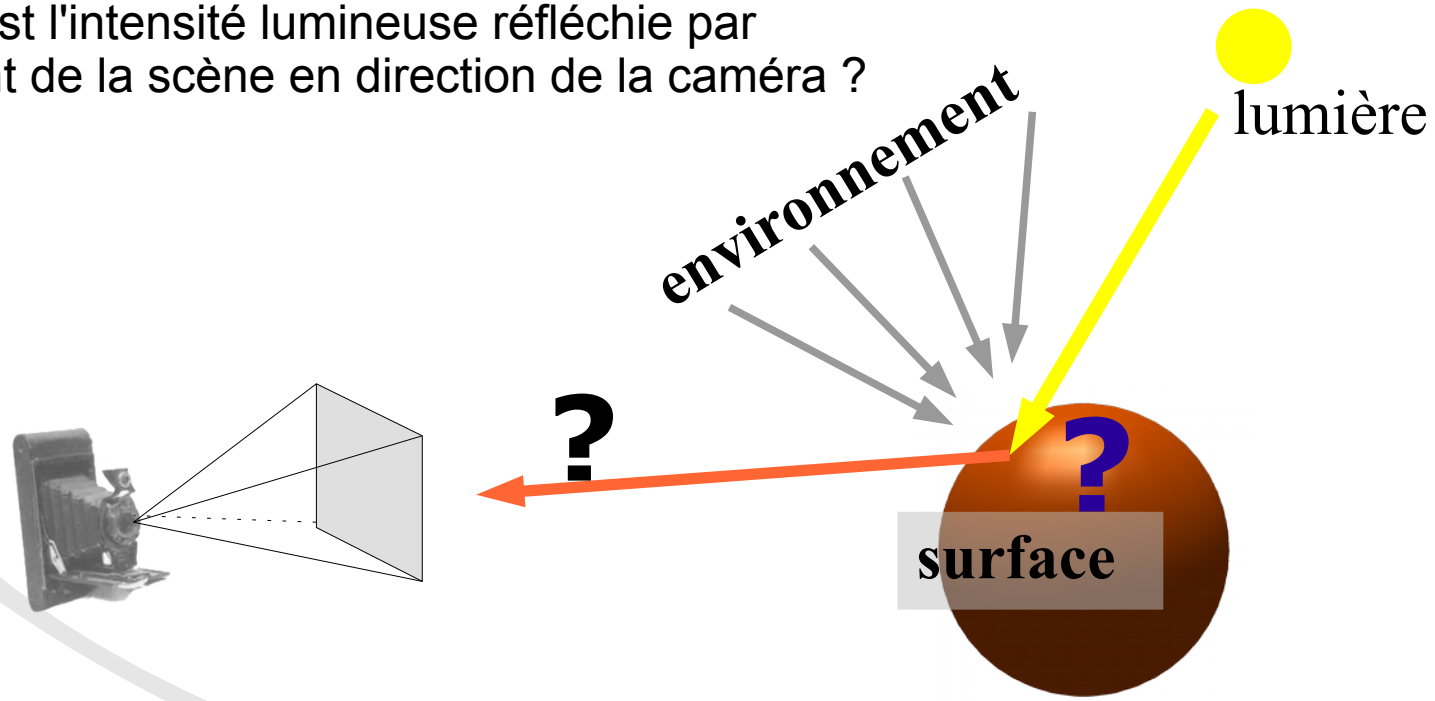
Shadows

gael.guennebaud@inria.fr

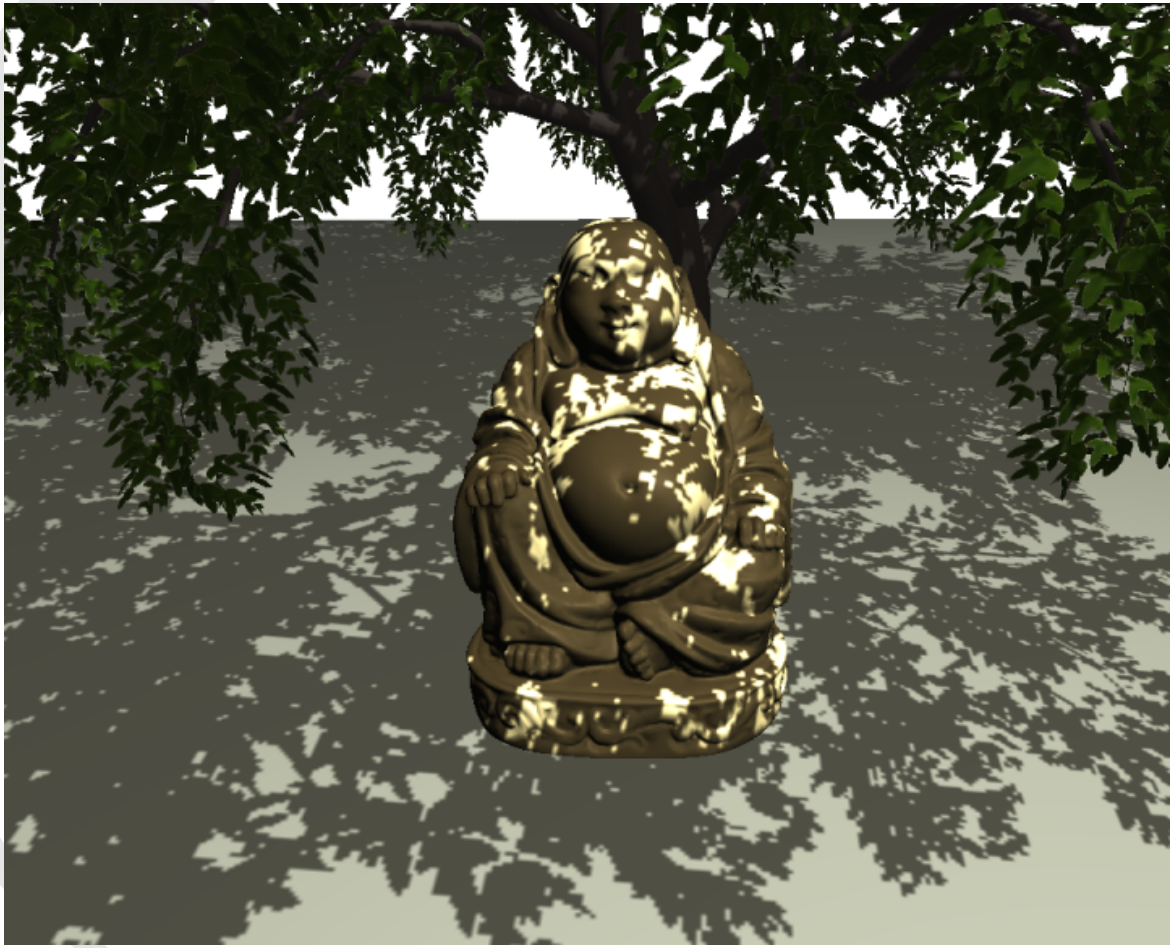
Simulation de l'éclairage

- **Simule la façon dont un objet “reflète” la lumière**
  - Propriétés de la surface
  - Propriétés des sources lumineuses
  - Propriétés de l'environnement
- **Problématique**
  - Quelle est l'intensité lumineuse reçu par chaque pixel de la caméra ?
  - i.e. Quelle est l'intensité lumineuse réfléchie par chaque point de la scène en direction de la caméra ?



→ réalisme

→ réalisme



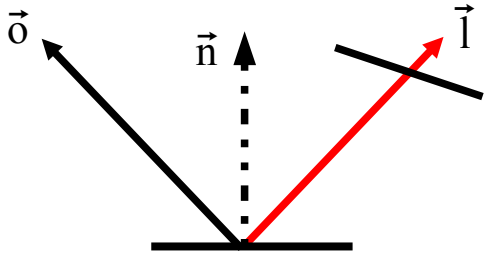
Shadows
-
Ombres portées

gael.guennebaud@inria.fr

Éclairage local & Visibilité

- **Équation**

$$\text{ambiant} + \sum_l \langle \vec{l}, \vec{n} \rangle \rho(\vec{l}, \vec{o}) L_l(s)$$



- **Que faire si la source n'est pas visible**
  - Terme de Visibilité
    - 0 si la source est cachée
    - 1 si visible

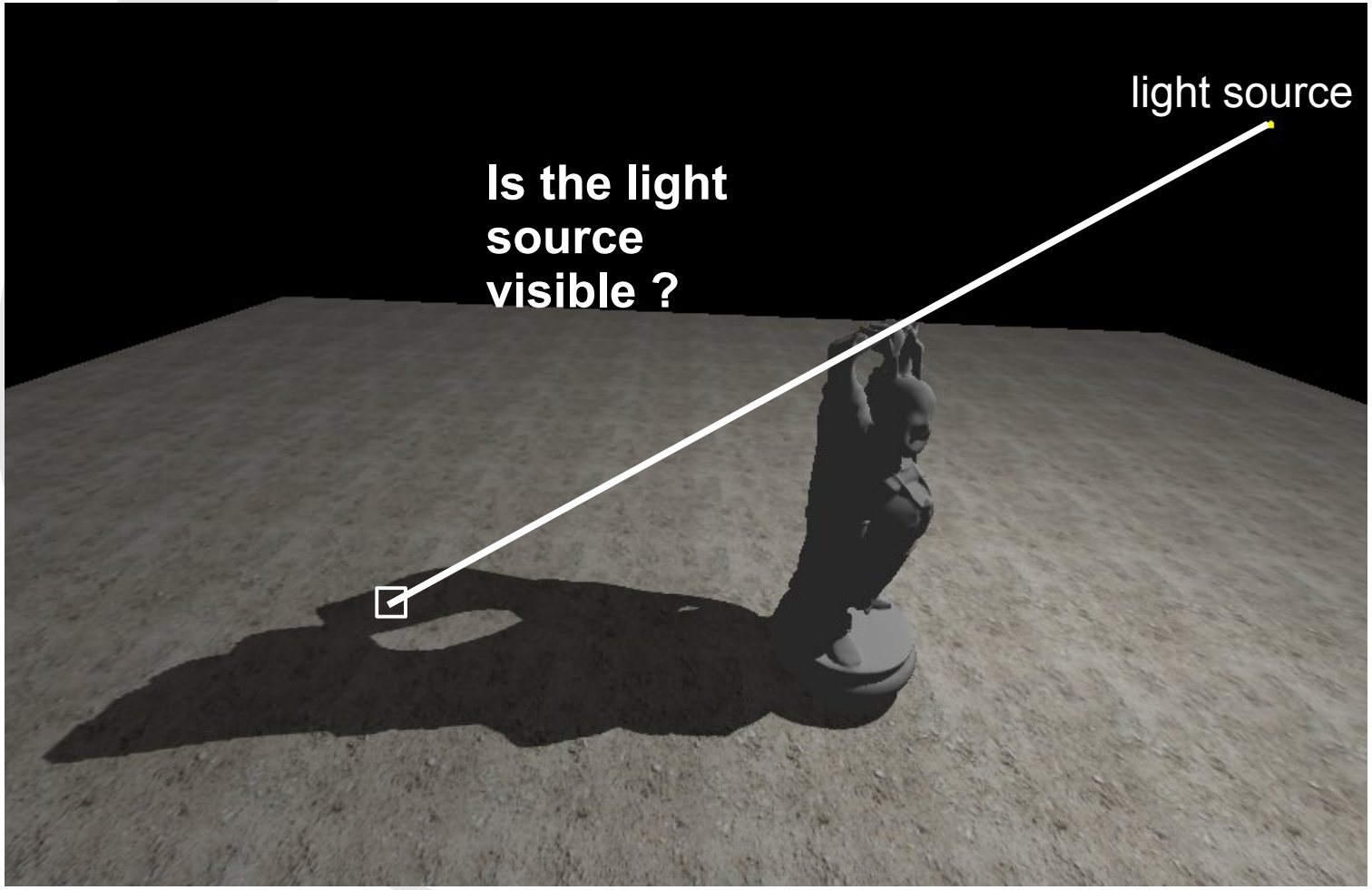
$$\text{ambiant} + \sum_l \langle \vec{l}, \vec{n} \rangle \rho(\vec{l}, \vec{o}) \mathbf{V(l,s)} L_l(s)$$

- **Notion d'ombre**
  - Zone de l'espace où la source de lumière est invisible

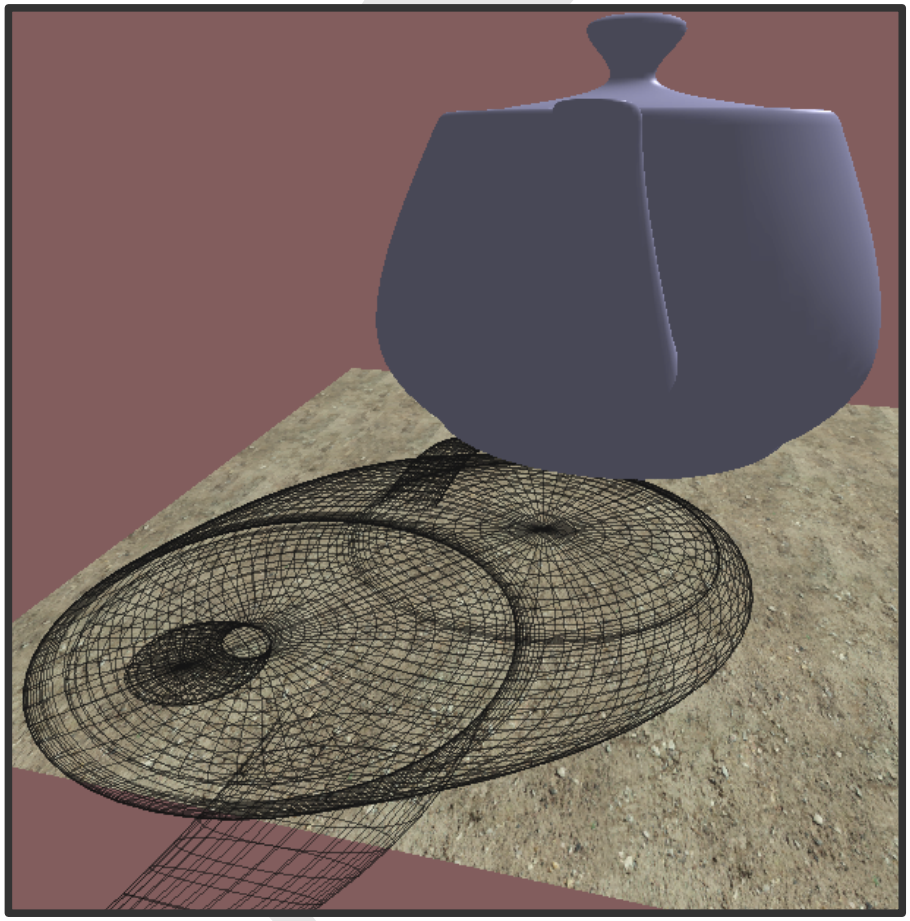
(vidéos)

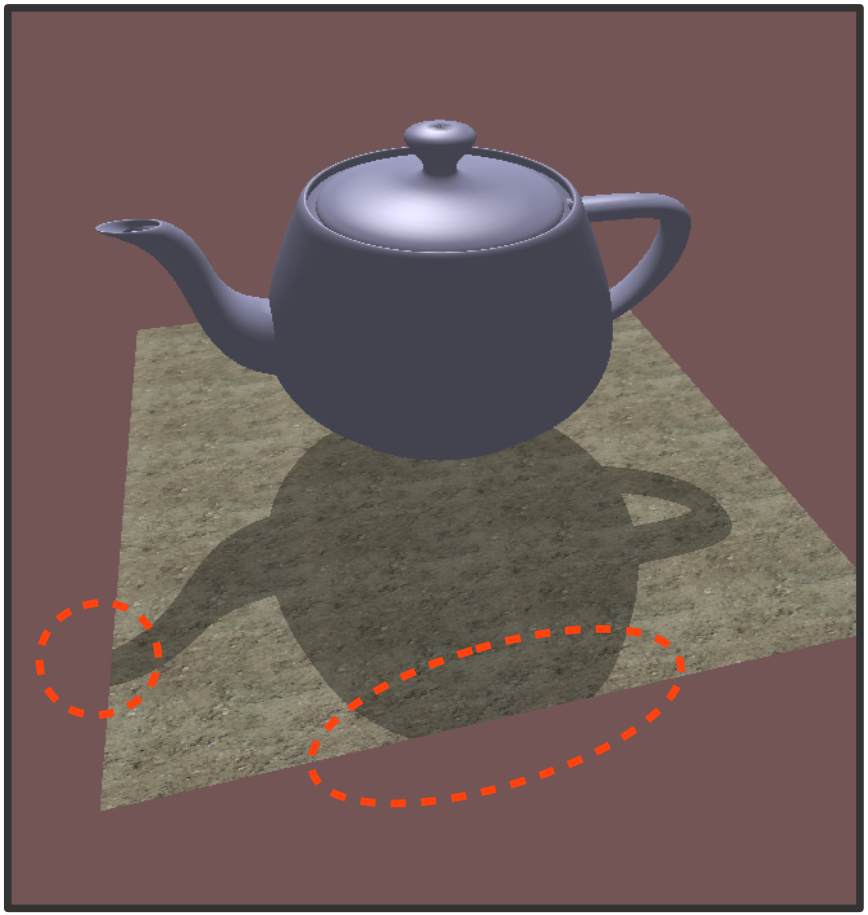
Définition : cas ponctuel

- **Visibilité par rapport à la source de lumière**
  - Visible (= pas d'objet occultant) => dans la lumière
  - Invisible (= au moins un objet occultant) => dans l'ombre



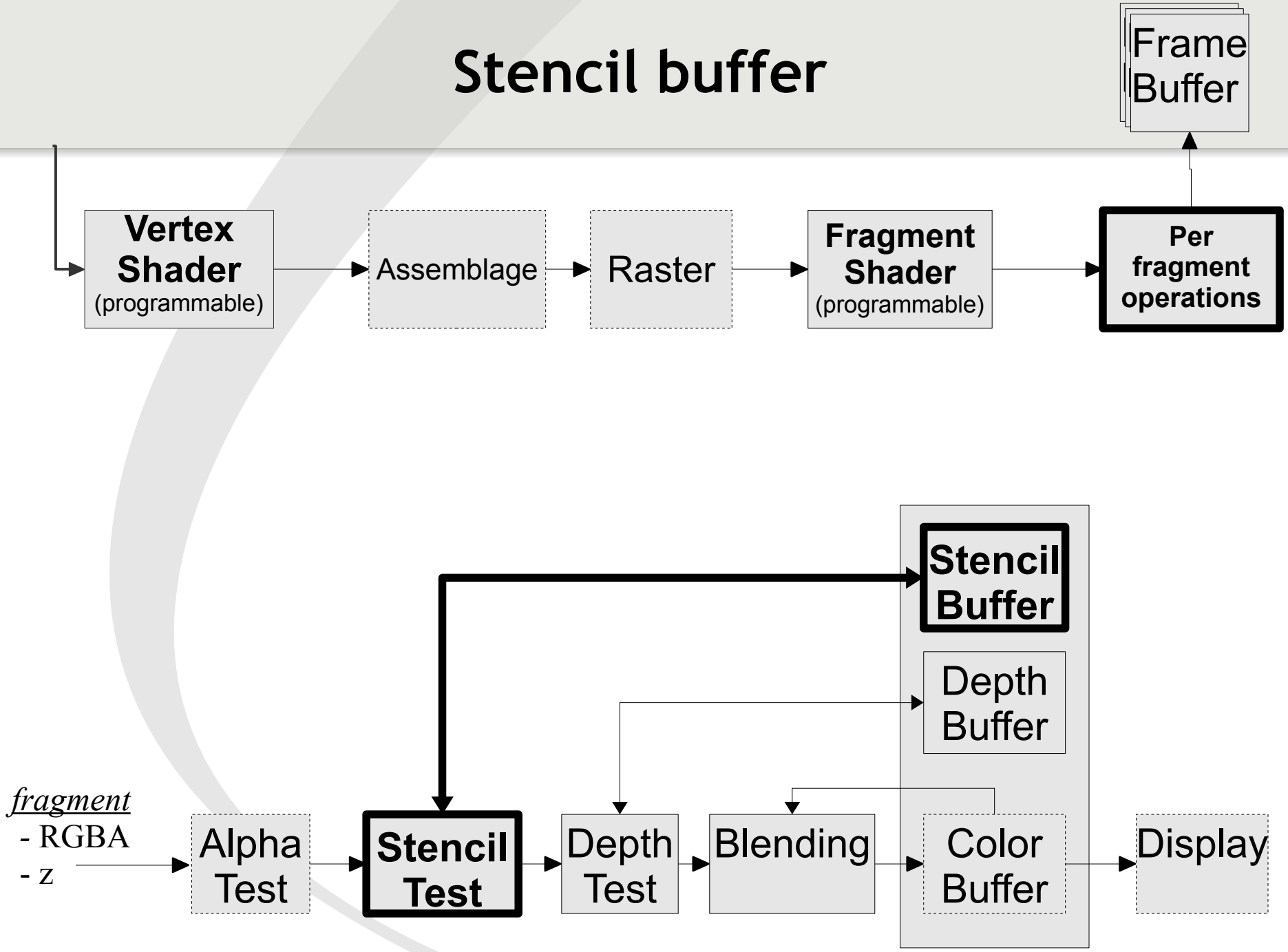
Ombres projetées





→ Simple mais limité aux receveurs plans
→ Questions : comment limiter l'ombre au receveur ?
comment ajouter les ombres une seules fois ?

Stencil buffer



Utilisation du stencil buffer

- **Test par pixel sur la valeur du stencil buffer**
  - Fragment rejeté si test raté
  - Test effectué avant le test de profondeur

Test = func (ref & mask , svalue & mask)

  - & = ET logique bits à bits
  - mask = entier 32 bits
  - ref = valeur de référence (! ce n'est pas un attribut du fragment !)
  - func = GL_{ALWAYS, NEVER, LESS, EQUAL, GREATER, NOTEQUAL, GEQUAL, LEQUAL}
  - svalue = valeur dans le stencil buffer
  - voir : glStencilFunc(), glEnable/Disable(GL_STENCIL_TEST), etc.

Utilisation du stencil buffer

- **Différentes opérations sur la valeur dans le stencil buffer selon**
  - test de stencil raté (fail)
  - test de profondeur raté (zfail)
  - test de profondeur réussi (zpass)
  - glStencilOp(fail, zfail, zpass)

- **Opérations possibles :**
  - GL_KEEP : garder la valeur
  - GL_REPLACE : remplacer par la valeur référence
  - GL_ZERO : remplacer par zéro
  - GL_INCR, GL_DECR : incrémenter, décrémenter
    (avec saturation, càd: 255++ => 255 et 0-- => 0)
  - GL_INVERT : inverser bit à bit
  - GL_INCR_WRAP, GL_DECR_WRAP : incrémenter, décrémenter
    (avec bouclage sur l'octet, càd: 255++ => 0 et 0-- => 255)

- **Variantes pour faces avant/arrière** (glStencilOpSeparate/glStencilFuncSeparate)

Exemple : les ombres projetés

- **Vider le stencil buffer (mettre à zéro)**

- **Tracer les ombres projetées sur le plan avec :**
  - mise à jour du z-buffer et color-buffer désactivée
  - mise à jour du stencil à 1 pour les pixels correspondants

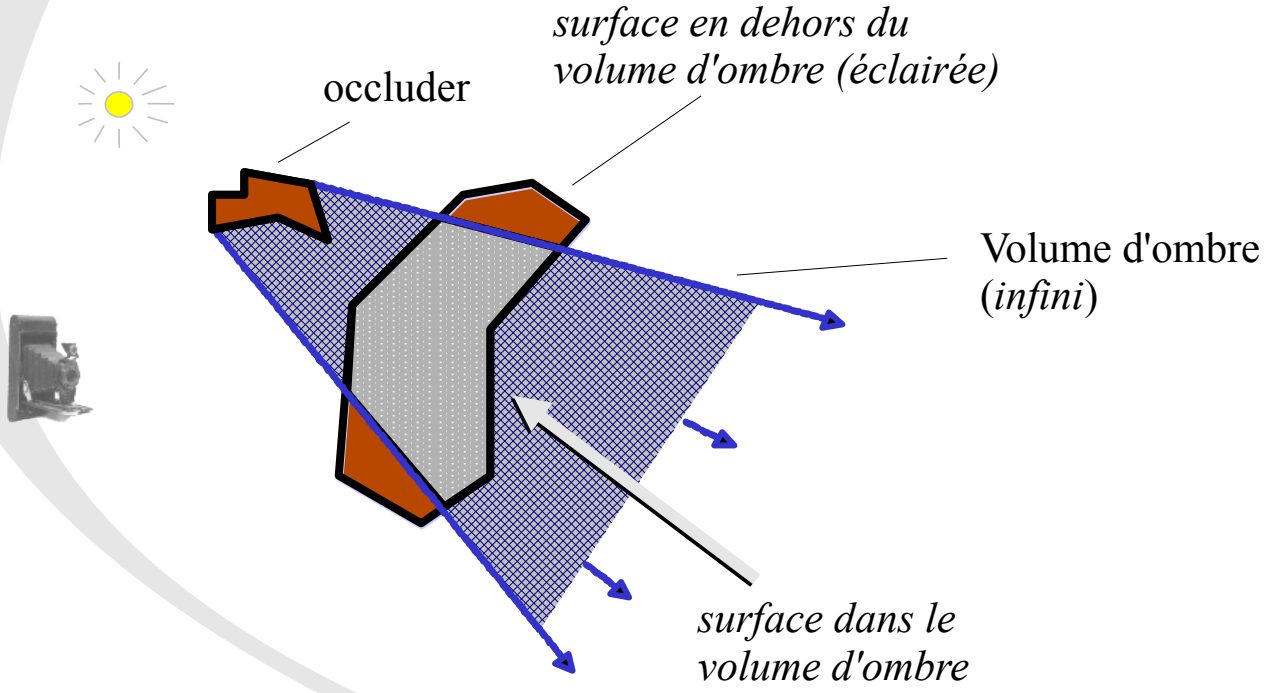
- **Tracer le plan receveur une 1ere fois avec :**
  - ambiant uniquement
  - test stencil activant uniquement les pixels à 1

- **Tracer le plan receveur une 2nd fois avec :**
  - éclairage complet
  - test stencil activant uniquement les pixels à 0

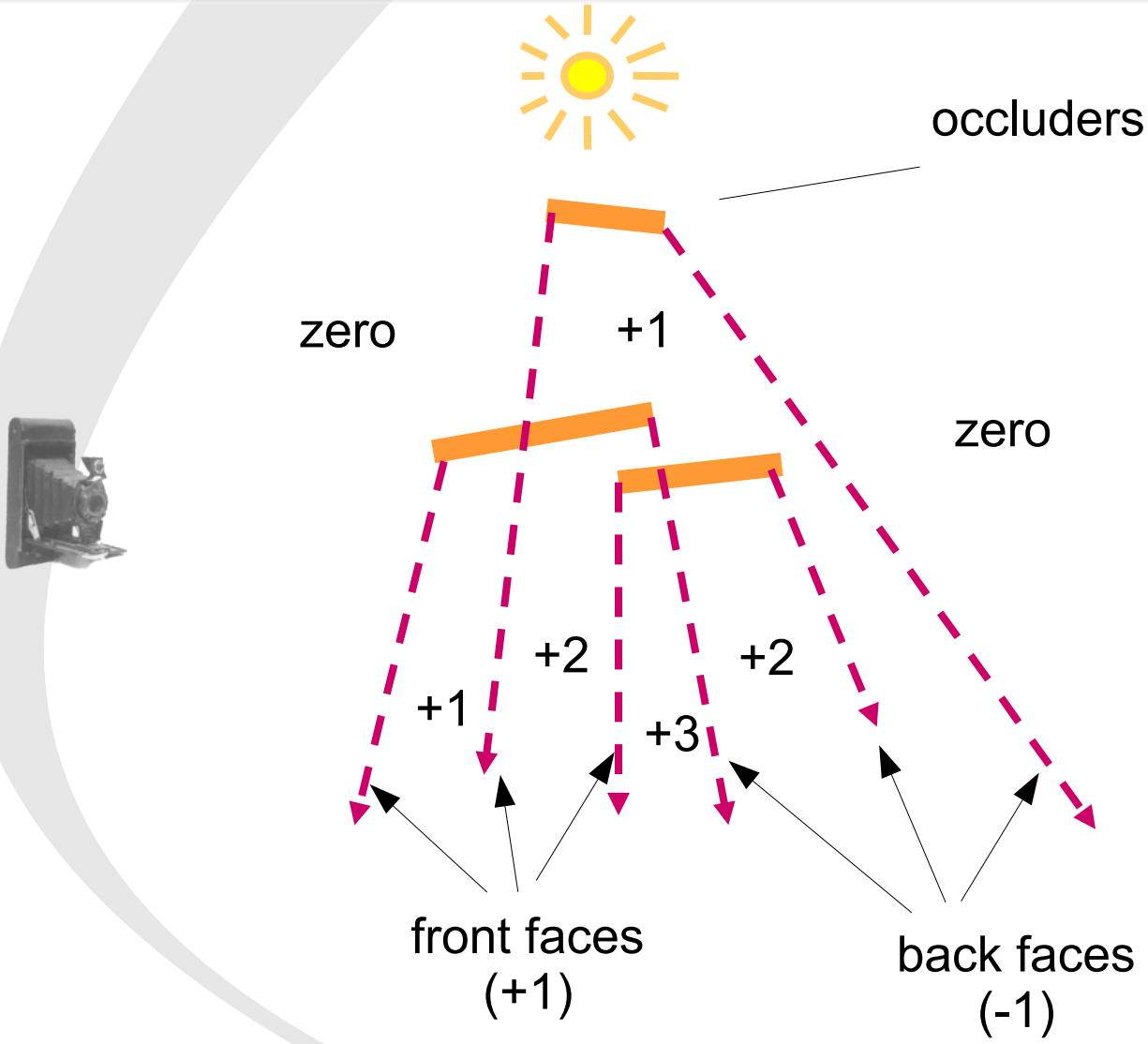
Shadow Volumes

Volume d'ombre (Shadow Volume)

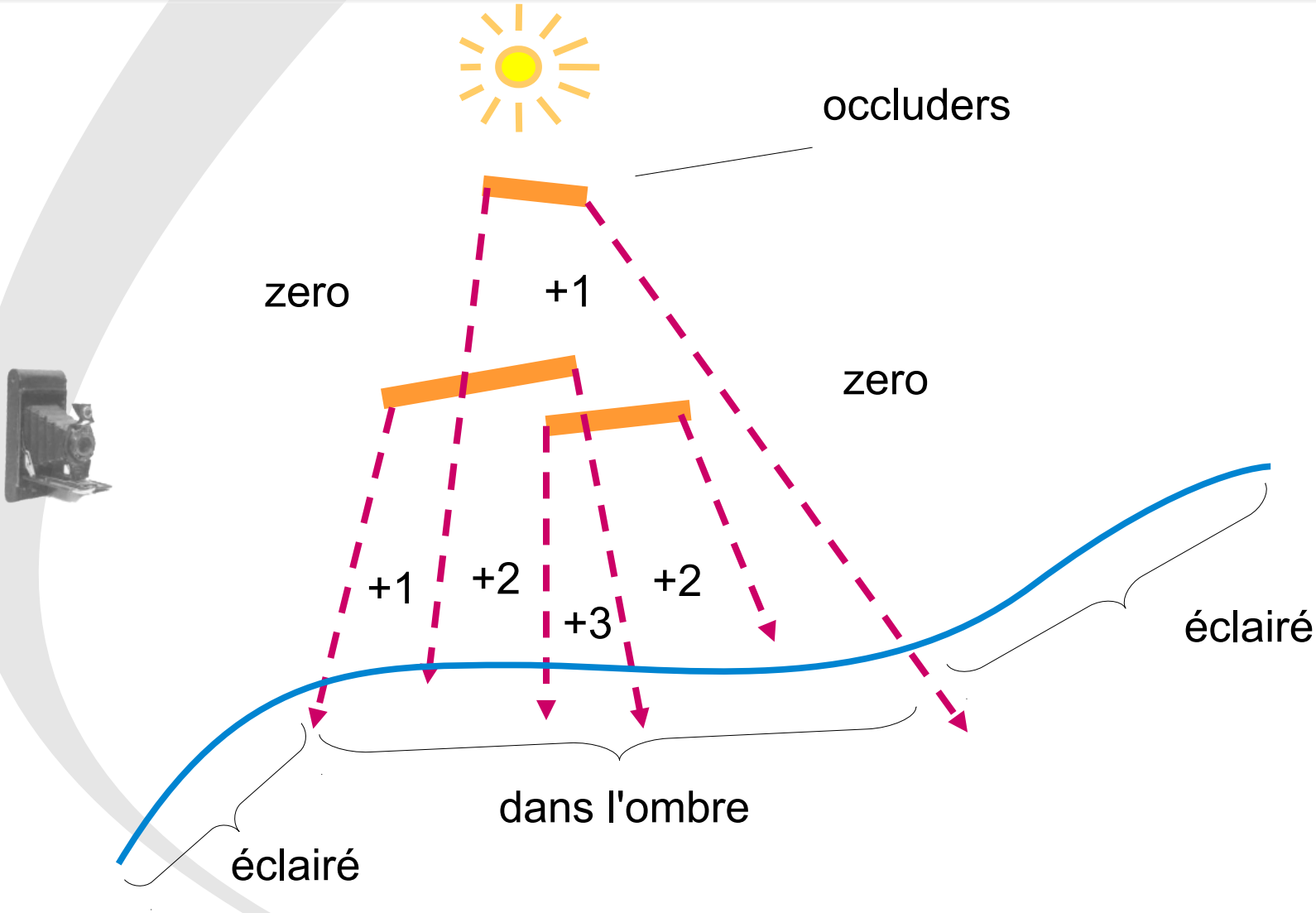
- **Principe de base**
  - Une lumière ponctuelle divise l'espace en deux :
    - les régions dans l'ombre
    - les régions éclairées
  - le **volume d'ombre** correspond à la frontière entre les deux régions



Comptage des entrées sorties



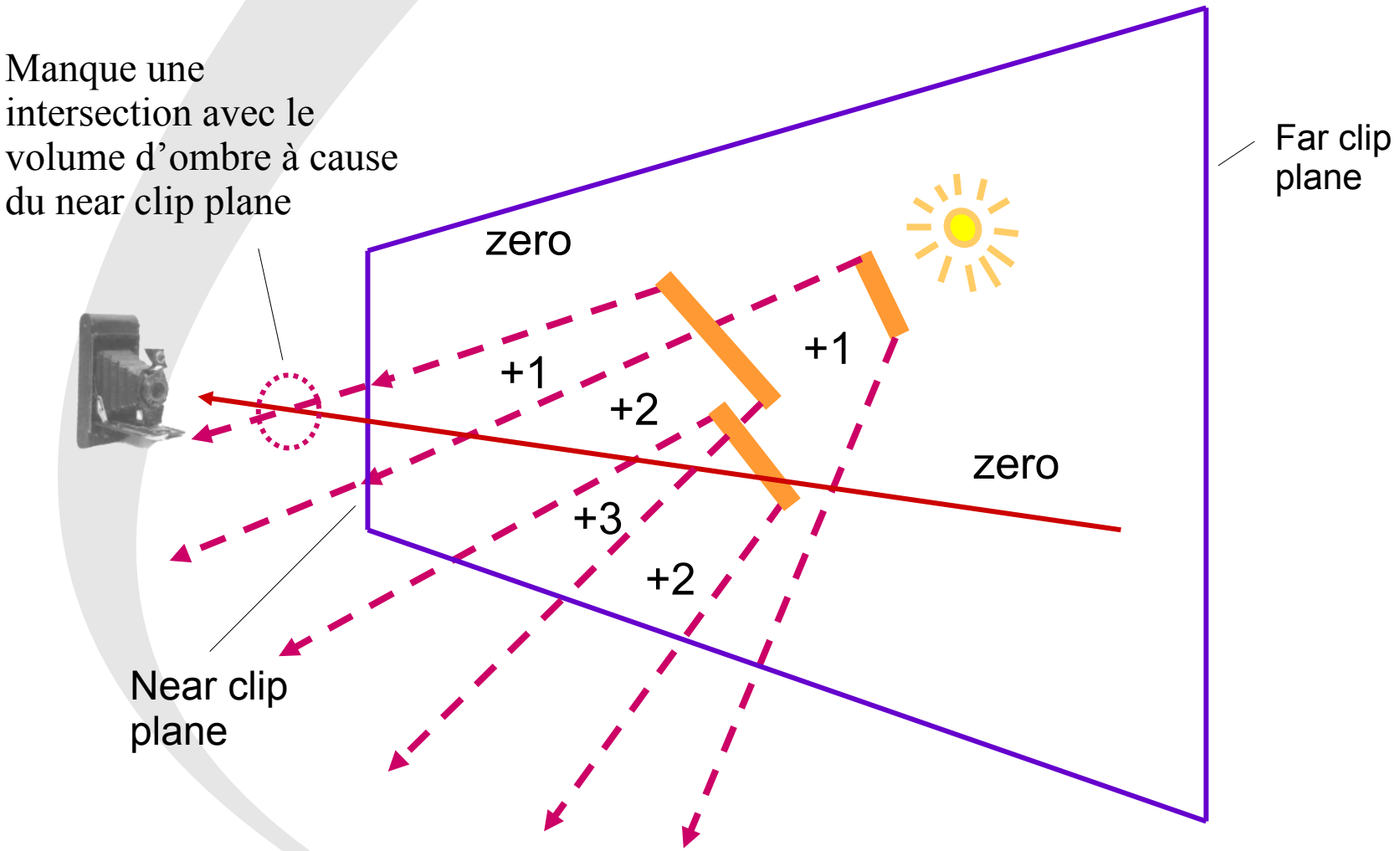
Comptage des entrées sorties



Algorithme (z-pass)

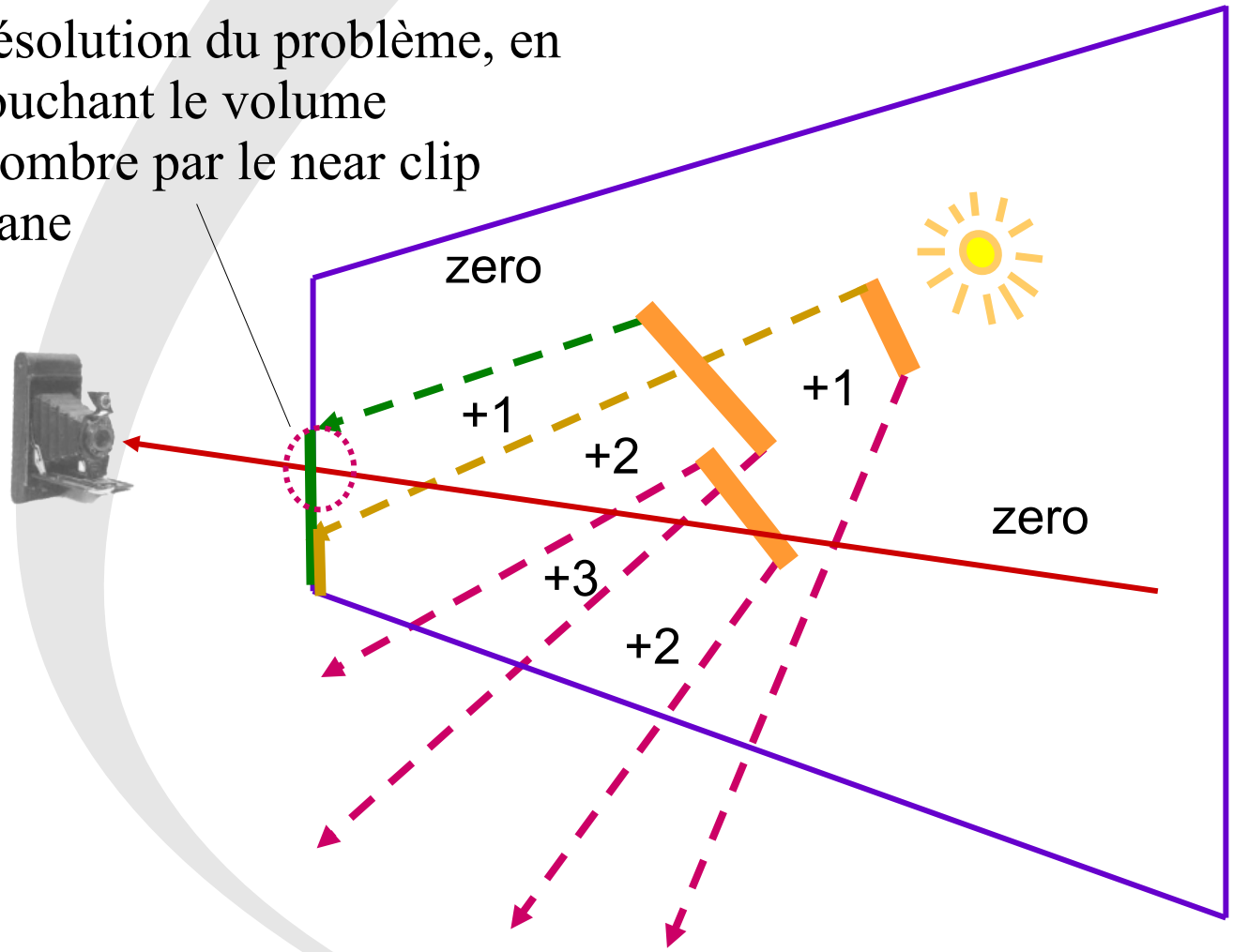
- **Initialiser le z-buffer**
  - par ex., tracer la scène lumière éteinte
- **Construction du Volume d'Ombre**
  - Détection de la silhouette vue de la lumière
- **Compter les entrées et sorties du volume d'ombre**
  - Configurer le stencil:
    - **back faces** : *décrémenter* le stencil si le test de profondeur réussi
    - **front faces** : *incrémenter* le stencil si le test de profondeur réussi
  - Tracer les faces du volume d'ombre
- **Tracer la scène lumière activée**
  - stencil != 0 → dans l'ombre !

Problème du near plane



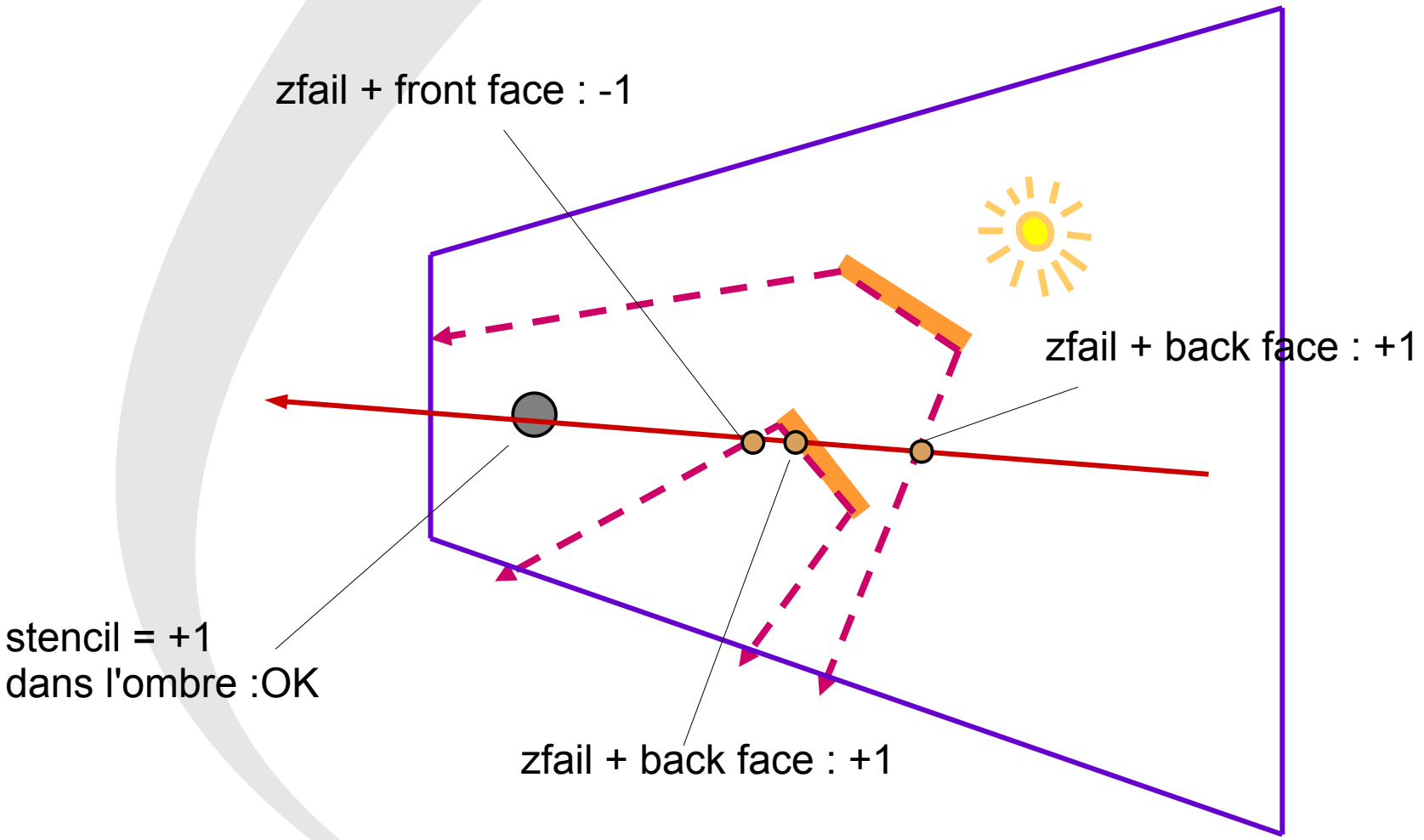
Problème du near plane

Résolution du problème, en bouchant le volume d'ombre par le near clip plane



La version en 3D de ce problème est beaucoup plus complexe que le cas 2D !

Approche zfail : compter après le receveur



Approche zfail

- **On se ramène à un problème de “far plane”, solution :**
  - mettre le plan *far* à l'infini (très loin !)
  - nécessite de boucher le volume d'ombre :
    - volume d'ombre += faces éclairées
      + faces éclairées à l'infini
  - plus coûteux que l'approche *zpass*
    → choisir selon l'objet l'approche *zpass* ou *zfail*

Algorithme (z-fail)

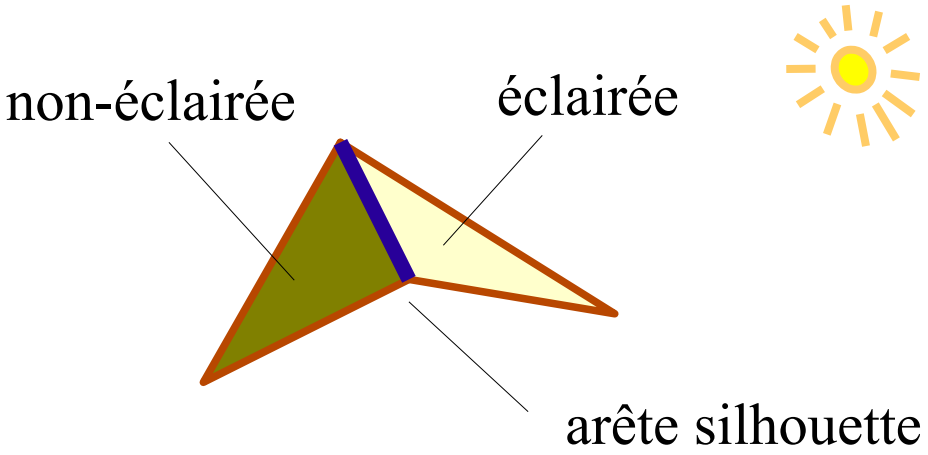
- **Initialiser le z-buffer (tracer la scène lumière éteinte)**
- **Construction du Volume d'Ombre**
  - Détection de la silhouette vue de la lumière
- **Compter les entrées et sorties du volume d'ombre**
  - Configurer le stencil:
    - **back faces** : *incrémenter* le stencil si le test de profondeur échou
    - **front faces** : *décrémenter* le stencil si le test de profondeur échou
  - Tracer les volumes d'ombres
- **Tracer la scène lumière activée**
  - stencil != 0 → dans l'ombre !

Calcul du shadow volume

- **Calcul de la silhouette**
  - Considérer toutes les arêtes potentiellement silhouette:
    - une face *éclairée* et une face *non-éclairée*
    - un bord (arête partagée par une seule face)
  - Structure d'arêtes (pré-calculée)

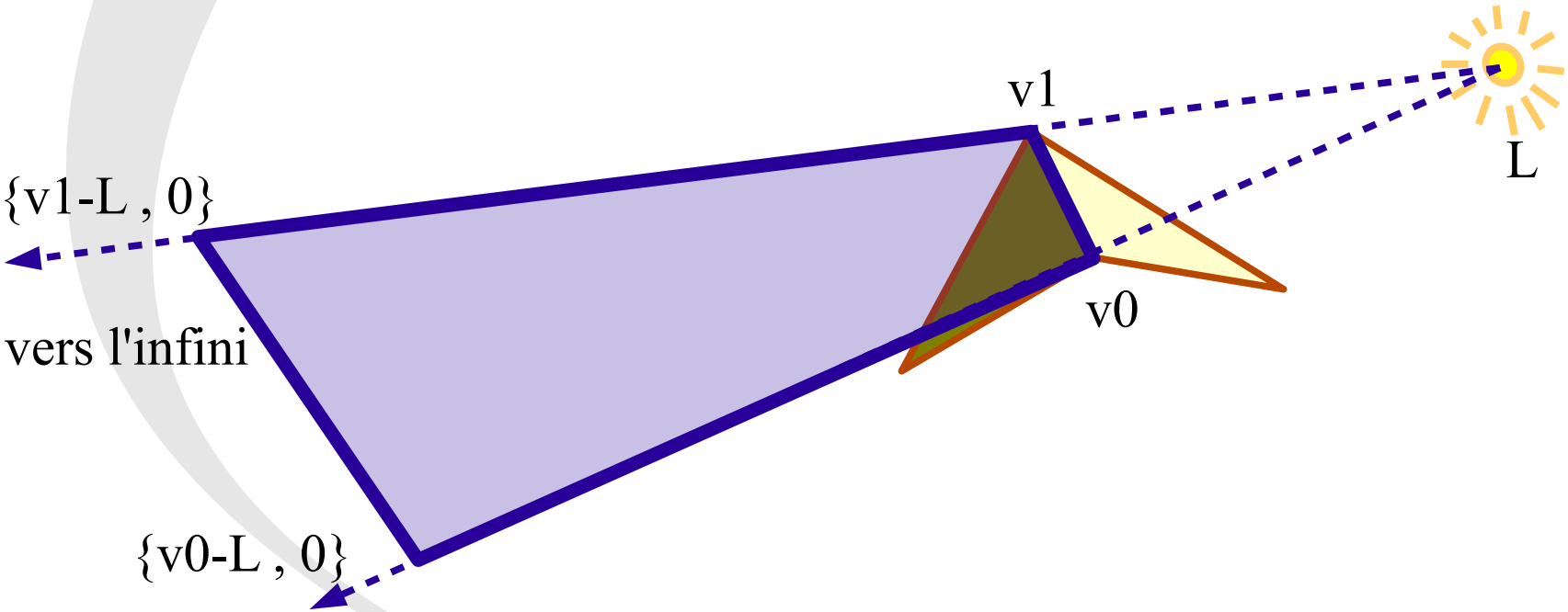
```
struct Edge {
  int facesId[2];          // index des 2 faces
  int verticesId[2];       // index des 2 extrémités
}
```

  - Face éclairée ou non ?
    - tester la position de la source lumineuse % au plan de la face (via l'équation du plan)



Rendu du shadow volume

- **Tracer un quad par arête potentiellement silhouette**
  - délimité par
    - les 2 extrémités de l'arête : *v0,v1*
    - les 2 extrémités de l'arête projetées à l'infini par rapport à la source lumineuse *L :*
      - *(v0-L)* et *(v1-L)* avec *w=0* (infini)



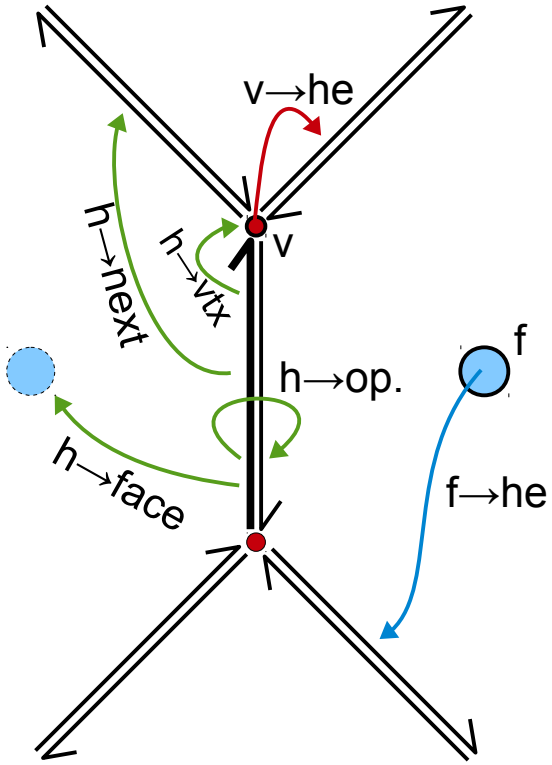
Halfedges

```
struct HalfEdge {
  HalfEdge     *next ;
  HalfEdge     *opposite ;
  Face         *face ;
  Vertex       *vertex;
};
struct Face { HalfEdge *he; };
```

```
struct Vertex { HalfEdge *he; };

struct Mesh {
  vector<HalfEdge>     halfedges ;
  vector<Vertex>       vertices;
  vector<Face>         faces;
} ;
```

- **Exercice : extraction de silhouette**

```
int n = mesh.halfedges.size();

for(i=0 ; i<n ; ++i) {


    HalfEdge *h = &mesh.halfedges[i];
    Face *f1 = ???;
    Face *f2 = ???;

    […]

}
```



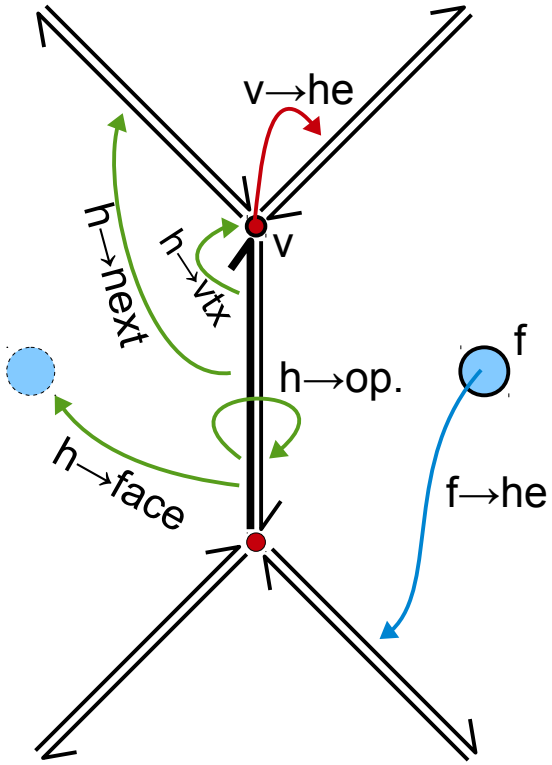
Halfedges

```
struct HalfEdge {
  HalfEdge     *next ;
  HalfEdge     *opposite ;
  Face         *face ;
  Vertex       *vertex;
};
struct Face { HalfEdge *he; };
```

```
struct Vertex { HalfEdge *he; };

struct Mesh {
  vector<HalfEdge>     halfedges ;
  vector<Vertex>       vertices;
  vector<Face>         faces;
} ;
```

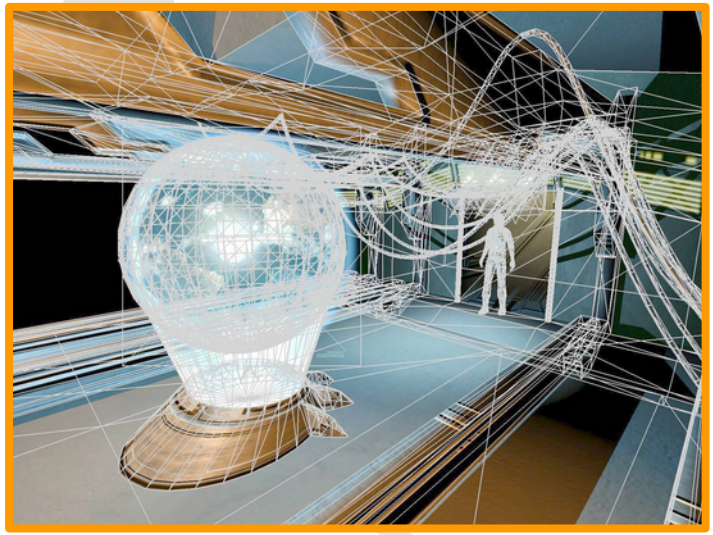
- **Exercice : extraction de silhouette**

```
int n = mesh.halfedges.size();
std::vector<bool> visited(n,false);
for(i=0 ; i<n ; ++i) {
  if( !visited[i] ) {
    visited[index(h->opposite)] = true;

    HalfEdge *h = &mesh.halfedges[i];
    Face *f1 = h->face;
    Face *f2 = h->opposite->face;

    […]
  }
}
```



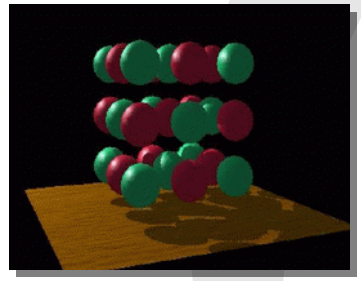
Shadow volumes : conclusion

- **Précision au pixel**
- **Sources ponctuelles /directionnelles**
- **Calcul de la silhouette**
  - Nécessite des objets manifold représentés par un maillage
- **La complexité de la géométrie des volumes d'ombre est supérieur à celle de la scène !**



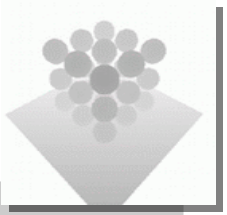
Shadow Mapping

Shadow Mapping



Dans l'ombre ?
Comparer la profondeur du point courant avec la profondeur stockée dans la shadow map

Source lumineuse (spot)



Shadow Map
(=texture de profondeur)

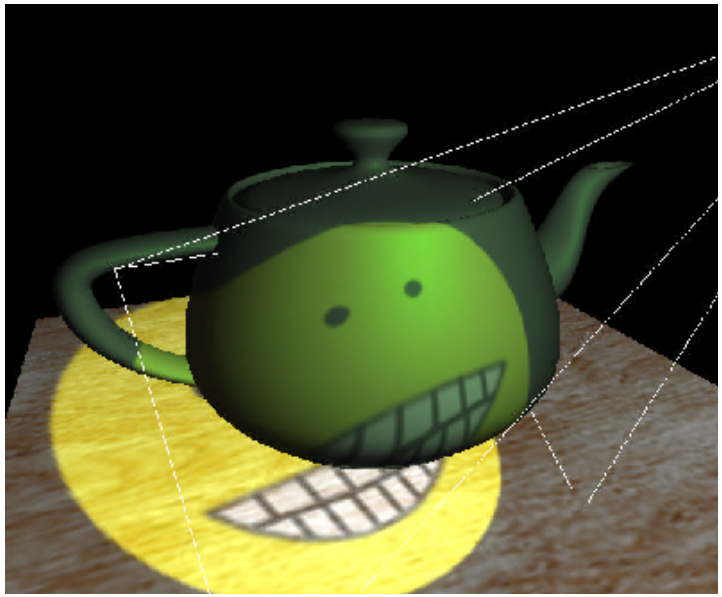
- contient la profondeur de tous les points visible à partir du spot
- calculée par OpenGL en traçant la scène à partir du spot

Point courant
doit être projeté
- dans le repère de la caméra
- & dans le repère du spot

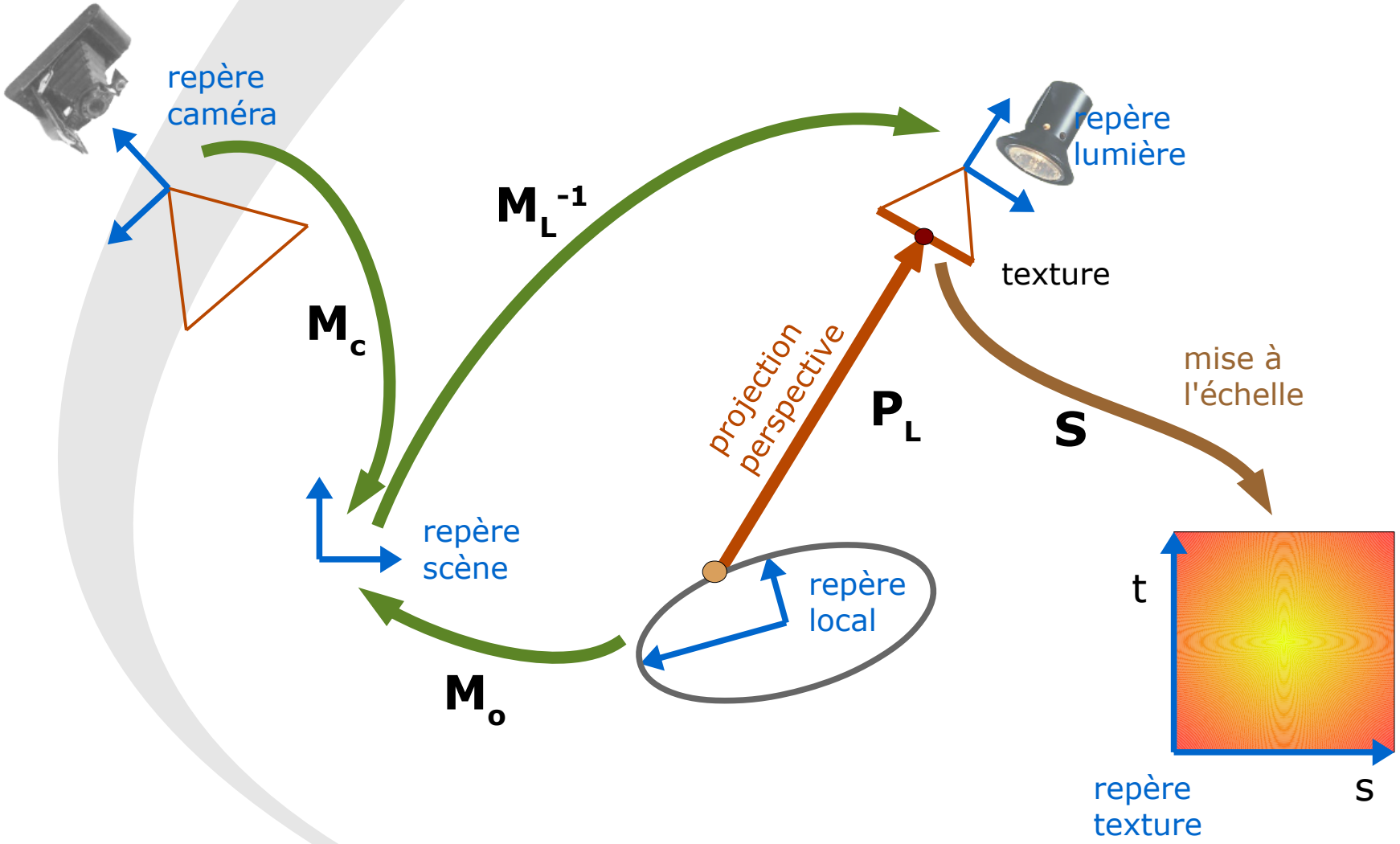
Algorithme

- **Algorithme**
  - Tracer la scène depuis la source de lumière en utilisant les FBO
    - Sauvegarder la profondeur dans une texture
  - Plaquer cette texture sur la scène
    - Texture projective
  - Pour chaque fragment/pixel depuis le point de vue
    - Comparer la distance à la source avec la profondeur issue de la texture

- **Texture projetées**

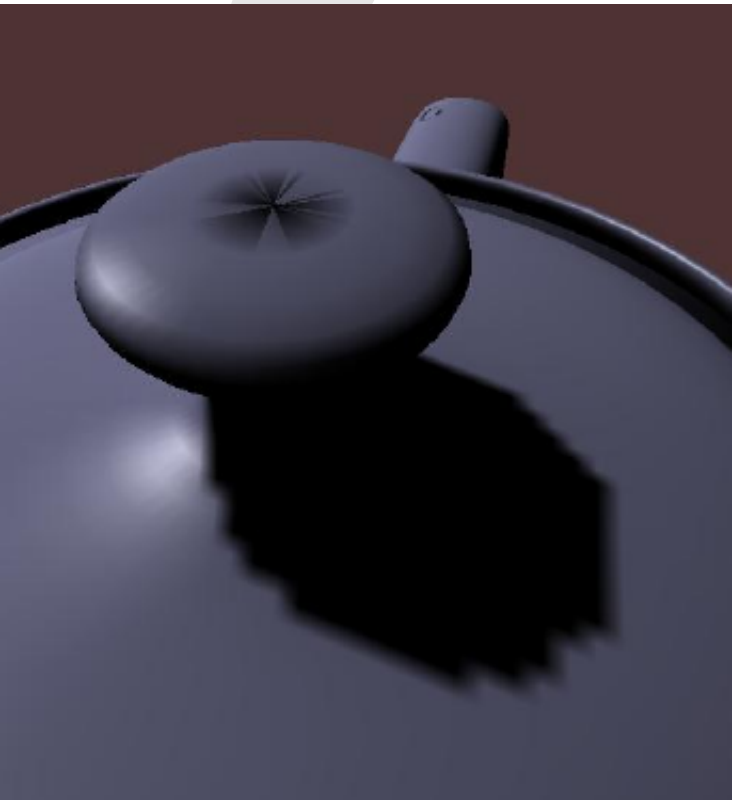


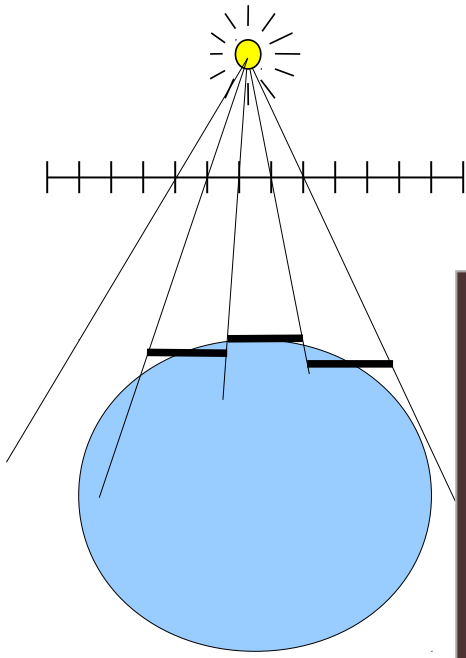
Shadow mapping



Shadow Mapping: aliasing

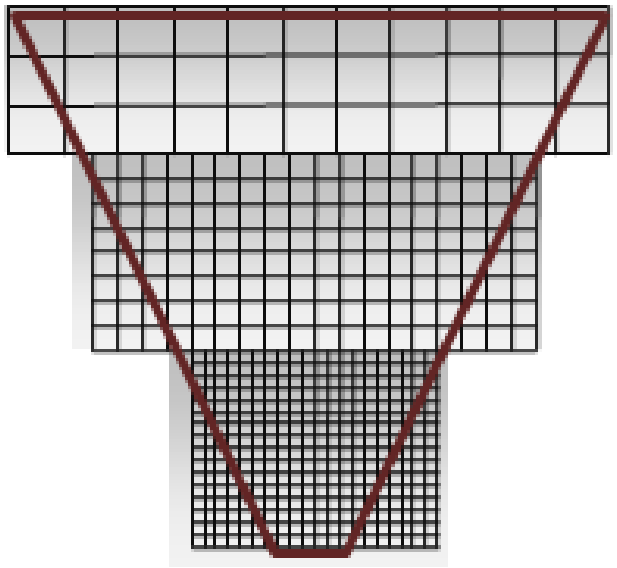
image des profondeurs
= discrétisation de la scène
=> aliasing





Shadow Mapping

- **Limitations**
  - Spots
    - 1 source omnidirectionnelle → 6 shadow-maps !
  - Pixelisation : approximation par morceau
    - Solutions :
      - Idéal : 1 pixel ↔ 1 texel
      - Augmenter la résolution
        - « Cascade shadow maps »



      - Texture d'ombre projective
        - Résolution dépendante de la distance à la caméra