De l'autre coté du miroir...

par Nithril

Les jeux actuels manquent cruellement de miroir. Outre la fonction de mirer son personnage dans tous les sens (ex. de manque: Heavy Metal Fakk 2), ils apportent une certaine touche esthétique (un plan d'eau reflétant l'environnement à la Xisle) mais aussi scénarique pouvant provoquer de savoureux effets dramatiques. Imaginez un monstre du Chaos aux griffes démesurées surgissant derrière vous et voulant vous enlacer en une mortelle étreinte. Mais vous, vous trouvant face à un miroir, d'une pirouette arrière vous lui défoncer ce qui semble sa tête. Intéressant... Nous allons donc aborder la conception de miroir accéléré. Si vous connaissez d'autres techniques, des optimisations, mail.

De ce qu'on attend d'un miroir...

Qu'il reflète correctement l'environnement pour produire un effet réaliste. Cela peut paraître aisé, mais concevoir un vrai miroir peut amener quelques problèmes qui n'ont pas forcement de solution accéléré (tout du moins pas encore).

De ce qu'il faut savoir à propos d'un miroir...

Ce qu'un miroir reflète est directement dépendant de l'orientation de la caméra. Soit la transformation T qui a un objet X associe son image X', cette transformation peut être assimilé à une symétrie planaire avec un plan P:
T(X,P) = X'.

Ce qui est vu par la caméra dans le miroir est l'image de la scène par la transformation T à du clipping prêt. On peut cependant faire la même chose en définissant l'image de la caméra par T et en rendant ensuite la scène par cette caméra.

Sur la fig. 1 on peut voir une scène comportant une caméra 1 Cam1, un miroir M, une caméra 2 Cam2 qui est l'image de Cam1 par T avec M et enfin un objet O. La fig. 2 et la fig. 3 nous montre les deux vues issues des deux caméras. En faisant un blending de ces deux vues nous verrions que les deux objets se recoupent. En affichant l'image de O par T en utilisant Cam1 nous aurions eu le même résultat

Du calcul de T

Il s'agit simplement de calculer la matrice qui pour un objet O nous donnera son image O'.

En utilisant le plan de réflexion nous allons construire l'espace vectoriel tel que dans cette espace il représente le plan XZ. Dans ce nouvel espace il suffira d'appliquer un scalaire k=-1 à la composante Y pour obtenir notre réflexion. Et enfin en multipliant par l'inverse de la matrice de changement de repère nous obtiendrons notre image O' dans le repère global.

Calculons la matrice de changement de repère:
Nous avons besoin de trois vecteurs v1,v2,v3 formant un repère orthonormé.

$$v_1 = normal\ du\ plan$$
$$v_2 = vecteur\ \in au\ plan$$
$$v_3 = v_1 \wedge v_2$$

Avec ces trois vecteurs nous allons créer la matrice de changement de repère M:

$$M = \begin{bmatrix} v_2 & v_1 & v_3 \end{bmatrix}$$

Calculons pour finir la matrice de réflexion R:

$$R = M * \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * M^{-1}$$

et voila...

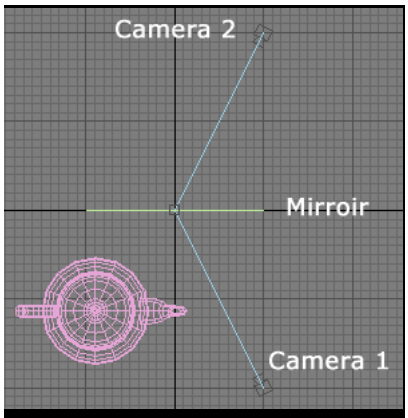

Fig. 1

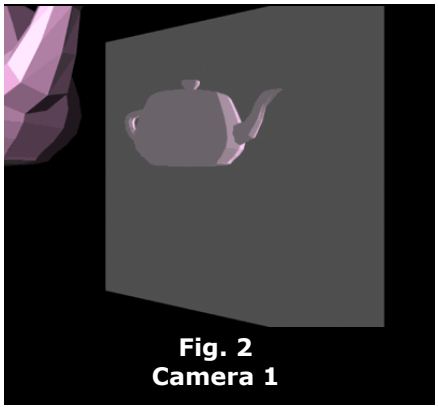
Fig. 2
Camera 1

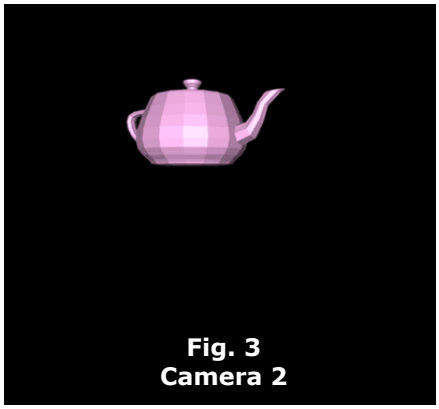
Fig. 3
Camera 2

De la théorie à la pratique

C'est ici que les choses deviennent plus embêtantes. Notre miroir n'est pas un plan mais un polygone, il faut donc clipper, et ca les cartes 3D ne savent pas encore le faire. Il va donc falloir faire avec ce que l'on a.

De l'utilisation du stencil buffer

La manière la plus simple de clipper est d'utiliser le stencil buffer. La grande majorité des cartes en sont maintenant équipées, ma Riva TNT de l'époque avait déjà cette feature, c'est dire.

Nous allons utiliser le stencil comme un masque. Pour cela il faut effectuer plusieurs étapes de pre-rendus:

1. Clear le zbuffer et le stencil
2. Rend le polygone qui servira de mask en activant l'écriture au niveau du stencil et en désactivant celle du zbuffer. On mettra par exemple le stencil à 1.
3. Rend l'images des objets selon T ou selon l'image de la caméra toujours selon T en testant le stencil buffer. Si la valeur du stencil > 0 alors j'écris le pixel.
4. Met à jour le zbuffer en affichant le polygone, qui sert de mask et de miroir, en désactivant l'écriture du pixel

On peut faire une pass supplémentaire en blendant le miroir qu'avec les objets reflétés ie. la valeur du stencil > 1, mais pour cela il faut incrémenter le stencil à chaque écriture d'un pixel des objets reflétés.

Pour plus de réalisme on peut paramètrer le polygone par une map d'opacité et un alpha-test.

Inconvénients:

- Le miroir ne peut être transparent
- Il faut clipper
- Il ne peut y avoir qu'un miroir

Avantage:

- La réflexion est juste
- On peut modifier sans crainte la taille du miroir

Du Chaos au Render to Texture

Allez hop, lançons winamp et «The God that failed» en boucle. Umhh..... ahhhhh..... yeahhh....., guitare électrique rulez. Continuons (taataataa taaaaa, fill the vibration, fill our destruction...), le render to texture a juste ce qu'il faut en plus de complication. Notre texture est rectangulaire, nous allons donc assimilé son pendant 3d à un rectangle. Ce rectangle est le plan de projection de notre nouvelle caméra image par F de la caméra courante. Cette nouvelle caméra doit satisfaire deux contraintes:

- que par F le point d'intersection entre le rayon incident et le plan du miroir o, et i le point d'intersection entre le rayon réfléchit et un objet quelconque soient identiques dans l'espace du plan pour i F(i) et dans l'espace de l'objet pour o et F(0).
- la caméra doit entièrement contenir le rectangle, les quatre droites composants le frustrum doivent contenir les quatre coins du rectangle.

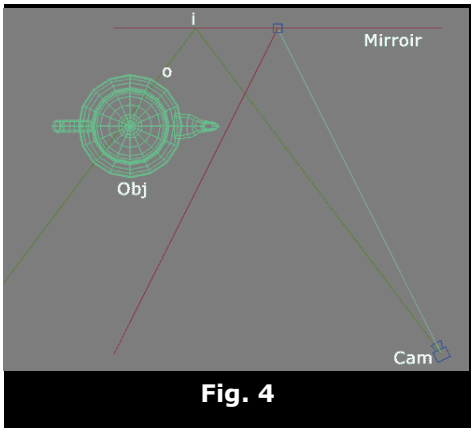

Fig. 4

Une rotation de la caméra ne conserverait pas la propriété i=F(i). Par contre translater Cam1 (l'image de la caméra courante par T) en Cam2 conserverait toutes les propriétés. Il ne faut que translater Cam1, la target quand à elle reste fixe, c'est pour cela que cette translation est un peu spéciale. Cette modification est appelée skew ou shear dans les logiciels de modélisation. Cette translation est fonction de Z à un facteur k pret.

skewx = curx + curz*k

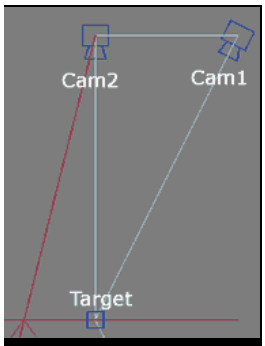

Fig. 5

Nous allons calculer F(T(Cam)) (Cam'') pour calculer le skew:

$$\vec{N} = Normal(Miroir)$$
$$Cam' = T(Cam)$$
$$Cam'' = (\vec{N} \bullet Cam') * \vec{N}$$
$$Dist = \|Cam''\|$$

Ces calculs sont effectués dans l'espace du miroir (Les calculs sont à quelques changements de repère pret :), où le centre du repère est le centre du rectangle.

$$T(Cam) = Cam'$$
$$F(Cam') = Cam''$$

Le skew n'est fait qu'avec X et Y. Z sera la composante du point dont on se servira pour faire la transformation. Calculons k_1 pour X. La fig. 6 est une vue du dessus du plan XZ (horz/vert) :

$$k_1 = Cam'.X / Dist$$

Idem pour k_2.

Construire la matrice est maintenant un jeu d'enfant dont je vous en laisse le plaisir.

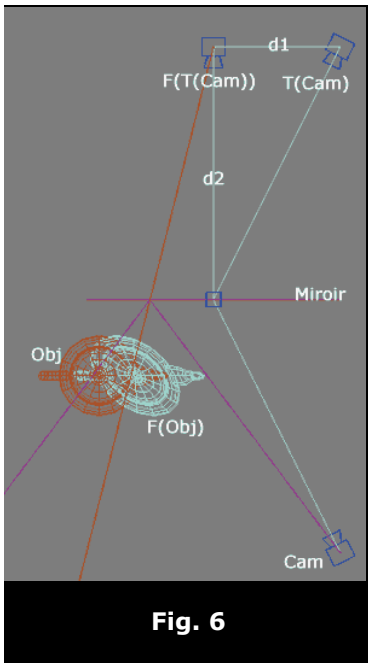

Fig. 6

Il faut simplement faire attention a flipper X et à mettre a jour la matrice de projection.

La scène est maintenant rendu de Cam''.

Inconvénients:

- Imprécision du au render to texture.
- Necessiter d'avoir une carte supportant des texturres de type rendertarget

Avantage:

- la réflexion est juste.
- On peut/doit mettre en place une sorte de mip-mapping.
- On peut faire de la transparence et plein d'autre sympathique effet
- Il n'est pas nécessaire de clipper (enfin je crois =)
- Il peut y avoir plusieur miroirs

Du clonage à JCVD

Il est cependant possible d'éviter le render to texture et l'utilisation du stencil-buffer en clonant par T d'une manière effective les objets visibles depuis le miroir. Cette technique est couramment utilisé pour par exemple simuler une réflexion de l'environnement au niveau du sol (ex: le jeu Revolt). Il faut cependant faire attention à ce qu'aucune face ne se trouve derrière le miroir et vienne parasiter le rendu. De plus, il faut impérativement qu'un polygone occlude les objets clonés non visible via la fenêtre qu'est devenu le miroir (ce polygone joue le rôle du stencil). Dans le cas d'une pièce, fig. 7, cette technique se prête admirablement bien, sous réserve qu'aucune pièce se trouve derrière le mur contenant le miroir. On peut affiner la technique en utilisant un algorithme de visibilité de type portal/cells, le miroir devenant lui même un portal un peu spécial nécessitant maintenant une réflexions dynamique.

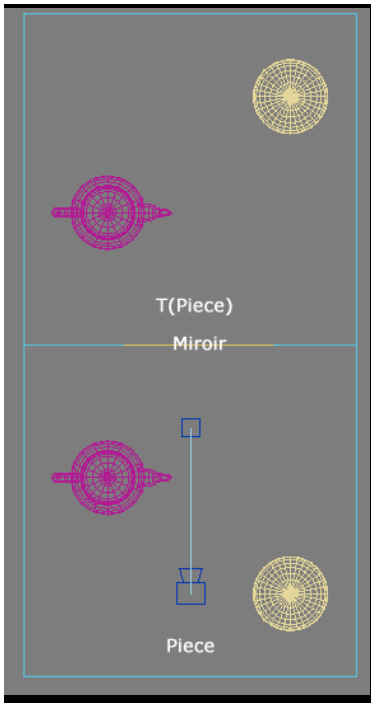

Fig. 7

Inconvénients:

- Un polygone doit clipper
- Le miroir doit être associé à ce polygone

Avantage:

- La réflexion est juste
- Il peut y avoir plusieurs miroirs
- s'associe bien avec les méthodes de visibilités de type portal/cell
- Permet de réfléchir le sol à peu de frais

De la conclusion sur une page

Voila, voila, c'est fini! Si vous relevez des fautes, faire des comments, mailez moi.

Vous pourrez trouver un tut. sur gamasutra sur le render to texture et implementant un miroir (DX7).

Personnelement je me suis fait un petit plugin sous 3dsmax de type texture pour faciliter l'interfaçage avec mon moteur. Je ne peux que vous conseiller d'en faire autant.

Jeux à ma connaissance implémentant des free miroirs:

- Quake3 (Id Software)
- The Devil Inside (Gamesquad)
- Duke Nukem 3D (3D Realms)
- Revolt
- Unreal (Epic)

Demos à ma connaissance implemantant des miroirs (cf. pouet.net):

- Sonnet (ThreeState)
- Rotaliator (Dubious)
- Jumpy (Digital Murder)
- En faites beaucoup de demos software (qui a dit software rulez???? :)

Remerciement

√ Gamasutra
√ Allergy
√ GrosLoup qui sans lui le «Du clonage à JCVD» n'aurait jamais eu lieu d'être