HAL
open science

Méthode temps réel simple pour la correction des mouvements pseudoscopiques en réalité virtuelle

Vincent Nozick, Venceslas Biri

To cite this version:

Vincent Nozick, Venceslas Biri. Méthode temps réel simple pour la correction des mouvements pseudoscopiques en réalité virtuelle. Association Française d'Informatique Graphique AFIG 2003, 2003, France. pp.231-240. hal-00733837

HAL Id: hal-00733837

https://hal.science/hal-00733837v1

Submitted on 19 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Méthode temps réel simple pour la correction des mouvements pseudoscopiques en réalité virtuelle

Vincent Nozick, Venceslas Biri

Université de Marne la Vallée, Institut Gaspard Monge, Equipe SISAR
6 cours du Danube, 77700 Serris, France
vnozick@univ-mlv.fr

Résumé : *Cet article présente une méthode pratique pour supprimer les mouvements pseudoscopiques perçus par un observateur face à une image stéréoscopique. Nous proposons une technique simple qui, à partir de la position de la tête de l'observateur, corrige lors de ses déplacements les déformations anormales de l'image stéréoscopique. Deux solutions sont proposées : soit le référentiel de la scène virtuelle est associé au référentiel de l'écran et alors l'observateur se déplace face à une scène virtuelle fixe, soit le référentiel de la scène virtuelle est associé à l'observateur qui, quelle que soit sa position, voit toujours exactement la même scène. Nous proposons une implémentation simple de ces deux méthodes en OpenGL.*

Mots-clés : réalité virtuelle, images stéréoscopiques, mouvements pseudoscopiques, positon orthostéréoscopique, OpenGL, projection.

1 Introduction

[BC93] définit un système de réalité virtuelle comme "*une interface qui implique de la simulation temps réel et des interactions via de multiples canaux sensoriels. Ces canaux sensoriels sont ceux de l'Homme*". Cette définition souligne l'aspect d'immersion et d'interactivité. Pour offrir une sensation d'immersion maximale, la réalité virtuelle se doit de stimuler tous les sens de l'être humain et notamment la vue, domaine que nous allons traiter plus particulièrement. Le caractère interactif d'un système de réalité virtuelle peut lui aussi être en relation avec certains de ses sens et, là encore, en particulier avec la vue. Le système peut par exemple s'adapter à l'utilisateur en captant la direction de son regard ou bien en adaptant les images générées au point de vue de l'observateur. C'est ce cas que nous allons traiter en l'appliquant à la vision en relief.

Une image stéréoscopique est composée de deux images, une pour chaque œil. En utilisant un appareillage adéquat (lunettes obturatrices, polarisées, etc.), un observateur peut fusionner les deux images et percevoir du relief. De telles images sont réalisables sur de nombreux supports tels que des écrans d'ordinateur [PAB+99], des bureaux immersifs [CPS+97][DMLT92] ou des surfaces plus grandes, éclairées par des vidéo-projecteurs comme les visio-cubes [CNSD93]. Quelque soit la taille de ce genre de dispositif, l'utilisateur est susceptible de se mouvoir devant ces supports. Si l'image affichée n'est pas modifiée, l'observateur verra les objets de la scène se déplacer de façon anormale et ses perceptions du relief s'en trouveront perturbées. Ce type de déformations est connu sous le nom de *mouvements pseudoscopiques* [FMP01]. Si la position de l'observateur est suivie, il est possible de corriger les images.

Dans un premier temps, nous décrirons plus en détails la nature de ces mouvements pseudoscopiques et leurs incidences sur l'observateur. Ensuite, nous proposerons deux méthodes permettant de générer un couple d'images stéréoscopiques dépourvu de mouvement pseudoscopique. La première méthode consiste à associer le référentiel de la scène virtuelle au référentiel de l'écran de manière à ce que l'observateur ait l'impression de se déplacer par rapport à une scène fixe. La seconde méthode associe le référentiel de la scène virtuelle à celui de l'observateur. Celui-ci voit donc exactement la même scène quelle que soit sa position et lorsqu'il se déplace, toute la scène virtuelle bouge avec lui. Enfin, pour chacune de ces deux méthodes nous proposerons une implémentation simple en OpenGL avec quelques images illustrant les résultats obtenus.

2 Mouvements pseudoscopiques et position orthostéréoscopique

Cette section propose une description des phénomènes observables face à une image stéréoscopique, notamment les mouvements pseudoscopiques ainsi qu'une de leurs conséquences, la position orthostéréoscopique.

2.1 Mouvements pseudoscopiques

Une image stéréoscopique est composée de deux images décrivant la même scène, mais d'un point de vue légèrement différent. Un point de la scène visible par les deux yeux de l'observateur sera présent sur les deux images, mais pas au même endroit. A l'aide d'un dispositif (lunettes obturatrices, polarisées, etc.), l'utilisateur va associer une image à chaque œil et percevoir du relief en fusionnant les deux images. Comme nous l'avons remarqué précédemment, lorsque l'observateur se déplace par rapport à l'écran, il constate que la scène se modifie d'une façon qui n'est pas naturelle. La figure 1 illustre ce phénomène en décrivant un déplacement de l'observateur en direction de l'écran. En même temps que l'observateur avance, il voit l'objet se rapprocher vers lui. Normalement, il devrait juste avoir l'impression de s'être rapproché d'un objet immobile.

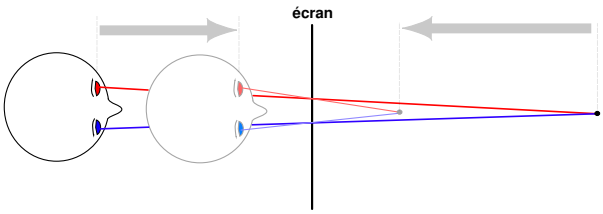


FIG. 1 – Mouvements pseudoscopiques vers l'avant

De la même manière, lorsqu'il effectue un déplacement latéral, l'objet lui semble se déplacer latéralement comme le montre la figure 2. Là encore, il n'a pas la sensation d'avoir à faire à un objet immobile.

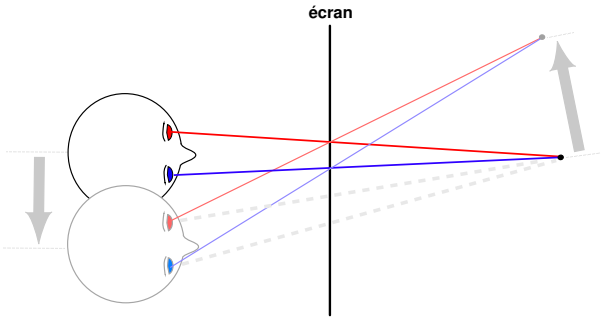


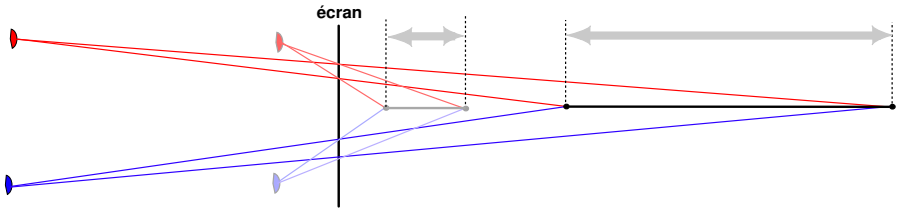
FIG. 2 – Mouvements pseudoscopiques latéraux

Ces mouvements des objets virtuels sensés être immobiles sont appelés *mouvements pseudoscopiques*. Ils ont une influence néfaste sur la sensation de relief perçue par l'observateur, c'est la raison pour laquelle il est nécessaire de les supprimer. En pratique, on constate qu'un observateur peut se permettre de bouger la tête sans subir trop de déformations lorsque l'amplitude du mouvement de la tête est petite par rapport à la taille de l'écran. Ainsi, dans le cas de plusieurs observateurs assis devant un grand écran, il n'est pas nécessaire de corriger l'image. Par contre, cela devient impératif pour un observateur bougeant la tête devant son écran d'ordinateur ou pour un observateur se déplaçant dans un visiocube ou devant un écran de grande taille.

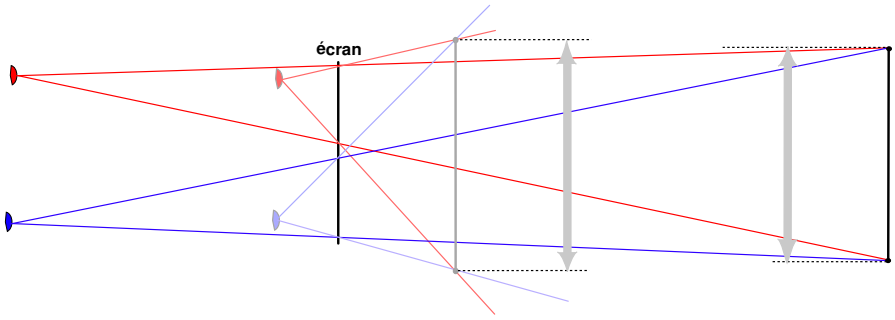
2.2 Position orthostéréoscopique

Il s'avère que lors du déplacement de l'observateur, les mouvements pseudoscopiques pour deux points distincts de la scène ne sont pas identiques et sont d'autant plus différents qu'ils sont éloignés l'un de l'autre. La figure $3(a)$ illustre un segment perpendiculaire à l'écran qui est dessiné vu de près et vu de loin. On constate que lorsque l'observateur se rapproche de l'écran, non seulement le segment s'en rapproche lui aussi mais on constate également que la longueur du segment diminue. Les mouvements pseudoscopiques provoquent donc des distorsions des distances sur les objets de la scène virtuelle. La figure $3(b)$ nous montre un segment positionné parallèlement à l'écran. Dans ce cas, pour le même déplacement de l'observateur que sur la figure $3(a)$, la taille du segment ne diminue pas, elle augmente plutôt. Les perceptions de la largeur et de la profondeur d'un objet virtuel ne sont donc pas

affectées de la même manière par le déplacement de l'observateur. Les mouvements pseudoscopiques sont donc à l'origine de déformations des objets virtuels : les proportions des objets virtuels ne sont plus toujours perçues par l'observateur de façon conforme à celles définies dans la scène virtuelle. En fait, il n'existe qu'une position permettant de restituer la scène dans les proportions désirées, il s'agit de la *position orthostéréoscopique* [Cor97].



(a) distorsion d'un segment perpendiculaire à l'écran



(b) distorsion d'un segment parallèle à l'écran

FIG. 3 – Distorsion des longueurs suite aux mouvements pseudoscopiques

Finalement, nous constatons que non seulement les objets se mettent à bouger quand l'observateur bouge (mouvements pseudoscopiques), mais qu'en plus ils se déforment et ne gardent pas forcément leurs proportions originales. Il a donc deux rectifications à effectuer.

3 Travaux préexistants

Le problème des mouvements pseudoscopiques a été abordé plusieurs fois et résolu de manières différentes. Dans [CNSD93], les auteurs corrigent les images pour un visiocube (*CAVE*) et proposent une matrice de projection associant un point de la scène à une position sur l'écran. Cette matrice est modifiée en fonction de la position de l'observateur. Le point faible de cette méthode est qu'elle ne permet pas un accès facile aux paramètres stéréoscopiques comme l'angle d'ouverture du champs de vision ou bien la distance focale.

Deux autres méthodes proposées dans [FMP01] permettent de générer des images stéréoscopiques et de les ajuster à une vision orthostéroscopique. Elles ne sont cependant valables que pour un observateur centré devant son écran. La première méthode procède par décalage des images générées. L'amplitude de ce décalage est calculée pour restituer une position orthostéroscopique mais ce faisant, on perd les pixels sur les bords de l'écran qui ne sont recouverts que par une seule image. Cette méthode est très répandue pour les films stéréoscopiques tournés avec des objectifs standard (sans objectif à bascule ou à décentrement) mais l'est moins pour les animations en images de synthèse. En effet, dans ce cas, il est possible d'éviter la perte des pixels situés près des bords de l'écran, nous reviendrons sur ce problème au chapitre 4.1. La seconde méthode consiste à faire converger les axes optiques vers "le centre de la scène virtuelle" et s'épargne ainsi un décalage des images. L'inconvénient de cette méthode est qu'elle génère de la parallaxe verticale et donc des images plus difficiles à fusionner. De plus, ces deux méthodes ne corrigent pas les mouvements pseudoscopiques pour des déplacements latéraux.

Enfin, [Ras00] propose une méthode de correction d'images projetées par vidéo-projecteurs décentrés. En plus de corriger les mouvements pseudoscopiques, cette méthode corrige la déformation de l'image projetée par un vidéo-projecteur qui ne serait pas positionné correctement face à l'écran mais un peu de travers. Cette méthode procède en deux étapes. Dans un premier temps, elle calcule l'image en modifiant les paramètres de projection de la caméra virtuelle en fonction des caractéristiques de l'écran (dimensions et position) et de la position de l'observateur. La seconde étape nécessite le calcul d'une homographie pour passer du point de vue de l'observateur à celui du vidéo-projecteur. Il suffit alors d'intégrer cette matrice dans le pipeline de construction de l'image d'OpenGL pour obtenir une image corrigée. Cette méthode est efficace cependant elle ne traite pas spécifiquement des images stéréoscopiques et nécessite le calcul et l'insertion d'une homographie et ce même dans le cas d'un vidéo-projecteur positionné face à l'écran.

4 Méthode de correction des mouvements pseudoscopiques

Dans cette partie, nous allons d'abord rappeler une technique répandue de construction d'images stéréoscopiques en utilisant la librairie graphique OpenGL [SA98]. Nous montrerons ensuite qu'il existe une analogie entre la scène réelle et la scène virtuelle nous permettant d'établir quelques relations d'équivalence. Nous proposerons alors deux méthodes fondées sur ces relations nous permettant de corriger les effets pseudoscopiques.

4.1 Création d'une paire d'images stéréoscopiques

La librairie OpenGL permet de définir des caméras non-symétriques par rapport à l'axe optique. La figure 4 nous montre que les paramètres $left$ et $right$ ne sont par nécessairement égaux. Il en est de même pour top et $bottom$.

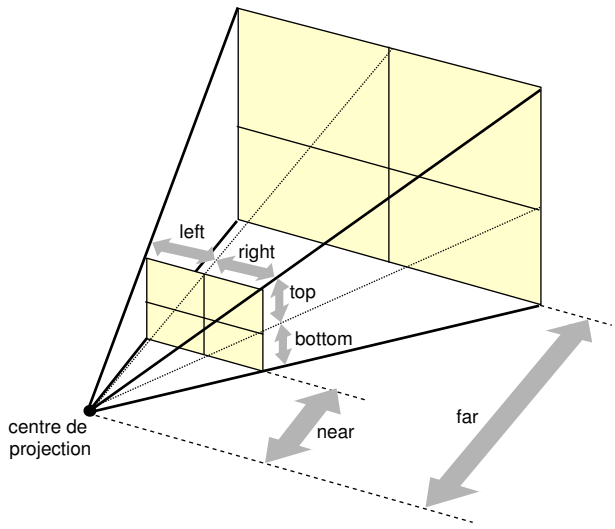


FIG. 4 – paramètres de la camera OpenGL (`frustum`) [WNDS99].

Ce caractère non-symétrique peut s'exploiter de manière à obtenir, pour les deux caméras, des champs de vision qui se croisent avec pourtant des axes optiques parallèles (figure 5).

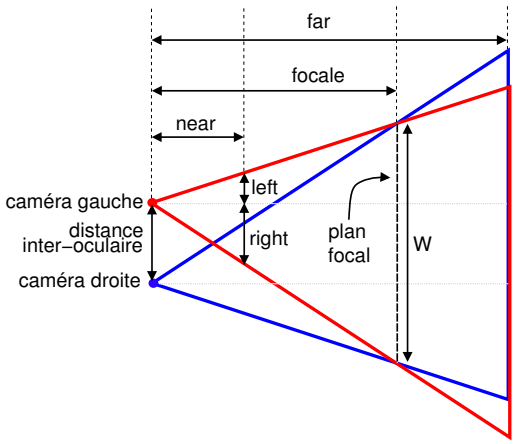


FIG. 5 – système de deux caméras virtuelles, vue de dessus. Les axes optiques sont parallèles.

avec :
- *distance inter-oculaire* (dio) : distance séparant les deux caméras. Notons que l'on s'autorise l'emploi de cette dénomination normalement réservée pour les yeux.
- W : la largeur du plan focal.

On remarque qu'un point situé sur le plan focal aura la même position en pixel sur les deux images donc une parallaxe nulle. Il sera donc perçu par l'observateur comme étant situé sur l'écran. Les objets situés derrière le plan focal seront observés derrière l'écran et réciproquement, les objets situés devant le plan focal seront observés devant l'écran.

Nous insistons sur le fait que les axes optiques soient parallèles. Dans le cas contraire, l'apparition de parallaxe verticale [FMP01] rend difficile la fusion des deux images. Prenons l'exemple d'un rectangle inscrit dans un plan vertical face aux deux caméras. La caméra de gauche étant plus près du bord gauche du rectangle que du bord droit, elle voit le rectangle déformé de telle sorte que son coté gauche apparaît plus grand que son coté droit (figure 6). Inversement, pour la caméra de droite, le rectangle apparaît plus grand du côté droit. Lors de la superposition des deux images, il est clair qu'en plus de la parallaxe horizontale s'est introduite de la parallaxe verticale.

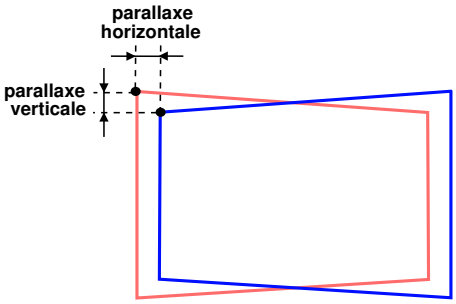


FIG. 6 – Parallaxe verticale.

4.2 Analogies entre la scène réelle et la scène virtuelle

Les deux images stéréoscopiques sont générées avec deux caméras virtuelles non-symétriques. Il ne nous reste plus qu'à placer ces caméras correctement et à les paramétrer en fonction de la position de l'observateur afin qu'à chaque instant celui-ci soit en position orthostéroscopique. Pour cela, nous allons établir un certain nombre de correspondances entre la scène virtuelle et la scène réelle. Le principe de cette démarche est d'associer une caméra

virtuelle à chaque œil de l'observateur, ce qui suppose une série d'analogies :

largeur de l'écran	$\leftrightarrow$	largeur du plan focal W
distance inter-oculaire$_{obs}$	$\leftrightarrow$	distance inter-oculaire$_{cameras}$
distance entre l'observateur et l'écran	$\leftrightarrow$	distance focale
champ de vision de l'observateur à travers l'écran	$\leftrightarrow$	champ de vision des caméras virtuelles à travers le plan focal
le rapport largeur / hauteur de l'écran	$\leftrightarrow$	le rapport largeur / hauteur du plan focal.

Pour passer du référentiel de l'écran à celui de la scène virtuelle, il faut respecter la relation suivante :

$$\frac{dio_{oeil}}{dio_{camera}} = \frac{focale_{oeil}}{focale_{camera}} = \frac{W_{ecran}}{W_{scene\ virtuelle}} = facteur\ d'echelle \tag{4.1}$$

Ces relations traduisent le fait que les deux modèles (yeux de l'observateur/écran et deux caméras/plan focal) sont identiques à un facteur d'échelle près. Certains paramètres de l'équation 4.1 dépendent de la scène réelle et sont donc fixés à l'avance comme dio_{oeil}, $focale_{oeil}$ et W_{ecran}. Il suffit alors de fixer un des paramètres du système de caméras virtuelles pour connaître les deux autres.

Il faut régler les paramètres de la projection perspective de façon à ce que "la pyramide de vision" de chaque caméra passe exactement par les bords de l'écran quelle que soit la position de l'observateur. La figure 7 représente une vue de dessus d'un système de caméras virtuelles compatible avec les yeux d'un observateur.

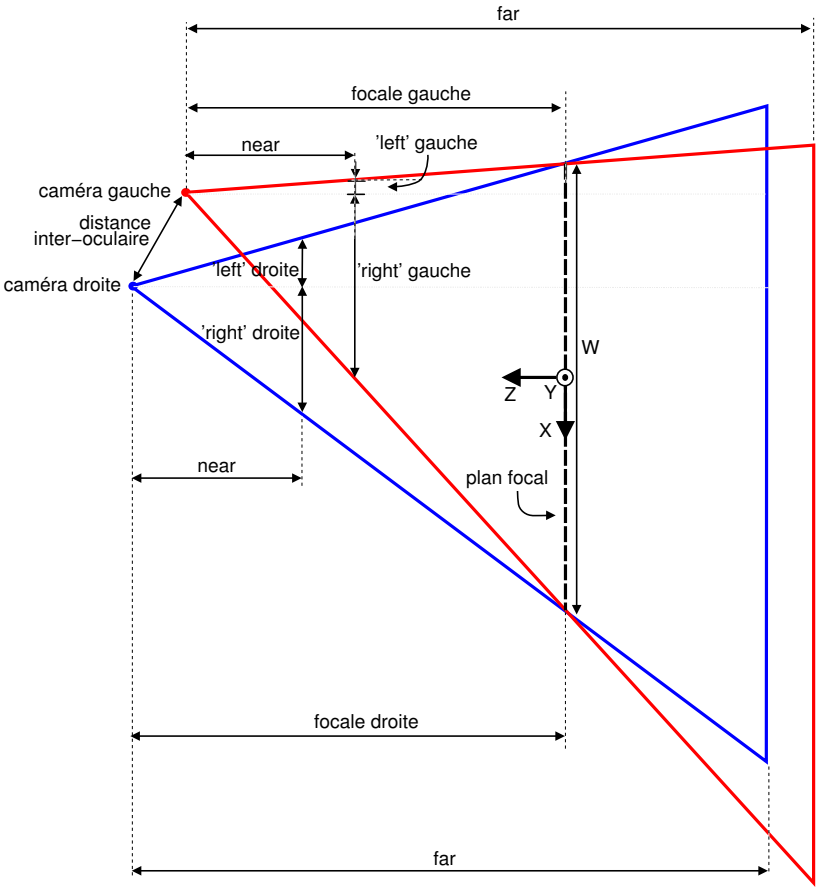


FIG. 7 – Système de deux caméras virtuelles, vue de dessus.

Voici la liste des relations que nous pouvons extraire de la figure 7 :

$$focale_g = \frac{obs_g.z}{facteur\ d'echelle} \tag{4.2}$$

$$focale_d = \frac{obs_d.z}{facteur\ d'echelle} \tag{4.3}$$

$$left_d = \frac{near}{2.focale_d}(W + \frac{2.obs_d.x}{facteur\ d'echelle}) \tag{4.4}$$

$$right_d = \frac{near}{2.focale_d}(W - \frac{2.obs_d.x}{facteur\ d'echelle}) \tag{4.5}$$

$$left_g = \frac{near}{2.focale_g}(W + \frac{2.obs_g.x}{facteur\ d'echelle}) \tag{4.6}$$

$$right_g = \frac{near}{2.focale_g}(W - \frac{2.obs_g.x}{facteur\ d'echelle}) \tag{4.7}$$

avec :
- $left_d$: le paramètre $left$ de la fonction de projection perspective pour la caméra de droite.
- $right_d$: le paramètre $right$ de la fonction de projection perspective pour la caméra de droite.
- $left_g$: le paramètre $left$ de la fonction de projection perspective pour la caméra de gauche.
- $right_g$: le paramètre $right$ de la fonction de projection perspective pour la caméra de gauche.
- $obs_g.x$, $obs_g.y$ et $obs_g.z$: position de l'œil gauche de l'observateur dans le référentiel de l'écran.
- $obs_d.x$, $obs_d.y$ et $obs_d.z$: position de l'œil droit de l'observateur dans le référentiel de l'écran.
- $focale_d$ et $focale_g$: distance focale pour la caméra de droite et pour celle de gauche.

En ce qui concerne les paramètres top et $bottom$ de la fonction de projection perspective (figure 8), on procède de la même façon que pour les paramètres $right$ et $left$. On en établit les relations suivantes :

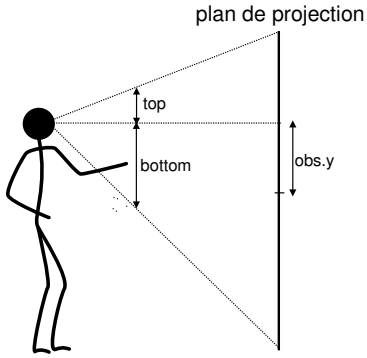


FIG. 8 – Système de deux caméras virtuelles, vue de côté.

$$top_d = \frac{near}{2.focale_d}\left(\frac{W}{ratio} - \frac{2.obs_d.y}{facteur\ d'echelle}\right) \tag{4.8}$$

$$bottom_d = \frac{near}{2.focale_d}\left(\frac{W}{ratio} + \frac{2.obs_d.y}{facteur\ d'echelle}\right) \tag{4.9}$$

où $ratio$ correspond au rapport largeur / hauteur de l'écran (et du plan focal). On en déduit les mêmes formules pour la caméra de gauche.

4.3 Correction des mouvements pseudoscopiques et position orthostéroscopique

Nous venons de voir un certain nombre d'analogies entre la scène réelle et la scène virtuelle. De ces analogies nous avons extrait des relations que nous allons pouvoir utiliser pour corriger les images. Nous proposons deux modes de résolution : soit on associe le référentiel de la scène virtuelle au référentiel de l'écran (figures 9), soit on l'associe au référentiel de l'observateur (figures 10).

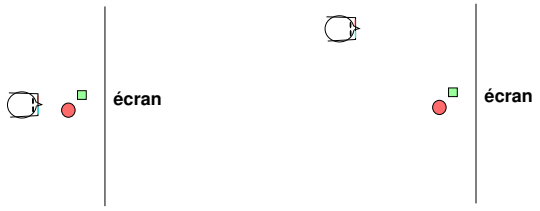


FIG. 9 – Scène liée à l'écran.

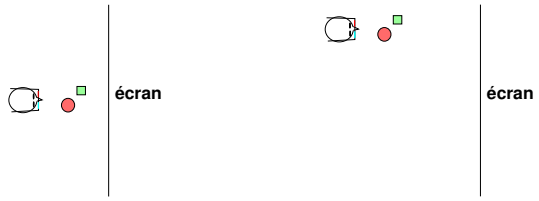


FIG. 10 – Scène liée à l'observateur.

4.3.1 Scène liée à l'écran

Avec ce modèle de restitution, l'observateur doit avoir la sensation que la scène virtuelle est fixe par rapport au référentiel de l'écran. Autrement dit, il peut voir cette scène selon différents points de vue. Cette méthode nécessite donc d'agir non seulement sur les paramètres de projection des caméras mais aussi sur leurs positions dans la scène virtuelle. En ce qui concerne les paramètres de projection, il suffit d'utiliser les équations du chapitre 4.2 qui nous assurent d'être à la position orthostéroscopique à chaque instant et évite ainsi tout mouvement pseudoscopique. Il reste ensuite à appliquer au système de caméras les mêmes déplacements que l'observateur, éventuellement à un facteur d'échelle près (équation 4.1).

4.3.2 Scène liée à l'observateur

Dans le second mode, l'observateur voit la même scène quelque soit sa position, il n'en a donc qu'un seul point de vue et il n'est alors pas nécessaire d'appliquer aux caméras virtuelles une quelconque modification de position. Il est suffisant de mettre à jour les paramètres de projection des caméras.

5 Implémentation

Une façon pratique d'utiliser un système de deux caméras est d'utiliser la fonction OpenGL `gluLookAt` [WNDS99] avec chacune des deux caméras. Cette fonction prend en paramètres les coordonnées de la caméra, les coordonnées du point visé (`focus`), et la direction du vecteur "vertical" dans le référentiel de la caméra (`up`). Il faut veiller à ce que les vecteurs `focus`, `up` et le vecteur reliant les deux caméras soient perpendiculaires deux à deux. Ceci nous assure que les axes optiques soient parallèles.

Quelle que soit le référentiel choisi, il faut commencer par initialiser certains paramètres comme la distance interoculaire (6,5 cm en moyenne) ou la taille de l'écran permettant de calculer le facteur d'échelle entre la scène virtuelle et la scène réelle (cf. équations 4.1). Il faut aussi déterminer certains paramètres de la matrice de projection.

```
double w_ecran   = W_ECRAN;                    // largeur de l'écran en cm
double w_virtuel = W_VIRTUEL;                  // largeur du plan focal souhaitée
double facteurEchelle = w_ecran / w_virtuel;
double ratio = w_ecran / hauteur_ecran;
double near = NEAR;
double far  = FAR;
```

5.1 Scène virtuelle lié à l'écran

Le chapitre 4.3.1 indique qu'il est nécessaire d'appliquer certaines transformations (translations et rotations) au système de caméras en plus du paramètrage de la fonction `frustum` gérant la projection sous OpenGL. Voici l'algorithme effectuant la correction des images dans le cas de la scène virtuelle liée à l'écran :

```
static void drawFunc(void){
...
```

```
// calculer le vecteur up
...

//camera droite
double focale = obsDroite.z / facteurEchelle;
double left   = near*(w_virtuel+2.0*obsDroite.x/facteurEchelle)/(2.0*focale);
double right  = near*(w_virtuel-2.0*obsDroite.x/facteurEchelle)/(2.0*focale);
double top    = near*(w_virtuel/ratio-2.0*obsDroite.y/facteurEchelle)/(2.0*focale);
double bottom = near*(w_virtuel/ratio+2.0*obsDroite.y/facteurEchelle)/(2.0*focale);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-left, right, -bottom, top, near, far);

// calculer le vecteurs focus
...

gluLookAt(cameraDroite.x + obsDroite.x / facteurEchelle,
          cameraDroite.y + obsDroite.y / facteurEchelle,
          cameraDroite.z + obsDroite.z / facteurEchelle,
          focus.x, focus.y, focus.z,
          up.x, up.y, up.z);

drawScene(); // affiche la scène OpenGL

//camera gauche
...
}
```

5.2 Scène virtuelle lié à l'observateur

Dans le cas où la scène virtuelle est liée à l'observateur, il suffit de remplacer la fonction `gluLookAt` du code précédent par :

```
gluLookAt(cameraDroite.x, cameraDroite.y, cameraDroite.z,
          focus.x, focus.y, focus.z,
          up.x, up.y, up.z);
```

5.3 Résultats

La figure 11 représente un observateur face à un écran de réalité virtuelle. La scène virtuelle représente une pièce dans laquelle se trouve un pilier surmonté d'une sphére. Les mouvements pseudoscopiques sont particulièrements présents dans ce genre de scènes où l'observateur voit les murs se déformer et se déplacer quand lui-même se déplace. La figure 12 illustre le la méthode consistant à associer le référentiel de la scène virtuelle avec le référentiel de l'observateur. La scène virtuelle suit les déplacements de l'obsevateur. Les figures 13 et 14 illustrent la méthode consistant à associer le référentiel de la scène virtuelle au référentiel de l'écran. Dans la figure 13, l'observateur s'est déplacer par rapport à l'écran et la paire d'image stéréoscopique a été modifiée en conséquence. Pour un observateur résté face à l'écran, la sphère ne prend plus une forme sphérique mais une forme elliptique. Ce n'est pas le cas pour l'observateur qui a une vision orthostéroscopique de l'image (figure 14).

6 Conclusion

Un observateur se déplaçant face à une paire d'images stéréoscopiques subit des mouvements pseudoscopiques perturbants. Nous avons présenté deux solutions pour corriger ces déformations et proposé pour chacune d'elles une implémentation simple en OpenGL. Ces deux méthodes garantissent une restitution de la scène virtuelle sans distorsion des objets. La première méthode donne à l'utilisateur la sensation de se déplacer face à une scène fixe

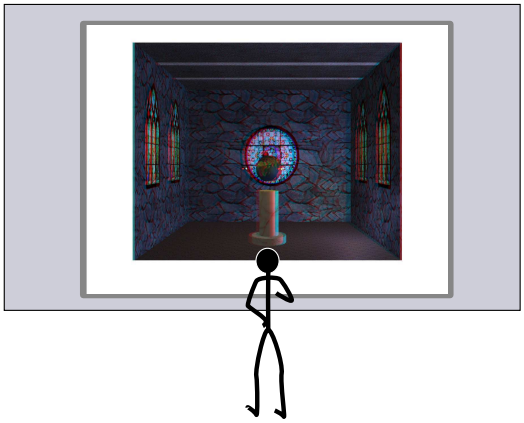


FIG. 11 – Observateur devant l'écran.

FIG. 12 – Scène liée à l'observateur.

FIG. 13 – Scène liée à l'écran.

FIG. 14 – Scène liée à l'écran vue par l'observateur.

par rapport à l'écran. Cette méthode lui permet donc de voir cette scène sous plusieurs points de vue. La seconde méthode associe le référentiel de la scène virtuelle à celui de l'observateur qui, par conséquent, voit exactement la même chose, quelle que soit sa position. Certaines applications deviennent alors accessibles aux enfants dont la petite taille constitue un handicap gênant pour une bonne perception d'une image en relief. En outre, cette méthode procure plus de confort à l'utilisateur qui peut se déplacer sans ressentir de gène.
Les méthodes proposées sont indépendantes du système de suivi de la tête de l'utilisateur ainsi que du mode de restitution des images stéréoscopiques (écran d'ordinateur, grand écran rétro-éclairé, ...). Elle sont simples a implémenter et fonctionnent sur n'importe quelle code source OpenGL. Elles présentent toutefois l'inconvénient d'être "mono-utilisateur" puisqu'une correction ne peut être effectuée que pour un observateur à une position déterminée.

Références

[BC93] Grigore Burdea and Philippe Coiffet. *La Réalité Virtuelle*. Hermes, 1993.

[CNSD93] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality : The design and implementation of the cave. *Proceedings of ACM SIGGRAPH'93*, pages 135–142, August 1993.

[Cor97] StereoGraphics Corporation. *Developer's Handbook : background on creating images for CrystalEyes and SimulEyes*. StereoGraphics Corporation, 1997.

[CPS+97] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, L. Dawe, and M. Brown. The immersadesk and infinitywall projection-based virtual reality displays. *Computer Graphics*, May 1997.

[DMLT92] N. A. Dodgson, J. R. Moore, S. R. Lang, and A. R. L. Travis. Time-multiplexed color autostereoscopic display. *IEE Colloquium on Stereoscopic Television*, 1992.

[FMP01] Philippe Fuchs, Guillaume Moreau, and Jean-Paul Papin. *Le traité de la réalité virtuelle*. Les Presses de l'Ecole des Mines, 2001. pages 330-333, 350-358.

[PAB+99] Dave Pape, Josephine Anstey, Mike Bogucki, Greg Dawe, Tom DeFanti, Andy Johnson, and Dan Sandin. The immersadesk3 - experiences with a flat panel display for virtual reality. *Proc. third Int'l Immersive Projection Technology Workshop, ACM Press*, pages 107–112, 1999.

[Ras00] Ramesh Raskar. Immersive planar display using roughly aligned projectors. *IEEE VR 2000*, pages 109–116, 2000.

[SA98] Mark Segal and Kurt Akeley. *The OpenGL Graphics System : A Specification (Version 1.2)*. Chris Frazier, 1998.

[WNDS99] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming guide : the official guide to learnig OpenGL, version 1.2, third edition*. OpenGL Architecture Review Board, 1999.