Real-time Shadows

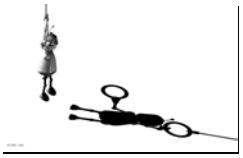
Xavier Décoret

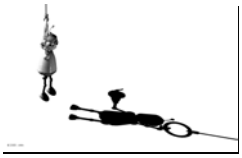
Ombre et lumière

- Les ombres augmentent le réalisme

Ombre et lumière

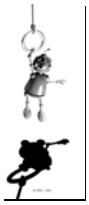
- Les ombres augmentent le réalisme
- Les ombres aident à percevoir
  - des objets cachés

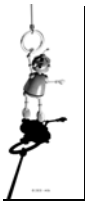




Ombre et lumière

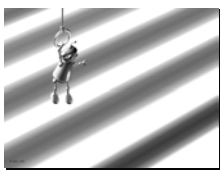
- Les ombres augmentent le réalisme
- Les ombres aident à percevoir
  - des objets cachés
  - le positionnement relatif des objets





Ombre et lumière

- Les ombres augmentent le réalisme
- Les ombres aident à percevoir
  - des objets cachés
  - le positionnement relatif des objets
  - la forme des objets



Ombre et lumière

- Contraintes pour les ombres en temps-réel
  - Lampes dynamiques
  - *Shadow casters* dynamiques
  - Surfaces ombrées dynamiques
  - Zombies en option

Ombre et lumière

- Contraintes pour les ombres en temps-réel
  - Lampes dynamiques
  - *Shadow casters* dynamiques
  - Surfaces ombrées dynamiques
  - Zombies en option
- Deux types d'ombres
  - Ombres dures
  - Ombres douces

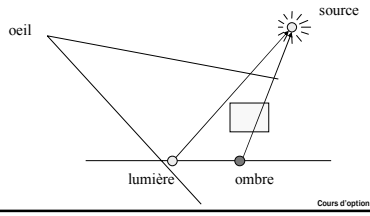
Ombres dures

- Deux approches possibles

- *Shadow Maps*
  - méthode "image"
- *Shadow Volumes*
  - méthode "géométrique"

Shadow Maps (1/3)

- Principe
  - Pour chaque fragment dessiné
    - ⊙ regarder dans la direction de la source
    - ⊙ tester si il y a un objet plus proche
      - – oui : le rayon est bloqué → on est dans l'ombre
      - – non: la source est visible → on est dans la lumière

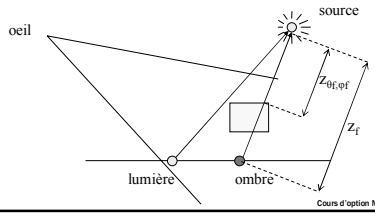


Shadow Maps (2/3)

- Comment?

une fois par frame
- Discrétiser les *shadow casters*
  - ⊙ échantillonner les directions θ,φ depuis la lampe
  - ⊙ stocker la distance $z_{\theta,\varphi}$ du premier objet dans chaque direction

pour chaque fragment f
- Calculer la direction θ_f,φ_f et la distance z_f pixel / source
- Comparer z_f et $z_{\theta f,\varphi f}$

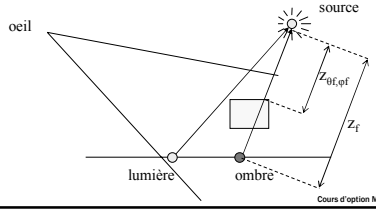


Shadow Maps (3/3)

- En pratique

une fois par frame
- Rendre une vue depuis la source
  - ⊙ conserver le *z-buffer* dans une texture T
  - ⊙ conserver la matrice M de projection

pour chaque pixel f
- Faire un *projective texture lookup* par M dans T
- Comparer z_f et $z_{\theta f,\varphi f}$ en utilisant une extension OpenGL



Rendre une vue depuis la source

- Rendre où?
  - il faut un buffer "caché" : un *pbuffer* de taille wxh
  - il faut une texture pour stocker le *z-buffer*

```
glGenTextures(1,&shadowMap);
glBindTexture(GL_TEXTURE_2D,shadowMap);
glTexImage2D(GL_TEXTURE_2D,0,GL_DEPTH_COMPONENT,
             w,h,0,GL_DEPTH_COMPONENT,GL_FLOAT,NULL);
```

- Récupérer le *z-buffer* et les matrices

```
glCopyTexSubImage2D(GL_TEXTURE_2D,0,0,0,0,0,w,h);
glGetFloatv(GL_MODELVIEW_MATRIX,m);
glGetFloatv(GL_PROJECTION_MATRIX,p);
shadowMtx = multMatrix(p,m);
```

```
GLuint shadowMap;
GLfloat shadowMtx[16];
```

Projective Texture Lookup

- Utiliser la matrice de texture

```
glMatrixMode(GL_TEXTURE_MATRIX);
glLoadMatrixf(shadowMtx);
```

- Passer les coord. 3D dans les coord. textures

```
glTexCoord3fv(v);glVertex3fv(v);
```

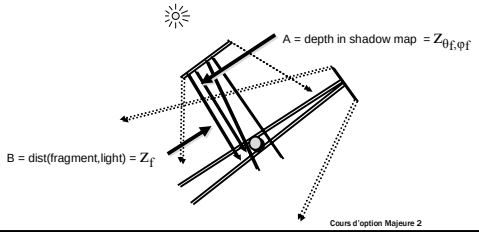
- Activer le mode projectif
  - extension OpenGL (standard maintenant)

```
glTexParameterf( GL_TEXTURE_2D,
                 GL_TEXTURE_COMPARE_MODE,
                 GL_COMPARE_R_TO_TEXTURE);
glTexParameterf( GL_TEXTURE_2D,
                 GL_TEXTURE_COMPARE_MODE,
                 GL_LEQUAL);
```

  - le fragment a une couleur 0 ou 1

Points techniques (1/3)

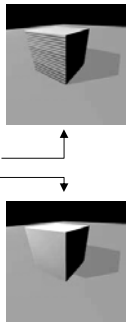
- Auto-ombrage des surface
  - précision limitée pour l'encodage du z
    - conflit de précision
      - en théorie A = B
      - en pratique A ≈ B



Points techniques (1/3)

- Auto-ombrage des surface
  - précision limitée pour l'encodage du z
    - conflit de précision
  - solution : biaiser la carte de profondeur
    - valeur idéale?
      - trop peu : les surfaces s'ombrent elles-mêmes
      - trop : certaines ombres disparaissent
    - solution acceptable en OpenGL

```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(3.0f,1.0f);
```



Points techniques (2/3)

- Rendu rapide de la shadow map
  - seul le z nous intéresse
    - on désactive tout le reste

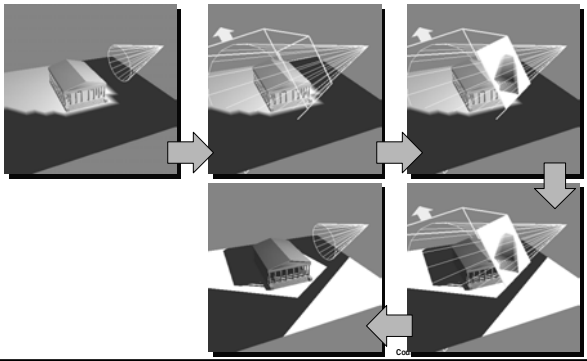
```
glColorMask(0,0,0,0);
glDisable(GL_LIGHTING);
```

  - on ne la met à jour que
    - quand la source bouge
    - quand les *shadow casters* bougent
- Choix des *shadow casters*
  - problème des torchères

Points techniques (3/3)

- Combiner avec l'éclairage OpenGL
  - on obtient une texture de 1 (lumière) et 0 (ombre)
  - on plaque cette texture en modulant la couleur
    - approche naïve
      - rendre la scène avec l'éclairage OpenGL
      - problème du spéculaire dans l'ombre!
    - approche correcte
      - rendre la scène avec juste l'éclairage ambiant
      - rendre la scène avec le diffus modulé par l'ombre
    - aujourd'hui
      - tout combiner dans un *fragment program*

Shadow map: bilan

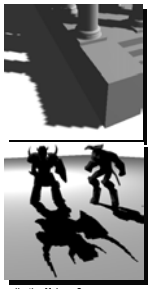


Shadow map: bilan

- Avantages
  - très simple à implémenter
  - marche pour des scènes quelconques
  - coût indépendant de la scène
- Inconvénients
  - deux (voir trois) rendus de la scène
  - sources omni-directionnelles?
  - problème d'aliassage

Aliassage

- Artefacts très visible et caractéristiques
- Solution : augmenter la résolution de la carte
  - limite liée à la carte
- Pas toujours suffisant :
  - problème théorique

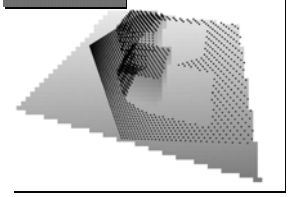


[Sen et al. 2003]

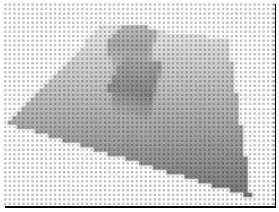
Aliassage: théorie (1/2)

vue de la caméra

reprojection dans la *shadow map*



échantillonnage dans la *shadow map*

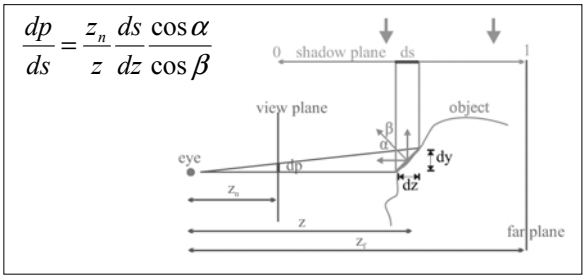


Points où la profondeur est demandée ≠ Points où la profondeur est connue

Aliassage: théorie (2/2)

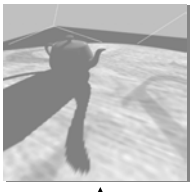
- *Perspective* vs. *projection aliasing*

$$\frac{dp}{ds} = \frac{z_n}{z}\frac{ds}{dz}\frac{\cos\alpha}{\cos\beta}$$



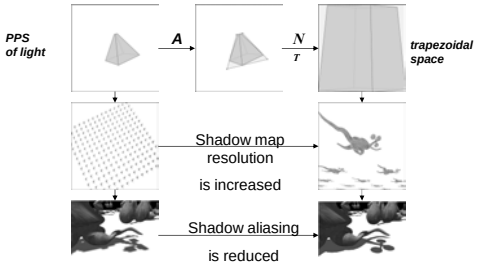
Aliassage

- Artefacts très visible et caractéristiques
- Solution : augmenter la résolution de la carte
  - limite liée à la carte
- Pas toujours suffisant :
  - problème théorique
  - cas idéal :
    - ⊙ source proche de la caméra
      - – lampe casque
  - cas le pire :
    - ⊙ source opposée à la caméra
      - – animal dans les phares (pas bon pour lui)



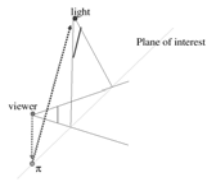
Aliassage: solutions (1/4)

Anti-aliasing and Continuity with Trapezoidal Shadow Maps [SoR04]
T. Martin, T.-S. Tan



Aliassage: solutions (2/4)

A Lixel for every Pixel [SoR04]
H. Chong, S. Gortler



Light Space Perspective Shadow Maps [SoR04]
M. Wimmer, D. Scherzer, W. Purgathofer

Aliassage: solutions (3/4)

Silhouette Shadow Maps [Sig03]
P. Seen, M. Cammarano, P. Hanrahan

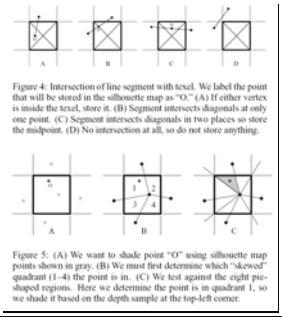
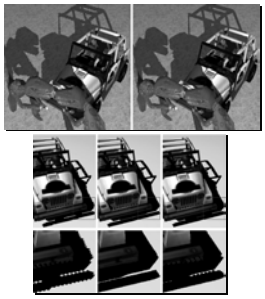


Figure 4: Intersection of line segment with texel. We label the point that will be stored in the silhouette map as "O." (A) If either vertex is inside the texel, store it. (B) Segment intersects diagonals at only one point. (C) Segment intersects diagonals in two places so store the midpoint. (D) No intersection at all, so do not store anything.

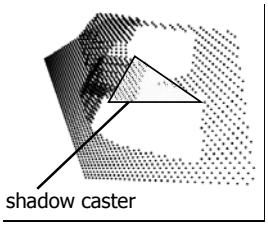
Figure 5: (A) We want to shade point "O" using silhouette map points shown in gray. (B) We must first determine which "skewed" quadrant (1–4) the point is in. (C) We test against the eight pie-shaped regions. Here we determine the point is in quadrant 1, so we shade it based on the depth sample at the top-left corner.

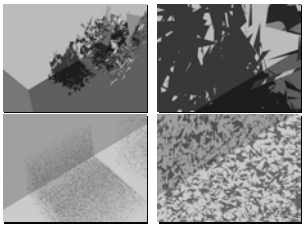


Aliassage: solutions (4/4)

Alias-Free Shadow Maps [SoR04]
T. Aila , S. Laine

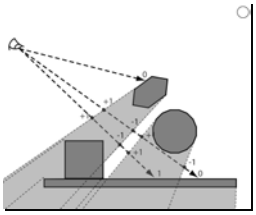
- rasterisation logicielle des shadow casters





Shadow Volumes (1/3)

- Principe
  - pour chaque *shadow casters*
    - ⊙ construire un volume d'ombre
  - pour chaque fragment dessiné
    - ⊙ compter combien de fois on entre/sort d'un volume
      - – > 0 : dans l'ombre
      - – = 0 : dans la lumière



Shadow Volumes (2/3)

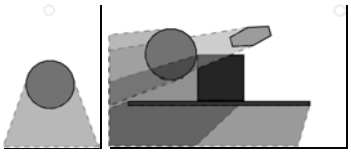
- Comment?
  - construire les volumes d'ombres
    - ⊙ trouver la silhouette des objets vus depuis la source
    - ⊙ construire des *quads* infinis s'appuyant
      - – sur la source
      - – sur chaque arête de silhouette
  - compter les entrées/sorties
    - ⊙ utiliser le *stencil buffer*

Trouver la silhouette

- Algorithme
  - pour chaque arête du modèle :
    - ⊙ identifier les faces gauche/droite et leurs normales
    - ⊙ calculer les prod. scal. normales/vecteur vers la source
    - ⊙ marquer comme silhouette si de signe différents
  - fait sur le CPU, possibilité sur le GPU
- Requiert les infos d'adjacence du maillage
- Calcule un sur-ensemble de la silhouette

Construire les volumes d'ombres

- On extrude les arêtes de silhouette vers l'∞
- On obtient des *shadow quads*
  - ces *shadow quads* sont orientés
    - ⊙ *front* ou *back facing*
  - ils forment des volumes d'ombres imbriqués
- Dans l'ombre = dans au moins un volume



Stencil Buffer

- Buffer auxilliaire non affiché
  - en plus couleur et profondeur
- Contrôle si un fragment "passe" ou pas
  - spécification d'un *stencil test*

```
glStencilFunc(GL_EQUAL,0,~0);
```

- Modifié lors de la rasterisation
  - incrémentable/décrémentable
  - trois actions spécifiables
    - ⊙ le fragment rate le *stencil test*
    - ⊙ le fragment passe le *stencil test* et passe/rate le *z-test*

```
glStencilOp(GL_KEEP,GL_KEEP,GL_DECR);
```


Stenciled Shadow Volumes (1/3)

- Premier rendu de la scène
  - Initialise le *Z buffer*
- Rendu du volume d'ombre
  - Pour chaque *front facing shad. quad*

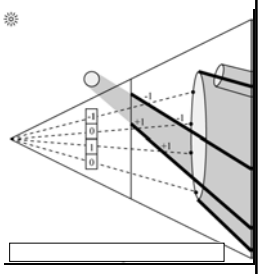
```
glStencilOp(GL_KEEP,GL_KEEP,GL_INCR);
```

  - Pour chaque *front facing shad. quad*

```
glStencilOp(GL_KEEP,GL_KEEP,GL_DECR);
```

- Deuxième rendu de la scène
  - Pour la partie éclairée

```
glStencilFunc(GL_EQUAL,0,~0);
```



Z pass

Code : rendu *shadow volumes*

Z pass

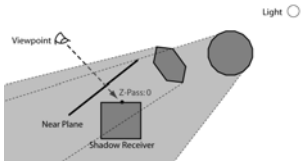
```
drawScene();              // La scène, écl. ambiant

glDepthMask(0);           // Ne pas écrire ds Z-buffer
glStencilFunc(GL_ALWAYS, 0, ~0);
glEnable(GL_STENCIL_TEST);
glEnable(GL_CULL_FACE);
glColorMask(0,0,0,0);     // pas modifier framebuffer
// front-facing quads
glCullFace(GL_BACK);
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
draw_shadow_quads();
// back-facing quads
glCullFace(GL_FRONT);
glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
draw_shadow_quads();
glColorMask(1,1,1,1);
glDepthMask(1);        // On peut écrire ds Z-buffer
```


Code : rendu de la scène

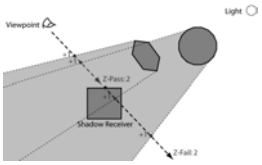
```
glStencilFunc(GL_EQUAL, 0, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glEnable(GL_STENCIL_TEST);
glDepthFunc(GL_EQUAL);
glEnable(GL_LIGHTING);
drawScene();
```


Z-pass ne marche pas!



- Valeur du stencil pour le point rouge incorrecte
  - des *shadow quads* vitaux ont été clippés par le *near plane*
- Comment faire?
  - il faut "capper" les *shadow volumes* sur le *near plane*
    - ⊙ garantir que les *shadows volumes* sont fermés
  - dur à faire en logiciel...

Stenciled Shadow Volumes (2/3)

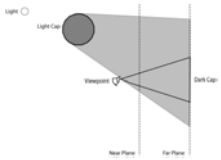


Z fail

[Carmack 00]

- Compter depuis l'∞ au lieu de depuis l'oeil
  - résultat équivalent dans le *stencil*
  - le *near plane* ne peut pas couper des volumes vitaux
    - ce qui est devant ce plan ne peut rater le *z test*!

Stenciled Shadow Volumes (2/3)



- Compter depuis l'∞ au lieu de depuis l'oeil
  - résultat équivalent dans le *stencil*
  - le *near plane* ne peut pas couper des volumes vitaux
    - ce qui est devant ce plan ne peut rater le *z test*!
- Oui mais problème symétrique avec le *far plane*?!?
  - heureusement on peut capper très simplement cette fois!

Z fail et cap

- Rendre les *shadow quads* comme avant
- Rendre le modèle
  - triangle face à la lumière → pas bougé *light cap*
  - triangle opposé à la lumière → envoyé à l'∞ *dark cap*
- Clamper à z=1 les fragments z>1
  - astuce: modifier la matrice de projection
  - mieux: extension `GL_NV_depth_clamp`

Code : rendu *shadow volumes*

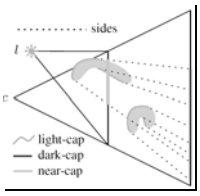
```
drawScene();              // La scène, écl. ambiant

glDepthMask(0);           // Ne pas écrire ds Z-buffer
glStencilFunc(GL_ALWAYS, 0, ~0);
glEnable(GL_STENCIL_TEST);
glEnable(GL_CULL_FACE);
glColorMask(0,0,0,0);     // pas modifier framebuffer
// front-facing quads caps
glCullFace(GL_BACK);
glStencilOp(GL_KEEP, GL_INCR, GL_KEEP);
draw_shadow_quads_and_caps();
// back-facing quads and caps
glCullFace(GL_FRONT);
glStencilOp(GL_KEEP, GL_DECR, GL_KEEP);
draw_shadow_quads_and_caps();
glColorMask(1,1,1,1);
glDepthMask(1);        // On peut écrire ds Z-buffer
```

Z fail

Stenciled Shadow Volumes (3/3)

- Z-fail bien mais coûteux
  - rendu des caps en plus
  - casse les optimisations OpenGL
    - display lists, triangle strips,etc..
- La solution Z-pass+
  - [Hornus05]
  - voir le papier
  - vraiment très bien



Points techniques (1/2)

- Envoyer un point à l'infini
  - déplacer d'une distance "grande"
    - nécessite calcul
    - et si on voit cette distance?
  - mettre à 0 la coordonnée homogène
    - direct!

```
glVertex4f(v[0],v[1],v[2],0);
```

    - envoie vraiment à l'infini!

Points techniques (2/2)

- Extensions OpenGL
  - `GL_NV_depth_clamp`
  - `GL_EXT_stencil_wrap`
    - ⊙ évite les dépassements dans le *stencil buffer*
  - `GL_EXT_stencil_two_side`
    - ⊙ Traiter les *front* et *back facing* en une seule passe
  - `GL_EXT_depth_bounds_test`
    - ⊙ Ne rasteriser que les primitives proches de la source

Shadow Volume Clamping (1/3)

- Les *shadow volumes* augmentent le *fill-rate*
- Certains *shadow quads/caps* sont inutiles
  - atténuation de la lumière

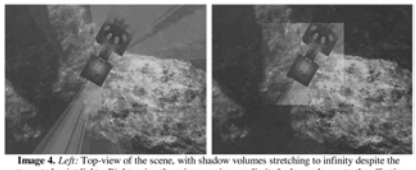


Image 4. *Left:* Top-view of the scene, with shadow volumes stretching to infinity despite the attenuated point light. *Right*: using the scissor region, we limit shadow volumes to the effective radius of the light and conserve fill rate.

Shadow Volume Clamping (2/3)

- Les *shadow volumes* augmentent le *fill-rate*
- Certains *shadow quads/caps* sont inutiles
  - atténuation de la lumière
  - inclusion dans un autre *shadow volume*

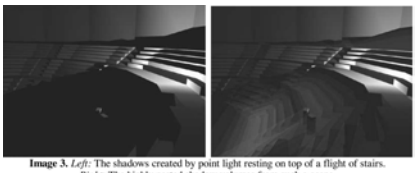


Image 3. *Left:* The shadows created by point light resting on top of a flight of stairs. *Right*: The highly nested shadow volumes from such a scene.

Shadow Volume Clamping (3/3)

- Les *shadow volumes* augmentent le *fill-rate*
- Certains *shadow quads/caps* sont inutiles
  - atténuation de la lumière
  - inclusion dans un autre *shadow volume*

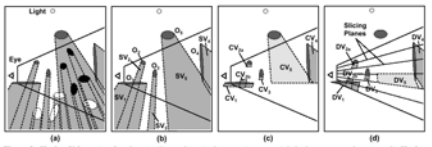


Figure 2: ***Shadow Volume Acceleration****: (a) Every object in the scene is a potential shadow caster and receiver. (b)* ***Shadow Volume Culling****: The shadow casters visible to the light are PSC = {O1,…,O5} and shadow receivers visible to the eye are PSR = {O1, O2, O3, O5}. (c)* ***Continuous Shadow Clamping:*** *Shadow volumes SV1 are clamped by using AABBs around the shadow receivers to compute clamped volumes CV1. (d)* ***Discrete Shadow Clamping:*** *SV1 are clamped to intervals defined by slicing planes. Testing for actual object containment results in a smaller DV1. Discrete clamping may be combined with continuous clamping to refine the poorly clamped CV1, producing a optimal set of shadow volumes {CV1…CV4, DV5} which significantly reduces fill requirements.*

CC Shadow Volumes
[SoR04]
B. Lloyd,
J. Wendt,
N. Govindaraju,
D. Manocha

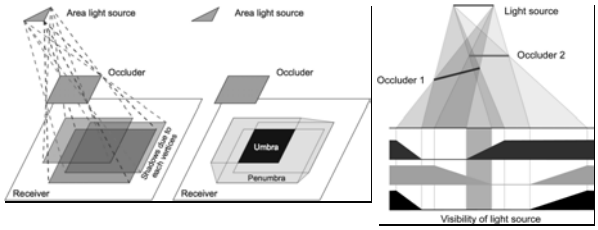
Shadow volumes: bilan

- Avantages :
  - ombres précises
  - positions quelconques source/caméra
  - robuste si bien programmé,
- Inconvénients :
  - calcul de la silhouette (sur CPU, év. long)
  - scènes bien modelisées préférables
  - deux rendus de la scène + rendu des volumes
  - *fill-rate* limité

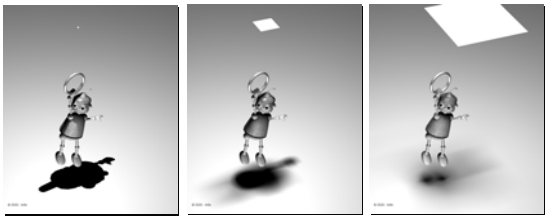
Ombres douces

- Algorithmiquement plus compliqué
  - problème de visibilité point-surface
    - ⊙ au lieu de point-point
    - ⊙ silhouette ?
  - ombre de la somme ≠ somme des ombres
  - problème de la variation de normale
    - ⊙ approximation par le % de visibilité
- Plusieurs algorithmes approximatifs
  - suffisants pour l'oeil humain
  - voir survey par Hasenfratz et al.

Ombres et pénombre



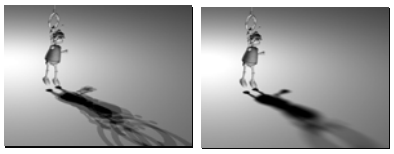
Problèmes de silhouette



Ombres douces par *sampling*

- Accumulation d'ombres :
  - calculer plusieurs ombres ponctuelles
  - additionner et moyenner les résultats
  - *accumulation buffer*
  - nombre d'échantillons élevés
    - ⊙ temps de calcul multiplié par # échantillons

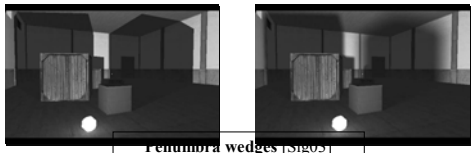
4 échantillons



1024 échantillons

Ombres douces (1/2)

- *Shadow volume* normal
- Pour chaque arête de silhouette :
  - ⊙ calculer volume englobant la pénombre
  - ⊙ pour chaque pixel dans ce volume
    - – calculer coefficient d'atténuation
- Beau, réaliste mais *fill-rate* multiplié par 2



Penumbra wedges [Sig03]
U. Assarson, T. Möller

Ombres douces (2/2)

Rendering Fake Soft Shadows with Smoothies[SoR03]
E. Chan, F. Durand

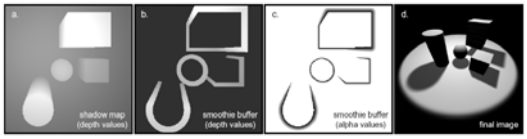


Figure 1: *Smoothie algorithm overview. (a) We first render a shadow map from the light's viewpoint. Next, we construct geometric primitives that we call* smoothies *at the objects' silhouettes. (b) We render the smoothies' depth values into the smoothie buffer. (c) For each pixel in the smoothie buffer, we also store an alpha value that depends on the ratio of distances between the light source, blocker, and receiver. (d) Finally, we render the scene from the observer's viewpoint. We perform depth comparisons against the values stored in both the shadow map and the smoothie buffer, then filter the results. The smoothies produce soft shadow edges that resemble penumbrae.*

Penumbra maps: Approximate soft shadows in Real-time [SoR03]
E. Chan, F. Durand

Résumé & conclusion

- Shadow maps :
  - Stable, robuste, facile, rapide, aliasage
- Shadow volumes :
  - Beau, difficile, complexité algorithmique
- Ombres douces
  - Complexe, lent, beau
- Beaucoup de papiers récents
  - Bibliographie en ligne
    `http://artis.imag.fr/~Xavier.Decoret/index.fr.html`