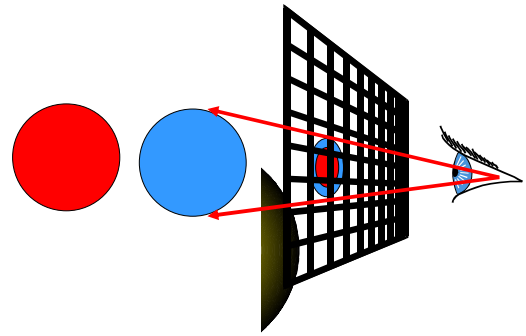
Synthèse d'Images

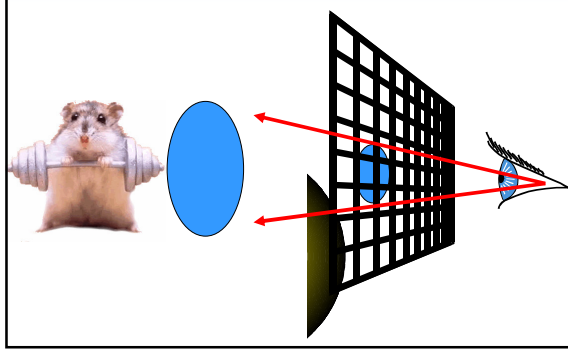
Elmar Eisemann
Elmar.Eisemann@inrialpes.fr

Basé sur les cours de
Frédo Durand, Barbara Cuttler, E. Boyer,
H. Briceno, N. Holzschuch, Alex Meyer

Pourquoi faire?



Autres raisons?

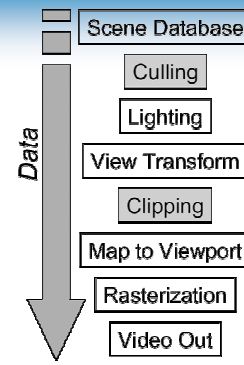


Que ne peut-on pas voir ?

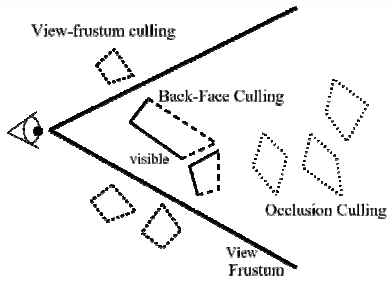
- Ce qui est dans l'ombre (?!)
- Ce qui est caché par un objet;
- Ce qui est en dehors du volume de vue.

⇒ Culling et clipping

⇒ Eliminer les objets le plus tot possible.



Culling :

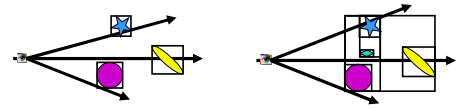


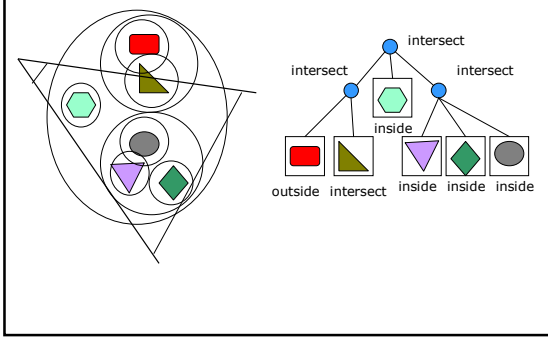
View frustum culling :

élimination des objets hors du cone de vue.

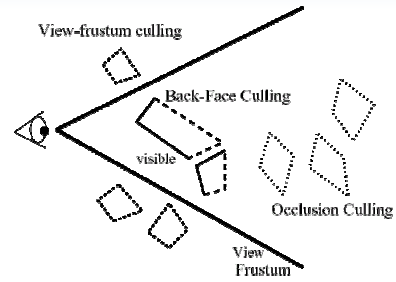
naïf : chaque primitive=> un test par polygone !!!

Boite/sphères englobante marche mieux.
Accélération : hiérarchie de boites englobantes.
Hors du volume de vue alors toutes ses filles aussi.
Problème de la construction de la hiérarchie.



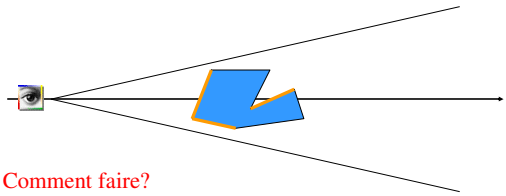


Culling :



Back Face Culling

- Éliminer tous les polygones qui ne sont pas tournés vers la caméra :



Comment faire?

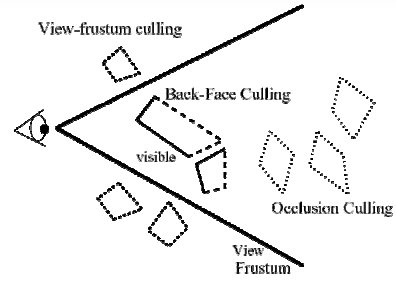
Back Face Culling : algorithme

- Si le point de vue n'est pas devant le polygone, on n'affiche pas le polygone
- Produit scalaire :
  - (Sommet-PointDeVue)*normale
  - > 0 : on garde le polygone
  - < 0 : on l'élimine

Back Face Culling

- Économise 50 % du temps de calcul
  - En moyenne
- Faible coût par polygone
- Étape préliminaire pour les autres algorithmes
- Suffisant pour un seul objet convexe

Culling :



Occlusion culling : *qui cache qui ?*

Souvent **PVS (potentially visible set)** :
objets potentiellement visibles pour un point de vue/region.

Différentes méthodes :
- Point / région
- Espace image / espace objet
- Portails / générique

Différents critères de choix :
Conservatif / approximatif / exact, objets non-bloqueurs (transparent), convexité des bloqueurs, fusion des bloqueurs, 2D / 3D, hardware, précalcul / à la volée, scènes dynamiques…

Élimination des parties cachées

- Espace image contre espace objet
- Algorithmes « objet »:
  - Backface culling
  - Tri par la profondeur
  - BSP-trees
- Algorithmes « image » :
  - Z-buffer
  - Coûts comparés, choix optimal

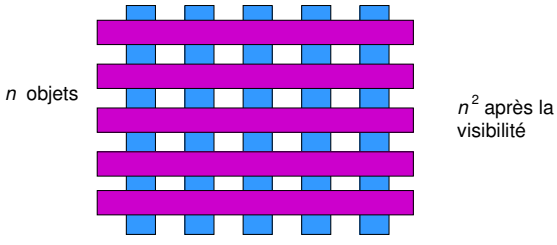
Élimination des parties cachées

- Détermination des surfaces visibles
- Résolution dans l'espace image ou dans l'espace objet
  - Espace image : précision limitée
  - Espace objet : précision infinie

Idée: Couper selon visibilité

Résolution dans l'espace objet

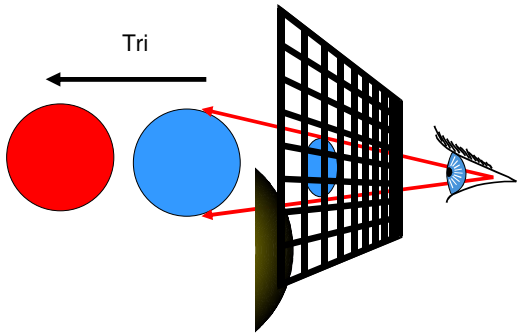
- Parfois nécessaire, mais complexité la pire :



Résolution dans l'espace image

- Permet d'utiliser la cohérence:
  - L'objet visible à un pixel sera sans doute visible sur les pixels voisins
- Plus rapide en moyenne
- Cas le pire : O(*np*)

Dessiner sur l'écran



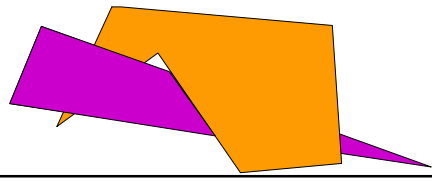
Tri par la profondeur

- Dit aussi « algorithme du peintre »
- On dessine d'abord les polygones les plus lointains, puis les polygones proches de l'œil
- Les polygones plus proches cachent les polygones lointains
- Comme un peintre dessine d'abord l'horizon, puis l'arrière-plan, puis le premier plan

Problème?

Algorithme simplifié

- Trier les polygones en fonction de la distance à l'œil
- Ordre incomplet
- Ambiguïtés à résoudre :

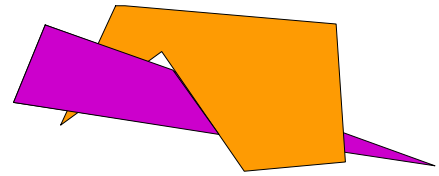


Pour ou contre

- Le plus intuitif des algorithmes
- Coût en mémoire :
  - Affichage direct à l'écran : O(*p*)
  - Il faut trier les polygones : O(*n*log*n*)
- Temps de calcul :
  - On affiche toute la scène
  - Efficace surtout sur des petites scènes

Example: Links 386

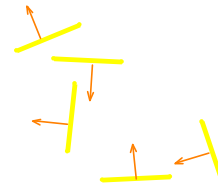
Comment regler le problème?



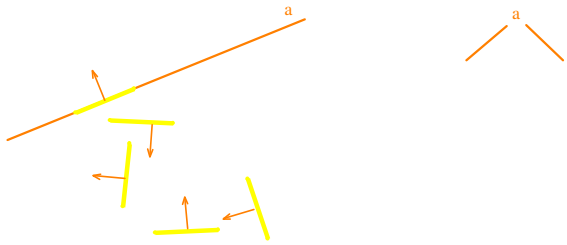
BSP-Trees

- On construit un BSP-Tree 3D pour toute la scène
  - En subdivisant
- Affichage par un parcous de l'arbre :
  - En premier les polygones qui sont derrière le nœud courant (par rapport à la caméra)
  - Puis le nœud courant
  - Puis, les polygones qui sont devant le nœud courant
- Sorte d'algorithme du peintre
  - Avec un ordre partiel
  - Mais suffisant

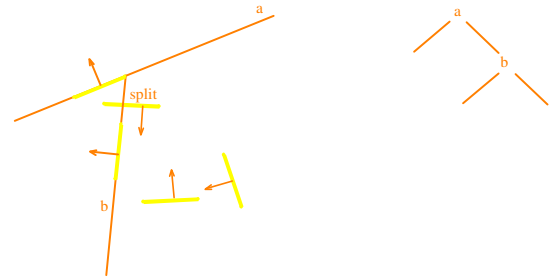
Construire un BSP-Tree (1)



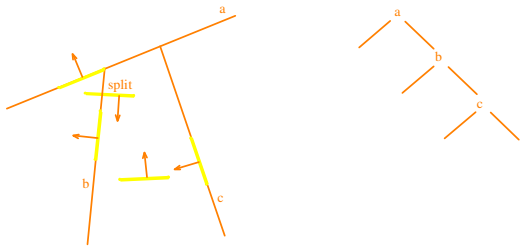
Construire un BSP-Tree (2)



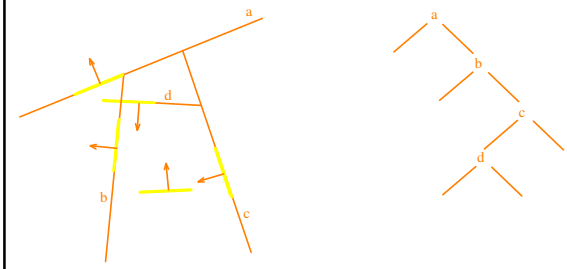
Construire un BSP-Tree (3)



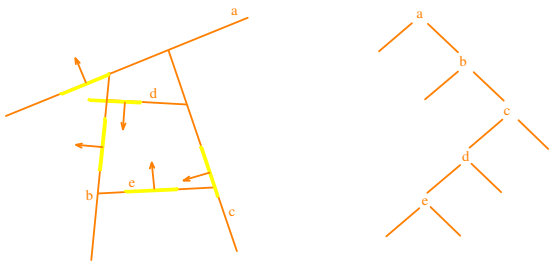
Construire un BSP-Tree (4)



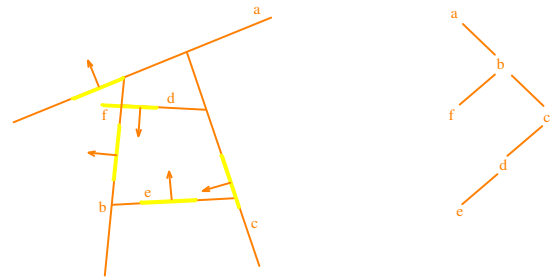
Construire un BSP-Tree (5)



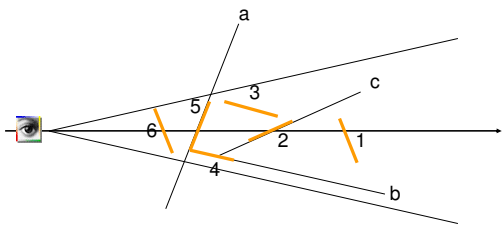
Construire un BSP-Tree (6)



Construire un BSP-Tree (7)



Utiliser un BSP-Tree



BSP Tree contre algorithme du peintre

- Le BSP-Tree fait des divisions de polygones
- À l'affichage plus direct
- BSP-Tree:
  - Pré-traitement plus long
  - Coût mémoire plus élevé
  - Temps par requête (position caméra) plus petit
- Algorithme du peintre:
  - Pas de pré-traitement
  - Temps par requête plus long
- Dans les deux cas : on affiche toute la scène

Z-Buffer

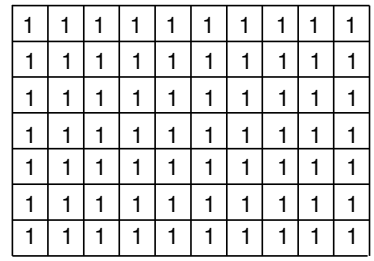
- Un tableau, de la taille de l'écran
- On stocke la valeur maximale de z pour chaque pixel
  - *z* est la direction de visée, exprime la distance à l'oeil
- Initialisation : tous les pixels à moins l'infini
- Projection de tous les polygones
  - On met à jour les pixels de la projection du polygone

Resoudre le problème de visibilité dans l'espace image

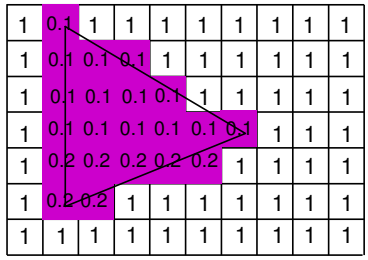
Z-buffer : algorithme

- Pour chaque polygone :
  - Projeter le polygone sur le plan image
  - Pour chaque pixel dans la projection du polygone
    - Calculer la valeur de *z* pour ce pixel
    - Si z est supérieur à la valeur courant de z max
      - Changer z maximal
      - Afficher le pixel à l'écran, de la couleur du polygone

Z-buffer (1)

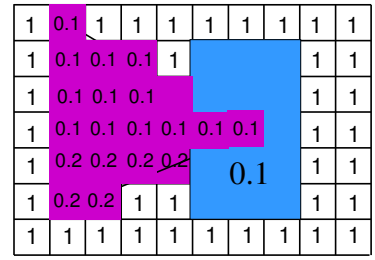


Z-buffer (2)



Z-buffer (3)

Dessiner un quad à une distance 0.1



Z-Buffer : pour ou contre

- Pour :
  - Facile à implémenter
  - Travaille dans l'espace image
    - Rapide
- Contre :
  - Coût en mémoire
  - Travaille dans l'espace image
    - aliasing
    - artefacts

Z-Buffer

- Combien d'information ?
  - Combien de bits pour z max ?
  - Limité par la mémoire :
    - 8 bits, 1024x1280: 1.25 Mb
    - 16 bits, 1024x1280: 2.5 Mb
  - Nécessaire pour la séparation des objets proches :
    - 8 bits, distance minimale entre objets de 0.4 % (4mm pour 1m)
    - 16 bits, distance minimale de 0.0001 % (1mm pour 1km)
    - Que se passe t-il en dessous de cette limite ?

Choix de l'algorithme ?

- L'algorithme idéal dépends :
  - De la complexité de la scène
  - Du matériel disponible
  - De ce qu'on veut faire en plus de l'affichage
- Z-Buffer en général :
  - Objets opaques
  - Fourni gratuitement dans la librairie graphique
  - Pas de pré-traitement
  - Quelques effets de bord si on le connaît mal

Calcul des ombres - plan

- Pourquoi les ombres ?
- Ombres planes
- Shadow map
- Shadow volume
- Ombres douces

Diapos de : Joëlle Thollot

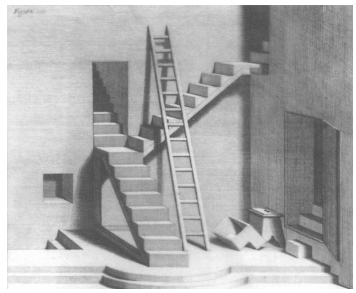
Importance des ombres

Position dans l'espace

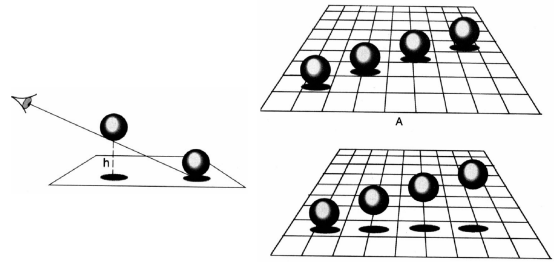
Points de contact

Eclairage

Réalisme

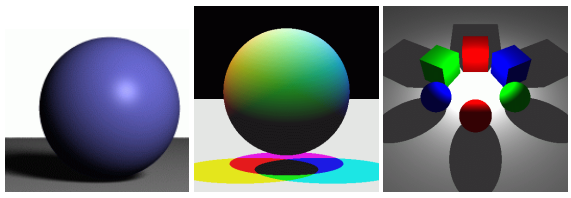


Indication de la position



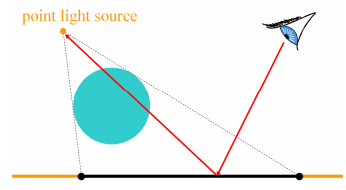
Indication de l'éclairage

- Position de la lampe
- Type de lampe
  - Directionnelle, ponctuelle, étendue
  - Colorée



Ombre dure

- Quand la source est ponctuelle
- Un point est dans l'ombre s'il ne voit pas la source

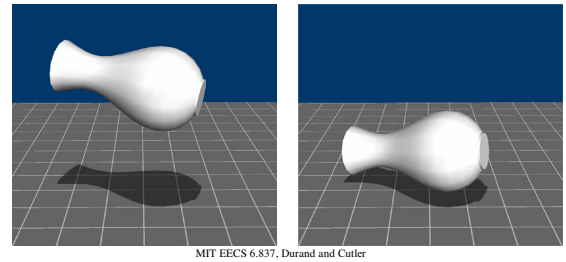


Calcul des ombres - plan

- Pourquoi les ombres ?
- Ombres planes
- Shadow map
- Shadow volume
- Ombres douces

Ombres planes

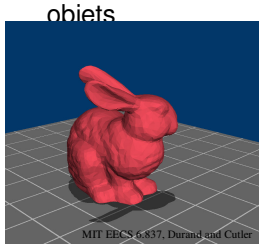
- Dessiner les primitives une seconde fois projetées sur le sol



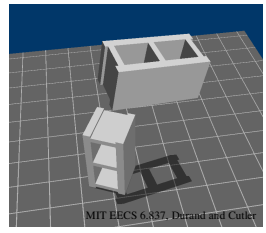
MIT EECS 6.837, Durand and Cutler

Ombres planes +/-

+ Simple et efficace
- Pas d'auto ombrage, pas d'ombres sur des surfaces courbes, sur d'autres objets



MIT EECS 6.837, Durand and Cutler



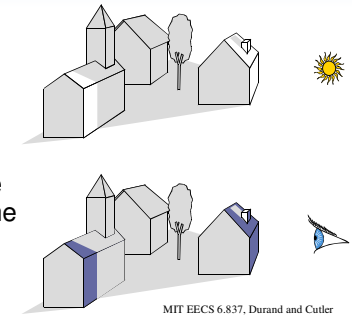
MIT EECS 6.837, Durand and Cutler

Calcul des ombres - plan

- Pourquoi les ombres ?
- Ombres planes
- Shadow map
- Shadow volume
- Ombres douces

Dualité ombre/vue

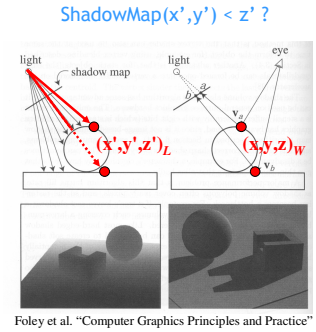
- Un point est éclairé s'il est visible de la source
- Le calcul des ombres est donc similaire au calcul d'une vue



MIT EECS 6.837, Durand and Cutler

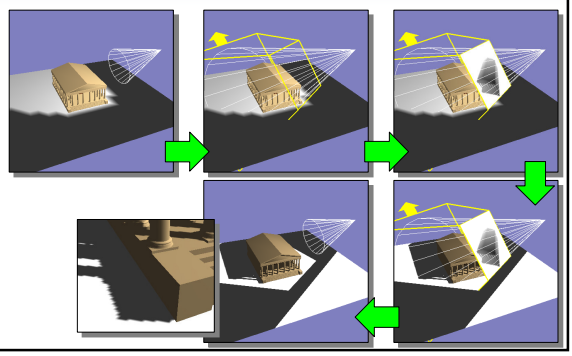
Shadow map (carte d'ombre)

- Texture et profondeur
- 2 passes de rendu :
  - Shadow map : image de profondeur vue de la source
  - Rendu final en regardant pour chaque pixel s'il est dans l'ombre ou pas



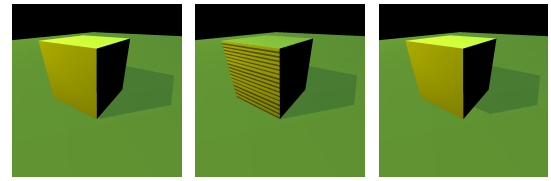
Foley et al. "Computer Graphics Principles and Practice"

Shadow map



Shadow map +/-

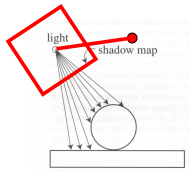
+ Implémentation en hard
- Pixel éclairé : ShadowMap(x',y') ≈ z'
=> ajouter un biais ShadowMap(x',y') + eps < z'



OK | biais trop faible | biais trop élevé

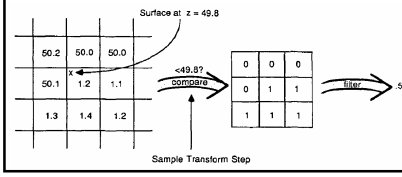
Shadow map +/-

- Angle de vue de la lampe
=> spot light ou cubic shadow map



Shadow map +/-

- Résolution de la carte de profondeur : aliassage
=> Trouver la bonne résolution
=> Perspective shadow maps
=> Filtrage de la réponse

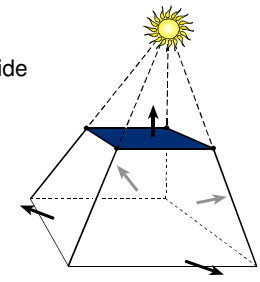


Calcul des ombres - plan

- Pourquoi les ombres ?
- Ombres planes
- Shadow map
- Shadow volume
- Ombres douces

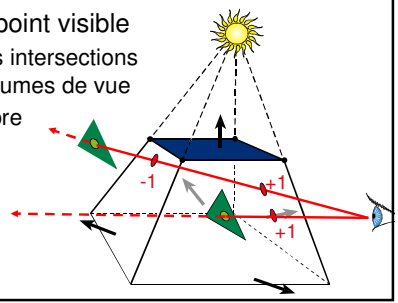
Shadow volume

- Représenter explicitement le volume d'ombre
- Pour chaque polygone
  - Construction de la pyramide
- Test d'ombre = clipping



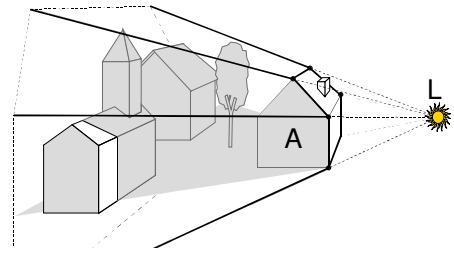
Shadow volume

- Un point qui est dans le volume d'ombre d'une source ne reçoit aucune lumière
- Calcul : par point visible
  - Compte des intersections avec les volumes de vue
  - Si ≠ 0 : ombre



Optimisation

- Ne calculer le volume d'ombre que pour les silhouettes des objets



Stencil Buffer

- Buffer pour compter sur la carte graphique
- glStencilFunc: spécifier condition par rapport aux valeurs courrantes (Always, Less, Greater, Never ...)
- glStencilOp: Modification selon résultat de la condition et du zTest

Shadow volume : implémentation

- Utilisation du stencil buffer :
  - Buffer pour marquer les pixel lors d'une passe de rendu
  - Puis stencil-test avant le z-test

Shadow volume algorithm

Initialize stencil buffer to 0

Draw scene with ambient light only

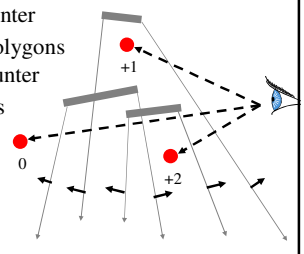
Turn off frame buffer & z-buffer updates

Draw front-facing shadow polygons
If z-pass → increment counter

Draw back-facing shadow polygons
If z-pass → decrement counter

Turn on frame buffer updates

Turn on lighting and redraw pixels with counter = 0



Shadow volume +/-

+ Calcul exact
- Rajoute de la géométrie => lent
- Rajoute des grands polygones => fill rate

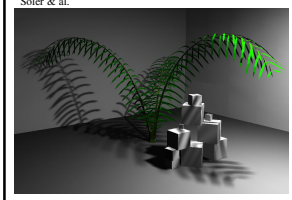
Calcul des ombres - plan

- Pourquoi les ombres ?
- Ombres planes
- Shadow map
- Shadow volume
- Ombres douces

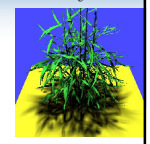
Ombre douce

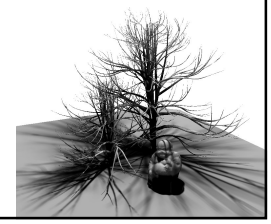
- Quand la source est étendue

Soler & al.



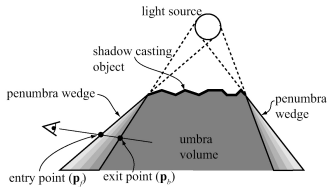
Agrawala & al.





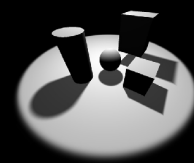
Ombre douce

- 3 zones :
  - Ombre : la source est totalement cachée
  - Pénombre : la source est partiellement cachée
  - Eclairée : la source est totalement visible

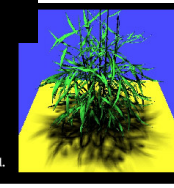


Ombres Douces

Rendering Fake Soft Shadows with Smoothies



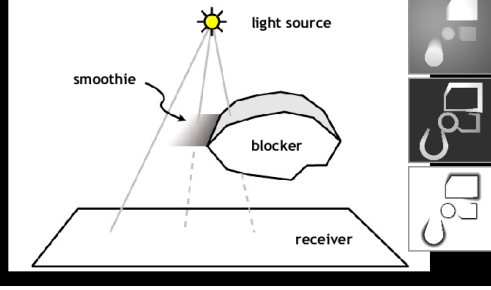
Eric Chan Frédo Durand



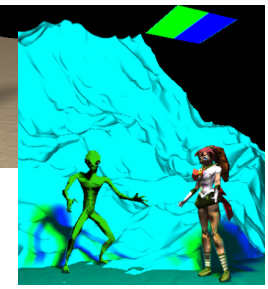
Agrawala et al.

Compute Shadows

Compute intensity using depth comparisons



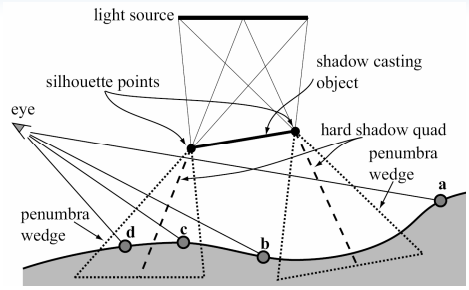
Penumbra Wedges



Compliqué....

Assarson et al.

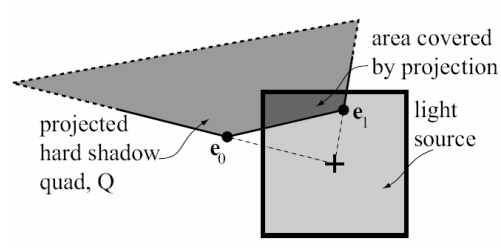
Penumbra Wedges



a = dans lumière c=d= penombre

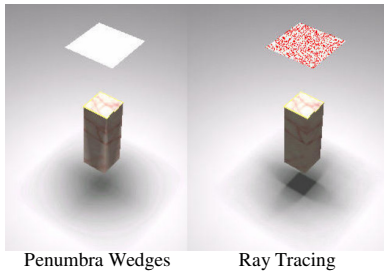
b= dans l'umbra

Penumbra Wedges



Penumbra Wedges

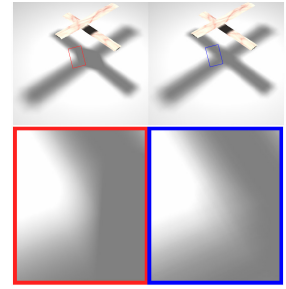
- Problème: Restriction aux silhouettes



Penumbra Wedges Ray Tracing

Penumbra Wedges

- Problème: Combiner les ombres



Penumbra Wedges Ray Tracing

Conclusion

- Shadow maps massivement utilisées dans les jeux vidéo
- Shadow volume en production mais reste coûteux

- Questions ouvertes :
  - Accélération des calculs
  - Précision
  - Ombres douces

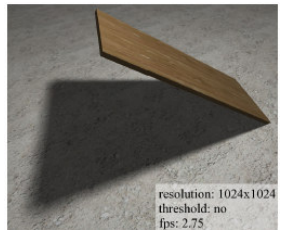
Reprojection Softshadows

- Guennebaud et al., Atty et al.

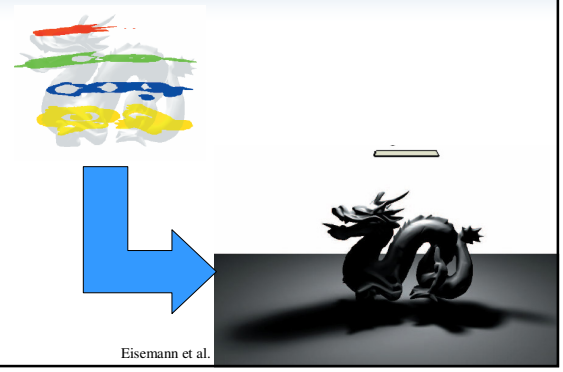
- Projeter depth map pixels sur la source
- Ici: 25 fps

Reprojection Softshadows

- Problem:



Occlusion Textures



Occlusion Textures

Slides

- Contributions de:
- **Briceno, H., Notes du cours SI, UFRIMA**
- **Boyer, E., Notes du cours SI , UFRIMA**
- **Holzschuch, N., « Notes du cours DEA-IVR, ENSIMAG, Création d'Images Virtuelles ». 2005-2006**
- **Frédo Durand and Barbara Cuttler, SI, MIT**
- **Joelle Thollot, INRIA**
- **Images taken from various sources:**

IF ANY IMAGE IN THIS PRESENTATION IS NOT ALLOWED TO BE USED, PLEASE CONTACT ME AND I WILL DELETE IT!

TO MY BEST KNOWLEDGE ALL IMAGES CAN BE USED FOR UNIVERSITY COURSES.