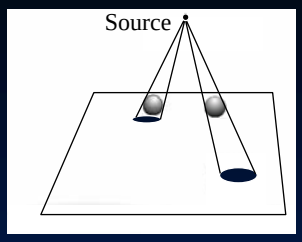
# Calcul des ombres

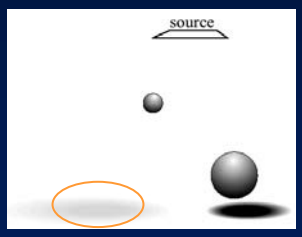
Cours de DEA
C.Soler
Jeudi 01/02/2007

## Introduction

- Ombres dures (*Hard shadows*)
  - Source ponctuelle
  - Détermination binaire
  - Renseignent sur la position des obstacles par rapport à l'oeil



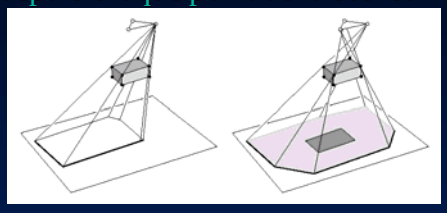

- Ombres douces (*Soft shadows*)
  - Source étendue
  - Variation continue
  - Renseignent sur les tailles et positions relatives entre source, obstacle et récepteur

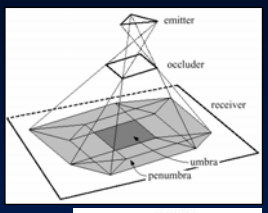


# Ombre et pénombre

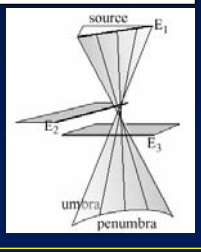
– La pénombre (sens large) est l'union des projections des obstacles depuis chaque point de la source

– L'ombre est l'intersection des projections des obstacles depuis chaque point de la source





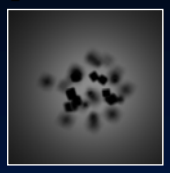

– Pour une source et des obstacles polygonaux:
  » Limite ombre-pénombre: quadriques reglées
  » Limite pénombre: polyhèdre





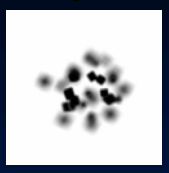

# Approximation courante

✓ Séparation éclairement direct/visibilité moyenne



#

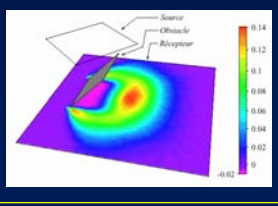
x



Illumination exacte — Illumination directe — Visibilite moyenne

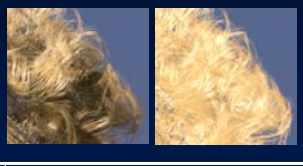
$$\int_{Source} I(\omega)v(x,\omega)\cos\theta d\omega \quad \# \quad \int_{Source} I(\omega)\cos\theta d\omega \quad \frac{1}{\Omega_S}\int_{Source} v(x,\omega)d\omega$$

✓ Erreur d'approximation
  – faible
  – cantonée dans les zones de forte correlation source-obstacle



# Caracteristiques des algorithmes

- Type de
  - la source: ponctuelle / sphérique / polygonale / à l'infini / uniforme …
  - l'obstacle: volumique / transparent / polygonal
  - récepteur: plan / courbe / volumique / complexe
- Exact / approximatif ⇨ différentes applications
- Auto-ombrage ?
- Vitesse:
  non interactif / interactif / RT
- Implémentation: soft/hard
- Coût
  CPU
  mémoire
- Espace:
  espace objet / espace image



| | |
|---|---|
| E | Nombre moyen d'aretes par polygone |
| P | Nombre de points pour echantillonner une source etendue |
| R | Resolution du buffer utilise |
| n | Nombre de primitives dans la scene |
| pxq | Resolution de l'image en pixels. |

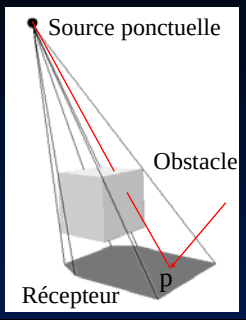


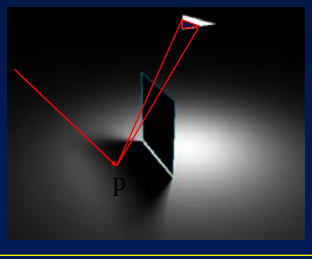
# Plan

- Espace objet
  - Lancer de rayons
  - BSP
  - Calculs géometriques explicites
- Espace image
- Méthodes hybrides

# Lancer de rayons

- Pour tous les points visibles *p*:
  - Lancer un rayon de *p* vers la source
  - Calculer si p voit la source par intersection avec la géometrie
- Complexité: coûteux.
- Si la source est étendue:
  - Calculer la proportion de la source qui est visible (Approx.)
    - » Echantillonnage de la source
    - » Plusieurs rayons
    - » Calcul integral (numérique)
  - Calcul de l'illumination (Exact)

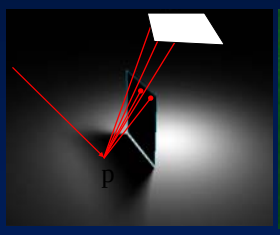


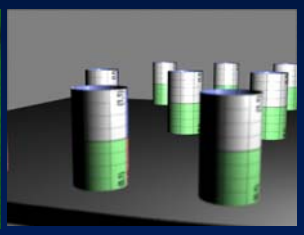






# Ray-tracing probabiliste (Cook'84)

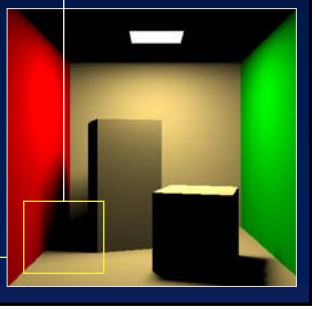
- Principe:
  échantillonner la source aléatoirement (angle ou espace).
- Extensions: *motion-blur*, *depth-of-field*
- Inconvenients:
  - Tres coûteux
  - Beaucoup de bruit (n rayons ⇨ log(n) bits)



# Qualité de l'échantillonnage

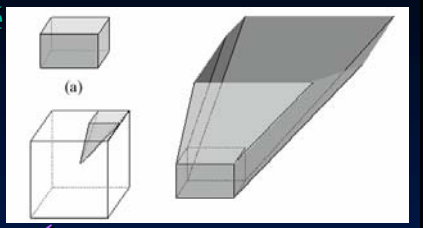

- Regulier:
  - correlation avec les obstacles
    ⇨ bandes d'éclairement
- Aléatoire uniforme:
  - bruit (Variance)
    ⇨ nécessite beaucoup de points
- stratifié:
  - variance réduite ⇨ images meilleures





# Classification des rayons (Arvo'87)

- L'espace $\Re^3 \times S^2$ est partagé en voxels.
  - Dans chaque voxel on stocke l'ensemble des objets rencontrables par ordre de profondeur.




- Inconvenients:
  - Coût mémoire

## Lancer de cones (Amanatides'84)

- Idée: remplacer les rayons d'ombre par des cones
  - *Anti-aliasing* automatique sur les ombres
  - Peut simuler des réflections mi-spéculaires

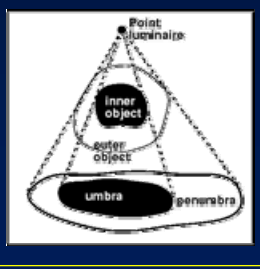
- Inconvenients:
  - Les lampes sont sphériques
  - Les intersections sont plus difficiles à calculer
    Cone-sphère, Cone-plan, Cone-polygone



## *Single sample soft shadows* (Parker'98)

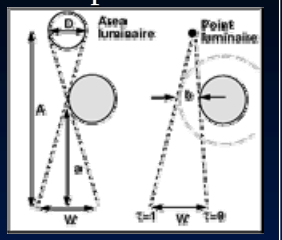
- Une ombre approximative est satisfaisante si:
  - Le calcul est rapide
  - La pénombre varie en fonction des distances obstacle/récepteur et la taille de la source
  - La méthode est stable dans l'espace et le temps
- Par ailleurs:
  - La forme de la source importe peu

⇒ Méthode *ad-hoc* pour une source sphérique



## *Single sample soft shadows* (Parker'98)

- ✓ Choix des paramètres




Source sphérique / obstacle plan:

$$s = (1 + \sin(\pi\tau - \pi/2))/2$$

approche par: $s = 3\tau^2 - 2\tau^3$

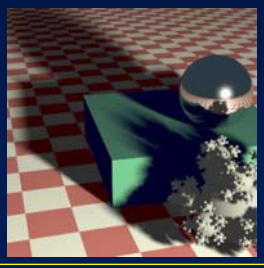
Composition des obstacles:

$$s = \min(s_1, s_2)$$

Création des obstacles virtuels:

$$b = aD/A$$

- ✓ Résultats



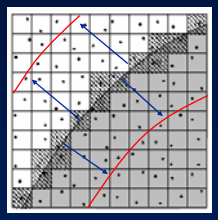


## Cohérence des ombres dans l'espace image

Deux points voisins sont souvent sous la même ombre

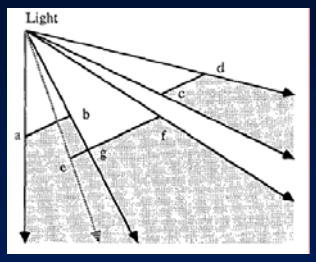
- Conserver un pointeur sur obstacle, et l'utiliser en priorité lors du calcul des points voisins
- Peut s'appliquer à la liste des obstacles.

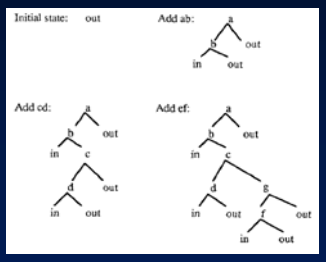
- ✓ Worley'97 - "*fast soft shadows*".
  - Lancer un rayon d'ombre par pixel.
  - Les pixels dont les voisins ne sont pas tous dans le même état sont sûrement dans la pénombre.
  - Recalculer ces pixels avec plus de rayons
  - Propager l'information



# *BSP* (Chin'89)

- Principe:
  - Deux arbres BSP: un pour la scene (#1) et un pour la source (#2).
  - Construire #2 en parcourant les polygones *front-to-back* (utiliser #1). Pour chaque polygone P:
    - » Découper P en fonction de #2
    - » Mettre a jour #2 avec les parties eclairées de P. Mettre à jour le noeud correspondant dans #1.

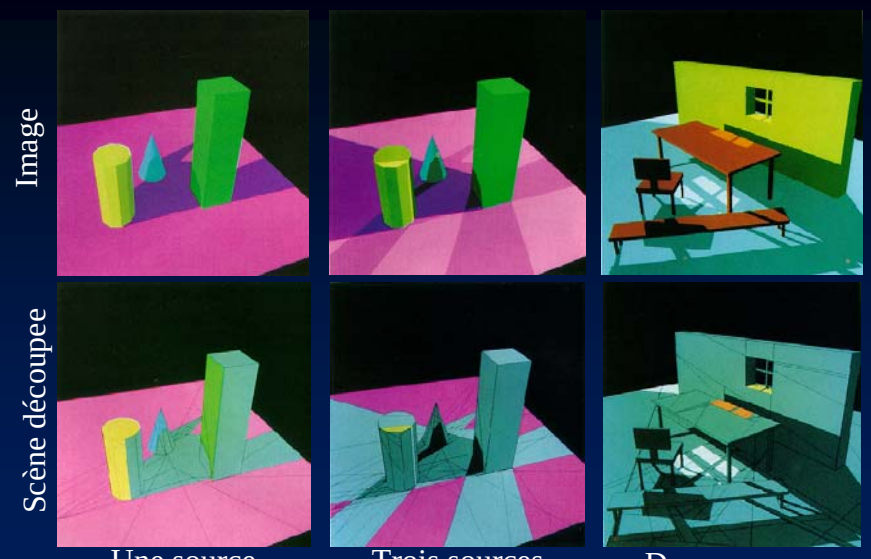



  - Rendu: rendre ombrés ou pas les sous-polygones des noeuds de #1. (On peut faire un z-buffer, ou un rendu *back-to-front* sans z-buffer grâce à #1).
- Sources multiples:
  - Utiliser chaque source successivement pour couper les polygones de #1



# *BSP* (Chin'89)

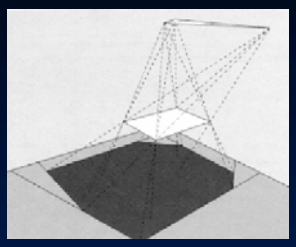


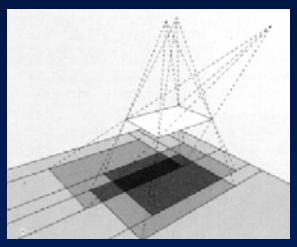
## *Soft-Shadows with BSP* (Chin'92)

- ✓ Deux BSP pour chaque source:
  - – un BSP pour la pénombre
    - » Plans VE et EV qui séparent la source de l'obstacle
    - » La pénombre est l'intersection demi-espaces correspondants.

    (definition équivalente à union des projections de l'obstacle)
  - – Un BSP pour l'ombre
    - » Plans VE seulement, et tels que la source et l'obstacle sont entièrement du même cote.
    - » L'ombre est l'intersection des demi-espaces correspondants.

    (définition équivalente à l'intersection des projections de l'obstacle)
- ✓ L'ombre est incluse dans la pénombre.



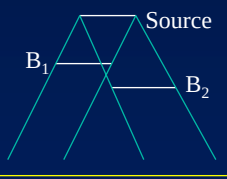




## *Soft-Shadows with BSP* (Chin'92)

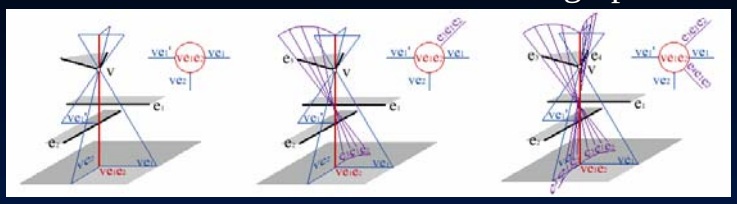
- ✓ Algorithme:
  - – Pre-processing:
    - » Calculer un BSP pour la scène
    - » Couper les sources par tous les plans des polygones

      Assure la cohérence de l'ordre de traversée des BSP des sources
    - » Couper tous les polygones par le plan de chaque source
  - – Classification et découpage des polygones, *front-to-back*
    - » Utiliser le BSP de pénombre et celui d'ombre
    - » Decouper les polygones selon leur classification (E/P/O)
  - – Rendu:
    - » Calcul explicite de l'eclairage dans les zones de pénombre (coûteux)
    - » Calcul rapide dans les zones eclairées (facteur de forme analytique)
- ✓ Défauts:
  - – La composition des obstacles est fausse
    - » L'union de deux volumes d'ombre n'est pas le volume d'ombre de l'union
    - » D'ailleurs, pas de triples-arètes (la limite d'ombre est une conique)

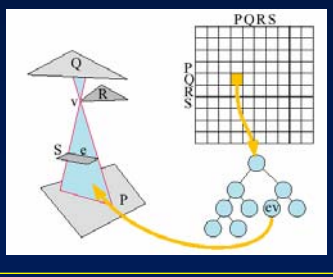


# Le squelette de visibilité (Durand'97)

- ✓ Enumération des discontinuités ⇨ graphe



- ✓ Structure de données:



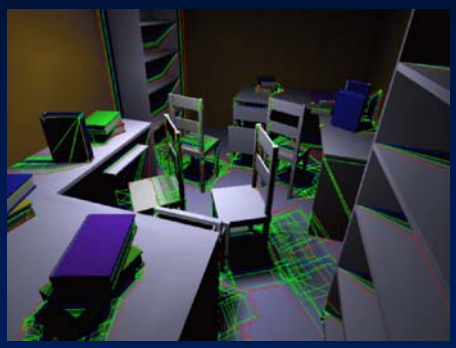

```
class Node {
   List<Arcs> adjacentArcs
   Face Fup, Fdown
   Point3D Pup, Pdown
}

class Arc {
   Node Nstart, Nend
   float tstart, tend
   Face Fup, Fdown
}
```

```
class EV : child of Arc {
   Edge e
   Vertex v
   }
class VisibilitySkeleton {
   tree<EV> ev[Fup][Fdown]
   tree<EEE> eee[Fup][Fdown]
   ...
}
```



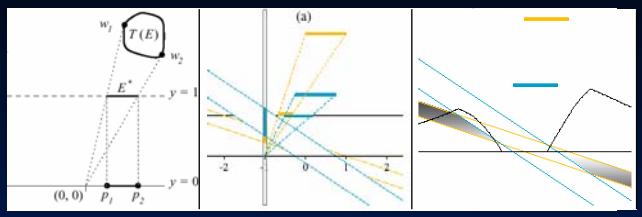
# Le squelette de visibilité (Durand'97)

- ✓ Requêtes à coût constant:
  - Liste exacte des obstacles
  - Calcul de l'ombre sans séparation de la visibilité.
  - Maillage de discontinuité



# Formulation analytique (stark'99)

- ✓ Étude en 2D




– Sans occlusion:

$$I(x) = k\int_E \frac{\cos\theta\cos\theta'}{d(x,u)}du = k\int_{p_1(x)}^{p_2(x)} \frac{du}{(1+u^2)^{3/2}}$$

– Cas général:

$$I(x) = k\int_{M_E(x,.)} w(u)du - k\int_{M_O\cap M_E(x,.)} w(u)du$$

⇨ Somme de 2 splines polyhèdrales ⇨ expression analytique

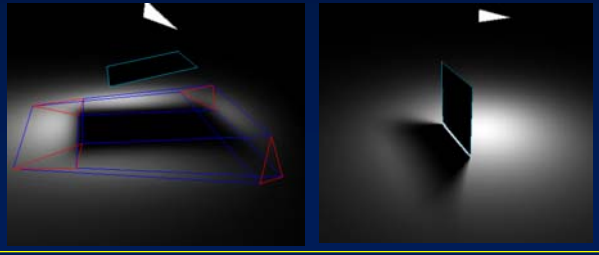


# Extension en 3D (stark'99)

- ✓ En trois dimensions:
  - – Splines polyhèdrales définies sur des trapèzes de dim 4
  - – Fonction poids:

$$w(x,y) = (1+x^2+y^2)^{-2}$$

  - – Formule d'évaluation analytique
    - » Assez complexe, mais utilisable. Sans de-corrélation.
    - » Restreinte à certaines configurations

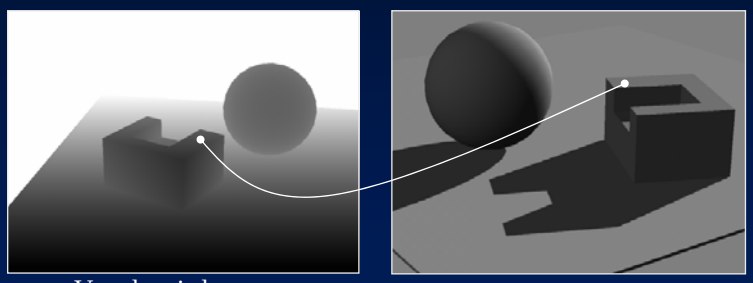


# Plan

- ✓ Espace objet
- ✓ Espace image
  - – Shadow maps
  - – Shadow volumes
  - – Méthodes à base de textures
- ✓ Méthodes hybrides

# *Shadow maps* (William'78)

- ✓ Algorithme:
  - – Rendre la scène vue depuis la source
  - – Lire le z-buffer
  - – Rendre la scène depuis l'oeil. Pour chaque pixel:
    - » Transformer le pixel dans l'image 1
    - » Si le z est plus petit, le pixel est eclairé, sinon il est à l'ombre.



Vue depuis la source
Z-buffer

Vue utilisateur

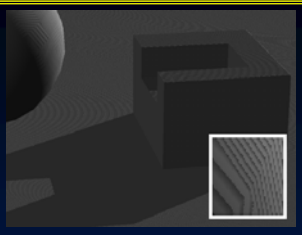
# *Shadow maps*:Implémentation

```
LightViewTransform();
RenderScene();
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT16, 0, 0, w, h, 0);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE_ARB,GL_COMPARE_R_TO_TEXTURE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC_ARB, GL_LEQUAL);
glTexParameteri(GL_TEXTURE_2D, GL_DEPTH_TEXTURE_MODE_ARB, GL_INTENSITY);

glTexGeni(GL_S,GL_TEXTURE_GEN_MODE,GL_EYE_LINEAR) ;      # faire de meme pour T,R et Q
glEnable(GL_TEXTURE_GEN_S) ;                             #

glEnable(GL_TEXTURE_2D);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glGetFloatv(GL_PROJECTION_MATRIX, proj_mat_scene);
glMatrixMode(GL_TEXTURE);
glLoadIdentity() ;
glMultMatrix(proj_mat_scene);
glMultMatrix(modelview_mat_scene) ;

glMatrixMode(GL_MODELVIEW);
EyeViewTransform() ;

RenderScene() ;
```

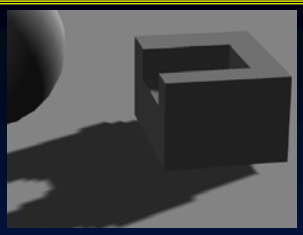
(C.f exemples sur le site NVidia)



# *Shadow maps* - défauts

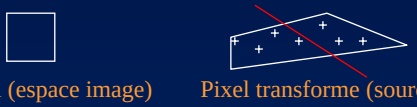


Effet d'auto-ombrage



*Aliasing*

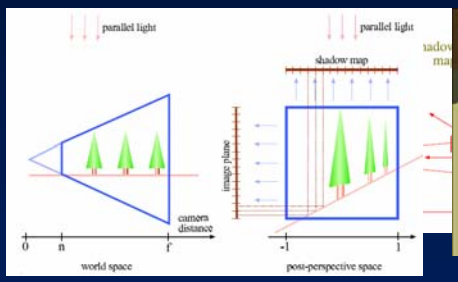
- ✓ Auto-ombrage (éclairage trop "étalé")
  - Correction: rajouter une constante (*bias*) à la *depth map*
- ✓ *Aliasing* (sous-echantillonnage de la *depth map*)
  - Correction: *Percentage-closer filtering* (Reeves'87)

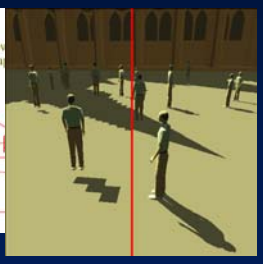



- ✓ Aliasing (sur-echantillonnage de la *depth map*)

# Ameliorations

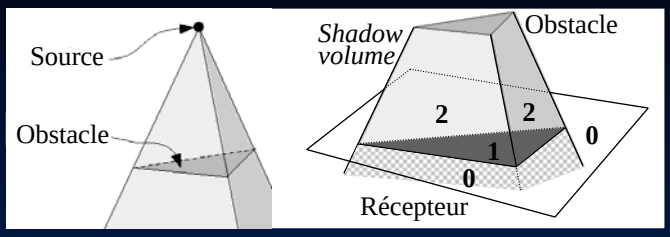
- Hourcade'85
  - utiliser un item buffer
- *Adaptive shadow maps* (Fernando'01)
  - la *shadow map* est stockée dans un *quadtree*
  - fabriquee en utilisant du mip-mapping
- *Perspective shadow maps* (Stamminger'02)



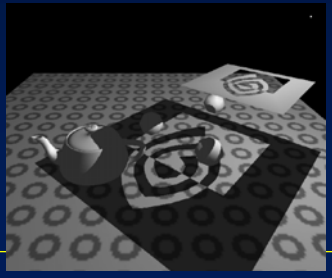



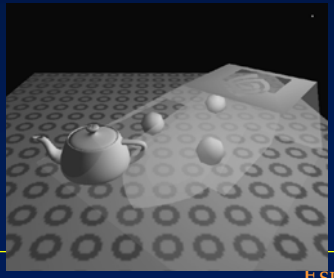
# *Shadow volumes*

- Idée: construire un volume d'ombre et tester la parité.




- Exemple:

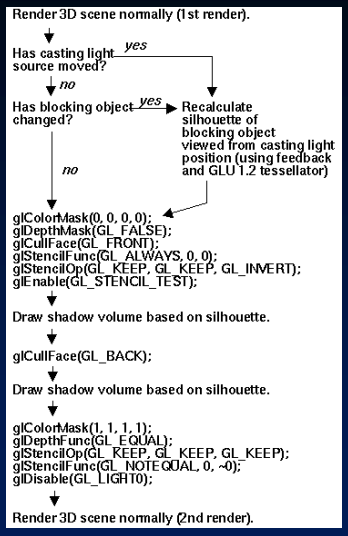




Espace image

# Shadow volumes

- Algorithme:
  - Rendre la scène une première fois
    - » pour remplir le z-buffer
  - Utiliser le *stencil buffer* pour créer un masque
    - » désactiver l'ecriture du FB et z-buffer
    - » rendre le *shadow-volume* coté fond, (inverse le stencil)
    - » rendre le *shadow-volume* coté face, (inverse le stencil)
  - Rendu de la scène (couleur ombre) à travers le masque
- Implémentation
  - Calcul du *shadow volume*
    - » *feed-back buffer* pour les contours
    - » analytique






# Améliorations

- Calcul de la silhouette:
  - en utilisant une *shadow map* (McCool'00)
    - » detections de contours
  - avec la carte graphique (Brabec'03)
    - » détection des silhouettes en hardware
    - » compatible avec les vertex programs

- Sans le stencil buffer (Roettger'02)
  - initialiser a ¼
  - glBlendFunc(GL_DST_COLOR,GL_ONE/GL_ZERO) avec resp. 1 et ½.
  - doubler/inverser/doubler en rendant des gros rectangles blancs
- En sens inverse (ZPass/ZFail: Everitt'02)
  - resout le problème de l'oeil dans l'ombre (+ d'autres)

# Méthode Herf (Herf'97) 1/2

- Idée:
  - Projeter les polygones causant l'ombre depuis plusieurs points situés sur la source et faire la moyenne des images obtenues.

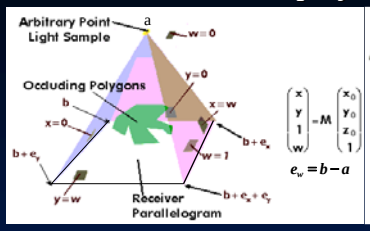
- Algo:

Pour chaque polygone récepteur:
pour chaque echantillon sur la source:
calculer la matrice de projection;
dessiner le récepteur;
dessiner les obstacles en noir;
ajouter l'image à l'accumulation buffer;
sauver l'accumulateur dans la mémoire de texture



# Méthode Herf (Herf'97) 2/2

- Calcul de la matrice de projection:



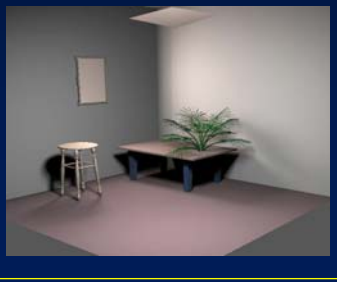

$$M = \begin{pmatrix} \alpha_x n_{xx} & \alpha_x n_{xy} & \alpha_x n_{xz} & -\alpha_x n_x \cdot b \\ \alpha_y n_{yx} & \alpha_y n_{yy} & \alpha_y n_{yz} & -\alpha_y n_y \cdot b \\ 0 & 0 & 0 & 1 \\ \alpha_w n_{wx} & \alpha_w n_{wy} & \alpha_w n_{wz} & -\alpha_w n_w \cdot a \end{pmatrix}$$

avec $n_x = e_w \times e_y$ $\alpha_x = 1/n_x \cdot e_x$

$n_y = e_x \times e_w$ $\alpha_y = 1/n_y \cdot e_y$

$n_w = e_y \times e_x$ $\alpha_w = 1/n_w \cdot e_w$

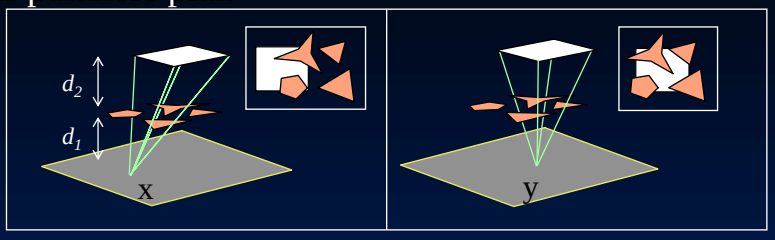
- Résultats



Trop peu de *samples*

# Méthode de convolution (Soler'98)

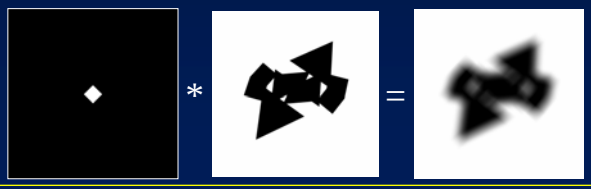
- Cas parallèle-plan




$$V(x) = \int_S v(x,x')dx' = \frac{1}{\alpha^2}(s_\alpha \otimes p_{1+\alpha})(x)$$

$$\alpha = \frac{d_1}{d_2} \quad p_{1+\alpha}(x) = P\left(\frac{1}{1+\alpha}x\right) \quad s_\alpha(x) = S\left(-\frac{1}{\alpha}x\right)$$

- Calcul:

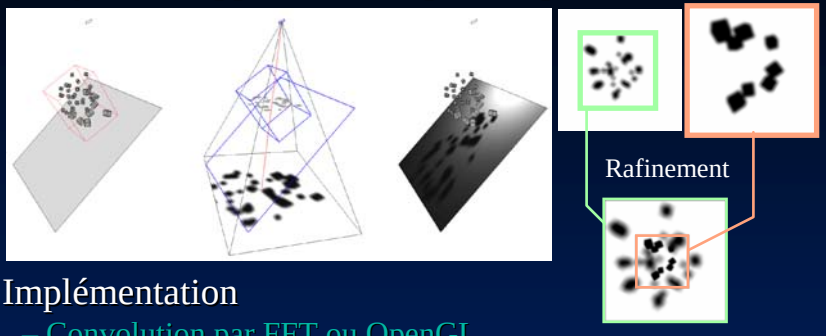



Hardware
Ou FFT

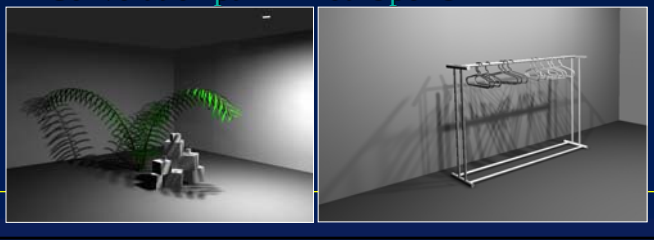


# Méthode de convolution

- Cas général: approximation par des plans




- Implémentation
  - Convolution par FFT ou OpenGL

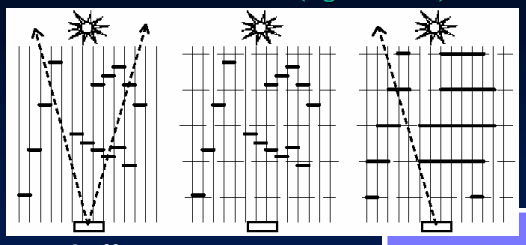


# Plan

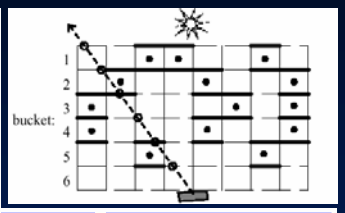
- ✓ Espace objet
- ✓ Espace image
- ✓ Méthodes hybrides

# Multi-layer depth images (Keating'99)

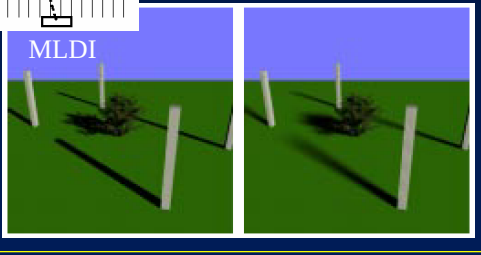
- ✓ Convertir la géometrie en un ensemble de couches
- ✓ Lancer des rayons vers la source à travers ces couches
  - – Eviter les trous (*light leaks*)



Z-buffer partitionnement



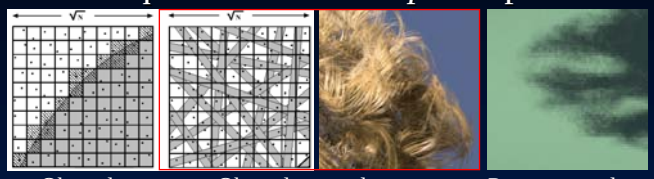

MLDI



  - – Stocker sous forme d' un tableau de bits.

# *Deep shadow maps* (Lokovic'00)

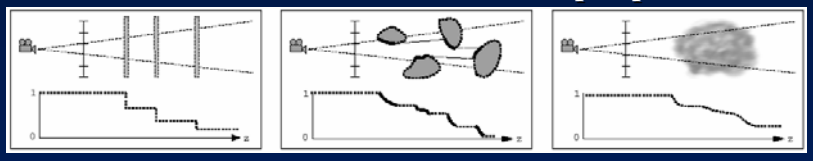
- Pour des obstacles complexes l'echantillonnage stochastique de *shadow map* a trop de variance.



Obstacle simple | Obstacle complexe | Percentage closer filtering $512^2$ pixels

Bruit ou Sur-coût

- Utiliser une fonction de transmittance par pixel



- Sampling: anti-aliasing à *z* constant dans la *DSM*.

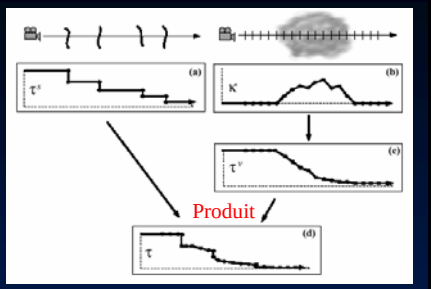


# *Deep shadow maps* (Lokovic'00)

- Calcul
  - Visibilité volumique

$$\frac{dI(s)}{ds} = -k(s)I(s) \qquad \tau(z) = e^{-\int_0^z k(t)dt}$$

  - Surfaces semi-transparentes
    - » Fonctions en escalier




- Stockage:
  - compression

# Références


*Ray tracing*

*Distributed ray tracing.* R.Cook, T. Porter, L.Carpenter. Siggraph'84

*Ray tracing with cones.* J.Amanatides. Siggraph'84

*Fast ray-tracing by ray classification.* J.Arvo, D.Kirk. Siggraph'87

*Shadow maps* - espace objet

*Casting surved shadows on curved surfaces.* William. Siggraph'78

*Rendering anti-aliased shadows with depth maps.* W.Reeves *et al.* Siggraph'87.

*Fast shadows and lighting effects using texture mapping.* M.Segal *et al.* Siggraph'92

*Rendering CSG models with a zz-buffer.* D.Salesin, J.Stolfi. Siggraph'90.

Arbres BSP

*Near real time shadow generation using BSP trees.* Norman Chin, Steven Feiner. Siggraph'89

*Fast Object-Precision Shadow Generation for Area Light Sources Using BSP Trees.* N.Chin, S. Feiner. Siggraph'92

Techniques analytiques

*Computing exact shadow irradiance using splines.* M.Stark *et al.* Siggraph'99

*The visibility skeleton: a powerful and efficient multi-purpose global visibility tool.* CG, 31(3A)p89,1997

Techniques diverses

*Single sample soft shadows.* S.Parker, P.Shirley, B.Smith.

*Efficient generation of soft-shadow textures.* M.Herf. Technical report CMU-CS-97-138. Carnegie Mellon University, Pittsburgh.

*Fast computation of soft shadow textures using convolution.* C.Soler, F.Sillion. Siggraph'98

*Surveys*

*A survey of shadow algorithms.* A.Woo, P.Poulin, A.Fournier. IEEE CG&A, 10(6):13-32, Nov 1990.

*A multidisciplinary survey on visibility.* F.Durand. ACM course notes *Visibility, problems and applications.* Siggraph'2000.

*Image-based*

*Efficient image-based methods for rendering soft shadows.* M.Agrawala *et al.* Siggraph'2000

*Deep shadow maps.* T. Lokovic, E.Veach. Siggraph'2000.