Shadows

David Odin

Forma3Dev pour CPE Lyon

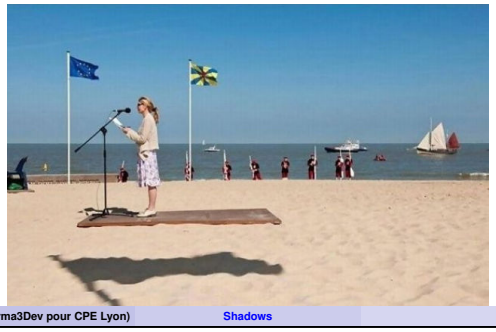
2016

Notes

INTÉRÊT

Les ombres portées sont importantes pour le réalisme de votre scène :

- Indice visuel (permet de connaître la hauteur d'un objet)
- Renforce la crédibilité des autres techniques d'éclairage



Notes

DIFFÉRENTES APPROCHES

- Projection du modèle sur un plan, en noir.
- Parallèle ou centrale
- Stencil shadows
- Shadow maps
- Soft edges
- Raytracing
- etc.

Notes

$z = 0$

- Technique d'ombrage la plus simple
- On dessine simplement le modèle en noir
- Les coordonnées des vertices sont alors $(x, y, 0)$

Notes

Résultat

Notes

Projection centrale

- Amélioration de la méthode $z = 0$
- Prise en compte de la position de la source de lumière
- Chaque point S de l'ombre est calculé comme le projeté du point V du modèle sur un plan P par la projection centrée sur la source de lumière L

On a les relations suivantes :

$$S = L + \alpha(V - L)$$

$$S \in P$$

avec

$$P = (a, b, c, d)$$

$$L = (l_x, l_y, l_z, l_w)$$

Notes

UN PEU DE MATH

On cherche la matrice M qui transforme V en S.
La formule de cette matrice est étonnament simple :

$$M = P \cdot LI - L^T xP$$

$$M = P \cdot L \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} l_x \\ l_y \\ l_z \\ l_w \end{pmatrix} \begin{pmatrix} a & b & c & d \end{pmatrix}$$

$$M = \begin{pmatrix} P \cdot L & 0 & 0 & 0 \\ 0 & P \cdot L & 0 & 0 \\ 0 & 0 & P \cdot L & 0 \\ 0 & 0 & 0 & P \cdot L \end{pmatrix} - \begin{pmatrix} l_x \cdot a & l_x \cdot b & l_x \cdot c & l_x \cdot d \\ l_y \cdot a & l_y \cdot b & l_y \cdot c & l_y \cdot d \\ l_z \cdot a & l_z \cdot b & l_z \cdot c & l_z \cdot d \\ l_w \cdot a & l_w \cdot b & l_w \cdot c & l_w \cdot d \end{pmatrix}$$

Notes

UN PEU DE MATH : RÉSULTAT

$$M = \begin{pmatrix} P \cdot L - l_x \cdot a & -l_x \cdot b & -l_x \cdot c & -l_x \cdot d \\ -l_y \cdot a & P \cdot L - l_y \cdot b & -l_y \cdot c & -l_y \cdot d \\ -l_z \cdot a & -l_z \cdot b & P \cdot L - l_z \cdot c & -l_z \cdot d \\ -l_w \cdot a & -l_w \cdot b & -l_w \cdot c & P \cdot L - l_w \cdot d \end{pmatrix}$$

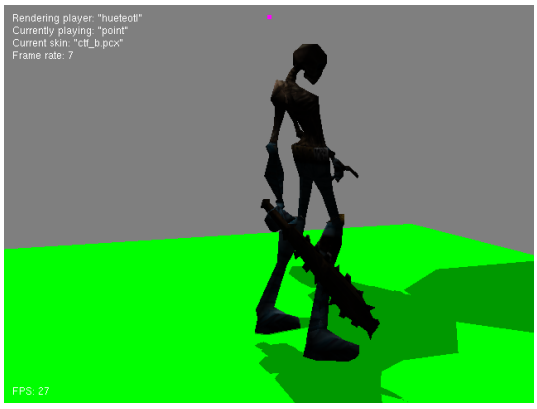
Note : Si $P = (0, 0, 1, 0)$ et $L = (0, 0, 1, 0)$, on retrouve le cas "$z = 0$".

$$M = \begin{pmatrix} 1-0 & -0 & -0 & -0 \\ -0 & 1-0 & -0 & -0 \\ -0 & -0 & 1-1 & -0 \\ -0 & -0 & -0 & 1-0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

C'est bien la matrice qui "fait $z = 0$".

Notes

RÉSULTAT



Notes

AVANTAGES / INCONVÉNIENTS

Avantages :

- Très facile à implémenter
- Très rapide dans ses cas d'utilisation

Inconvénients

- Limité aux surfaces planes illimitées.
- Ombre unie ou pleine d'artefacts, souvent noire.
- Assez peu réaliste finalement

Notes

UTILISATION D'UN STENCIL BUFFER

L'utilisation d'un stencil buffer peut améliorer le processus pour :

- limiter les ombres aux surfaces réelles
- utiliser une texture différente pour les parties à l'ombre.

Fonctions importantes :

```
1 void glStencilFunc (GLenum func,  // type de test
2                     GLint  ref,   // valeur de comparaison
3                     GLuint mask); // masque de bits pour la comparaison.
```

Cette fonction permet de choisir la fonction de comparaison du test stencil ainsi que la valeur de référence et l'ensemble des bits concernés. La fonction de comparaison est à prendre parmi GL_NEVER, GL_LESS, GL_LEQUAL, GL_GREATER, GL_GEQUAL, GL_EQUAL, GL_NOTEQUAL, ou GL_ALWAYS

```
1 void glStencilMask (GLuint mask); // mask vaut souvent ~0
```

permet de choisir quels bits du stencil buffer peuvent être écrits.

```
1 void glStencilOp (GLenum sfail,  //  si le test stencil foire
2                   GLenum dpfail, //  si le test de profondeur foire
3                   GLenum dppass); //  si les tests reussissent.
```

C'est avec cette fonction que l'on peut modifier le stencil buffer pour chaque pixel, suivant la réussite ou l'échec des tests stencil et de profondeur. Chacun des paramètres peut être : GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_INCR_WRAP, GL_DECR, GL_DECR_WRAP, ou GL_INVERT

Notes

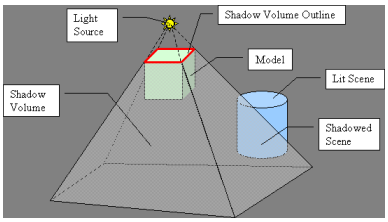
THÉORIE

Les "stencil shadows" sont une méthode pour obtenir des ombres en implémentant des "shadows volumes" à l'aide du stencil buffer. L'algo se décompose en plusieurs étapes :

- Création de la géométrie du *shadow volume*
- Dessin de la scène éclairée
- Dessin du *shadow volume* dans le stencil buffer d'une manière bien particulière
- Redessin de la scène ombrée, en utilisant le résultat du stencil buffer
- Note : contrairement à ce qu'il se passe avec les shadows maps, la caméra ne change pas de position pendant tout le procédé.

Notes

CRÉATION DU SHADOW VOLUME

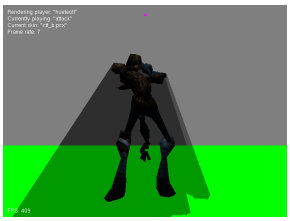


- Le shadow volume est un tronc de pyramide limité par une face du modèle (en rouge) et quatre (si la face est un quadrilatère) quads (en clair sur la figure).
- Ces quads sont théoriquement illimité (et on peut créer des quads illimités en OpenGL. . .) même si en pratique, on les limite souvent à la taille de la scène.

Notes

SHADOW VOLUME

- Chaque bord $[AB]$ crée donc un quad $(A, B, B1, A1)$
- $A1 = L + (A - L) * infini$ (en pratique infini peut être égal à 100 ou 1000 si la lampe ne touche pas le modèle)
- Attention ! L'ordre des points est important.
- Ces quads sont faciles à calculer, et si on les affiche, on obtient quelque chose comme ceci :



Notes

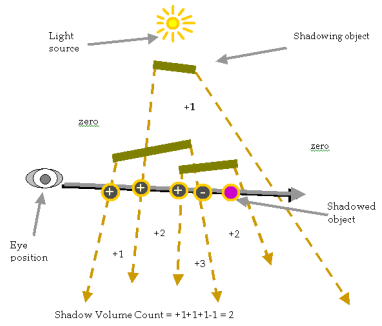
INTERACTION AVEC LE STENCIL BUFFER

- Maintenant, on peut dessiner les quads dans le stencil buffer, et savoir quels sont les points à l'ombre.
- Les quads sont orientés et on sait s'ils vont vers nous ou pas (front ou back).
- On fait donc deux passes dans le stencil buffer. La première avec les quads 'front' qui vont incrémenter le stencil buffer, la seconde avec les quads 'back' qui vont le décrémenter.
- Le dessin se fait en désactivant l'écriture dans les buffers de couleurs et de profondeurs et le test de profondeur est réglé sur inférieur ou égal (les quads qui sont derrière le décor ne nous intéressent pas et ne doivent pas être pris en compte).

Notes

ILLUSTRATION

L'image suivante explique comment fonctionne ce stencil buffer.



Notes

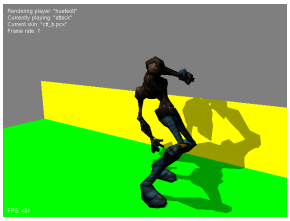
OÙ EN EST-ON ?

- À ce point, le stencil buffer contient des 0 pour les pixels eclairés et une valeur positive pour les points à l'ombre. C'est presque gagné.
- Et non ! Ça ne marche pas du tout !
- Il ne faut pas générer des quads d'ombres pour les bords de toutes les faces !
- Comme nos quads sont orientés, les faces qui "regardent" la lumière engendrent des cônes d'ombre, mais les autres génèrent des cones de lumière !!!
- Il est donc important de ne tracer que les quads engendrés par les bonnes faces
- Cela fait plus de travail pour le CPU, mais la moitié pour le GPU.

Notes

C'EST MIEUX !

En prenant tout cela en compte et en traçant un gros quad noir semi-transparent qui recouvre tout l'écran et en prenant le stencil en compte, on obtient ceci :



C'est bien mais pas parfait : il y a un point lumineux en plein milieu de l'ombre du crane !!! (D'autres points de ce type apparaissent suivant l'angle de vision et l'animation.)

Notes

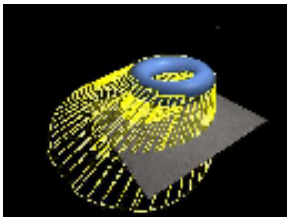
DES P'TITS TROUS...

- D'où proviennent les points lumineux ?
- Ce sont des erreurs d'approximation.
- Beaucoup de "bords" de faces sont utilisés deux fois car ils appartiennent à deux faces qui regardent la lumière toutes les deux.
- Ces bords engendrent deux quads dont les effets s'annulent l'un l'autre.
- Si les deux quads ne sont pas rendus exactement de la même façon, un trou peut se former.
- Ces bords sont donc à éviter puisqu'ils n'apportent que des bétises.
- Il ne faut donc prendre en compte que les bords qui appartiennent à deux faces dont l'une regarde la lumière et l'autre pas.
- Ces bords forment la silhouette.

Notes

ILLUSTRATION

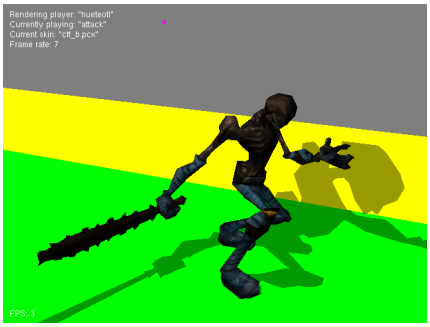
Sur cette image, on voit quels sont finalement les quads à dessiner dans le stencil buffer :



Notes

RÉSULTAT

Une fois ces défauts corrigés, on obtient l'image suivante :



Notes

IMPLÉMENTATION

Passons maintenant à la partie implémentation, qui reprendra le même plan :

- calcul des adjacences
- calcul de la visibilité des faces par la lumière
- dessin de la scène dans les buffers de couleurs et de profondeur
- dessin du *shadow face.volume* dans le stencil buffer.
- utilisation du stencil pour dessiner les ombres.

Notes

ADJACENCES

- Étape très dépendante des données
- Peut être faite une fois pour toute à l'initilisation
- L'idée est de pouvoir retrouver rapidement les trois faces adjacentes à une face donnée.
- Ajout d'un tableau `next` contenant les indices des faces adjacentes.

Notes

VISIBILITÉS

- Il s'agit de la visibilité de la face par rapport à la lumière.
- Nos faces étant orientées, on peut trouver une normale à la face.
- Un simple produit scalaire (dot) permet de connaître la visibilité de la face
- Si une face n'est pas visible, on ne créera aucun quad d'ombre pour cette face
- Cette étape doit être faite à chaque fois que la lampe ou le modèle change (à chaque frame dans notre cas).

Notes

DESSIN DE LA SCÈNE

Simple dessin de la scène, sans le modèle qui fait de l'ombre.

Notes

PRÉPARATION

Préparation du stencil buffer et autres paramètres :

```
1 glColorMask(0, 0, 0, 0);  // on laisse tranquille ces deux
2 glDepthMask(GL_FALSE);    // buffers.
3
4 glEnable(GL_STENCIL_TEST); // on est là pour ça...
5 glStencilFunc(GL_ALWAYS, 1, 0xffffffff); // on compare à un, réussit tout le temps.
6
7 glEnable(GL_CULL_FACE); // On active le back-face culling,
8 glCullFace(GL_BACK);    //   pour que seuls les faces utiles soient tracées.
9 glDepthFunc(GL_LEQUAL);
```


Notes

PREMIÈRE PASSE

On trace alors les quads d'entrée en zone d'ombre dans le stencil buffer :

```
1  glFrontFace(GL_CCW); // on ne tracera que les faces qui sont dans le sens trigo
2                       // de notre point de vue !
3  glStencilOp(GL_KEEP, // n'arrive jamais !
4              GL_KEEP, // profondeur échoue => l'ombre est derrière, on ne fait rien
5              GL_INCR);// profondeur réussit => on incrémente le pixel
6  pour toutes les faces
7    si elle est visible
8      pour les 3 bords
9        si la face adjacente à la face par ce bord n est pas visible
10         tracer le quad A B B1 A1
```


Notes

SECONDE PASSE

Puis on fait pareil mais pour les autres polygones et en décrémentant :

```
glFrontFace(GL_CW);   // on ne tracera que les faces qui ne sont pas dans le sens trigo
                      // de notre point de vue !
glStencilOp(GL_KEEP,  // n'arrive jamais !
            GL_KEEP,  // profondeur échoue => l'ombre est derrière, on ne fait rien
            GL_DECR); // profondeur réussit => on décremente le pixel
pour toutes les faces
  si elle est visible
    pour les 3 bords
      si la face adjacente à la face par ce bord n est pas visible
        tracer le quad A B B1 A1
```


Notes

UTILISATION DU STENCIL

À ce point, on peut faire ce que l'on veut avec le stencil. Le plus simple est de faire un gros quad de tout l'écran qui ne noircira que les zones où le stencil n'est pas nul :

```
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE); // on rebranche l'écriture

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glStencilFunc(GL_NOTEQUAL, 0, 0xffffffff); // on ne trace que la ou le stencil n'est pas à 0
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP); // sans jamais le modifier.

glColor4f(0, 0, 0, 0.4); // on dessine avec du noir semi transparent
// un gros quad.
glPushMatrix();
glLoadIdentity();
glBegin(GL_TRIANGLE_STRIP);
  glVertex3f(-0.1f, 0.1f,-0.10f);
  glVertex3f(-0.1f,-0.1f,-0.10f);
  glVertex3f( 0.1f, 0.1f,-0.10f);
  glVertex3f( 0.1f,-0.1f,-0.10f);
glEnd();
glPopMatrix();
glDisable(GL_BLEND);
```

Mais on peut imaginer plein d'autres choses.

Notes

AVANTAGES / INCONVÉNIENTS

Avantages

- Assez facile à implémenter.
- Le shadow volume garantit que l'on peut avoir des ombres projetées sur n'importe quelle forme.

Inconvénients

- Demande de dessiner le modèle 3 fois !
- Gros problèmes si la caméra est à l'ombre
- Demande des calculs de visibilité, ce qui est limitant si l'application est juste au niveau CPU
- Les ombres sont dures.

Notes

GL_EXT_STENCIL_TWO_SIDE

Depuis OpenGL 2.0
Cette extension permet de résoudre le problème du dessin multiple du modèle en donnant des réglages différents au niveau du stencil pour les faces avant (front) et arrière (back).
Elle ajoute les fonctions suivantes :

```
1  void glStencilFuncSeparate (GLenum face,   // GL_FRONT ou GL_BACK
2                              GLenum func,   // type de test
3                              GLint  ref,    // valeur de comparaison
4                              GLuint mask); // masque de bits pour la comparaison.
5  void glStencilMaskSeparate (GLenum face,   // GL_FRONT ou GL_BACK
6                              GLuint mask); // mask vaut souvent ~0
7  void glStencilOpSeparate (GLenum face,     // GL_FRONT ou GL_BACK
8                            GLenum sfail,    //  si le test stencil foire
9                            GLenum dpfail,   //  si le test de profondeur foire
10                           GLenum dppass);  //  si les tests reussissent.
```

Ces fonctions ont le même rôles que leur équivalent sans le suffixe Separate, mais elles permettent de choisir des réglages différents pour les deux types de faces. On gagne donc une passe.

Notes

AMÉLIORATIONS ET EXTENSION

- L'utilisation des vertex shaders (et éventuellement des geometry shaders) peut permettre d'accélerer la création du shadow volume.
- Carmark's reverse pour gérer le cas où la caméra est dans l'ombre
- Suivant les cas, on utilisera z-pass ou z-fail.
- Un peu de flou sur les contours de l'ombre.
- En ajoutant une passe et un FBO on peut même obtenir une ombre dont le flou dépend de la distance au point de contact (très réaliste)
- etc.

Notes

Notes