- notions de lumière, matières,
- ombres, reflets, transparence,
- couleur,

pas facile de calculer des ombres, des reflets, des transparences . . .

Objectif : lancer de rayons

algorithme :

- pour chaque pixel :
- déterminer la direction de la droite passant par l'observateur et le centre du pixel,
- calculer l'intersection de la droite avec les objets de la scène,
- ne conserver que la plus petite : l'objet visible, le plus proche de l'observateur,
- calculer l'énergie arrivant sur l'observateur à travers le pixel,
- déterminer la couleur correspondante.

Objectif : lancer de rayons

et alors ?

- mêmes idées que l'affichage par découpage + fragmentation,
- mais la réalisation est différente (*pipeline* différent),
- très simple de savoir si 2 points se "voient" et de calculer le transfert d'énergie.

solution directe :

pour les ombres, les reflets, etc.

comment calculer l'intersection d'une droite avec les objets de la scène ?

Objectif: calcul d'intersections

cas simples :

- plan,
- triangle,
- sphere,
- cube.

question bonus :

et les primitives non planes (PN Triangle) ?

Intersection : rayon / plan

rappels :

- $p(t) = o + t \cdot \vec{d}$,
- $\vec{n} \cdot \overrightarrow{ap} = 0$ sur le plan de normale $\vec{n}$ passant par a et $p \in plan(a, \vec{n})$.

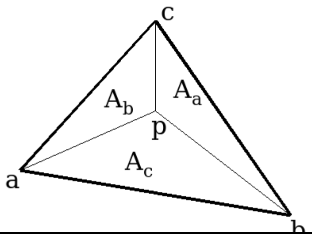
résultat :

- $\vec{n} \cdot \overrightarrow{ap(t)} = 0$
- $\vec{n} \cdot ((o + t \cdot \vec{d}) - a) = 0$
- $\vec{n} \cdot ((o - a) + t \cdot \vec{d})) = 0$
- $t = \frac{(a-o) \cdot \vec{n}}{\vec{d} \cdot \vec{n}}$

Intersection : rayon / triangle

rappels :

- $p(t) = o + t \cdot \vec{d}$,
- $p(\alpha, \beta) = \alpha a + \beta b + (1 - \alpha - \beta)c$ si $p \in triangle(a, b, c)$
- $\alpha = A_a/A$, $\beta = A_b/A$
- $\gamma = 1 - \alpha - \beta = A_c/A$



Intersection : rayon / triangle

résultat :
"Fast, Minimum Storage Ray-Triangle Intersection"
code + détails

Intersection : rayon / sphere

rappels :

- $p(t) = o + t \cdot \vec{d}$,
- $(p_x - c_x)^2 + (p_y - c_y)^2 + (p_z - c_z)^2 - R^2 = 0$ si $p \in sphere(c, R)$
- $(p - c) \cdot (p - c) - R^2 = 0$

résultat :

- $(o + t \cdot \vec{d} - c) \cdot (o + t \cdot \vec{d} - c) - R^2 = 0$
- $(\vec{d} \cdot \vec{d})t^2 + 2\vec{d} \cdot (o - c)t + (o - c) \cdot (o - c) - R^2 = 0$

Intersection : rayon / boite orientée

rappels :

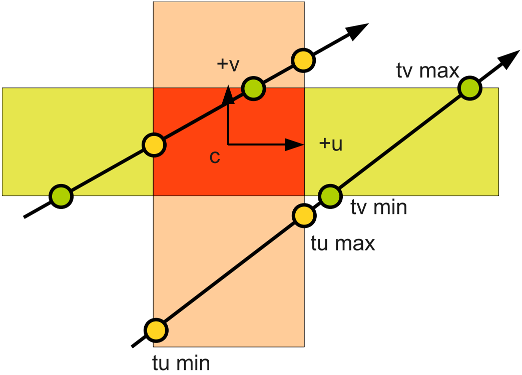
- $p(t) = o + t \cdot \vec{d}$,
- $p \in boite(c, \vec{u}, \vec{v}, \vec{w})$
- la boite est l'intersection de 3 paires de plans parallèles :
- $plan(c - \vec{u}, -\vec{u})$, $plan(c + \vec{u}, \vec{u})$,
- $plan(c - \vec{v}, -\vec{v})$, $plan(c + \vec{v}, \vec{v})$,
- $plan(c - \vec{w}, -\vec{w})$, $plan(c + \vec{w}, \vec{w})$,

résultat :

- calculer t_{min} et t_{max} pour chaque paire de plans (cf. intersection rayon / plan),

Intersection : rayon / boite orientée

- l'intersection existe si l'intersection des intervalles n'est pas vide : $max(t^u_{min}, t^v_{min}, t^w_{min}) < min(t^u_{max}, t^v_{max}, t^w_{max})$



Intersection : rayon / boite alignée sur les axes

détails et astuces de calculs :
"An Efficient and Robust Ray-Box Intersection Algorithm"
code

Intersection : rayon / PN Triangle

ou se trouve la surface du pn triangle ?

- idée 1 : se ramener à un cas simple,
- idée 2 : ?

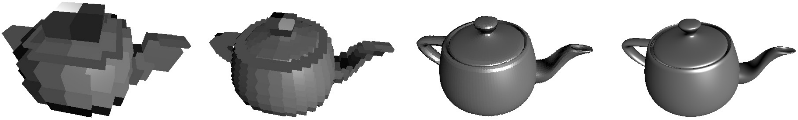
Intersection : rayon / PN Triangle

idée 1 : se ramener à un cas simple

- subdivision dans le domaine paramétrique (cf. tp1),
- construire un englobant de chaque sous domaine,
- tester les englobants,
- subdiviser les sous domaines dont l'englobant intercepte le rayon.

"Improving Ray Tracing Precision by Object Space Intersection Computation"
H. Dammertz, A. Keller, irt 2006

Intersection : rayon / PN Triangle



subdivision régulière d'une surface "complexe".

Intersection : rayon / PN Triangle

critère d'arret de la subdivision :

- fixer un nombre de subdivisions ?
- fixer une dimension maximale ? dans l'espace objet ?
- mais : problèmes de raccord, cf cm3 sur les pn triangles.

continuer jusqu'à la limite de précision des *float*.
la division par 2 est une opération *exacte* sur des floats.

Objectif : plusieurs objets

pour chaque pixel (x, y)

- générer le rayon $r_{(x,y)} = (o(x, y),\ \overrightarrow{d(x, y)})$,
- pour chaque objet :
- calculer l'intersection du rayon et de l'objet, t_{objet}
- si $t_{objet} < t_{min}$
- $t_{min} = t_{objet}$
- calculer la position du point le long du rayon : $r_{(x,y)}(t_{min})$

Objectif : ombres

pour chaque point visible :

- créer un rayon vers la source de lumière,
- calculer les intersections avec les objets de la scène,
- s'il y a une intersection :
- le point est à l'ombre,
- sinon, il est éclairé.

on peut aller plus vite : il suffit de vérifier l'existance d'une intersection.

sources non ponctuelles : pénombres ...

Objectif : reflets

pour chaque point sur un miroir :

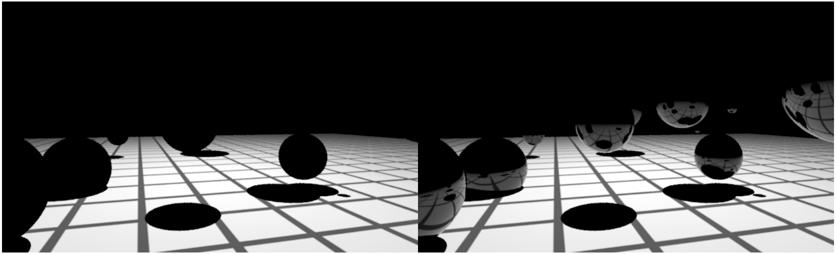
- calculer la direction réfléchie,
- trouver le point le plus proche visible dans cette direction,
- calculer l'énergie qu'il "envoie" vers le point sur le miroir,
- éventuellement recommencer, si le point est encore sur un miroir ...

l'algorithme complet est récursif.

Exemples :

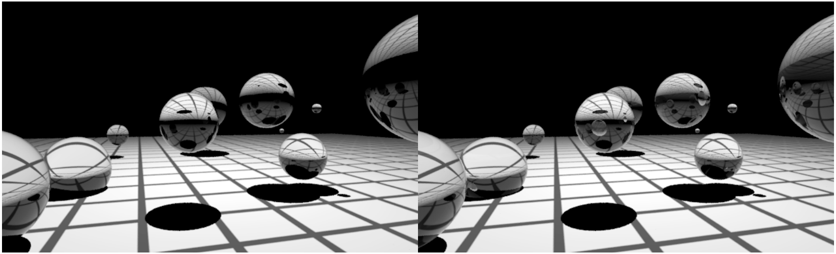
profondeur 10 $== 2^{10}$ chemins explorés par pixel...

Exemples :



profondeur 1 (à gauche), profondeur 2 (à droite)

Exemples :



profondeur 3 (à gauche), profondeur 6 (à droite) $== 2^6$ chemins explorés par pixel...

Accélération

quelle complexité ?

- pour chaque pixel : calculer les intersections avec tous les objets !
- plus les rayons pour les ombres, les reflets . . .

idée :

grouper les objets et ne tester les objets du groupe que si l'englobant du groupe intercepte le rayon.