# Architecture Logicielle

Le prototype scientifique
du projet Toute La Lumière Sur L'Ombre
(TLLSLO)

Wai Kit Chan, LIMSI-CNRS
août 2010

V.01

# Table des matières

# I. Introduction

**Toute La Lumière Sur L'Ombre** (TLLSLO) est un projet Partenariat Institutions Citoyens pour la Recherche et l'Innovation (PICRI), associant scientifiques, artistes et grand public. L'utilisation de l'ombre comme espace de projection numérique pour l'augmentation du monde réel est le thème central du projet.
Les éléments du dispositif sont principalement un projecteur de lumière pour la création de l'ombre, une caméra pour la capture du monde physique, un ordinateur pour l'analyse d'image et la synthèse d'effets visuels, et un vidéoprojecteur pour la projection de l'augmentation numérique. À partir de ces éléments, un ensemble de configurations a déjà été réalisé au cours du projet par les partenaires.

Le document contient des explications sur le mécanique du logicielle de rendu graphique Virtual Choroegrapher (VirChor). Au sein du logicielle, l'ensemble des opérations sont organisés en différents modules. Ces modules regroupent un ensemble de passes. Une passe permet de calculer une image à partir des données préexistantes. Chaque passe est composée d'opérations. Ces différents niveaux d'organisation logicielle forment l'architecture logicielle.

# II. Dispositif scientifique

Dans le cadre du projet TLLSLO, deux applications fonctionnent indépendamment l'un de l'autre : le prototype scientifique et **ToLaLuSuLo**. La première a été mise en place à partir d'un moteur rendu 3D nommé **VirChor**, qui nécessite des connaissances approfondies en programmation graphique. **VirChor** est utilisé par les scientifiques pour expérimenter les algorithmes de traitement d'images et les effets spéciaux. Pour chaque fonctionnalité testée et approuvée au sein du prototype scientifique, elle est alors retranscrite dans **ToLaLuSuLo** selon sa priorité et sa faisabilité.
**ToLaLuSuLo** est une application basée sur **TapeMovie** qui est un environnement logiciel modulaire dédié à l'écriture et au contrôle intermédia (son, vidéo, 3D, lumière) en temps-réel. Il est utilisé par les artistes, vidéastes et régisseurs des compagnies de théâtre.

# III. Matériels

Afin de réaliser le prototype scientifique, il est nécessaire de procurer la liste des matériels ci-dessous, leur description et les étapes d'installation sont documentés dans « **Manuel d'utilisateur** » :

- Un projecteur qui éclaire le performeur avec une lumière infrarouge avec ou sans lumière visible. La lumière visible est indépendamment de chaine de traitement d'images.
- Une caméra Infrarouge qui capture l'ombre infrarouge et le corps éclairé (ou la silhouette selon la configuration).
- Un convertisseur qui convertit le flux vidéo analogique en numérique.
- Un ordinateur de traitement qui reçoit le flux vidéo numérique, réalise un ensemble d'opérations sur l'image capturée dans le but d'extraire les éléments intéressants (ombre, corps éclairé, silhouette, fond, …) puis de créer des effets artistiques.
- Un vidéoprojecteur qui diffuse l'image sortie de l'ordinateur dans l'environnement réel.
- Une Wiimote qui contrôle le dispositif à distance.
- Une clé Ilok qui permet l'utilisation du logiciel Max/MSP/Jitter.

Mise à part le projecteur, le reste des éléments sont connectés directement ou indirectement à l'ordinateur, l'élément central du dispositif (voir illustration 1).

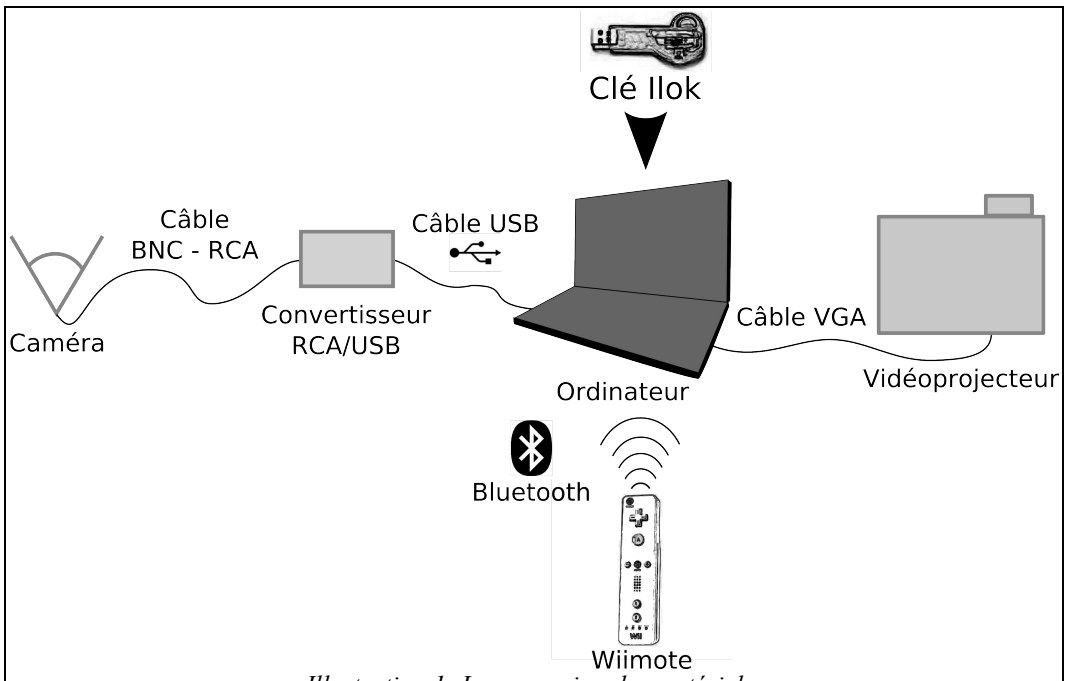


*Illustration 1: La connexion des matériels.*

# IV. Logiciels

Le prototype scientifique fonctionne avec un ensemble de logiciels ayant des fonctionnalités différentes :

- **Virtual Choreographer** (**VirChor**) est un moteur de rendu 3D en temps réel. La représentation des données géométriques et des interactions se fait en XML. Il permet une programmation haut niveau des scènes en **OpenGL**, en programmation graphique **GPU** et en communication réseau avec des scripts de commande écrits en **XML**. Cet outil a été développé par Christian Jacquemin et ses collaborateurs au **LIMSI-CNRS**, et est disponible en Open Source sur SoureForge (http://sourceforge.net/projects/virchor/).

- **OpenGL** (**Open G**raphics **L**ibrary) est une spécification qui définit une API multi-plateforme pour la conception d'applications générant des images 3D (mais également 2D). Elle utilise en interne les représentations de la géométrie projective pour éviter toute situation faisant intervenir des infinis.

- **C for Graphics** (CG) est un langage de programmation de haut niveau pour Shader développe par **NVIDIA**. Comme son nom l'indique, sa syntaxe est très similaire au langage de programmation C. De manière simple, les Shader sont des petits programmes exécutés par chaque vertex, chaque primitive et chaque pixel : le Vertex Shader (ou le Vertex Program) est exécuté sur chaque vertex par l'API graphique, le Geometry Shader (ou le Geometry Program) est exécuté sur chaque primitive, et enfin le Pixel Shader (ou le Fragment Program) est exécuté sur chaque pixel du rendu.

- **Java** est une technologie développée par Sun Microsystems : (la technologie Java™). Elle correspond à plusieurs produits et spécifications de logiciels qui, ensemble, constituent un système pour développer et déployer des applications. Java est utilisée dans une grande variété de plates-formes depuis les systèmes embarques et les téléphones mobiles jusqu'aux serveurs, les applications d'entreprise, les superordinateurs et dans une moindre mesure pour les interfaces graphiques comme les applets Java du Web.

- **Max/MSP/Jitter :**
  - **Max/MSP** est un logiciel musical permettant de faire de la synthèse sonore, de l'analyse, de l'enregistrement, ainsi que du contrôle d'instrument MIDI.
  - **Jitter** est une bibliothèque supplémentaire de fonctions ajoutée à Max (comme l'est MSP) et permettant de travailler sur des matrices. Son champ d'application est, par conséquent multiple : traitement d'image en temps réel (le plus fréquemment relié à la pratique du Vjing (http://fr.wikipedia.org/wiki/Vjing), mais également synthèse audio matricielle (spectrographies FFT), matriçage mathématique ou encore modélisation matricielle 3D en OpenGL (http://fr.wikipedia.org/wiki/OpenGL).

## *A) Outils de contrôle*

Les périphériques de contrôle sont principalement une **Wiimote**, une **souris** et un **clavier**. L'outil de rendu **VirChor** reçoit des événements directement de la souris ou du clavier, et des messages OSC via UDP (des messages réseaux encodés selon le protocole **Open Sound Control**). Les messages reçus par **VirChor** sont envoyés par une application **Max/MSP/Jitter**. Cette application est constituée d'une interface graphique de contrôle où **VirChor** ne dispose pas. L'application **Max/MSP/Jitter** reçoit des événements souris et des messages **UDP – OSC** d'une application **Java**. L'application **Java** reçoit directement les événements de la **Wiimote** via **Bluetooth** et renvoie des messages encodés en OSC via UDP (voir illustration 2).

La mise en marche des périphériques qui contrôlent l'installation implique la connexion de la Wiimote à l'ordinateur, et l'envoi de messages de la Wiimote au moteur de rendu VirChor via une application JAVA et un patch Max/MSP/Jitter. Pour avoir plus de détails sur la mise en marche, veuillez consulter le document « **Manuel d'utilisateur** ».

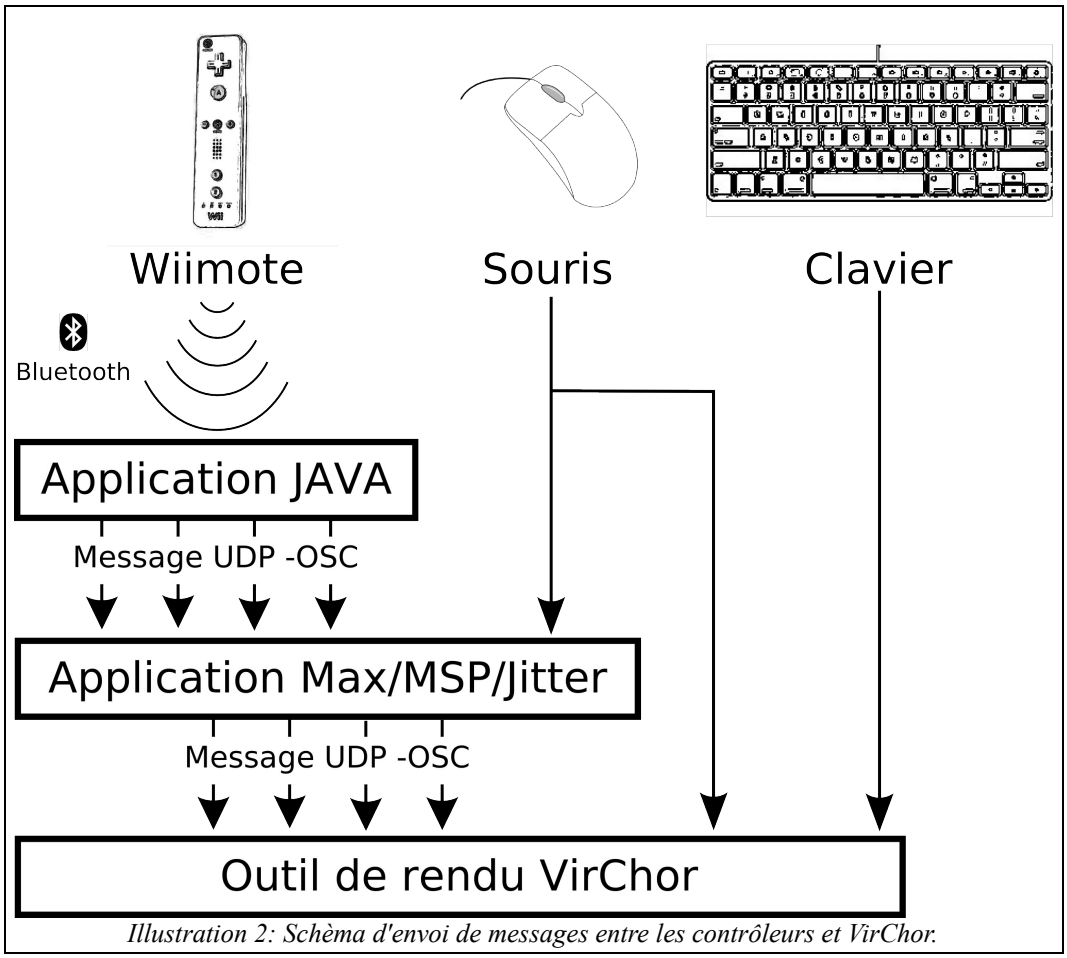


*Illustration 2: Schèma d'envoi de messages entre les contrôleurs et VirChor.*

## *B) Moteur de rendu VirChor*

L'application principale du dispositif est le moteur de rendu graphique. Elle est réalisée à partir de VirChor. Elle a pour but de créer des effets spéciaux pour être projetés dans l'environnement réel.

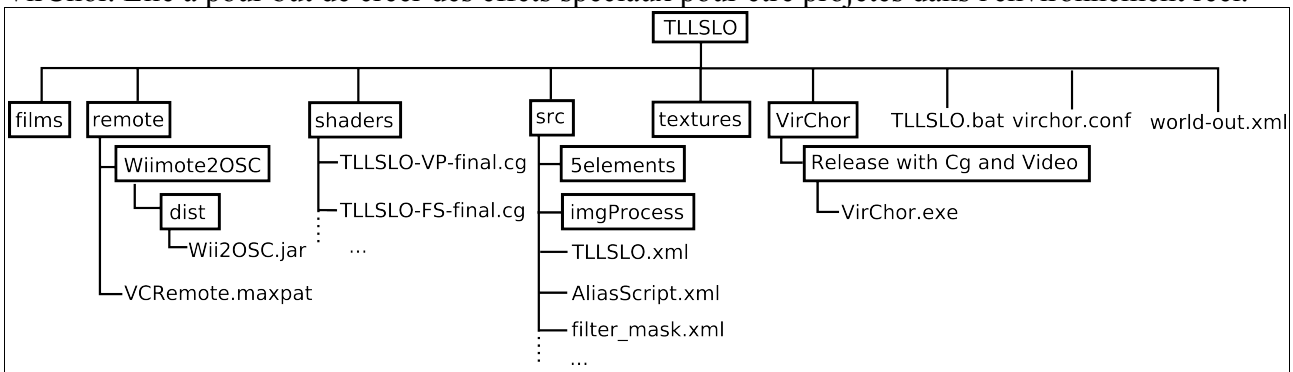


*Illustration 3: L'arborescence logicielle.*

Le répertoire « **TLLSLO** » est souvent accompagné par la version classée selon la date (ex : « TLLSLO_10_01_15 » pour la version du 15 janvier 2010).
L'organisation de l'application au niveau des répertoires :

- Le dossier « **films** » contient les vidéos utilisés lorsque la caméra live est inactivée ainsi que des vidéos pour les effets spéciaux.
- Le dossier « **remote** » contient l'exécutable de l'application Java « **Wii2OSC.jar** ». Elle est utile pour recevoir les actions de la Wiimote, et du patch de contrôle Max/MSP/Jitter.
- Le dossier « **shaders** » contient tous les fichiers shaders (programmes cartes graphique) de l'application.
- Le dossier « **src** » contient tous les fichiers XML de l'application.
- Le dossier « **textures** » contient toutes les images utiles à l'application.
- Le dossier « **VirChor** » contient l'exécutable VirChor et les librairies (du type *.dll) nécessaires.

L'exécutable **« TLLSLO.bat »** contient l'instruction suivante :
« **.\VirChor\Release with Cg and Video\VirChor.exe" virchor.conf .\src\TLLSLO.xml** »

Il fait appel aux différentes parties de l'application.
La première est l'ensemble du code de **VirChor** (sous forme d'un exécutable de type *.exe) sur lequel le moteur de rendu s'appuie. Pour télécharger VirChor, l'acquisition se fait via le site http://virchor.wiki.sourceforge.net/.
La seconde est un fichier de configuration nommé « **virchor.conf** », et permet de configurer l'application et de définir par exemple :

- la position et la taille de la fenêtre.
- la fréquence de rafraichissement (nombre d'images par seconde) de l'application.
- le(s) serveur(s) local(aux) et le(s) client(s) à distance.
- la déclaration et initialisation des constantes.

La troisième partie comprend le fichier « **TLLSLO.xml** ». C'est le fichier principal qui relie tous les implémentations de la scène VirChor, soit dans le fichier lui-même, soit en faisant appel à d'autres fichiers XML inclus lors de la compilation. De manière générale, il définit principalement un utilisateur, son mode de projection (orthographique ou en perspective), une camera qui lui est associée. Dans le cadre de ce projet, il fait appel aux autres fichiers XML pour le traitement d'images.

# 1. Traitement d'image en GPU

Les algorithmes itératifs de restauration d'images (élimination du bruit et du flou introduits par les systèmes d'acquisition d'image ou vidéo) et de reconstruction d'images (notamment dans le domaine médicale et en astronomie) offrent une très bonne qualité d'image au prix d'un temps de calcul bien souvent trop important (plusieurs heures voire jours de calculs) pour être employés en temps réel.

L'implémentation de ces algorithmes sur les processeurs graphiques (GPU) de la nouvelle génération permet d'accélérer de manière significative ces algorithmes.

L'application **VirChor** a pour finalité de réaliser à la fois des fonctionnalités d'analyse d'images, mais aussi de synthèse d'images. Pour le rendu de la géométrie dans VirChor, il est possible de faire appel à des Shaders à travers le langage Cg de NVIDIA.

Un Shader (anglais, du verbe « to shade » signifiant ombrager, estomper ou nuancer) est un programme utilisé dans le domaine de la synthèse d'images pour paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel. Dans le cadre du projet TLLSLO, tout les traitements d'images, tant au niveau de l'analyse qu'au niveau de la synthèse, sont réalisés par des Shaders.

L'implémentation des Shaders dans VirChor est composée de deux parties :

- Le mécanisme de Shader dans VirChor
- Les fichiers au format Cg chargés par VirChor

Les cours de programmation graphique de christian jacquemin :
http://perso.limsi.fr/jacquemi/RGA-TD/index.html

Une petite présentation assez complète :
http://www.artiflo.net/2009/10/presentation-d%E2%80%99un-gpu/2/

## 2. Principe de la programmation GPU (Graphic Processing Unit) multi-passe

Une technique fréquemment employée par les programmeurs de GPU consiste à créer une texture à partir de l'image affichée. Cela permet de faire du rendu multi-passe. Le principe est le suivant : nous affichons notre géométrie dans une texture. Cette texture est ensuite plaquée sur un quadrilatère qui fait la même taille que le volume de vue (viewport), c'est-à-dire ce qu'on voit à l'écran afin de faire des traitements ultérieurs. Cela offre de nombreux avantages : utilisation et combinaison de textures plutôt que de la géométrie, opérations directement sur les pixels et inspection des voisinages, etc.

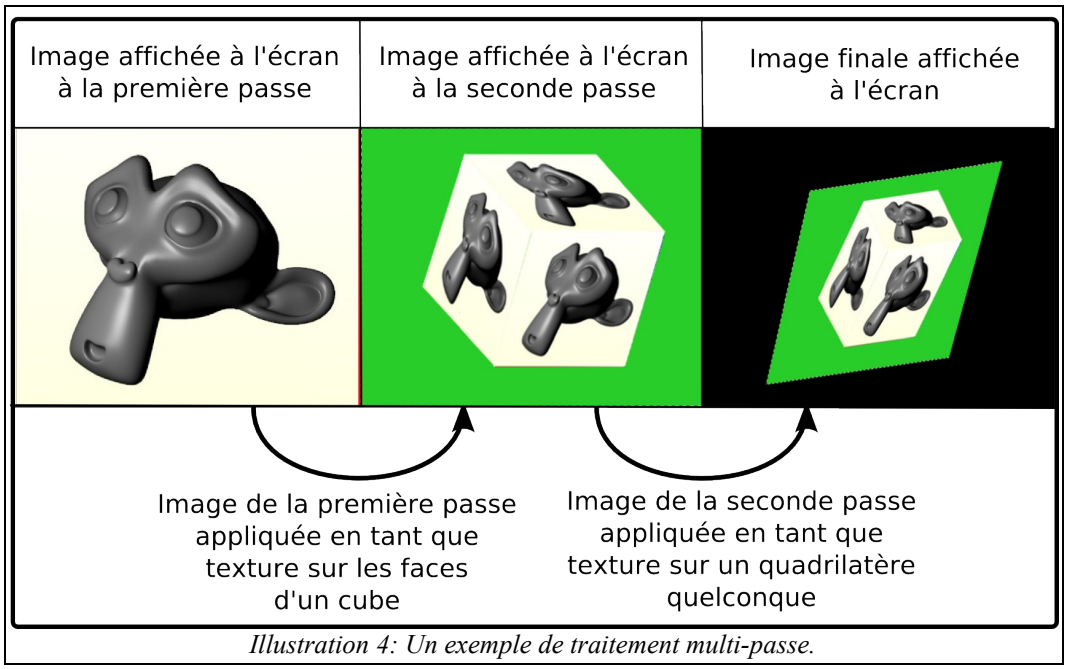


*Illustration 4: Un exemple de traitement multi-passe.*

Une image complexe est donc rendue en plusieurs passes successives, chacune modifiant ou complétant le résultat des précédentes (voir illustration 4). Ainsi les opérations de mélange permettent de tracer une première couche de surface, puis d'autres qui affectent la teinte (vernis) ou la luminosité (ombres fines), puis ajoutent des reflets, etc.

## 3. Achitecture logicielle globale

Les étapes de traitement d'images au sein du moteur 3D VirChor sont représentées sous forme d'un graphe. Au début de la chaîne de traitements, la caméra fournit une image capturée de l'environnement réel. Cette images brute (originale) peut être tout simplement vidéoprojetée sans traitement. Néanmoins, l'intérêt du projet et du dispositif est d'utiliser cette image comme base pour créer des effets artistiques et de les diffuser. L'image brute subit un ensemble d'opérations comme le débruitage dans le but d'améliorer la qualité de l'image, ou la calibration dans le but compenser la déformation due à la position de la caméra. Cela donne une image propre et ajustée (niveau caméra). L'image à la sortie du « **module de calibrage et de traitement d'images** » permet au « **module d'analyse et de détection d'ombre** » de produire une image masque (par exemple, tous les pixels appartenant à une ombre sont en blanc, et le reste en noir). À partir de l'image bicolore, le « **module de Tracking** » calcule la boîte englobante de l'ombre ainsi que son centre. À partir de cette même image, le « **module du moteur et de rendu des particules** » calcule comment les particules doivent interagir avec l'ombre et comment les particules sont affichées. Grâce au système de multi-pass (à tout moment, il est possible d'utiliser une image après une opération par une ou plusieurs autres opérations). Le « **module de création d'effets artistiques** » permet de créer un ensemble d'effets spéciaux selon les inspirations artistiques. Enfin le « **module de compositing** » permet de rassembler l'ensemble des images utiles à la calibration et images remplis d'effets dans le but de décider l'image à diffuser. Cette image sera envoyée au vidéoprojecteur (voir illustration 5).

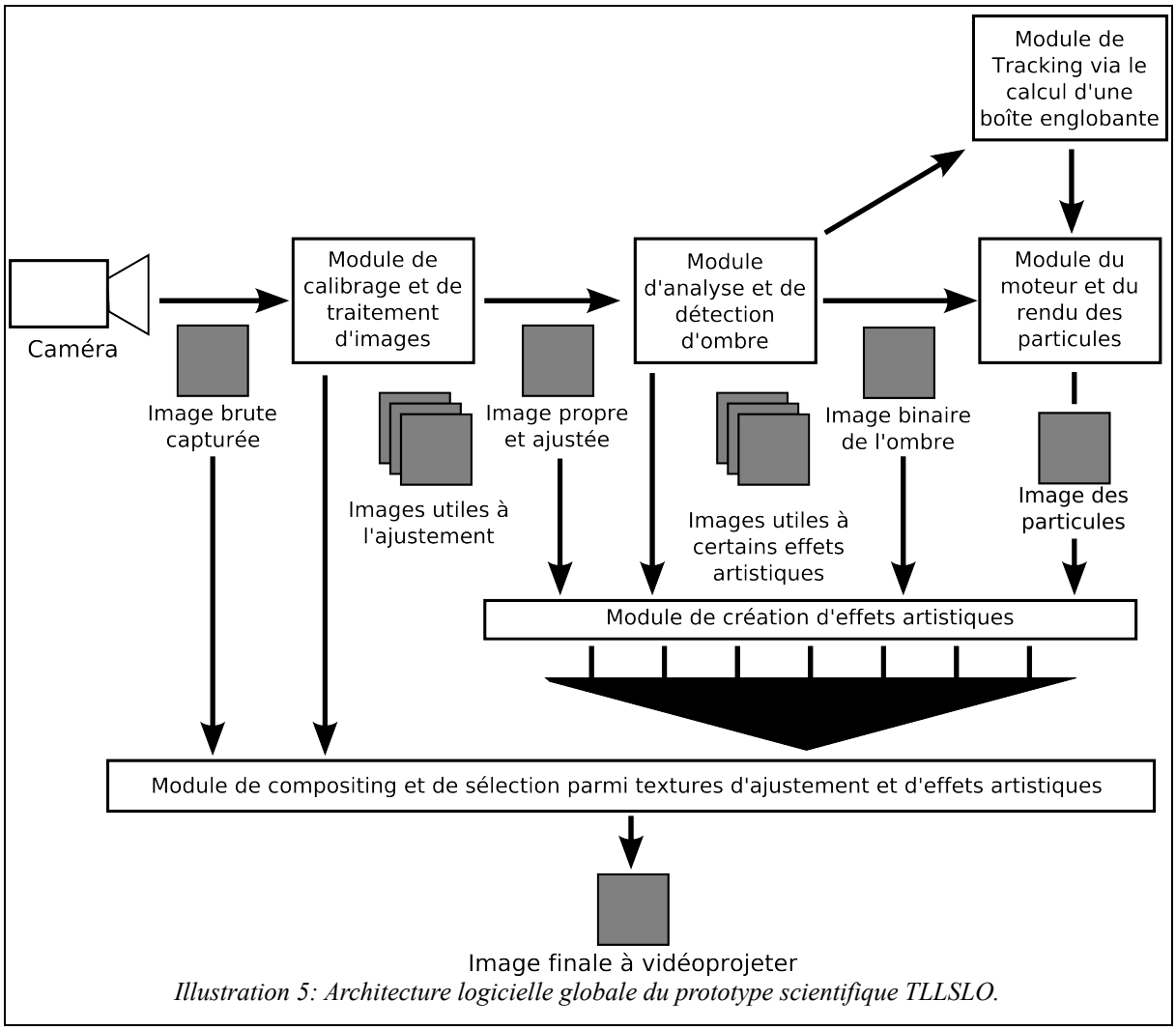


*Illustration 5: Architecture logicielle globale du prototype scientifique TLLSLO.*

# 4. Module de traitement d'images et de calibrage

Le module de traitement d'images et de calibrage est composé des six filtres, donc six passes d'opérations qui traitent une ou plusieurs images pour générer une image résultat. La passe « deinsterlaced » permet de régler d'anomalie d'entrelacement vidéo lors de l'acquisition. La passe « radialecorrection » permet de corriger le focal grand angle utilisé. Les passes « rotationcorrection » et « keystone » permettent de réaliser le calibrage entre la surface de projection et la caméra. Enfin les passes « median » et « ROADBI » permet de supprimer les bruits impulsionnel vidéo sans trop perdre d'informations de l'image capturée. A la sortie de ses opérations, l'image capturée à l'entrée est plus propre et ajustée pour les traitements postérieurs (voir illustration 6).

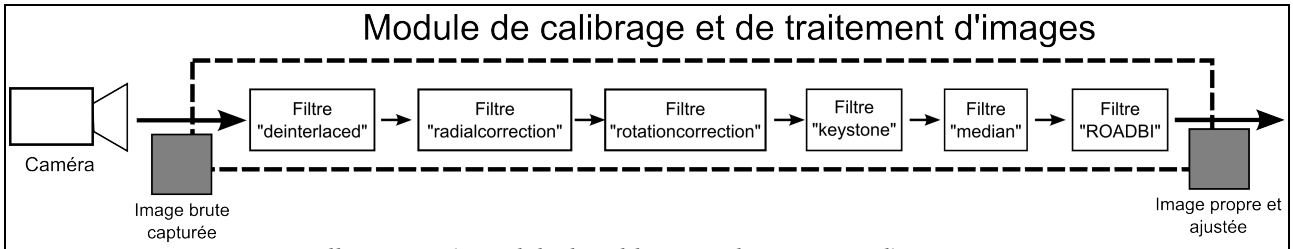


*Illustration 6: Module de calibrage et de traitement d'images.*

## *a) Désentrelacement*

**Nom texture** : « FrameBuffer_deinterlaced_filter ».

**Fonction** : désentrelacement de l'image.

**Nom fichier** : « filter_deinterlaced.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-deinterlaced.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_bypass » (0 pour activer l'effet et 1 pour désactiver).

**Texture (s) d'entrée (s)** : image entrelacée et capturée par la caméra grâce à au tag « vidéo ».

**Détail** :

Lors de l'acquisition de l'image à partir de la caméra, l'image est entrelacée, à savoir qu'elle ne capte pas 30 images par seconde. Car en réalité, la caméra ne capte que 60 demi-images par seconde. C'est en combinant deux demi-images consécutives, encore appelées trames (field), qu'on obtient une seule image (frame). La combinaison de deux trames est appelée l'entrelacement.

L'apparition de matériels affichant des images entières (on parle d'affichage en mode progressif) tels que les écrans d'ordinateur nécessite d'effectuer la conversion de séquences vidéo du mode entrelacé au mode progressif. Cette action est le désentrelacement :

1. Méthode la plus simple : les lignes de chaque demi-image sont doublées pour obtenir une image complète.
2. Méthode du filtre spatial (adaptée aux images en mouvement) : on calcule les lignes manquantes de chaque trame à partir des lignes voisines de la même trame
3. Méthode du filtre spatio-temporel (adaptée aux images fixes) : les lignes manquantes de chaque trame sont calculées à partir des lignes voisines de la même trame et des lignes voisines ou identiques des trames voisines.
4. D'autres méthodes existent encore et les études ne sont pas terminées afin d'améliorer les résultats. Étant donné la mobilité de notre diapositif, nous avons opté pour le filtre spatial.
   Le filtre de désentrelacement fut l'un des premiers filtres réalisés, car il joue un rôle crucial dans les différents domaines de mise en forme du signal vidéo (filtrage, réduction de bruit,

détection de contours,…).

Dans le cadre du projet TLLSLO, le choix se porte sur la seconde méthode. La première permet d'obtenir une qualité moindre. La troisième est optimisée que pour les flux vidéo à faible mobilité. Et la dernière est trop couteuse en ressources pour la réaliser.

Voici l'extrait du fichier « TLLSLO-FS-deinterlaced.cg »

**La correction entre film pré-enregistré et flux vidéo en direct à une symétrie vertical près**

```
if (cg_fs_videoLive!=1.0)
{
	decalCoords.y=1-decalCoords.y;
}
```

Le paramètre « cg_fs_videoLive » passé depuis VirChor, déterminé par la variable d'initialisation « Video » du fichier de configuration « virchor.conf ». si « Video » est à « false » (cela signifie que le moteur est en mode flux vidéo pré-enregistré) alors « cg_fs_videoLive » est à 0 alors il y a l'application d'une symétrie verticale. À savoir que la taille de texture est de 1, donc il suffit d'effectuer « 1-nb ligne » pour faire la symétrie verticale.

```
float2 coordTmp;
coordTmp = decalCoords.xy*cg_fs_2fv_size;
```

Redimensionner l'image de traitement : « decalCoords.xy » est la position en x et y du pixel dans l'image de taille 1, pour obtenir la position en unité pixel, il faut multiplier ce dernier par la taille de la texture, donc dans ce cas là du « 640*480 ».

```
int tmp = mod(int(coordTmp.y) ,2.);
```

Le calcul de desentrelacement : la variable « tmp » reçoit une valeur de 0 ou de 1 pour déterminer si la ligne « coordTmp.y » de la texture est paire ou impaire. En fonction de cette valeur, deux types de traitement sont effectués:

```
if (tmp !=0)
{
	return float4(0.5 * texRECT(MaVideo, float2( coordTmp.x , (coordTmp.y)-1 )).bgra
	+ 0.5 * texRECT(MaVideo, float2( coordTmp.x , (coordTmp.y)+1 )).bgra);
```

Si c'est impaire, l'intensité lumineuse de la ligne est calculée en réalisant la moyen entre l'instensité de la ligne du dessus et de dessous.

```
}else{
	return texRECT(MaVideo,coordTmp).bgra;
}
```

Si c'est paire, alors on renvoie directement l'intensité lumineuse de la ligne sans modifications.

### *b) Correction radiale*

**Nom texture** : « FrameBuffer_radialCorrection_filter ».

**Fonction** : correction de la déformation radialement de l'image due à la déformation grand angle(voir illustration 7).

**Nom fichier** : « filter_radialCorrection ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-radialcorrection.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_bypass » (1 pour activer l'effet et 0 pour désactiver) et « cg_fs_2fv_coef » contient 2 valeurs; la première est le coefficient pour l'axe X et la seconde l'axe Y.

**Texture (s) d'entrée (s)** : « FrameBuffer_deinterlaced_filter ».

**Détail** :

*Illustration 7: La correction Radiale.*

l'extrait du fichier « TLLSLO-FS-radialcorrection.cg ».

```
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
```

Redimensionne la taille de la texture de 1 à la taille prédéfinie.

```
...
float2 CoordsBis = CoordsTmp-cg_fs_2fv_size/2.0;
float x= CoordsBis.x/(cg_fs_2fv_size.x/2.0);
float y= CoordsBis.y/(cg_fs_2fv_size.y/2.0);
```

Par défaut l'orgine du repère est en bas à gauche avec une intervalle de 0 à la taille, le changement de repère permet de mettre l'instervalle de -taille sur 2 et traille sur 2. De ce faite, l'orgine de l'image n'est plus en bas à gauche de l'image pas au centre de l'image.

```
//passage coordonnée polaire atan2( y ,x );
float r;
r= sqrt(x*x+y*y);
float rho;
rho = atan2(y,x);//2.0 * atan((y/r)/(1+ (x/r)));
rho = mod(rho, 2.0* pi);
```

Cet extrait permet de transformer l'écriture cartiésienne (coordonnée en x et y) du pixel en coordonnée polaire.

```
float distorX = cg_fs_2fv_coef.x - 1.5;
float f2rX = ((cg_fs_2fv_coef.x * distorX) /(cg_fs_2fv_coef.x-r))- distorX;
float distorY = cg_fs_2fv_coef.y - 1.5;
float f2rY = ((cg_fs_2fv_coef.y * distorY ) /(cg_fs_2fv_coef.y-r))-distorY ;
```

A partir du rayon « r » du coordonnée polaire, il y a l'application d'une fonction linéaire pour réaliser la déformation radiale selon les coefficients « cg_fs_2fv_coef.x » (déformation horizontale) et « cg_fs_2fv_coef.y » (déformation verticale) de manière indépendante.

```
float2 CoordX = float2(f2rX*cos(rho),f2rX*sin(rho));
float2 CoordY = float2(f2rY*cos(rho),f2rY*sin(rho));
```

Opération inverse permet de transformer les coordonnées polaires en cartésiennes.

```
CoordX*=cg_fs_2fv_size/2.0;
CoordY*=cg_fs_2fv_size/2.0;
if ((CoordX.x<-(cg_fs_2fv_size.x/2.))||(CoordX.x>(cg_fs_2fv_size.x/2.))||
(CoordY.y<-(cg_fs_2fv_size.y/2.))||(CoordY.y>(cg_fs_2fv_size.y/2.)))
{
```

```
        return float4(0.0, 0.0, 0.0, 0.0);
}else{
        return texRECT(MaVideo,float2    (CoordX.x+cg_fs_2fv_size.x/2.,CoordY.y+cg_fs_2fv_size.y/2.)).rgba;
}
```

Le changement de repère s'effectue de manière inverse et envoie le résultat avec un traitement de bord (colorier en noir tout zone qui ne correspond à l'extérieur de l'image de départ (avant la déformation de l'image par l'opération radiale).

### *c) Correction de la rotation et des symétries horizontale et verticale*

**Nom texture** : « FrameBuffer_rotationCorrection_filter ».

**Fonction** : corriger la déformation radialement de l'image due à la déformation grand angle.

**Nom fichier** : « filter_radialCorrection ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-rotationcorrection.cg » et « TLLSLO-FS-rotationcorrection .cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_bypass » (1 pour activer l'effet et 0 pour désactiver), « cg_fs_angle » pour l'angle de la rotation, « cg_fs_flipH » (1 pour appliquer la symétrie horizontale) et « cg_fs_flipV » (1 pour appliquer la symétrie verticale).

**Texture (s) d'entrée (s)** : « FrameBuffer_radialCorrection_filter ».

**Détail** :

De manière générale, ces opérations (rotation et symétries) sont des transformations de l'image, leur utilisation peut tantôt être pour le calibrage, tantôt pour un effet artistique, ou encore un élément scénaristique.

L'extrait du fichier « TLLSLO-FS-rotationcorrection.cg ».

```
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
...
if (cg_fs_flipH == 1.0){
        decalCoords.x = 1 - decalCoords.x;
}
```

Le paramètre « cg_fs_flipH » permet d'activer ou desactiver la symétrie horizontale.

```
if (cg_fs_flipV == 1.0){
        decalCoords.y = 1 - decalCoords.y;
}
```

Le paramètre « cg_fs_flipV » permet d'activer ou desactiver la symétrie verticale.

```
float angle = cg_fs_angle;
```

Le paramètre « cg_fs_angle » est l'angle de rotation en unité radian.

```
float2x2 mat = {
{cos(angle),sin(angle)},
{-sin(angle),cos(angle)}};
```

« mat » est la matrix de rotation en fonction de l'angle.

$$\begin{bmatrix} \cos(angle) & \sin(angle) \\ -\sin(angle) & \cos(angle) \end{bmatrix}$$

```
decalCoords.xy -= 0.5 ;
```

Déplacer l'origine du repère au centre de l'image.

```
decalCoords.xy = mul(mat, decalCoords.xy);
```
Appliquer la transformation de la  rotation.

```
decalCoords.xy += 0.5 ;
```
Retrouver l'origine de départ par déplacement linéaire.

```
float2 CoordsTmp = decalCoords.xy;
if ((CoordsTmp.x<0)||(CoordsTmp.x>1.)||(CoordsTmp.y<0)||(CoordsTmp.y>1.))
{
```
Les conditions de gestion de bord de l'image qui permettent de mettre en noir l'extérieur de l'image tournée.

```
        return float4 (0,0,0,0);
}else{
        return texRECT(MaVideo,CoordsTmp * cg_fs_2fv_size).rgba ;
```
Renvoie de la valeur (l'intensité lumineuse) du pixel à la sortie du programme graphique.
```
}
```

### *d) keystone*

**Nom texture** : « FrameBuffer_keystone_filter ».

**Fonction** : compensation de la déformation de l'image (calibration) en fonction des 4 points.

**Nom fichier** : « filter_keystone.xml »

**Nom (s) fichier(s) shader(s)** : « TLLSLO-VP-final.cg » et « TLLSLO-FS-keystone.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_bypass » : paramètre à valeur 1 pour ne pas utiliser le traitement et 0 pour l'utiliser.
- « cg_fs_2fv_coeffHG », « cg_fs_2fv_coeffHD », « cg_fs_2fv_coeffBG » et « cg_fs_2fv_coeffBD » sont 4 paires de valeurs pour la position en x et y des 4 coins de la texture (HG pour haut gauche, HD pour hautdroite, BG pour bas gauche et BD pour bas droite). Ces valeurs varient entre -4 à 4.
- « cg_fs_4fv_outColor » est un tableau de 4 variables pour la couleurs du bord de l'image (RGBA).

**Texture (s) d'entrée (s)** : « FrameBuffer_radialCorrection_filter ».

**Détail :**

**Le principe de calibrage**

Le calibrage est l'action de régler un appareil par rapport à des données de références afin d'obtenir un comportement fidèle lors de l'enregistrement ou la reproduction. La déformation des quadrilatères (l'image capturée et l'image diffusée) se font par le déplacement des coordonnées de textures selon une fonction bijective de la déformation. En général, si le vidéoprojecteur est perpendiculaire à l'axe de projection, l'image n'est pas déformée. Mais dans le cas contraire, il faut déformer l'image projetée pour compenser cette déformation (voir illustration 8).

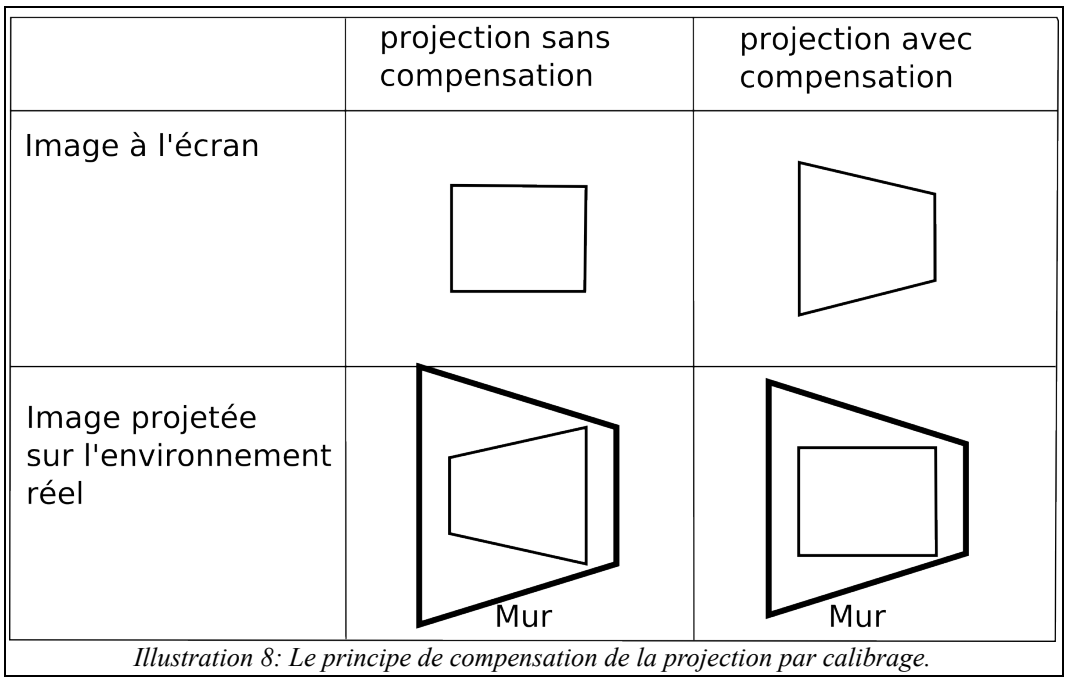


*Illustration 8: Le principe de compensation de la projection par calibrage.*

Dans le cadre du projet TLLSLO, la calibration a pour but de superposer fidèlement l'ombre numérique à l'ombre naturelle ou à l'ombre infrarouge. L'indice de superposition est considéré comme un critère de qualité. Après avoir installé les matériels et les logiciels, avoir mis en marche les logiciels de contrôle (Java, Max/MSP/Jitter) et le moteur de rendu graphique VirChor, il est temps de mettre relation ces différents éléments pour que l'installation soit fonctionnelle.

### *e) Médian spatial*

**Nom texture** : « FrameBuffer_median_filter ».

**Fonction** : suppression du bruit vidéo.

**Nom fichier** : « filter_median.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-median.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_onOff » paramètre pour activer ou désactiver

**Texture (s) d'entrée (s)** : « FrameBuffer_keystone_filter ».

**Détail :**

La *médiane* est une mesure statistique qui représente une alternative robuste à la moyenne. Considérons $n$ valeurs numériques $x_1, ..., x_n$ (pas nécessairement distinctes), où $n$ est *impair*. On les ordonne de la plus petite à la plus grande, ce qui donne la suite permutée $x_{i1}, ..., x_{in}$, où $\{i_1, ..., i_n\}$ est une permutation de {1, ..., n}. La médiane est alors la valeur placée au milieu de cette suite ordonnée, à savoir $x_{im}$ pour $m = (n+1)/2$.
Par exemple, soit $n = 9$, et considérons les 9 valeurs 17, 1, 3, 5, 3, 1, 12, 2, 7. Notons que certaines valeurs peuvent être répétées, et qu'il ne faut pas supprimer les répétitions ! En les ordonnant de la plus petite à la plus grande, on obtient la suite 1, 1, 2, 3, 3, 5, 7, 12, 17. La valeur au milieu de cette suite ordonnée est la 5ème, à savoir 3, qui est donc la médiane des 9 valeurs.
Le filtre médian permet, de manière efficace supprimer les bruits impulsif tout en conservant les fortes variation.
La définition est tirée de : http://arthur.u-strasbg.fr/~ronse/TIDOC/FILTER/median.html

L'extrait du fichier « TLLSLO-FS-median.cg ».

```
static const int dim = 3;
```

```
static const int d1 = dim*dim;
static const int d2 = dim*dim-1;
static const float d3 = (dim*dim)/2;
static float data[dim*dim];
static const float2 dir[9] = { {0,0},{1,1}, {1,-1}, {-1,-1}, {-1,1} ,{2,2}, {2,-2}, {-2,-2}, {-2,2}};
```
Les constantes et les variables prévu pour l'algorithime de tri.

```
void bubblesort()
{
	float minor, major;
	for( int i=0; i<d1; ++i) {
		for( int j=0; j<d2; ++j) {
			minor = min( data[j], data[j+1] );
			major = max( data[j], data[j+1]);
			data[j] = minor;
			data[j+1] = major;
		}
	}
}
```
La fonction de tri « bubblesort ».

La suite est dans la fonction principale « main ».
```
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
```
Redimensionner la texture unitéraire à l'unité pixel (640*480).

```
if (cg_fs_onOff == 1.0)
{
```
La condition d'activation de traitement.

```
	return texRECT(MaVideo,CoordsTmp).rgba;;
}else{
	//median
	//trier les 9 valeurs
	float grayMedian;
	float4 colorMedian;
	for( int i=0; i<d1; ++i) {
		colorMedian = texRECT( decal,CoordsTmp.xy+dir[i] );
		grayMedian = colorMedian.r*0.3+colorMedian.g*0.59+colorMedian.b*0.11;
		data[i] = grayMedian;
	}
```
Remplissage du tableau de tri.

```
	bubblesort();
```
Application du tri.

```
	return float4( float3(data[d3]), 1.0);
```
Renvoie de la valeur d'intensité résultante qui est la valeur central du tableau trié.
```
}
```

### *f) Filtrage Bilatéral*

**Nom texture** : « FrameBuffer_ROADBI_filter ».

**Fonction** : diminution du bruit impulsif.

**Nom fichier** : « filter_ROADBI.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP.cg » et « TLLSLO-FS-ROADBI.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_2fv_para » et « cg_fs_onOff ».(1 pour activer l'effet et 0 pour désactiver).

**Texture (s) d'entrée (s)** : «  FrameBuffer_median_filter »

**Détail :**

Le filtrage bilatéral est un filtrage non-linéaire que ne détruit pas les discontinuités. Si on applique un filtrage gaussienne simple, on aura une moyenne basée sur les aspects spatiales de l'image, plus les pixels voisins seront proche du pixel traité, plus le poids sera élevé. On ajout, donc, un poids par rapport à l'intensité des pixels considérés. Deux pixels peuvent, donc, être près dans l'espace ou peuvent avoir des valeurs similaires (près dans l'ensemble image).
http://www-igm.univ-mlv.fr/labinfo/rapportsInternes/2008/04.pdf
La méthode est implémenté selon l'article [1] Garnett Roman et al.

## 5. Module d'analyse d'image et de détection d'ombre

Une fois que l'image capturée par la caméra est débruité, elle va être analysé pour en extraire l'ombre et le corps éclairé (à savoir création des masques) et plus précisément décider un ensemble de pixel appartenant ou pas à l'ombre ou au corps éclairé (voir illustration 9).

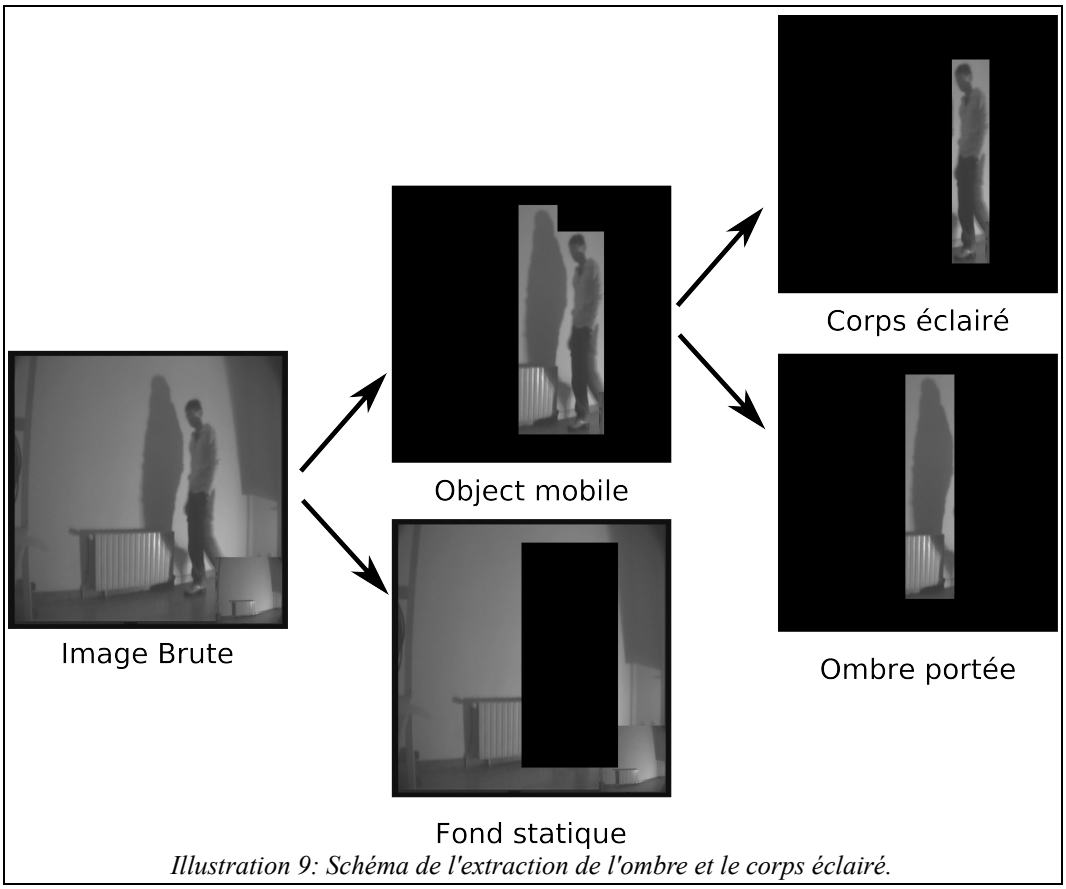


*Illustration 9: Schéma de l'extraction de l'ombre et le corps éclairé.*

La première étape est de distinguer si un pixel appartient à l'objet mobile ou pas. La solution est l'utilisation d'une méthode de traitement d'images nommée « la soustraction de fond ». Elle prend une image de référence qu'on appelle « le fond » ; c'est l'image statique qui ne contient pas d'objet mobile. Pour réaliser l'opération, il faut soustraire la valeur de l'intensité lumineuse du pixel de l'image courante avec la valeur du même pixel (position spatiale) du fond. En théorie, si un pixel de l'image courante appartient au fond, alors la différence est nulle ou proche de zéros, par contre si un pixel appartient à l'objet mobile alors la différence est significative. Enfin un seuillage permet de prendre une décision binaire. L'opération de soustraction de fond est réalisée dans le filtre « **SsFondLOG** ». La soustraction s'effectue dans le domaine logarithmique pour rehausser les valeurs d'intensité lumineuse intermédiaire.
La méthode de soustraction de fond est assez performante si l'illumination de l'environnement ne varie pas. Car si l'intensité lumineuse globale de l'environnement varie, alors les valeurs des pixels supposés immobiles changent de valeurs par rapport à l'image « fond » donc tous les pixels de l'image sont alors considérés comme mobile.
Donc il faut un fond qui s'adapte au changement lumineuse globale de l'environnement. Cette méthode s'appelle le « fond adaptatif ».

La seconde étape, pour distinguer si un pixel appartient à une ombre ou corps éclairé. Il y a deux critères de distinctions. Le premier est que le corps éclairé de l'image courante peut avoir une

intensité lumineuse supérieure à l'image « fond » alors que l'ombre propre de l'image courante aura toujours une intensité lumineuse inférieure à l'image « fond ».
Le deuxième critère est, avec l'hypothèse que le fond est homogène (le mur ou le sol), que l'ombre projetée sur une surface de projection homogène restera homogène alors que le corps éclairé sera beaucoup plus contrasté (plie de vêtements etc.). Donc la distinction entre l'ombre et le corps éclairé se calcul à partir d'une valeur d'homogénéité. La possibilité de distinguer l'ombre propre liée au corps éclairé et l'ombre portée reste peut robuste.

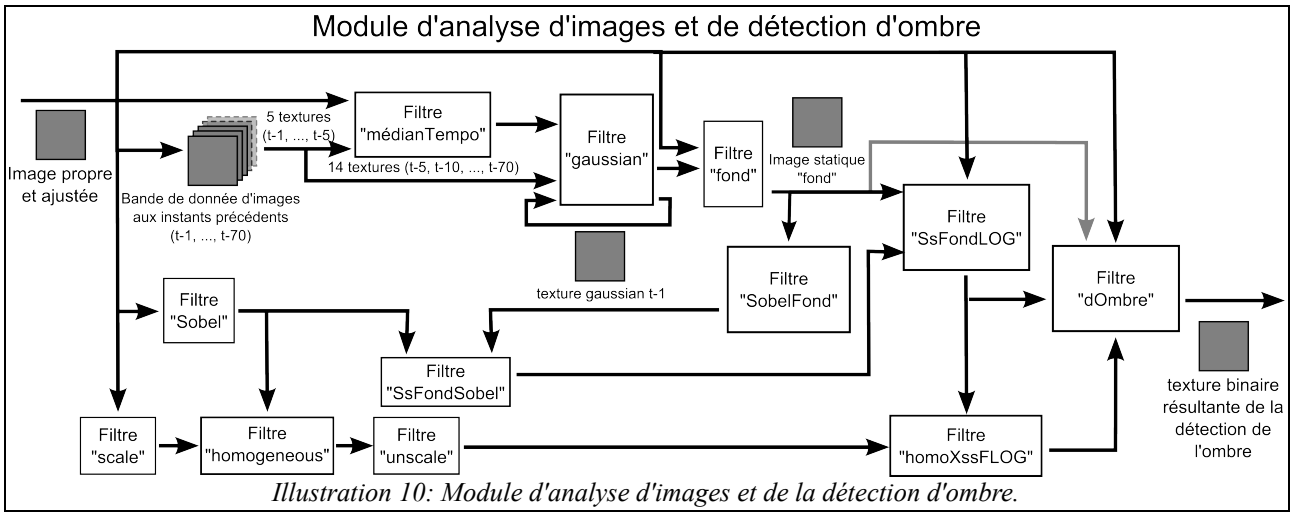

*Illustration 10: Module d'analyse d'images et de la détection d'ombre.*

Le module d'analyse d'images et de détection d'ombre prend à son entrée « image propre et ajustée » résultante de la passe « **ROADBI** » expliqué dans la partie précédente (« **module de traitement d'images et de calibrage** »). Cette image est la matière première pour tout ce module dans le but de distinguer l'ombre, le corps en mouvement et le fond statique. Cette image permet avant tout de constituer une bande de donnée d'images temporelle, Elle contient 70 textures, l'image courante « **FrameBuffer_ROADBI_filter** », puis l'image à l'instant d'avant (t-1) « **imgM01** », ainsi de suite jusqu'à l'instant t-70 « **imgM70** ». l'ensemble des transferts de textures sont dans le fichier « **imgSwitchx5.xml** », qui permet de passer t-1 à t-2, t-2 à t-3, …, t-69 à t-70. La passe « **médianTempo** » permet de créer une image qui supprimer les grandes variations temporelles. La passe « **gaussian** » permet de calculer la valeur d'intensité lumineuse du fond adaptatif selon une fonction gaussienne. La passe « fond » remanie les valeurs la texture « **gaussian** » (qui ne sont pas simplement du rouge, vert, bleu et alpha, mais la valeur d'intensité lumineuse, et 3 coefficients de la fonction gaussienne). À partir de l'image propre et ajustée résultante de « **ROADBI** », et l'image du fond, la passe « **ssFondLoG** » réalise la soustraction de fond dans le domaine logarithmique. Mais pour améliorer soustraction, Un travail de contour a été réalisé. Grâce à l'algorithme de détection de contour Sobel (sur une image un pixel est blanc, si il a une valeur d'intensité très différent de ses voisins). La passe « **sobel** » calcule le contour de l'image courante ( c'est à dire l'image propre et ajustée). La passe « **sobelFond** » calcule le contour de l'image fond. La passe « **ssFondSobel** » permet de faire le soustraction de fond dans le domaine de la contraste. Elle permet de renforcer la qualité d'information de l'image à la sortie de « **ssFondLOG** ».
A cet étape d'analyse d'images, le système est en mesure de distinguer les objets en mouvement (ombre, corps en mouvement ou la silhouette) et le fond immobile. Enfin, le calcule du critère d'homogénéité permet de faire une distinction faible entre l'ombre et le corps en mouvement. À partir de l'image propre et ajustée,La passe « **scale** » permet de diminuer la taille de la texture, car lors de la réduction d'une taille d'une texture, il entraine automatiquement la suppression de la contraste. Justement dans notre cas, cette perte d'information est profitable pour le calcul de l'homogénéité de la passe suivante « **homogeneous** »Enfin après cette passe, la texture retrouve sa taille originale grâce à la passe « **unscale** ». pour l'instant le critère d'homogénéité est appliqué sur

l'ensemble de tout l'image. Avec la passe « **homoXssFLOG** » qui multiplie l'image à la sortie de « **unscale** » (blanc pour les zones homogènes, et noir dans les zones contrastées) et l'image à la sortie de « **ssFondLOG** » (blanc pour les zones en mouvement et noir les zones statiques). Cette multiplication permet d'obtenir une image où les zones blanches correspondent à l'ombre (en mouvment et homogène).
L'ensemble des résultats est envoyé à la passe finale « **dOmbre** » du module pour prendre des décisions, et d'autres calculs selon d'autre critère de sélection (voir illustration 10).

### *a) Médian Temporel*

**Nom texture** : « FrameBuffer_medianTempo_filter ».

**Fonction** : diminution des bruits impulsifs et suppression des mouvements rapides des objets mobiles.

**Nom fichier** : « filter_medianTempo.xml »

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP.cg » et « TLLSLO-FS-medianTempo.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_onOff ».

**Texture (s) d'entrée (s)** : 5 images consécutives (images courante « FrameBuffer_ROADBI_filter » , image t-1 « imgM01 », t-2 « imgM02 », t-3 « imgM03 », t-4 « imgM04 »).

**Détail :**

Le filtre médian temporel est une filtre non-linéaire, et similaire au filtre médian spatial. Pour une pixel donné, il récupère sa valeur à des instants différents ( instant courant t, instant avant t-1, puis t-2, jusqu'à t-4). Ainsi, la valeur de sortie de chaque pixel est calculé à partir de 5 valeurs. Le filtre trie les 5 valeurs par ordre croissantes dans un tableau de valeur. La valeur de sortie du filtre la valeur médian du tableau (ex s'il y a 5 valeurs, alors c'est la 3°, si la taille du tableau est de 9, alors c'est la 5° valeurs...)
Ce filtre permet de supprimer les pixels avec des valeurs instables dans le temps (ex : à l'entrée, nous avons 56,80,58,204,76, la valeur médian est 76).
L'image obtenue ne remplace pas l'image courante (propre et ajustée), elle donne une impression saccadé du résultat.
Il permet juste à l'utilisation du fond adaptatif, car il supprime une partie des mouvements rapids et les bruits vidéos.

### *b) Gaussian*

**Nom texture** : « gaussian1 ».

**Fonction** : calcul du fond adaptatif.

**Nom fichier** : « filter_gaussian1x5.xml »

**Nom (s) fichier(s) shader(s)** : « TLLSLO-VP-final.cg » et « TLLSLO-FS-gaussian1.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_update » : 1 pour activer le test de mise à jour (après il y a 2 autres tests de mise à jour, donc même si ce premier test est activé que le fond va forcément mise à jour)du fond et 0 pour désactiver (que le fond est sure de ne pas être mise à jour)
- « cg_fs_capture » : permet de réinitialiser les valeur du fond, lors ce paramètre est supèrieur à 0,75, le système réinitialise les valeurs du fond avec l'image « medianTempo », sinon il utilise la texture « gaussian1M1 » (les valeurs calculées à l'instant précédente) et le réactualise avec l'image « medianTempo » si nécessaire.

**Texture (s) d'entrée (s) :**

- « FrameBuffer_medianTempo_filter » : la texture médian temporel pour la mise à jour.
- « gaussian1M1 » : la texture qui contient les données de la gaussien à l'instant t-1.
-13 textures des instants précédents tous les 5 images (image t-5, t-10 … t-70) qui seront utilisées pour le 2° test de mise à jour.

**Détail :**

Ce filtre « gaussian » est principalement constitué de deux parties; La première partie permet de tester si la valeur des pixels de l'image courant de mettre à jour le fond ou pas. La deuxième partie de l'implémentation permet de mettre à jour la texture (image) de fond.
En général, une texture visuel est constituée de quatre canaux (rouge, vert, bleu, et alpha pour la transparent). Mais pour réaliser ce filtre gaussien, deux valeurs est utilisé : la moyenne et la variance.
Pour un pixel donné, La première valeur est la moyenne dans le temps (temporelle). Elle est la valeur d'intensité lumineuse probable du fond. La seconde valeur est la variance. Plus cette valeur est grande, plus la valeur de l'image courante a de probabilité d'être mise à jour; c'est une valeur de seuillage.

**Les trois tests de mise à jour de la gaussienne**

a) Le paramètre « cg_fs_update » est le premier des tests de mise à jour, si il est à 0 alors, le fond ne fera pas de mise à jour. Par contre si il est à 1 alors le deuxième test aura lieu
b) Le deuxième test de mise à jour a pour principe de considérer la stabilité de l'image selon un ensemble échantillon temporel. Pour un pixel donné, la mise à jour de la valeur du fond se réalise que s'il n'y a pas trop de changement de valeur d'intensité lumineuse (différence puis un seuillage pour pouvoir prendre une décision) entre l'image courant et un ensemble d'image antérieur échantillonnée dans le temps (tout les 5 images).
c) Le dernière test de mise à jour (si les deux test avant sont positifs) utilise la valeur variance définir par la méthode de la gaussienne.

Si ces trois tests ont été positifs, alors il y aura la mise à jour de la gaussienne définie par la valeur moyenne et la variance.

**La mise à jour de la gaussienne**

La mise à jour de la moyenne est une combinaison entre la ancienne valeur et la valeur de l'image courante :
*nouvelle moyenne = (1-coef) * ancienne moyenne de la gaussienne + coef * intensité du pixel de l'image courant*
De même que la mise à jour de la variance est une combinaison entre la ancienne valeur et la valeur de l'image courante :
*nouvelle variance = (1-coef) * (ancienne variance de la gaussienne)² + coef * (intensité du pixel de l'image courant - nouvelle moyenne)²*
La méthode du fond adaptatif est tirée de l'article [2].
La méthode pour la réalisation du deuxième test de mise à jour est tirée de l'article [3] de A. Leone et al.

### *c) Fond*

**Nom texture** : « FrameBuffer_fond_filter ».

**Fonction** : génération de l'image fond.

**Nom fichier** : « filter_fond.xml »

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP.cg » et « TLLSLO-FS-fond.cg ».

**Paramètre (s) d'entrée (s)** : cg_fs_capture; la valeur par défaut est 1, elle diminue et tente vers 0, et tant que cette valeur est supérieure à 0.75, l'image courante est considérer comme le fond (permet d'initialiser le système au lancement). Il suffit de remettre la variable à 1 pour refaire le fond manuellement.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_ROADBI_filter » : l'image courante pour la mise à jour manuelle.
- « gaussian1 » la texture qui contient la moyenne et la variance de la gaussienne.

**Détail :**

**L'extrait du fichier «** TLLSLO-FS-fond.cg »

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
if (cg_fs_capture>0.75){
        return texRECT( MaVideo,texCoord ).rgba;
```

« MaVideo » est l'image courante « FrameBuffer_ROADBI_filter » pour la réinitialisation du système.

```
}else{
        float tmp;
        tmp = saturate(texRECT( gauss1,texCoord ).r);
        float4 result = float4(tmp,tmp,tmp,1.0);
        return result;
```

Le programme renvoie la valeur de la moyenne de la gaussienne « gauss1 » (dans le canal rouge), sur les 3 canal de couleur à la sortie du programme.

```
}
```

### *d) Détection de contour Sobel*

**Nom texture** : « FrameBuffer_sobel_filter ».

**Fonction** : calcul du contour (gradiente) de l'image courante

**Nom fichier** : « filter_sobel.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-sobel.cg ».

**Paramètre (s) d'entrée (s) :** aucun.

**Texture (s) d'entrée (s) :** « FrameBuffer_ROADBI_filter » l'image courante.

## Détail :

L'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords.

**L'extrait du fichier «** TLLSLO-FS-sobel.cg »

```
//sobel
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
// frame buffer reading
float4 sobelH =
        -       texRECT(decal,float2( CoordsTmp.x - 1 , CoordsTmp.y - 1 ))
        - 2 *   texRECT(decal,float2( CoordsTmp.x - 1 , CoordsTmp.y     ))
        -       texRECT(decal,float2( CoordsTmp.x - 1 , CoordsTmp.y + 1 ))
        +       texRECT(decal,float2( CoordsTmp.x + 1 , CoordsTmp.y - 1 ))
        + 2 *   texRECT(decal,float2( CoordsTmp.x + 1 , CoordsTmp.y     ))
        +       texRECT(decal,float2( CoordsTmp.x + 1 , CoordsTmp.y + 1 ));
float4 sobelV =
        -       texRECT(decal,float2( CoordsTmp.x - 1 , CoordsTmp.y - 1 ))
```

```
- 2 *  texRECT(decal,float2( CoordsTmp.x     , CoordsTmp.y - 1 ))
-      texRECT(decal,float2( CoordsTmp.x + 1 , CoordsTmp.y - 1 ))
+      texRECT(decal,float2( CoordsTmp.x - 1 , CoordsTmp.y + 1 ))
+ 2 *  texRECT(decal,float2( CoordsTmp.x     , CoordsTmp.y + 1 ))
+      texRECT(decal,float2( CoordsTmp.x + 1 , CoordsTmp.y + 1 ));
```

Le calcul de la valeur « sobelH » correspond à l'opération matricielle « Gx ». Et le calcul de la valeur « sobelV » correspond à l'opération matricielle « Gy ».

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{et} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

*Illustration 11: Les opérations matricielles des gradients horizontales et verticales.*

```
float4 sobel;
sobel = sqrt(sobelV*sobelV + sobelH * sobelH);
```

Le faite de calculer la racine au carrée pour le vecteur de direction constitué de la paire (Gx,Gy), L'inversion des signes de calcule lors des opérations matricielles n'as pas d'importance.

```
float graysobel = saturate(0.33 * sobel.r + 0.33 * sobel.g + 0.33 * sobel.b);
```

L'image supposée en couleur est convertie en dégradé de gris dans le but de rentre le programme shader plus générique).

```
graysobel= saturate (graysobel*2.0);
```

Réhaussement de la valeur de contour de manière arbitaire.

```
return float4(graysobel,graysobel,graysobel,1.0);
```

Renvoie de la valeur de contour.

### *e) Détection de contour Sobel de l'image fond*

**Nom texture** : « FrameBuffer_sobelFond_filter ».

**Fonction** : calcul du contour (gradiente) de l'image fond.

**Nom fichier** : « filter_sobelFond.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-sobel.cg ».

**Paramètre (s) d'entrée (s)** : aucun.

**Texture (s) d'entrée (s)** : « FrameBuffer_fond_filter » : l'image fond.

## Détail :

Le Programme shader est identique à celui de « Détection de contour Sobel », seul diffère l'image à l'entrée du programme qui est l'image de fond.

### *f) Soustraction de fond en Détection de contour « ssFondPlusSobel »*

**Nom texture** : « FrameBuffer_ssFondSobel_filter ».

**Fonction** : la soustraction entre 2 textures : « FrameBuffer_sobel_filter » et « FrameBuffer_sobelFond_filter ».

**Nom fichier** : « filter_ssFondSobel.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-ssFondSobel.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_start » permet à la réintialisation.

**Texture (s) d'entrée (s)** : image gradient courante (FrameBuffer_sobel_filter) et l'image gradient fond (FrameBuffer_sobelFond_filter)

**Détail :**

Ce filtre renforce le système du la soustraction de fond, et rend le résultat meilleur.

L'extrait du fichier « TLLSLO-FS-ssFondSobel.cg »

```
static const	float2 dir[5] = { {0,0},{1,1}, {1,-1}, {-1,-1}, {-1,1} };
static const	float weight[5] = { 3, 0.5, 0.5, 0.5,0.5};
```

Le tableaux « dir » définit les pixels locaux utilisés pour l'opération.
Le tableaux « weight » définit le poids associé au pixels voisins locaux.

La suite est dans le programme principale « main ».

```
float4 color ;
float4 textureIn;
float4 wallpaper;
float3 diff ;
if (cg_fs_start>0.8)
{
	return float4(0.5,0.5,0.5,1.0);
```

Le renvoi d'une couleur neutre pour la réinitialisation.

```
}else{
	float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
	for( int d = 0 ; d < 5 ; d++ )
	{
		textureIn  += texRECT( imageSob , texCoord.xy + dir[d] ).rgba * weight[d];
		// image ref en capture dynamique de taille de l'écran
		wallpaper  += texRECT( fondSob , texCoord.xy + dir[d] ).rgba * weight[d];

	}
```

Les opérations consistent à calculer la somme des 5 valeurs ponctuées par leur poids respectif à la fois pour l'image Sobel courante « imageSob » et l'image Sobel fond « fondSob ».

```
	textureIn.rgb/=5.0;
	textureIn.r = textureIn.r *0.3+ textureIn.g *0.59+ textureIn.b*0.11;
	wallpaper.rgb/=5.0;
	wallpaper.r = wallpaper.r *0.3+ wallpaper.g *0.59+ wallpaper.b *0.11;
```

La division permet de normer le poids associé à chaque pixel.
La seconde opération permet à l'image supposée en couleur d'être convertie en dégradé de gris dans le but de rentre le programme shader plus générique).

```
	float diff = abs(textureIn.r – wallpaper.r);
```

L'opération de la soustraction de fond consiste de calculer la valeur absolue de la différence entre un pixel de l'image sobel courante et l'image sobel fond à la même position.

```
	return float4(diff,diff,diff,1.0);
}
```

### *g) Soustraction de fond dans le domaine logarithmique « SsFondLOG »*

**Nom texture** : « FrameBuffer_ssFondLOG_filter ».

**Fonction** : application du méthode de détection de mouvement par soustraction de fond (voir illustration 12).

**Nom fichier** : »filter_ssFondLOG.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-ssFondLOG.cg ».

**Paramètre (s) d'entrée (s)** : aucun.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_ROADBI_filter » : l'image courante.
- « FrameBuffer_fond_filter » : l'image fond.
- « FrameBuffer_ssFondPlusSobel_filter » : l'image de soustraction de fond de gradient.

**Détail :**

Le filtre permet de réaliser la soustraction de fond. Mais au lieu d'avoir les valeurs d'intensité lumineuse de manière affine (x = y), il y a conversion des valeurs de l'intensité lumineuse dans le domaine logarithmique. Cette transposition de domaine permet de relever les valeurs intermédiaires (entre 0 et 255). donc la différence entre les 2 valeurs d'un pixel (image courante et l'image fond) est plus flagrant.

La méthode de soustraction de fond dans le domaine logarithmique est tirée de l'article [3] A. Leone et al.

L'extrait du fichier « TLLSLO-FS-ssFondSobel.cg »

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
float3 img;
float3 fond;
float3 ssFondPlusSobel;
img = texRECT(MaVideo, texCoord ).rgb;
fond = texRECT(f, texCoord ).rgb;
ssFondPlusSobel = texRECT( ssFondPlusSob, texCoord ).rgb;
```

La récupération de la valeur du pixel des 3textures.

```
//convertir en log
img.r = img.r * 0.3 +img.g * 0.59 +img.b * 0.11;
img.r = log(img.r * 255.0);
fond.r = fond.r * 0.3 +fond.g * 0.59 +fond.b * 0.11;
fond.r = log(fond.r * 255.0);
```

La première opération consiste à convertir l'image couleur (RGB) en l'intensité lumineuse (une seule valeur).
La seconde opération est de mettre les valeurs dans le domaine logarithmique.

```
float diff = abs(img.r – fond.r);
```

Le calcul de la différence est la valeur absolue de la différence entre les textures.

```
ssFondPlusSobel.r = ssFondPlusSobel.r * 0.3 +ssFondPlusSobel.g * 0.59 +ssFondPlusSobel.b * 0.11;
```

L'image de renforcement de contour est à priori en couleur (pour la généricité), donc il faut aussi le convertir en une seule valeur d'intensité lumineuse.

```
//soustraction de fond consiste à la fois de :
//difference entre 2 images, mais aussi différence entre les contours des 2 images
return float4 (float3(diff+ssFondPlusSobel.r),1.0);
```

Le renvoi de la valeur de soustraction du fond : plus l'image courante est différentes de l'image fond, plus l'intensité résultantes est blanche.

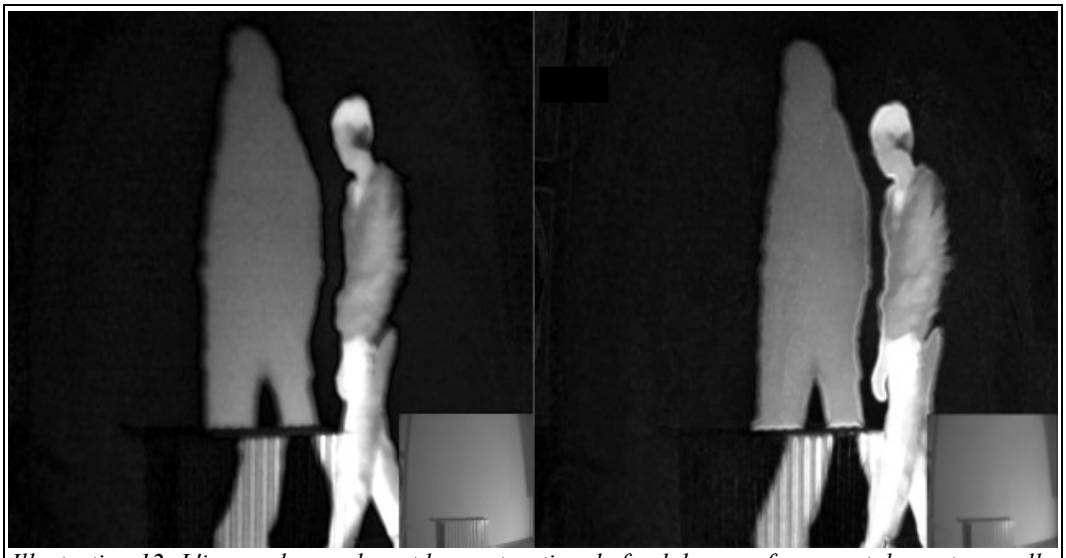

*Illustration 12: L'image de gauche est le soustraction de fond dans renforcement de contour, celle de droite avec.*

### *h) Rétrécissement*

**Nom texture** : « FrameBuffer_scale_filter ».

**Fonction** : rétrécir l'image à l'entrée.

**Nom fichier** : « filter_homogeneous.xml ».

**Nom (s) fichier(s) shader(s) :** « Fullscreen-VP.cg » et « TLLSLO-FS-scale.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_scale » permet de régler la taille de la texture lors de la rétrécissement.

**Texture (s) d'entrée (s)** : « FrameBuffer_ssFondLOG_filter » : l'image résultante de la soustraction de fond

**Détail :**

L'extrait du fichier « TLLSLO-FS-scale.cg »

```
static const float dim = 5.0;
static const float d1 = dim * dim;
static const float2 dir [25] = {
{0,0},{-1,-1},{-1,0},{-1,1},{0,-1},{0,1},{1,-1},{1,0},{1,1},
{-2,-2},{-2,-1},{-2,0},{-2,1},{-2,2},
{-1,-2},{-1,2},{0,-2},{0,2},{1,-2},{1,2},
{2,-2},{2,-1},{2,0},{2,1},{2,2},
};
```

Le tableau « dir » désigne les pixels voisinages au pixel local qui seront utilisés pour le calcul.

L'implémentation dans le programme principale « main ».

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size*cg_fs_scale;
```

Redimensionne la texture en fonction du coéfficient « cg_fs_scale ».

```
if ((texCoord.x > cg_fs_2fv_size.x)||(texCoord.y > cg_fs_2fv_size.y))
        return float4(0.0,0.0,0.0,1.0);
```

Renvoie noir si le pixel local n'appartient pas à la texture rétrécie.

```
float3 color;
for (int i =0 ; i < d1 ; i ++ )
{
        color += texRECT(MaVideo, texCoord + dir[i]).rgb;
}
color/=d1;
```
« color » est la moyenne 25 valeurs.

```
return float4(color.rgb, 1.0);
```
Renvoie la valeur résultante.

### *i) Homogénéité*

**Nom texture** : « FrameBuffer_homogeneous_filter ».

**Fonction** : calcul le critère d'homogénéité pour distinguer l'ombre et le corps éclairé. image distinguant les zones homogènes ou contrastées(voir illustration 13).

**Nom fichier** : « filter_homogeneous.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-homogeneous.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_scale » : permet de régler la taille de la texture lors des opérations de calcule d'homogénéité.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_ssFondLOG_filter » : l'image de soustraction de fond.
- « FrameBuffer_sobel_filter » : l'image courante gradient.

**Détail :**

Pour distinguer l'ombre et la silhouette, l'hypothèse principale est que l'ombre et une zone plus homogène que la silhouette. Pour calculer le critère d'homogénéité, c'est à la fois la gradient spatial et la variance spatial de l'image courante.
Le calculer pour la variance spatiale, qui utilise beaucoup de ressource, peut être optimiser si la calcule est réalisée sur une image de taille inférieur.

L'extrait du fichier « TLLSLO-FS-homogeneous.cg »
```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
```
Redimensionne la texture à l'unité pixel.

```
float2 texCoordBis = decalCoords.xy*cg_fs_2fv_size * cg_fs_scale;
```
Redimensionne la texture en fonction du coéfficient « cg_fs_scale ».

```
float3 img;
float sobel;
float3 diff ;
if ((texCoordBis.x > cg_fs_2fv_size.x)||(texCoordBis.y > cg_fs_2fv_size.y))
        return float4(0.0,0.0,0.0,1.0);
```
Renvoie noir si le pixel local n'appartient pas à la texture rétrécie.

```
float3 color [d1];
float intensite [d1];
float mean = 0.0;
float EType = 0.0;

//replissage des différents donnée
for( int i = 0 ; i < dim ; i++ ) {
```

```
		for( int j = 0 ; j < dim ; j++ ) {
			color[i*dim+j] = texRECT( MaVideo, float2 (texCoord.x+j-d2, texCoord.y+i-d2)).rgb;
			intensite[i*dim+j] = color[i*dim+j].r*0.3+color[i*dim+j].g*59+color[i*dim+j].b*0.11;
```
Remplissages des tableaux « color » et « intensite ».
```
			mean += intensite[i*dim+j];
```
Le calcul de la somme pour la moyenne « mean ».

```
		}
}

mean/= d1;
```
La division pour la moyenne.

```
//variance
for (int i = 0 ; i < d1 ; i++)
{
		EType += (intensite[i] - mean)*(intensite[i] - mean);
}
```
L'opération de l'écarte type ou de la variance calcule la différence d'intensité lumineuse spatiale et locale du pixel. Plus cette valeur est petite, plus c'est homogène.

```
//sobel
sobel = texRECT(sob,texCoordBis).r * 0.3 + texRECT(sob,texCoordBis).g * 0.59+ texRECT(sob,texCoordBis).b *
0.11;
```
Récupération de la valeur contour. Plus la valeur du variable du contour est petite, plus c'est homogène.

```
//homogeneous
float h;
sobel *= 10.0;
EType *= 10.0;

float moyenne = saturate(sobel * 0.5 + 0.5* EType);
h = 1.0 - moyenne;
```
La valeur d'homogénéité est le complémentaire de la moyenne entre la valeur de l'écart-type et du coutour Sobel.
```
return float4(h,h,h,1.0);
```

*Illustration 13: image de gauche est l'homogénéité, image du milieu est la soustraction de fond, image de droite est le produit pixel à pixel.*

### *j) Aggrandissement*

**Nom texture** : « FrameBuffer_homogeneous_filter ».

**Fonction** : agrandir l'image à l'entrée.

**Nom fichier** : « filter_homogeneous.xml ».

**Nom (s) fichier(s) shader(s) :** « Fullscreen-VP.cg » et « TLLSLO-FS-unscale.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_scale » permet de régler la taille de la texture lors de l'agrandissement. Cette valeur est inversement proportionnelle à « cg_fs_scale » des 2 passes précédentes (« Rétrécissement » et « Homogénéité »).

**Texture (s) d'entrée (s)** :

- « FrameBuffer_homogeneous_filter » : l'image résultante de la passe de calcul d'homogénéité.

**Détail :**

Le programme shader de l'agrandissement est similaire à celui du rétrécissement, l'essentiel résident sur le paramètre d'entrée « cg_fs_scale » qui est inversement proportionnelle au « cg_fs_scale » du rétrécissement. Dans le cas du projet, « cg_fs_scale » du rétrécissement est de 4 (voir au début du fichier principale « TLLSLO.xml »). Alors « cg_fs_scale » du agrandissement sera de ¼ (=0,25).

### *k) Filtrage avec soustraction de fond et homogénéïté*

**Nom texture** : « FrameBuffer_homoXssFLOG_filter ».

**Fonction** : Filtrage des zones homogènes et en mouvement. image avec des zones de forte intensité pour les zones mobiles et homogène

**Nom fichier** : « filter_homoXssFLOG.xml ».

**Nom (s) fichier(s) shader(s) :** «TLLSLO-VP-final.cg » et « TLLSLO-FS-homoXssFLOG.cg ».

**Paramètre (s) d'entrée (s)** : aucun.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_homogeneous_filter » : image distinguant les zones homogènes ou contrastées
- « FrameBuffer_ssFondLOG_filter » : l'image résultante du système soustraction de fond.

**Détail :**

La multiplication de ces 2 textures permet de révéler les pixels à la fois mobiles et homogènes.

L'extrait du fichier « TLLSLO-FS-homoXssFLOG .cg »

```
float2 CoordsTmp = decalCoords.xy*cg_fs_2fv_size;
```

Conversion de l'unité unitaire en unité du pixel des coordonnées du texture du pixel local.

```
float3 homogeneous = texRECT(homo, CoordsTmp).rgb;
float3 ssFondLOG   = texRECT(ssFLOG, CoordsTmp).rgb;
```

Récupération des données de couleur du pixel.

```
homogeneous.r = homogeneous.r * 0.3 + homogeneous.g * 0.59 + homogeneous.b * 0.11;
ssFondLOG.r = ssFondLOG.r * 0.3 + ssFondLOG.g * 0.59 + ssFondLOG.b * 0.11;
```

Conversion les valeurs couleurs en intensité lumineuse.

```
return float4( float3(homogeneous.r * ssFondLOG.r), 1.0);
```

Revoie du résultat avec la multiplication des 2 intensités lumineuse.

### *l) Détection d'Ombre*

**Nom texture** : « FrameBuffer_dOmbre_filter »

**Fonction** : calcul le masque booléen pour distinguer l'ombre, le corps éclairé ou la silhouette du

fond.l'extraction de l'ombre selon les critères de sélection (FrameBuffer_dOmbre_filter)

**Nom fichier** : « filter_dOmbre.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-dOmbre.cg ».

**Paramètre (s) d'entrée (s) :**

« cg_fs_4fv_coef » : un tableau de 4 paramètre, le premier permet de définir le seuil du soustraction du fond, plus le seuil tente vers 1, moins il y a de pixels mobiles. Le deuxième paramètre permet de définir le seuil d'homogénéité, plus le seuil tente vers 1, moins il y a de pixels homogènes.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_ROADBI_filter » : l'image courante.
- « FrameBuffer_fond_filter » : l'image fond.
- « FrameBuffer_ssFondLOG_filter » : l'image de soustraction de fond.
- « FrameBuffer_homoXssFLOG_filter » : image distinguant les zones homogènes ou contrastées.

**Détail :**

Le filtre rassemble tout les différents texture contenant les informations utiles à l'analyse d'image. Il prendre des décisions à partir en effectuant des seuillages. Il permet d'extraire l'ombre comme un objet mobile (par soustraction de fond), homogène (par le critère d'homogénéité) et une diminution d'intensité lumineuse.

L'extrait du fichier « TLLSLO-FS-dOmbre.cg »

```
float4 textureInSum = float4 (0.0,0.0,0.0,1.0);
float4 wallpaperSum = float4 (0.0,0.0,0.0,1.0);

float3 coef = cg_fs_4fv_coef.xyz;

float4 currentSupBGColor;
float4 homoColor;
float4 noHomoColor;

//binaire mode
if (coef.z==1.){
	currentSupBGColor = BLACK;
	homoColor = WHITE;
	noHomoColor = WHITE;
}else{
	currentSupBGColor = RED;
	homoColor = WHITE;
	noHomoColor = GREEN;
}
```

Les options d'affichage : binaire ou colorié.

- Si « coef.z » est à 1, alors l'image résultante de la passe détection d'ombre sera une image binaire noir et blanc. En noir, les pixels appartiennent au fond statique. En blanc, les pixels appartiennent à l'ombre et le corps en mouvement.
- Si « coef.z » est à 0, alors l'image résultante de la passe détection d'ombre sera une image tricolore.
  - En rouge, les pixels appartiennent au corps en mouvement, car cette zone ou l'intensité lumineuse est supérieure à celle du fond statique.
  - En blanc, les pixels appartiennent strictement à l'ombre (à la fois mobile, zone où l'intensité lumineuse de l'image courante est inférieure à celle de l'image fond, enfin homogène d'après le critère d'homogénéité).
  - En vert, les pixels appartiennent au zone mobile mais pas homogène.

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
// img reference en dure taille video
float ssFondLOG = texRECT( ssFPLog , texCoord ).r;
```

Le premier test est le seuillage du soustraction de fond. Plus il y a de bruit, plus il faut rencontre strict le seuillage, plus il y a de perte d'information sur l'objet mobile.

```
if (ssFondLOG >coef.x )
```

```
{

	//si c'est un pixel mobil
	//replissage des différents donnée
	for( int i = 0 ; i < dim ; i++ ) {
		for( int j = 0 ; j < dim ; j++ ) {
			textureIn[i*dim+j]= texRECT( texIN  , float2 (texCoord.x+j-d2, texCoord.y+i-d2) ).rgba;
			textureInSum.rgb += textureIn[i*dim+j].rgb;
			wallpaper[i*dim+j]= texRECT( fond   , float2 (texCoord.x+j-d2, texCoord.y+i-d2)).rgba;
			wallpaperSum.rgb += wallpaper[i*dim+j].rgb;
		}
	}
```

Le tableau « textureIn » contient les 25 valeurs en couleur (RGB) du pixel local et ses voisins de la texture de l'image courante « FrameBuffer_ROADBI_filter ».
Le tableau «wallpaper» contient les 25 valeurs en couleur (RGB) du pixel local et ses voisins de la texture de l'image du fond « FrameBuffer_fond_filter ».

```
	//moyenne des 2 images
	textureInSum.rgb *= d3;
	wallpaperSum.rgb *= d3;
```

La variable « textureInSum » est la moyenne du pixel de l'image courante et ses 24 voisins.
La variable «wallpaperSum» est la moyenne du pixel de l'image du fond et ses 24 voisins.

```
	//test si le pixel de l'image courant est plus clair que le pixel fond : ça ne peut pas être une ombre
	float textureInSumGray = textureInSum.r * 0.3 +textureInSum.g * 0.59 +textureInSum.b * 0.11;
	float wallpaperSumGray = wallpaperSum.r * 0.3 +wallpaperSum.g * 0.59 +wallpaperSum.b * 0.11;
```

Le seconde test distinct si l'image courante a une intensité supérieure ou pas à l'image fond.

```
	if (textureInSumGray<wallpaperSumGray)
	{//c'est un pixel ombre ou ombre sur la silhouette

		float4 homogeneous = texRECT( homo  , texCoord ).rgba;
```

Le tableau « homogeneous » récupère les valeurs d'homogénéité de la texture respective calculée précedemment.

Le troisième test est un seuillage sur l'homogénéité.

```
		if (homogeneous.r>coef.y)
		{
			return homoColor;
		}else{
			return noHomoColor;
		}
	}else{
		//pixel avec une fort variance avec ses voisons, soit le contour de l'ombre soit la silhouette
		return currentSupBGColor;
	}
}else{
	//si le pixel est un fixel fond
	return float4 (0.0,0.0,0.,1.0);
}
```

## 6. Module de détection de mouvement

Le module de détection de mouvement (tracking) permet à partir de l'image de l'ombre (calculée à partir des opérations du module précédent), de suivre en temps-réel le déplacement d'un ou des objets en calculant le centre ou l'axe central de la boîte englobante (voir illustration 14).

La boite englobante désigne un objet 2D/3D qui renferme un élément complexe. L'élément complexe est l'objet mobile. Dans le cas du projet TLLSLO, cette boîte 2D est définie par 4 valeurs cardinaux (xmin, xmax, ymin et ymax). Ces 4 derniers permettent de définir 4 points de la boîte (HG, HD, BG et BD) où chaque point est une position en X et en Y. A partir de ses informations, il y a plusieurs possibilités de calculs; par exemple, calculer le centre entre les 4 points, ou juste le centre entre les 2 ponts en haut de l'image (HG et HD) …

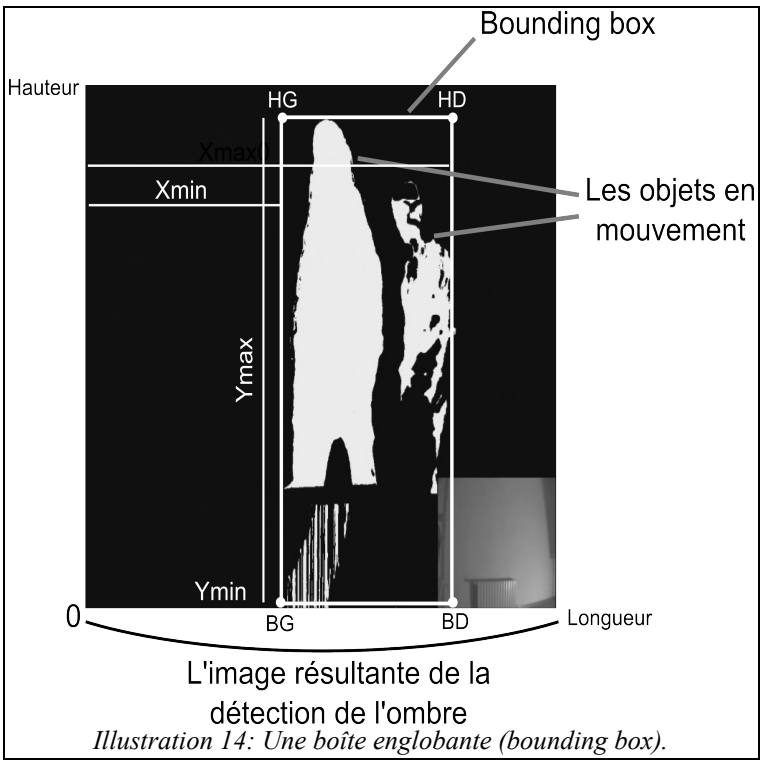


*Illustration 14: Une boîte englobante (bounding box).*

Pour que le système soit en temps-réel, il n'est impossible d'effectuer le calcul de la boîte englobante au niveau CPU. Le choix de l'implémentation se portes sur la programmation GPU.

Au niveau de la programmation GPU shader, il y a principalement 2 programmes; Les « vertex program », permet de travailler sur les vertex, qui se compile pour chaque vertex. Les « pixel program », permet de travailler sur les pixels, qui se compile pour chaque pixel. Dans le premier cas, le calcul de la boite englobante sera répété 3 fois. Et dans le seconde cas, il sera répété avec le nombre de pixel visible par la caméra. Les dernières versions des pilotes de la programmation graphique permettent au informaticiens d'intervenir à un autre niveau du pipeline graphique. Après avoir la capacité d'intervenir au niveau vertex, puis au niveau pixel, ils ont maintenant capable d'intervenir au niveau primitif : c'est le **Geometry Shader** (une petite introduction en anglais : http://en.wikipedia.org/wiki/Geometry_shader).

Si l'objet de base pour le calcul de la boîte englobante est juste un primitif avec 3 vertex, alors, ce calcul sera effectué qu'une seule fois.

### *a) Fonctionnement du système de tracking dans VirChor*

Dans le fichier « TLLSLO.xlm », un ensemble d'étape ont été réalisé pour le système de suivie.

Variable

« boundingBoxCenter » : un tableau de taille 2 contenant la position relative au résultat de la boîte englobante.

« boundingBoxCenterM1 » : un tableau de taille 2 contenant la position relative au résultat de la boîte englobante à l'instant d'avant.

Les étapes du fonctionnement du système de tracking dans VirChor :

- copier « boundingBoxCenter » dans « boundingBoxCenterM1 ».
- calculer la boîte englobante grâce à la « geometry shader » dans le fichier « boundingBoxGS.xml ». Le résultat final sera rendu sous la forme d'une couleur où le canal rouge sera la position de l'axe horizontal et le canal vert sera la position de l'axe vertical.
- Placer le pointeur de lecture de la couleur affichée à l'écran grâce au tag XML « pixel_reader ». La couleur du pixel sera automatiquement affectée au variable « pixel_color_r », « pixel_color_g » et « pixel_color_b » du nœud de configuration,
- copier les valeurs « pixel_color_r » et « pixel_color_g » dans le tableau « boundingBoxCenter ».
- envoyer les tableaux « boundingBoxCenter » et « boundingBoxCenterM1 » au paramètre « cg_fs_4fv_pt » (un tableau de 4 valeurs) pour le moteur de particule (contenu dans le nœud « wordPosComputing ») pour le mode d'interaction d'attraction des particules vers le centre de l'objet en déplacement.

### *b) Calcul de la boîte englobante de l'objet en mouvement*

**Nom texture** : aucun.

**Fonction** : calcul de la boîte englobante et renvoie la position d'un pixel de l'image (voir illustration 15).

**Nom fichier** : « boundingBoxGS.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-BoundingBox.cg », « GS-bounding.cg » et « TLLSLO-FS-bounding.cg ».

**Paramètre (s) d'entrée (s)** : « cg_gs_2fv_start »

**Texture (s) d'entrée (s)** : « FrameBuffer_dOmbre_filter »

**Détail :**

L'extrait du fichier « boundingBoxGS.xml »

Il est nécessaire de détailler un peu l'implémentation de cette passe de calcul au niveau XML.

```
<vertexShader xmlns:xlink = "xlink" xlink:href = "shaders/TLLSLO-VP-BoundingBox.cg"/>
<geometryShader xlink:href = "shaders/GS-bounding.cg" output_type = "triangle"/>
<fragmentShader xmlns:xlink = "xlink" xlink:href = "shaders/TLLSLO-FS-bounding.cg"/>
```

L'utilisation exceptionnelle de la balise « geometryShader » charge le fichier pour intervenir durant l'étape de la création des primitifs.

Du faite qu'il y a l'intervention au niveau primitif, alors il suffit de créer un seul primitif (une

triangle) pour que le calcule de la boîte englobante soir se réalise qu'une seule fois.

```
<tabvertex size = "3">
        <vertex index = "1" x = "0" y = "4" z = "-11"/>
        <vertex index = "2" x = "-4" y = "-4" z = "11"/>
        <vertex index = "3" x = "4" y = "-4" z = "11"/>
</tabvertex>
```

## L'extrait du fichier « GS-bounding.cg »

Au niveau du passage du paramètre d'entrée-sortie du programme principal « main ».

```
uniform samplerRECT lookupTable1 : TEXTURE1,
```

Cette texture est l'image résultante de la passe détection de l'ombre « FrameBuffer_dOmbre_filter », elle sert au calcule de la boîte.

```
AttribArray<float4> position : POSITION,
AttribArray<float2> dc : TEXCOORD0,
AttribArray<float4> color : COLOR
```

Pour ce primitif, il a 3 vertex définies par leur position xyzw, il a 3 pairs de valeur (pour chaque vertex) en tant que coordonnées de textures, et il a 3 couleurs (pour chaque vertex) définies en rgba.

Au sein du programme « main ».

```
float2 border = float2(5,5);
float minX = cg_gs_2fv_start.x -border.x;
float minY = cg_gs_2fv_start.y -border.y;
float maxX = border.x ;
float maxY = border.y ;
```

Initialisation des valeurs cardinales « minX », « minY », «  maxX » et « maxY ».

```
float intensite;
float intensiteTotal = 0;
float window = 10;
float2 dir [5] = {{0,0},{-window,-window},{window,-window},{window,window},{-window,window}};
for (int i = border.x ; i < cg_gs_2fv_start.x-border.x ; i = i+8) {
        for (int j = border.y ; j < cg_gs_2fv_start.y-border.y ; j =j+8) {
                for(int k = 0 ; k < 5 ; k++){
                        intensiteTotal += texRECT( lookupTable1, float2(i, j)+dir[k] ).r;
                }
```

Pour la fiabilité des valeurs cardinales, au lieu de prendre que l'intensité lumineuse du pixel local, le choix se porte sur 5 valeurs à 10 pixels de distance. C'est une optimisation pour la performance temps-réel, le choix de départ se porte sur la valeur du pixel local et ses 24 voisins, mais le temps de calcul devient trop important.

```
        if (intensiteTotal > 4.5){
```

Si l'intensité de la somme dépasser ce seuil, alors il faut réactualiser l'une des 4 valeurs cardinales.

```
                        if ( minX > i)
                                minX= i ;
                        if ( maxX < i)
                                maxX = i ;
                        if (  minY > j)
                                minY =j;
                        if ( maxY < j)
                                maxY =j;
                }
```

```
                intensiteTotal = 0.;
        }
}
// emission face originale
float4 p1 = position[0].xyzw ;
```

Tout le traitement de la chercher des 4 valeurs cardinales a été traité dans selon des unités pixels. Il faut les reconvertir en valeur unitaire.

```
minX = minX/cg_gs_2fv_start.x*2.-1;
maxX = maxX/cg_gs_2fv_start.x*2.-1;
minY = minY/cg_gs_2fv_start.y*2.-1;
maxY = maxY/cg_gs_2fv_start.y*2.-1;

float2 cardinal [4] = {{minX,minY},{minX,maxY},{maxX,maxY},{maxX,minY}};
float2 central = float2((maxX+minX)/2.0,(maxY+minY)/2.0);
```

Définir le point central de la boite englobante.

```
float2 centraltop = float2((maxX+minX)/2.0,maxY);
```

Définir le point en haut centré sur l'axe horizontal.

```
float2 dc1 = dc[0].xy ;
float2 dc2 = dc[1].xy ;
float2 dc3 = dc[2].xy ;

float promiSize = 0.01;

//bas gauche
emitVertex( float4(cardinal[0].x,cardinal[0].y+promiSize,p1.zw) : POSITION,  dc1 : TEXCOORD0, white: COLOR);
emitVertex( float4(cardinal[0].x-promiSize,cardinal[0].y-promiSize,p1.zw) : POSITION,  dc2 : TEXCOORD0 , white: COLOR );
emitVertex( float4(cardinal[0].x+promiSize,cardinal[0].y-promiSize,p1.zw) : POSITION,  dc3 : TEXCOORD0 , white: COLOR );
restartStrip();
```

La création d 'un petit marqueur pour visualiser le point en bas à gauche de la boîte englobante avec les valeurs cardinales « cardinal[0] », dont la paire de valeur « {minX,minY} ».

Ce marqueur et un primitif triangle, constitué de 3 vertex, accompagnés de chacun une coordonnée de texture, et une couleur. Chaque vertex est créé par la fonction « emitVertex() ».

Une fois que les 3 vertex définis, il y a création du primitif avec la fonction « restartStrip() ».

```
//haut gauche
...
//bas droit
...
//haut droit
…

// central top
emitVertex( float4(centraltop.x,centraltop.y+promiSize,p1.zw) : POSITION, dc1 : TEXCOORD0, white: COLOR);
emitVertex( float4(centraltop.x-promiSize,centraltop.y-promiSize,p1.zw) : POSITION,  dc2 : TEXCOORD0, white: COLOR );
emitVertex( float4(centraltop.x+promiSize,centraltop.y-promiSize,p1.zw) : POSITION,  dc3 : TEXCOORD0, white: COLOR );
restartStrip();
```

La définition du marqueur en haut au centre de la boîte englobante.

```
float2 valuePos = float2(0.,cg_gs_2fv_start.y)/cg_gs_2fv_start*2.-1.;
float4 centralValue = float4((centraltop+1)/2., 0,1);
emitVertex( float4(valuePos.x,valuePos.y+promiSize,p1.zw) : POSITION,  dc1 : TEXCOORD0, centralValue: COLOR );
emitVertex( float4(valuePos.x-promiSize,valuePos.y-promiSize,p1.zw) : POSITION,  dc2 : TEXCOORD0, centralValue: COLOR );
emitVertex( float4(valuePos.x+promiSize,valuePos.y-promiSize,p1.zw) : POSITION,  dc3 : TEXCOORD0, centralValue: COLOR );
restartStrip();
```

La définition d'un primitif en haut à droit de l'image de manière statique, et qui a comme couleur la valeur de la position du point en haut au centre de la boîte englobante.

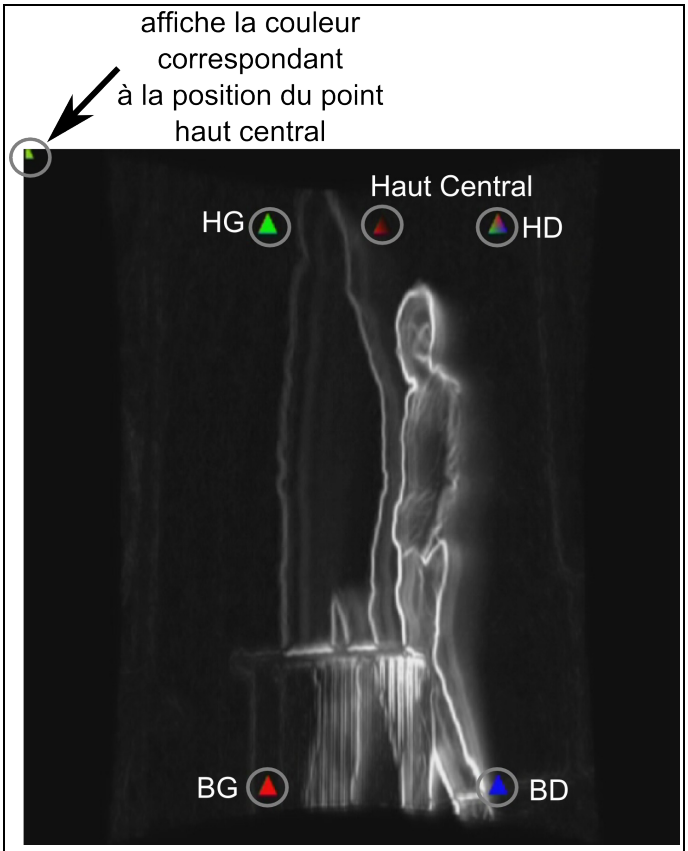


*Illustration 15: Les 5 marqueurs de la boîte englobante et le primitif qui donne la position d'un point.*

## 7. Module du moteur des particules

Les moteurs ou les systèmes de particules, en infographie, sont utilisés pour la synthèse de phénomènes physiques (pluie, fumée, feu, explosions, …) et les objets déformables (cordes, chaînes, tissus, gels, …).
Un moteur de particules modélise un système physique par un ensemble de particules élémentaires soumises à des forces (gravité, vent, …). L'évolution du moteur est calculée par simulation discrète temporaire : on fixe un pas de temps dt et on calcule l'état du système à t+dt en fonction de l'état à l'instant t. Le système est constitué de deux composants, un composant de simulation et un composant de rendu graphique:
-Le composant de simulation calcule l'évolution du système au cours du temps.
-Le composant de rendu affiche l'état du système à chaque pas de simulation.

**Principe**
Le principe de ce moteur physique de particules est d'utiliser deux textures qui stockent la position de chaque particule à l'instant t-1 et à l'instant t-2 pour calculer la position prévisionnelle à l'instant t (à l'instant courant). Cette position prévisionnelle est mise en interaction avec l'image capturée lors de l'acquisition vidéo.
Une collision existe lors qu'il y a une forte variation d'intensité lumineuse de l'image capturée entre la position à l'instant précédant (t-1) et la position prévisionnelle (voir Illustration 16). La collision de la particule correspond à un rebondissement sur l'objet en interaction selon la pente (représenté par le vecteur normale) de l'endroit où la particule va frapper. S'il y a collision, alors la position prévisionnelle de la particule sera modifiée, sinon cette position sera conservée.

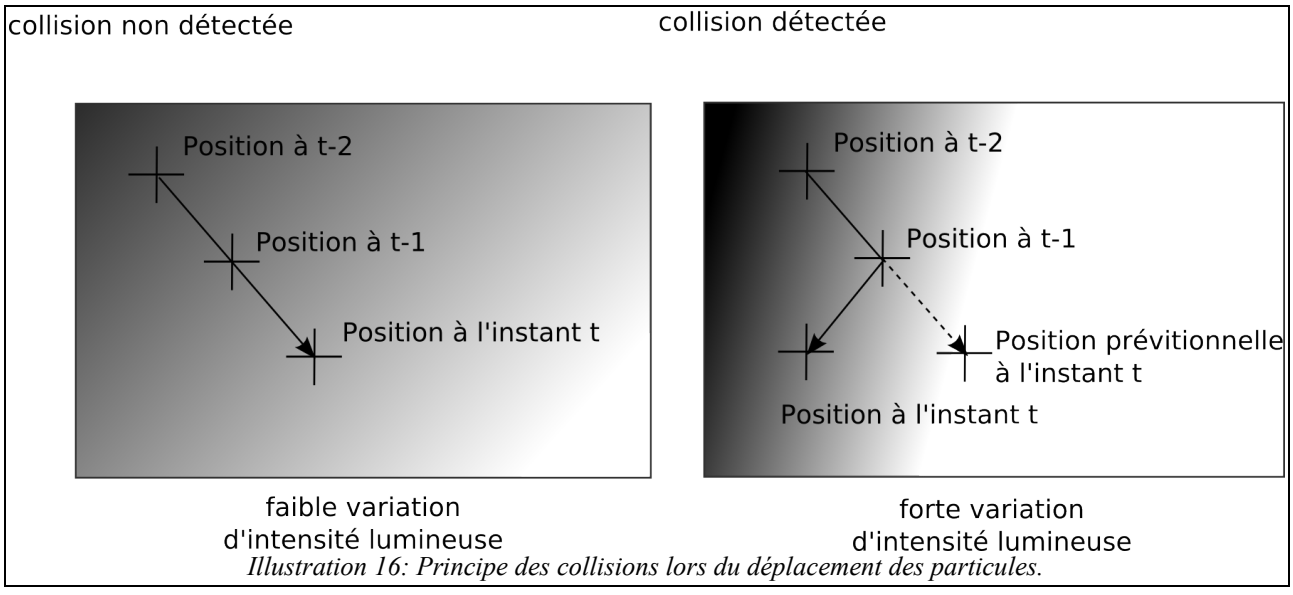

*Illustration 16: Principe des collisions lors du déplacement des particules.*

Une particule est le plus petit élément du système. Dans le moteur de simulation, une particule est définie par sa position en x et en y (dans le cadre de notre dispositif, les particules se déplacent sur un plan), et par son identifiant (dans un tableau à double entrée ligne/colonne). Les coordonnées de l'ensemble des particules sont stockées dans une texture (voir Illustration 17).

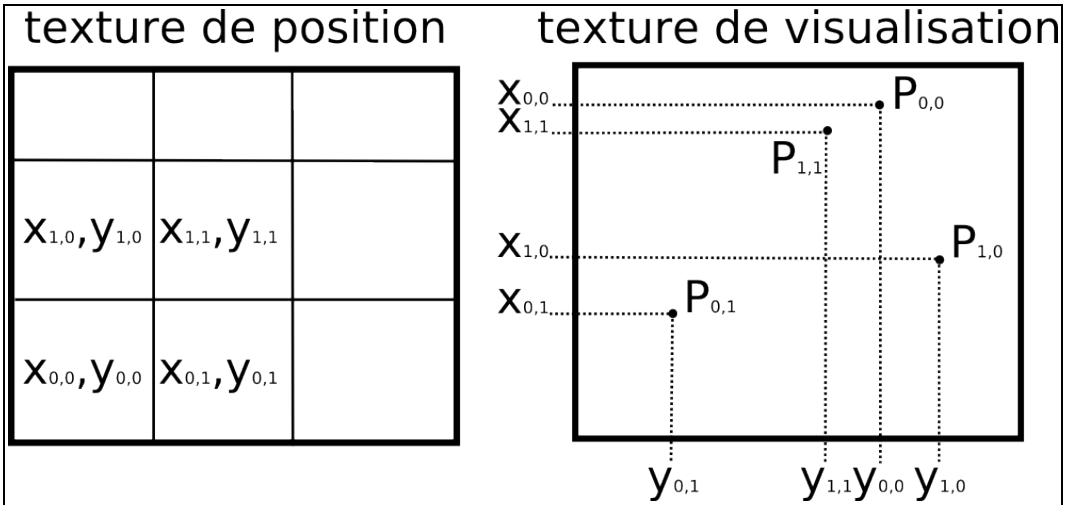


*Illustration 17: Relation entre la texture de position et la texture de visualisation.*

Le module du moteur des particules est le principale atout du dispositif d'interagir avec le monde monde réel. . Le monde réel est capturé par la caméra infrarouge, et c'est à partir de cette image que le système a pu définir les objets en mouvements. L'image d'entrée du module est l'« **image de la détection d'ombre** » que les particules font interagir avec. Avant d'aller au cœur du système des particules, il y a un ensemble d'opération à pré-calculer. D'abord il y a la passe « **normalMap** », il établit pour chaque pixel considérer comme contour, sa pente (la tangente) et son vecteur normal (orthogonale à la surface de rebond). Comme ce calcul de la normal map a été effectué de manière locale, les vecteurs normaux sur une petite surface peux avoir de forte variation. Donc il faut une autre opération « **normalMapBlur** » pour atténuer une trop grande variations entre les vecteurs normaux locaux.

Néanmoins, la valeur de la position d'une particule est d'une valeur discrète. Alors que les informations contenant dans une images, l'objet en mouvements est sous forme de données indiscrètes; à savoir qu'il n'as pas d'identifiant, et il peut disparaître, démultiplier, fusionner... De cette nature, il n'est impossible que toutes les particules rebondissent de manière absolue sur la parois de l'objet. La passe « **dOmbreEcho** » permet de rendre le système plus performante.

La passe « PrograssWordPart » calcule le déplacement des particules en temps-réel. Elle reçoit, les images contenant des informations pré-calculées et analysées de l'image courante (ex : « normalMapBlur »), mais aussi les textures de positions des particules des instants précédents (t-1 et t-2 : « wordPosM1 » et « wordPosM2 »). Une fois la textures de position à l'instant t calculée, elle devient une texture d'entrée pour deux passes de rendu indépendante :

-La passe « quadRendering » permet d'obtenir un rendu visuel sous forme de goutte.

-La passe « wordQuadRendering » permet d'obtenir un rendu visuel sous forme de mot.

Enfin la passe « 2particle » permet de choisir le quel des 2 rendus est pris en compte (voir illustration 18).

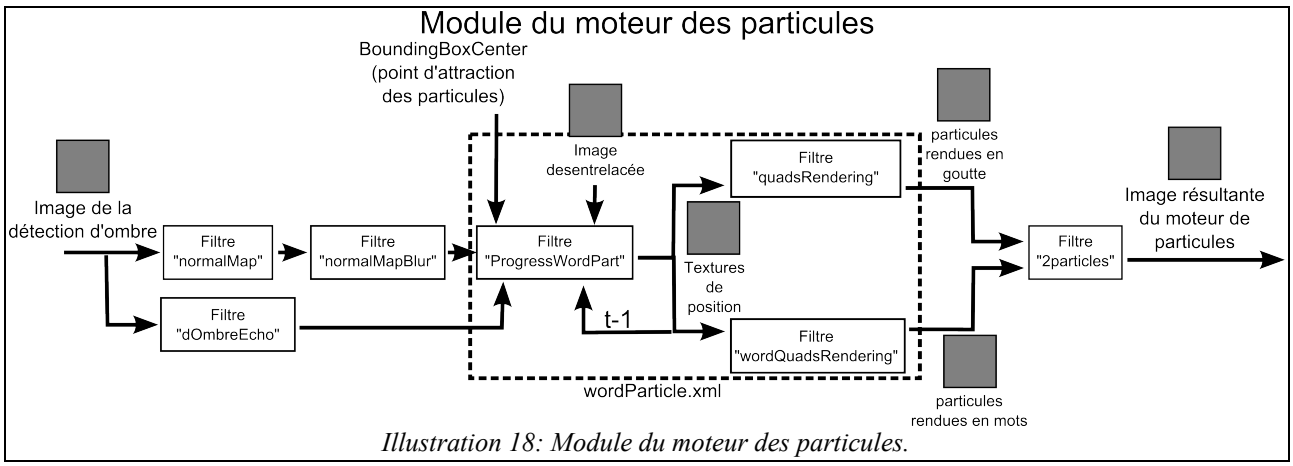


*Illustration 18: Module du moteur des particules.*

### *a) FrameBuffer_normalMap_filter*

**Nom texture :** « FrameBuffer_normalMap_filter ».

**Fonction :** calculer le vecteur normale du contour de l'objet en mouvement.

**Nom fichier :** « filter_normalMap.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-normalMap.cg ».

**Paramètre (s) d'entrée (s) :** aucun.

**Texture (s) d'entrée (s) :** « FrameBuffer_dOmbre_filter » l'image binaire résultante de la détection de l'ombre.

**Détail :**

L'extrait du fichier « TLLSLO-FS-normalMap.cg »

Le calcule de « sobelH » (le gradient horizontale) et « sobelV » (le gradient vertical) est similaire aux opérations du section « Détection de contour Sobel ». Sauf qu'au lieu calculer la norme du vecteur, « normalMap » conserve la nature du vecteur (exprimer avec 2 valeurs réelles relatives).

```
//sobel
sobelV.r = 0.3 * sobelV.r+ 0.59 * sobelV.g+ 0.11 * sobelV.b;
sobelH.r = 0.3 * sobelH.r+ 0.59 * sobelH.g+ 0.11 * sobelH.b;
float2 Sobelnormalized = float2(sobelH.r, sobelV.r);

if ((Sobelnormalized.x!=0.)||(Sobelnormalized.y!=0.))
{
	Sobelnormalized = normalize(Sobelnormalized);
```

Il faut normaliser les valeurs être dans une intervalle entre -1 et 1.

```
	Sobelnormalized /=2.;
}
Sobelnormalized +=0.5;
```

Comme l'intervalle des valeurs de sortir est de 0 à 1. alors il faut diviser les valeurs du vecteur par 2 (pour avoir une intervalle de -0,5 à 0,5), puis ajouter 0,5 pour avoir du 0 à 1.

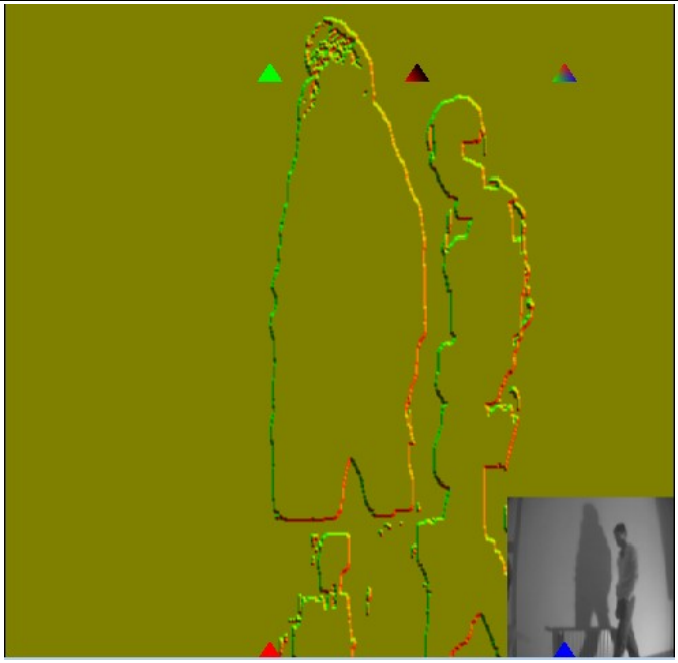

*Illustration 19: NormalMap : contenant le vecteur normal des pixels contour.*

### *b) FrameBuffer_normalMapBlur_filter*

**Nom texture** : « FrameBuffer_normalMapBlur_filter »

**Fonction** : atténuer une trop grand variation de direction des vecteurs normaux des pixels de contour (voir illustration 19).

**Nom fichier** : « filter_normalMapBlur.xml »

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-normalMapBlur.cg ».

**Paramètre (s) d'entrée (s)** : aucun.

**Texture (s) d'entrée (s)** : « FrameBuffer_normalMap_filter » : image contenant les valeurs du vecteur normal des pixels de contour.

**Détail :**

De manière grossière, l'illustration 20 représente l'avant et l'après avoir effectué une moyenne du vecteur normal du pixel contour avec ses deux voisins proches. Avant, la direction de rebond calculé à partir de la position de la particule et le vecteur normal peut varier de manière importante à une pixel prés. Après avoir faire une atténuation de la variation du vecteur normal du pixel contour avec ses 2 voisins proches, La direction du vecteur du pixel varie moins avec ses voisins.

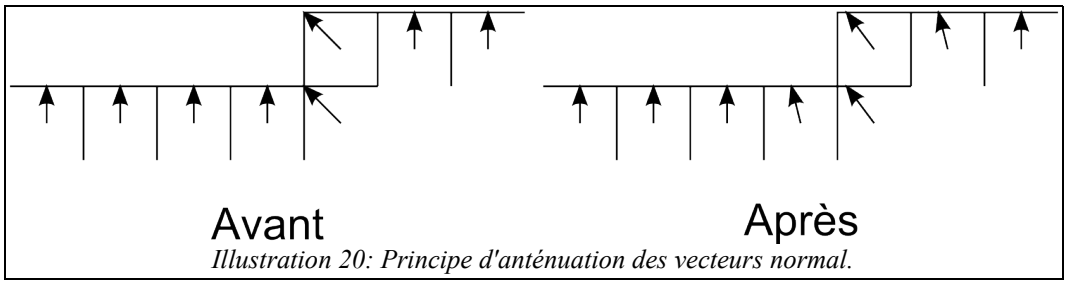


*Illustration 20: Principe d'anténuation des vecteurs normal.*

Cette amélioration permet à une échelle macroscopique, de rendre le modèle de rebond des particules plus réalise. Car l'œil nu voit pas exactement quel pixel contour la particules va rebondir. Donc l'utilisation peut à priori voir 2 particules rebondir à un même endroit, et de repartir à des direction complétement différentes: c'est à cela que le système veut éviter.

L'opération d'atténuation a été effectué en réalisant un traitement successifs de 6 fois l'opération de la moyenne des vecteurs normaux.

L'extrait du fichier « TLLSLO-FS-normalMapBlur.cg »

```
float2 coordTmp;
coordTmp = decalCoords.xy*cg_fs_2fv_size;
float4 vecNormal =  texRECT(MaVideo,coordTmp).rgba;
if (!((vecNormal.r < 0.50)&&(vecNormal.r > 0.49)&&(vecNormal.g < 0.50)&&(vecNormal.g > 0.49)))
{
```

Il y a pas de traitement si le pixel local n'est pas un pixel contour (donc vecteur normal nul).

```
	float2 dir [9] = {{-1,-1},{-1,0},{-1,1},{0,-1},{0,0},{0,1},{1,-1},{1,0},{1,1}};
	float4 result = float4 (0.,0.,vecNormal.b,0.);
	float coef=0;
	float4 normalTmp;
	for(int i = 0; i < 9 ; i ++){
		normalTmp = texRECT(MaVideo,coordTmp+dir[i]).rgba;
		if (!((normalTmp.r < 0.50)&&(normalTmp.r > 0.49)&&(normalTmp.g < 0.50)&&(normalTmp.g > 0.49))){
```

Ne pas prendre en compte les pixels non contour au tour du pixel local lors de la réalisation de la moyenne.

```
			result +=normalTmp;
			coef++;
		}
	}
	result /=coef;
	result.xy = (result.xy-0.5)*2.;
```

Mettre les valeurs dans l'intervalle -1 à 1 pour la normalisation du vecteur.

```
	result.xy = normalize(result.xy);
	result.xy = (result.xy*0.5)+0.5;
```

Remettre les valeurs dans l'intervalle 0 à 1 (domaine de la couleur de sortie).

```
	return result;
}
return vecNormal ;
```

### *c) Echo de la détection d'Ombre*

**Nom texture** : « FrameBuffer_dOmbreEcho_filter »
**Fonction** : Renforcer les rebonds des particules.
**Nom fichier** : « filter_dOmbreEcho.xml ».
**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-dOmbreEcho.cg ».
**Paramètre (s) d'entrée (s)** :
**Texture (s) d'entrée (s)** : « FrameBuffer_dOmbre_filter »
« FrameBuffer_dOmbreM1_filter »
**Détail :** aucun**.**

L'extrait du fichier « TLLSLO-FS-dOmbreEcho.cg »

```
float2 CoordsTmp = decalCoords.xy*cg_fs_2fv_size;
```

Redimensionner en unité pixel.

```
float3 dOmbre = texRECT(dOm, CoordsTmp).rgb;
float3 dOmbreM1 = texRECT(dOmM1, CoordsTmp).rgb;
```

Récupération de la valeur en couleur de l'image résultante de la détection de l'ombre et celle de l'instant d'avant.

```
dOmbre.r = dOmbre.r * 0.3 + dOmbre.g * 0.59 + dOmbre.b * 0.11;
dOmbreM1.r = dOmbreM1.r * 0.3 + dOmbreM1.g * 0.59 + dOmbreM1.b * 0.11;
```

La moyenne est calculée à partir de ces 2 valeurs du même pixel et est renvoyée à la sortie du programme.

```
return float4( float3(0.5 * dOmbre.r+ 0.5 *dOmbreM1.r), 1.0);
```

### *d) Moteur physique de particules*

**Nom texture** : « wordPos »

**Fonction** : calculer la position des particules en interaction avec l'image courante.

**Nom fichier** : « wordParticle.xml ».

**Nom (s) fichier(s) shader(s) :** « FullscreenPart-VP.cg » et « ProgressWordPart-FS.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_4fv_pt »: contient 4 valeurs, dont 2 point (position en x et en y), « boundingBoxCenter » le point d'attraction à l'instant t et « boundingBoxCenterM1 » le point d'attraction à l'instant t-1. Ces deux point forme le vecteur de déplacement de l'objet mobile (voir la section « **Module de détection de mouvement** »).
- « cg_fs_start » : permet à la réinitialisation du système de particule.
- « cg_fs_partmode » : 0 pour le mode de gravité, et 1 pour le mode d'attraction.

**Texture (s) d'entrée (s)** :

« FrameBuffer_dOmbreEcho_filter » : renforcement du rebond des particules qui ont traversés le parois de l'objet en mouvement.

« wordPosM1 » et « wordPosM2 » : les textures de positions des particules à l'instant t-1 et t-2.

« FrameBuffer_normalMapBlur_filter » : contient les vecteurs normaux de direction des pixel de contour.

« FrameBuffer_deinterlaced_filter » : permet à la réinitialisation de la position des particules.

**Détail :**

#### Choix du mode de déplacement des particules

Avant tout calcul, il faut choisir le mode de déplacement des particules selon la valeur de « cg_fs_partmode » :

- « 0 » pour le mode de gravité : à savoir que les particules tombes selon un vecteur de déplacement et en fonction de l'objet en déplacement (contour et l'intérieur des contours),
- et 1 pour le mode d'attraction selon un point, les particules se déplacent vers ce point, mais en fonction du chemin que l'objet en mouvement lui offre.

#### Principe de calcul de la position de la particule.

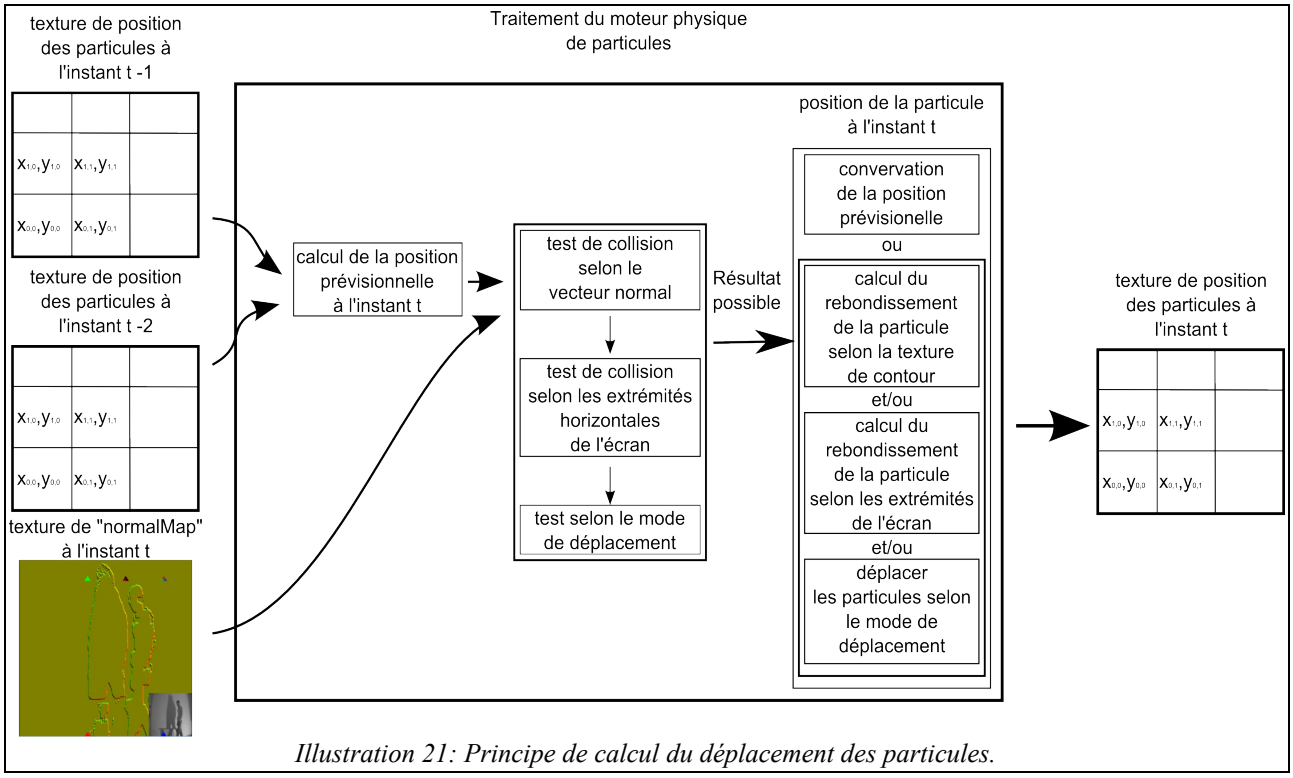


*Illustration 21: Principe de calcul du déplacement des particules.*

Pour une particule donnée, la position prévue est définie par sa position aux instants d'avant (t-1 et t-2). Puis Cette position est analysée pour savoir si c'est au bord, à l'intérieur ou l'extérieur de l'objet en mouvements (ombre, corps éclairé ou silhouette). Après il y a test sur le rebond à l'extrémité de l'image. Enfin, il y a un test de rebonds selon le mode choisi. Ces tests permettent de définir une nouvelle position que celle prévue (s'il y a collisions). Et à la sortie du système, il y a aura une nouvelle position de la particule (voir illustration 21).

## L'extrait du fichier « ProgressWordPart-FS.cg »

Le programme shader est subdivisé en 2 grandes parties selon les 2 modes. La description qui suit est dans la même l'ordre pour la compréhension.

### 1. Mode d'attraction autour de l'objet en mouvements

```
float4 result;
float4 previousValuesT0,previousValuesT1;
float2 sizeCoefAttracPoint = float2(640,480)/cg_fs_2fv_size;
float2 attracPoint = cg_fs_4fv_pt.xy * sizeCoefAttracPoint;
float2 attracPointM1 = cg_fs_4fv_pt.zw * sizeCoefAttracPoint;
```

Convertir la valeur des coordonnées des 2 point d'attraction du 0 à 1 en proportion de la texture de traitement et la taille actuelle de la fenêtre.

```
if (screenLoc.x>4 || screenLoc.y>4){
```

Limiter le traitement de la taille de la texture de position des particules revient à définir le nombre de particules que le systèmes utilisera (dans ce ça, seul 16 particules sont utilisées). La position des autres particules ne subisse aucune modification.

```
        return float4(0.,0.,0,1);
```

```
}else{
	if (cg_fs_start>0.1){
		return float4(
		screenLoc.x/4*sizeCoefAttracPoint.x,
		screenLoc.y/4*sizeCoefAttracPoint.y,
		0,1);
```

L'initialisation de la position de départ des 16 particules est sous forme de tableau (ligne colonne).

```
	}else{
		previousValuesT0 = texRECT(positions0,screenLoc.xy);
		previousValuesT1 = texRECT(positions1,screenLoc.xy);
```

Récupération de la position de la particule aux instant précédentes (t-1 et t-2).

```
		float2 sizeCoef = float2(screenLoc.xy/decalCoords.xy)*cg_fs_2fv_size/float2(640,480);
```

Les variables et leur définition:

- « screenLoc.xy » : « ScreenLoc.x » est l'index colonne et « ScreenLoc.y » est l'index ligne. Ces 2 valeurs sont la position du pixel (« WPOS ») directement récupérer de la carte graphique. Elles varient de 0 à la taille de la fenêtre.
- « decalCoords.xy » : coordonnées de texture qui varient de 0 à 1.
- « cg_fs_2fv_size » : la taille de la fenêtre, qui varient de 0 à la taille, passée en paramètre en tant que « uniform » (la même valeur pour tous les pixel à un moment donné).
- « float2(640,480) » une paire de valeurs fixes pour des texture de traitement. Le faite d'être le dénominateur de « cg_fs_2fv_size » permet de tous convertir à 0-1.

```
		float2 speed; float2 speedInit ;
		float2 supposedNextPos ;  float2 shadowSpeed;
		float collision ; float intensite ;
		//test de collision la position de particle doit être dans un pixel noir de valeur nulle
		//la silhouette, donc la rebord est blanc
		speed = previousValuesT0.xy - previousValuesT1.xy ;
```

Le vecteur vitesse prévisionnelle de la particule.

```
		speedInit = (screenLoc.xy/4.*sizeCoefAttracPoint) - previousValuesT0.xy;
		shadowSpeed = attracPoint - attracPointM1;
```

Le vecteur de déplacement de l'objet en mouvement selon 2 points d'attraction.

```
		//speed computing about attract point
		if (attracPoint.y> 0.02){
			//if the bounding box point is above the particle position
			if (attracPoint.y>previousValuesT0.y){
				//attract to bounding box Y axis center
				speed.x = attracPoint.x - previousValuesT0.x;
			}else{
				//attract to bounding box center
				speed = attracPoint - previousValuesT0.xy;
			}
```

Vue que le point d'actraction n'est pas le centre de la boîte englobante mais le point centre haut de cette dernière.

Si la position de la particule est plus haut que le point centre haut de la boîte, alors la particule s'attire vers ce point (à la fois sur l'axe X et Y. Si la position de la particule est plus bas que le point centre haut, alors elle s'attire que vers l'axe

verticale du centre de la boîte (conservation de la valeur de y).

```
}else{
        //return to init position
        speed = speedInit*0.5 ;
}
```

Avec l'hypothèse que si le point centre haut de la boîte englobante est très bas, alors il y a pas d'objet en mouvements.

```
speed *= 0.075;
if (texRECT (video,previousValuesT0.xy* sizeCoef).r>0.5){
```

Si la positon actuelle de la particule, elle se situe déjà à l'intérieur de l'objet en mouvements. C'est le cas de la « fuite des particules » dans le sens ou l'objet en mouvement n'est pas une entité discrete. Dans ce cas là, pas de rebond possible à partir du vecteur normal du pixel contour. Nous exerçons tout simplement une force opposée pour refaire sortir la particule.

```
        int todo1 = 0;
        for (float i = 1. ;  i < 20. ; i = i+1 ){
                if (todo1==0){
                        if (texRECT (video,(previousValuesT0.xy-(speed*i))* sizeCoef).r<0.5){
                                speed = - (speed*i);
                                todo1 = -1;
                        }
                        if (texRECT (video,(previousValuesT0.xy+(speed*i))* sizeCoef).r<0.5){
                                speed =  (speed*i);
                                todo1 = -1;
                        }
                }
        }
        if (todo1==0)
                return float4 (previousValuesT0.xy+(shadowSpeed*0.5 + 0.5 *speed),0.,1.);
        else
                 return float4 (previousValuesT0.xy+speed,0.,1.);
}
supposedNextPos = previousValuesT0.xy +speed;
```

« supposedNextPos » est la position prévisionnelle de la particule.

```
collision = collisionDetection(video,previousValuesT0.xy*sizeCoef,supposedNextPos*sizeCoef);
```

« collisionDetection » est une fonction qui prend en paramètre l'image binaire de la détection d'ombre, la position actuelle de la particule et sa position prévisionnelle. Renvoi 1 si il y a collision (la couleur de l'une des 2 positions est noire et l'autre blanche).

```
if (collision >0.5){
```

s'il y a collision d'après une valeur de seuillage par défaut (0,5), alors il faut chercher la paire de valeur du vecteur normal du pixel de contour.

```
        float2 SobelValue;
        //cherche le point d'impact
        float step = 500;
        float2 vecRecheche = speed/step;
        int todo = 0;
        //le vecteur AO est le vecteur entre la position de particule au point d'impact
```

```
                        float2 vecAO;
                        //récuperation de la valeur
                        for (int i = 0 ; i < step ; i ++){
                                if (todo==0){
                                        collision= collisionDetection(video,previousValuesT0.xy*sizeCoef,
(previousValuesT0.xy+vecRecheche*(i))*sizeCoef);
                                        if (collision >0.5){
                                                todo = -1;
                                                //point impact trouvé
                                                vecAO = vecRecheche*(i);
                                                //récuperation de la valeur
                                                SobelValue = texRECT(lookupTable3,
(previousValuesT0.xy+vecAO)*sizeCoef).rg;
                                                SobelValue -= float2(0.5);
                                                SobelValue *= float2(2.0);
```

Une fois que le vecteur normal trouvé, il faut le convertir du 0 à 1 à -1 à 1.

```
                                                float2 ptImpact = previousValuesT0.xy+vecAO;
```

« ptImpact » est la position précise où la particule va rebondir.

« vecAO » est la différence entre « ptImpact » et la position prévues « previousValuesT0 ».

```
                                                float2 normalD = SobelValue;
                                                float2 vecAOD = normalize (vecAO);

                                                float2 rebond;
                                                rebond = reflect(vecAOD,normalD);
```

La fonction « reflect » est prédéfinie par le langage Cg, est à priori utilisé pour le Raytracing (luminosité multipasse). Et permet de calculer le rebond du vecteur incident (point d'impact – position prévue) et le vecteur normal.

```
                                                float2 rebondD = normalize(rebond);
                                                return float4 (ptImpact+ rebondD*0.02,0.,1.);
```

Renvoie du résultat obtenue du rebond à la point d'impact « ptImpact » à la direction vecteur rebond « rebondD ».

```
                                        }
                                }
                        }
                }

                speedInit = ((screenLoc.xy/4.*sizeCoefAttracPoint) - previousValuesT0.xy)*0.5 + 0.5 *(attracPoint.xy
– previousValuesT0.xy);
```

Les tests de collision de rebord et le changement de direction.

```
                //border collision
                if (supposedNextPos.x<-0.1)
                        speed.x = speedInit.x;
                if (supposedNextPos.x>1.1)
                        speed.x = speedInit.x;

                //border collision
```

```
                if (supposedNextPos.y<-0.1)
                        speed.y = speedInit.y;
                if (supposedNextPos.y>1.1)
                        speed.y = speedInit.y;
```

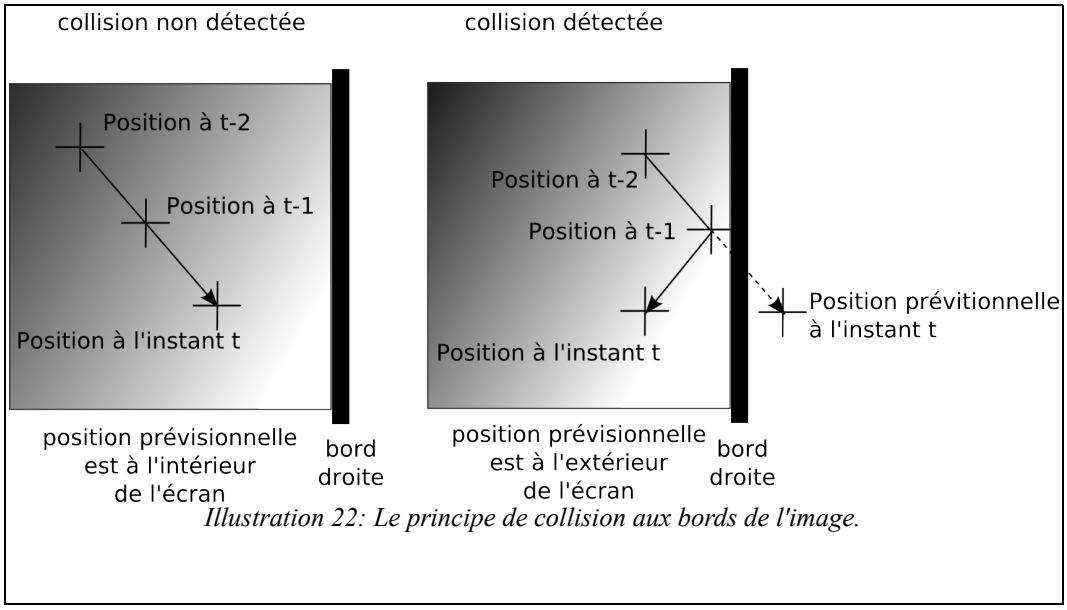


*Illustration 22: Le principe de collision aux bords de l'image.*

```
                float2 nextPos = previousValuesT0.xy + speed;
                return float4(nextPos,0.,1);
                }
        }
}
```

2. Mode gravitation

Le mode gravitation est moins complèxe que le mode d'attraction, du faite que les particules ne s'attire pas vers un axe, mais simplement vers le bas de l'image. Sinon la gestion du rebond avec l'objet en mouvements et le bord est identitique.

### *e) Filtre d'intensité lumineuse des particules en goutte*

**Nom texture** : «part».

**Fonction** : à partir de la position d'une particules, il faut affecter une texture pour avoir un visuel. Cette texture est sous forme de goutte d'eau semi transparente (voir illustration 25).

**Nom fichier** : «quads.xml ».

**Nom (s) fichier(s) shader(s) :** « quadsRendering-VP.cg » et « quadsRendering-FS.cg ».

**Paramètre (s) d'entrée (s)** : « cg_vp_particleSize » : définit la taille des particules.

**Texture (s) d'entrée (s)** :

- « goutte4.png » : la textures goutte semi-transparente
- « wordPos » : la texture de la position des particules.

**Détail :**

Pour l'instant, une particule est définie par un couple d'indice ligne/colonne et sa position dans un plan (x et y). Le but de ce filtre et de rendre visible les particules, leurs donner des propriétés

géométrique (taille, couleur, transparence, etc.).

Et pour créer l'objet géométrique rectangle, chaque particule nécessite 4 vertex (sommet). Ses vertex sont regroupés 4 par 4. L'ensemble des vertex sont défini dans un objet virtuel sous forme d'un maillage dans le fichier « quads.xml ». Ce fichier définit les propriétés de l'objet virtuel et fait appel au fichier «subdiContentQuad.xml » qui contient les données géométrique du maillage (vertex, face, normal, …).

L'extrait du fichier « subdiContentQuad.xml »

```
//les 4 vertex concernant la particule situant à la colonne 0 et à la ligne 0 de la texture de position
<vertex index="1" x="0" y=" 0" z="0" />
<vertex index="2" x="0" y=" 0" z="1" />
<vertex index="3" x="0" y=" 0" z="2" />
<vertex index="4" x="0" y=" 0" z="3" />
//les 4 vertex concernant la particule situant à la colonne 0 et à la ligne 1 de la texture de position
<vertex index="5" x="0" y=" 1" z="0" />
<vertex index="6" x="0" y=" 1" z="1" />
<vertex index="7" x="0" y=" 1" z="2" />
<vertex index="8" x="0" y=" 1" z="3" />
```

Les attributs « **x** » et « **y** » sont les indices qui désignent la ligne et la colonne qu'il faut lire dans la texture de position pour récupérer les coordonnées de la particule. Et l'attribut « **z** » permet d'indiquer les 3 coins du triangle (voir figure 23).

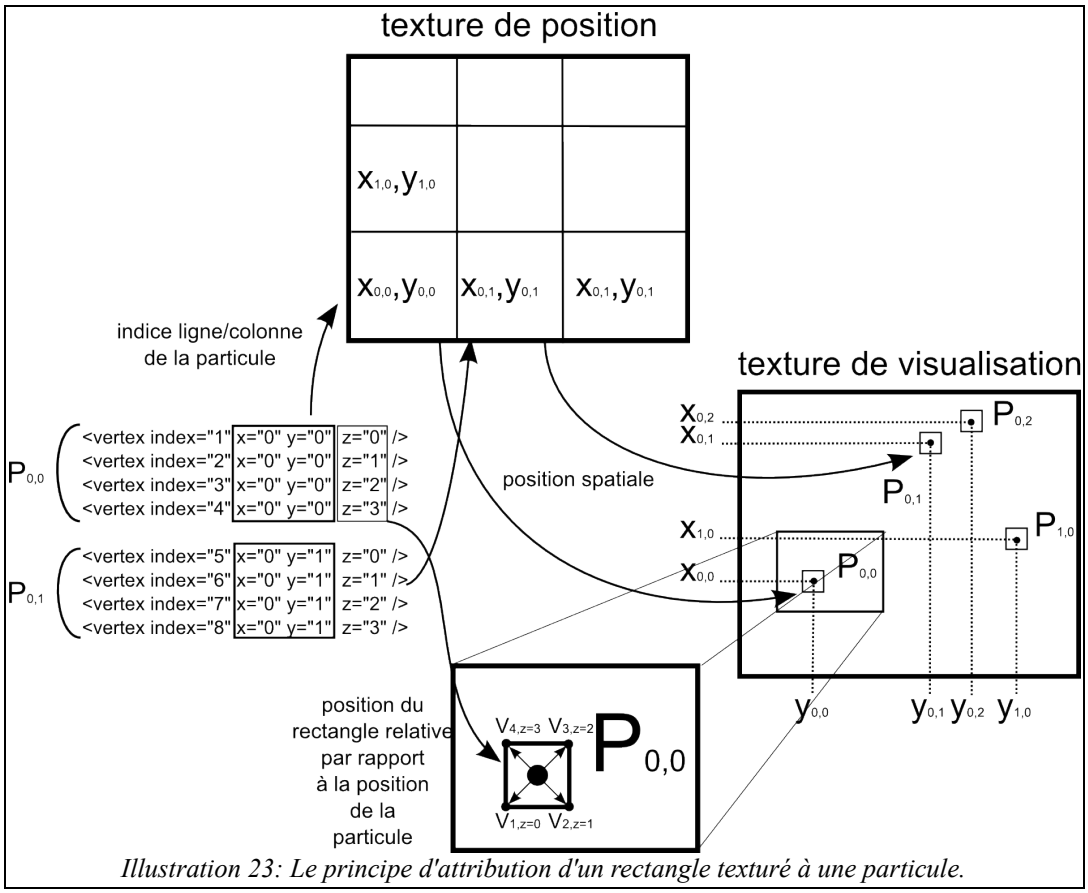


*Illustration 23: Le principe d'attribution d'un rectangle texturé à une particule.*

L'extrait du fichier « quadsRendering-VP.cg »

```
//lire la position des particules dans la texture de position
float4 screenLoc = texRECT(lookupTable1,position.xy+0.5);

float size = cg_vp_particleSize;
HPosition = float4( screenLoc.xy*2.0-1.0, 0.0,1.0);
//coordonnes de textures en fonction de l'index de position.x
//une texture de 4 mots sous formes horizontal
int wordIndex = mod (position.x,4.);
float4 texCoord;
```

L'attribution des valeurs des coordonnées de textures selon l'index du vertex.

```
if (wordIndex ==0){
        texCoord.xy= float2(0.,0.);
        texCoord.zw= float2(1.,0.25);
}else{
        if (wordIndex ==1){
                texCoord.xy= float2(0.,0.26);
                texCoord.zw= float2(1.,0.5);
        }else{
                if (wordIndex ==2){
                        texCoord.xy= float2(0.,0.51);
                        texCoord.zw= float2(1.,0.75);
                }else{
                        texCoord.xy= float2(0.,0.76);
                        texCoord.zw= float2(1.,1.);
                }
        }
}
```

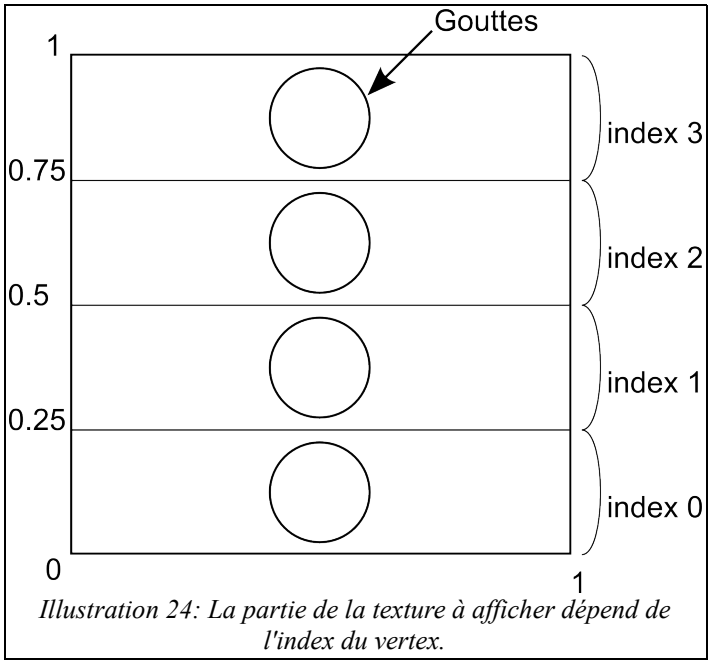


*Illustration 24: La partie de la texture à afficher dépend de l'index du vertex.*

En tant normal, un vertex a une position qui est constituée de 4 valeurs (xyzw qui sont des coordonnées homogène).

Mais dans le cadre du traitement d'image 2D, il y a, dans la plupart du temps, que l'utilisation des 2 premières données (x et y). Dans le même ordre d'idée, ce vertex est aussi définis par un coordonnées de texture de 4 valeurs (uvst), et dans la plupart du temps, il y a seulement l'utilisation des 2 premières (u et v), Et l'utilisation détournée de ses 4 valeurs permettent de définir 2 positions (la position en bas à gauche et la position en haut à droit) dont 4 coordonnées textures.

L'attribution du décalages de la position des vertex autour de la position de la particule selon l'index du vertex.

« Hposition.x » est la position du vertex sur l'axe X avant et après le décalage par « +/- w*size » selon l'index.

« Hposition.y » est la position du vertex sur l'axe Y avant et après le décalage par « +/- h*size » selon l'index.

« w » et « h » sont la longueur et la hauteur du quad (constitué de 2 triangles).

Enfin « size » et la taille à l'entrée du paramètre « cg_vp_particleSize » qui permet de contrôler la taille de manière temps-réel par l'utilsateur.

```
float w=0.05, h=0.02;
if (position.z< 2){//les 2 points en bas
	HPosition.y -= h*size;
	if (position.z==0){//bas gauche
		HPosition.x -= w*size;
		decalCoords = float4(texCoord.x,texCoord.y,HPosition.xy);
	}else{//bas droite
		HPosition.x += w*size;
		decalCoords = float4(texCoord.z,texCoord.y,HPosition.xy);
	}
}else{
//les 2 points en haut
	HPosition.y += h*size;
	if (position.z==3){//haut gauche
		HPosition.x -= w*size;
		decalCoords = float4(texCoord.x,texCoord.w,HPosition.xy);
	}else{//haut droite
		HPosition.x += w*size;
		decalCoords = float4(texCoord.z,texCoord.w,HPosition.xy);
	}
}
index = float3(size,position.x,screenLoc.z);
```

Les variables « decalCoords » et « index » font être récupèrer par le programme fragment shader « quadsRendering-FS.cg »

## L'extrait du fichier « quadsRendering-FS.cg »

```
decalCoords.x = saturate(decalCoords.x);
decalCoords.y = saturate(decalCoords.y);
float4 color = tex2D(decal,decalCoords.xy);
```

« decal » est la texture « goutte4.png », la couleur du pixel « color » est définie selon la zone définie par les coordonnées de texture précalculé dans le programme pixel shader.

```
return float4(color.rgb,color.a*(1.0-saturate(100*index.z)));
```

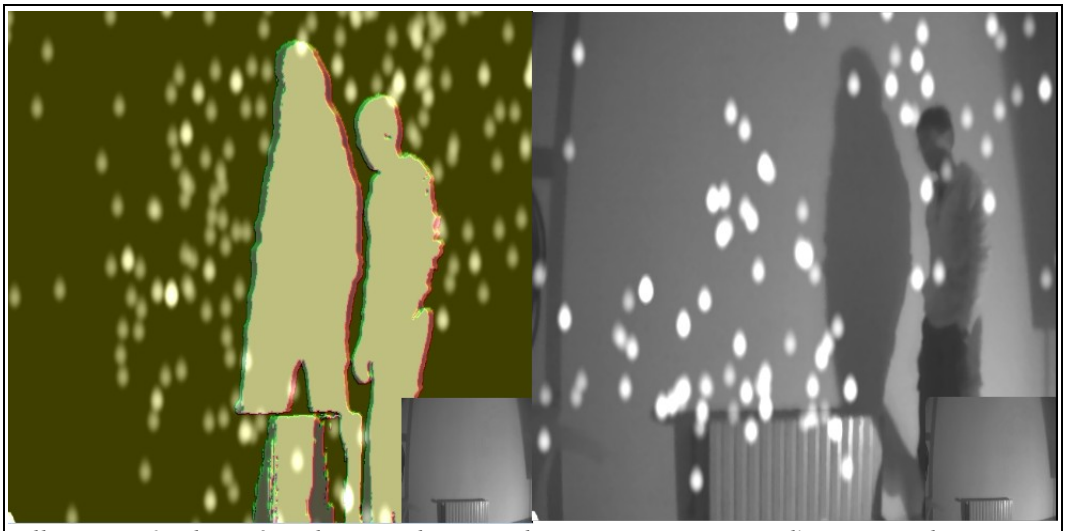

*Illustration 25: le système de particules en mode gravitation et goutte : l'image avec les vecteur normal en couleur à gauche, le rendu visuel à droite.*

### *f) Filtre d'intensité lumineuse des particules en mots*

**Nom texture** : « wordPart ».

**Fonction** : à partir de la position d'une particules, il faut affecter une texture pour avoir un visuel. Cette texture est sous forme de mots (voir illustration 26).

**Nom fichier** : « wordQuads.xml »

**Nom (s) fichier(s) shader(s) :** « wordQuadsRendering-VP.cg » et « wordQuadsRendering-FS.cg ».

**Paramètre (s) d'entrée (s)** : « cg_vp_particleSize » : définit la taille des particules.

**Texture (s) d'entrée (s)** :

- « wordTexture.png » : texture contenant 4 mots.
- « wordPosM1 » : texture de position des particules.

**Détail :**

De même que « goutte4.png », « wordPosM1 » est subdivisée en 4 partie. Cette passe est très identique au « Filtre d'intensité lumineuse des particules en goutte ».

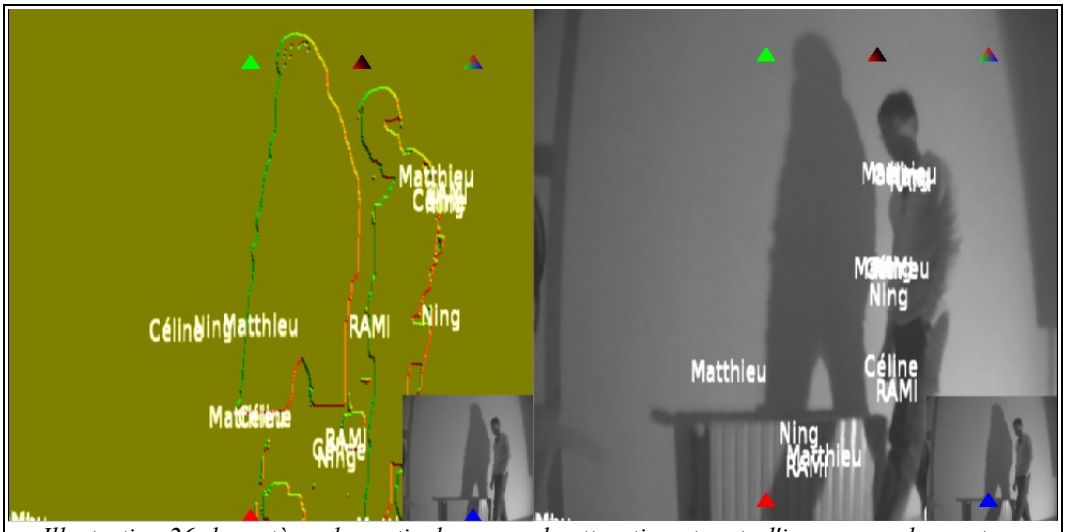


*Illustration 26: le système de particules en mode attraction et mot : l'image avec les vecteur normal en couleur à gauche, le rendu visuel à droite.*

### *g) Mode de vision du rendu*

**Nom texture** : «FrameBuffer_2particles_filter».

**Fonction** :

**Nom fichier** : «filter_2particles.xml »

**Nom (s) fichier(s) shader(s) : «** TLLSLO-VP-final.cg » et « TLLSLO-FS-2particles.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_partmode » : 0 pour le mode attraction avec les mots et 1 pour le mode gravitation avec les gouttes.

**Texture (s) d'entrée (s)** :

- « wordPart » : La texture de rendu « mot ».
- « part » : La texture de rendu « goutte ».

**Détail :**

Le changement de mode de déplacement est d'affichage interviennent à plusieurs endroits. Contrôler par une application tierce (Puredata, Max/MSP … via message OSC/UDP), L'ensemble des actions à réaliser lorsqu'on changement de mode est définis dans le fichier « **AliasScript.xml** » qui contient les parseurs pour message OSC/UDP.

Le message reçu est composé d'une entête nommée « /partmode » et suivi d'un chiffre (float). Ce changement intervient à 2 endroits. Au nœud « wordPosComputing » pour le calcul de la positon des particules et « subdivision_2particles » pour le choix du mode d'affichage

L'extrait du fichier « AliasScript.xml »

```
<command pattern = "/partmode f">
	<action>
		<set_material_attribute_value operator = "=">
			<shader_source cg_fs_partmode = "({$1})" />
		</set_material_attribute_value>
		<target type = "single_node" value = "#wordPosComputing"/>
	</action>
	<action>
		<set_material_attribute_value operator = "=">
			<shader_source cg_fs_partmode = "({$1})" />
		</set_material_attribute_value>
		<target type = "single_node" value = "#subdivision_2particles"/>
	</action>
</command>
```

# 8. Module de création d'artistique

Ce module n'a pas un but unique, à savoir donner un seul résultat escompté. Mais il contient un ensemble d'effets artistiques, de tests, et des expérimentations diverses et variés.

## *a) Compensation colorimétrique*

**Nom texture** : « FrameBuffer_colorCorrection_filter ».

**Fonction** : colorier l'image binaire (en noir et blanc) par 2 couleurs (voir illustrations 27 et 28).

**Nom fichier** : « filter_colorCorrection.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-colorCorrection.cg ».

**Paramètre (s) d'entrée (s)** :

-« cg_fs_switch » : 0 conservation des couleur au zone prédéfinie, 1 inverser les couleurs.

-« cg_fs_4fv_color1 » : couleur remplissant la zone de l'objet en mouvement (ombre, corps en mouvements ou silhouette).

-« cg_fs_4fv_color2 » : couleur remplissant la zone statique (ombre, corps en mouvements ou silhouette).

**Texture (s) d'entrée (s)** : « FrameBuffer_ROADBI_filter » est la texture de l'image courante et « FrameBuffer_dOmbre_filter » est la texture binaire (noir et blanc) résultante de la détection de l'ombre.

**Détail :**

Cet effet est plus considérer comme une expérimentation qu'un effet artistique. Il est utilisé pour annuler l'ombre. Lorsque le performeur est dans l'espace de jeu, si le projecteur n'est pas équipé d'une filtre infrarouge, alors une ombre réel est créé. Le but est de définir une couleur qui colorierait la zone d'ombre pour être à la même couleur que le fond statique qui est éclairé par le projecteur.

Dans le cadre de l'expérimentation à LIMSI, c'est une mandarine qui projette une lumière jaune, La couleur de compensation est dans l'ordre du saumon (rgba : 0.787234,0.8968,1,1.0). Cette correction doit prendre en compte beaucoup de paramètre: la lumière du vidéoprojecteur, lumière naturel si il y a, réflexion de la surface de projection.

Nous remarquons que cette correction colorimétrique varie si c'est vue à l'œil nu, ou pré enregistré par une caméra.

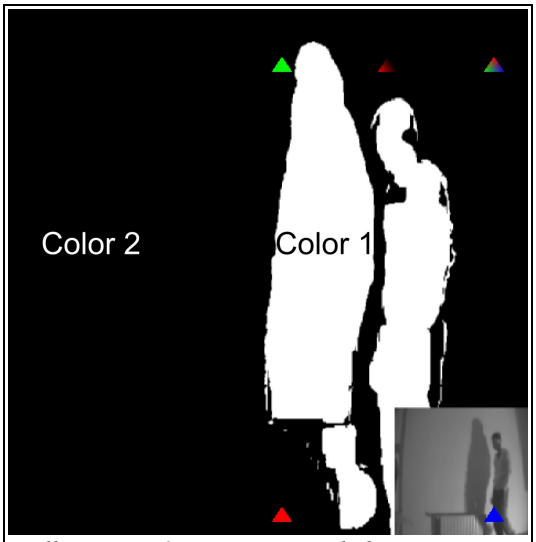


*Illustration 27: Le principe de la correction colorimétrique.*

*Illustration 28: Application sur l'annulation d'ombre.*

### *b) Effet flash*

**Nom texture** : « FrameBuffer_flash_filter ».

**Fonction** : faire apparaître des sprites texturées sous forme de flash en fonction d'une position (x et y) fournie pour un logicielle tierce (Max/MSP).

**Nom fichier** : « filter_flash.xml »

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-flash.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_bypass » : 1 pour désactiver l'effet, 0 pour l'activer.

- « cg_fs_4fv_pos » : Les 2 premières est la position « x » et « y ». La 3° est la variable déclencheur « z », s'il est à 1 alors on affiche le flash sinon rien. Et la dernière « w » est la taille du flash.

**Texture (s) d'entrée (s) :**

- « FrameBuffer_ROADBI_filter » : l'image courante qui se sert de fond à l'effet

- « FrameBuffer_flash_filter » : l'image flash d'instant précédente, pour créer l'effet d'écho (effacement progressif du flash).

- « star2.png » : La texture flash à afficher.

**Détail :**

L'extrait du fichier « TLLSLO-FS-flash.cg »

```
 float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
float4 resultColor ;
if (cg_fs_bypass == 1.0){
        return texRECT(MaVideo,CoordsTmp).rgba;
```

Renvoie l'image courante si l'effet désactivé.

```
}else{
        if (cg_fs_4fv_pos.z == 1.){
```

Si le flash est déclenché.

```
                float2 pos =cg_fs_4fv_pos.xy-0.5;
                float size = cg_fs_4fv_pos.w;
```

Récupération des valeurs positions (et translation du repère au centre de la texture flash) et la valeur taille.

```
                float2 coordTmp;
                coordTmp =(decalCoords.xy-pos-0.5)/size;
                if (coordTmp.x>-0.5 && coordTmp.x<0.5 && coordTmp.y>-0.5 && coordTmp.y<0.5){
                        float4 rond = tex2D(rd,coordTmp+0.5).rgba;
                        return rond* (rond.a) + texRECT(imgM1,CoordsTmp).rgba * (1 -rond.a);
```

« coordTmp » contient les coordonnées de texture flash en fonction de la valeur « size », et « CoordsTmp » contient les coordonnées de l'image courante. Le mé

```
                }else{
                        resultColor = texRECT(imgM1,CoordsTmp).rgba;
                        return resultColor;
```

Si le pixel n'est pas dans la zone où il y a la texture flash, alors on renvoie simple l'intensité de l'image courante.

```
                }
        }else{
                resultColor = texRECT(imgM1,CoordsTmp).rgba;
                return resultColor * 0.95;
```

s'il y a pas de déclenchement flash, il y aura juste renvoie de l'mage.

```
        }
}
```

### *c) Contraste et lluminosité*

**Nom texture** : « FrameBuffer_contrast_filter ».

**Fonction** : correction de la contraste et de la luminosité à titre expérimental (pas d'influence sur le reste du système).

**Nom fichier** : « filter_contrast.xml »

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-contrast.cg »

**Paramètre (s) d'entrée (s)** : « cg_fs_2fv_conLum » est un tableau de 2 variables, la première pour la correction contraste (varie de 0 à 128, 0 l'image ne subi pas de changement, 1 l'image devient binaire). Et la seconde permet de régler la luminosité.

**Texture (s) d'entrée (s)** : « FrameBuffer_ROADBI_filter ».

**Détail :**

Le filtre de contraste est un filtre de correction lumineuse qui diminue l'écart de luminosité entre les hautes et les basses lumières en entrée Il redistribue cet intervalle réduit sur l'ensemble du spectre des niveaux de gris augmentant ainsi le contraste des images dont les niveaux de gris se situent principalement dans les gris moyens. La figure 10 montre l'image vidéo traitée selon les différents niveaux de contraste.

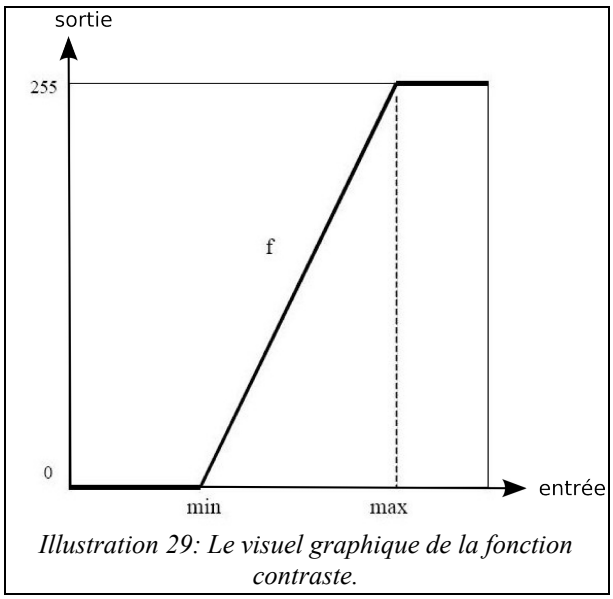

*Illustration 29: Le visuel graphique de la fonction contraste.*

Le graphe de la fonction de correction lumineuse pour le contraste (voir figure 11) : si le modificateur est à 0, le min des luminosités en entrée est à 0 et le max à 255. S'il est à 64, alors le min est à 64 et le max à 191. Enfin s'il est à 127, le min et le max sont à 127. Dans ce dernier cas, on obtient une fonction escalier qui retourne comme valeur soit 0, soit 255.

La luminosité est l'ajout ou le retranchement d'une valeur constante dans l'intensité lumineuse. Ce paramètre de variation permet d'assombrir et éclaircir l'image (voir figure 12).

La formule de modification de luminosité :

*Intensité obtenue = Intensité initiale + Modification de luminosité.*

### *d) Bascule entre l'image d'ombre et l'image courante*

**Nom texture** : « FrameBuffer_interpolROADBIDOmbre_filter ».

**Fonction** : à but artistique, ce filtre permet de transiter de manière douce l'image courante et l'image binaire résultante de la détection de l'objet en mouvement.

**Nom fichier** : « filter_interpolROADBIDOmbre.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « shaders/TLLSLO-FS-interpolROADBIDOmbre.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_coef »: varie entre 0 et 1

**Texture (s) d'entrée (s)** : « FrameBuffer_ROADBI_filter » l'image courante et « FrameBuffer_dOmbre_filter » l'image résultante de la détection d'objet en mouvements.

**Détail :**

L'extrait du fichier « shaders/TLLSLO-FS-interpolROADBIDOmbre.cg ».

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
 float4 texture1 = texRECT( decal , texCoord ).rgba;
float4 texture2 = texRECT( lookupTable1 ,texCoord).rgba;
return lerp (texture1,texture2,cg_fs_coef );
```

« lerp » est une fonction prédéfinie (veuillez visiter ce site pour le reste de fonction de CG par défaut :http://http.developer.nvidia.com/CgTutorial/cg_tutorial_appendix_e.html).

| | |
|---|---|
| | Linear interpolation: |

| | |
|---|---|
| `lerp(a,b,f)` | `(1 - f)*a + b*f`<br>where ***a*** and ***b*** are matching vector or scalar types. ***f*** can be either a scalar or a vector of the same type as ***a*** and ***b*** . |

### *e) Plasma*

**Nom texture** : « FrameBuffer_plasma_filter ».

**Fonction** : création d'un effet onirique et de légèreté (voir illustration 30).

**Nom fichier** : « filter_plasma.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-plasma.cg ».

**Paramètre (s) d'entrée (s) :** « cg_fs_start » : la variable temps pour l'angle des rotations de l'effet.

**Texture (s) d'entrée (s) :**

-« FrameBuffer_interpolROADBIDOmbre_filter » :

-« plasma.png » : la texture plasma permet de d'une part généré des valeurs aléatoires, mais aussi des couleurs aléatoires.

-« FrameBuffer_plasma_filter » : la texture plasma à l'instant d'avant.

**Détail :**

Le filtre Plasma réalise une colorisation arbitraire de l'image initiale à partir de l'image de l'acquisition. Le filtre Plasma contient trois effets :

1. un effet d'écho (ou de traînée) qui mélange entre l'image actuelle et l'image précédente,
2. un jeu de colorisation à partir de la texture « plasma.png »,
3. un jeu de déformation avec un déplacement des coordonnées de textures.

L'effet artistique dégage un sentiment onirique, vivant et léger en raison de la fluidité visuelle

Dans le but de colorier l'image, nous allons traiter les canaux rouge, vert et bleu séparément. Comme leurs implémentations sont similaires, seul le canal rouge est expliqué (voir équation 3). Implémentation shader du filtre plasma du fichier « FES-FS-texPros_MATT.cg » :

L'extrait du fichier « TLLSLO-FS-plasma.cg »

Le calcul du canal de couleur rouge se réalise par la lecture de la texture « texM1 » avec un changement de coordonnées de texture pour créer cet effet de vague et de déformation. Les déplacements de coordonnées de textures se réalise avec une variable temporaire « cg_fs_start » et des fonctions trigonométriques dans le but de créer cet effet de tourbillon

```
float R =
        texRECT(texM1, (decalCoords.xy+0.01*(tex2D(Plasma,decalCoords+float2( 0,
0)+float2(cos(cg_fs_start*0.51),sin(cg_fs_start*0.04+5.5))).xy*2.0-1.0))*cg_fs_2fv_size).r*0.2+
        texRECT(texM1, (decalCoords.xy+0.01*(tex2D(Plasma,decalCoords+float2( 0,
5)+float2(cos(cg_fs_start*0.51),sin(cg_fs_start*0.04+5.5))).xy*2.0-1.0))*cg_fs_2fv_size).r*0.2+
        texRECT(texM1, (decalCoords.xy+0.01*(tex2D(Plasma,decalCoords+float2( 0,-
5)+float2(cos(cg_fs_start*0.51),sin(cg_fs_start*0.04+5.5))).xy*2.0-1.0))*cg_fs_2fv_size).r*0.2+
        texRECT(texM1, (decalCoords.xy+0.01*(tex2D(Plasma,decalCoords+float2( 5,
0)+float2(cos(cg_fs_start*0.51),sin(cg_fs_start*0.04+5.5))).xy*2.0-1.0))*cg_fs_2fv_size).r*0.2+
        texRECT(texM1, (decalCoords.xy+0.01*(tex2D(Plasma,decalCoords+float2(-5,
0)+float2(cos(cg_fs_start*0.51),sin(cg_fs_start*0.04+5.5))).xy*2.0-1.0))*cg_fs_2fv_size).r*0.2;
```

Une fois que les valeurs des trois canaux (rouge, vert et bleu) d'intensité de couleur du pixel local sont

calculées (seul le canal rouge est expliqué en détail), elles sont mélangés avec les trois canaux de couleur de l'image à l'instant courant.

```
color.rgb =  texRECT(MaVideo, (decalCoords.xy*cg_fs_2fv_size)).bgr*0.2 + float3(R,G,B)*0.79 ;
```

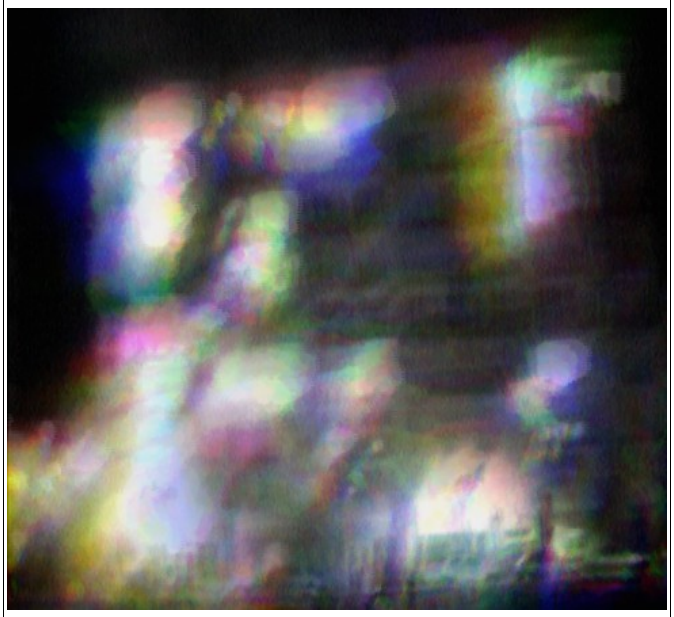

*Illustration 30: L'effet Plasma.*

## *f) Echo*

**Nom texture** : « FrameBuffer_echo_filter »

**Fonction** : création d'un effet écho avec l'image d'instant précédente (voir illustration 31).

**Nom fichier** : « filter_echo.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-echo.cg ».

**Paramètre (s) d'entrée (s)** : aucun

**Texture (s) d'entrée (s)** :

-« FrameBuffer_dOmbre_filter » : l'image résultante de la détection d'ombre.

-« FrameBuffer_echo_filter » : est la texture résultante de l'instant précédente.

**Détail :**

L'extrait du fichier « TLLSLO-FS-echo.cg »

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
float4 textureIn  = texRECT( texIN , texCoord ).rgba;
float4 testureTM1 = texRECT( texTM1 , texCoord ).rgba;
float4 color= float4(0.0,0.0,0.0,1.0);
color.rgb = textureIn.rgb + testureTM1.rgb * 0.95 ;
return color ;
```

Le résultat renvoyé est la superposition des 2 images, avec une atténuation à l'image d'instant précédente.

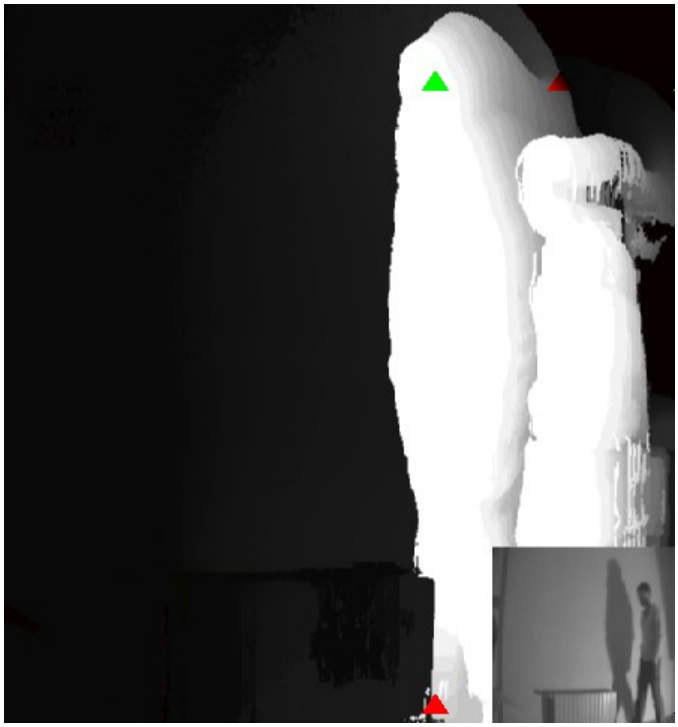

*Illustration 31: Effet Echo.*

### *g) Masque*

**Nom texture** : « FrameBuffer_mask_filter »

**Fonction** : Très similaire à la correction colorimétrique « Compensation colorimétrique » (voir illustration 32).

**Nom fichier** : « filter_mask.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-mask.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_inv » : par défaut, la texture « plasma.png » est appliquée à la zone blanche (l'objet en mouvement), parcontre, si elle est à 1 alors c'est le fond noire qui est remplacée par la texture.

**Texture (s) d'entrée (s)** :

-« FrameBuffer_echo_filter » : l'image résultante de la passe « echo ».

-« plasma.png » : texture colorié à appliquer sur le masque

**Détail :**

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
 float4 textureIn  = texRECT( texIN , texCoord ).rgba;
textureIn.rgb = textureIn.rgb * cg_fs_inv + (1-textureIn.rgb) * (1-cg_fs_inv);
```

Activer le négatif de l'image binaire est en fonction de la variable «  cg_fs_inv ».

```
float4 wallpaper  = tex2D( wall , decalCoords.xy ).rgba;
```

Récupération de la couleur de la texture plasma.

```
float4 masque = float4(textureIn.rgb, textureIn.a) * wallpaper.rgba;
```

Le masquage est traduit par une multiplication, seul les pixels blancs du masque prennent la couleur du plasma.

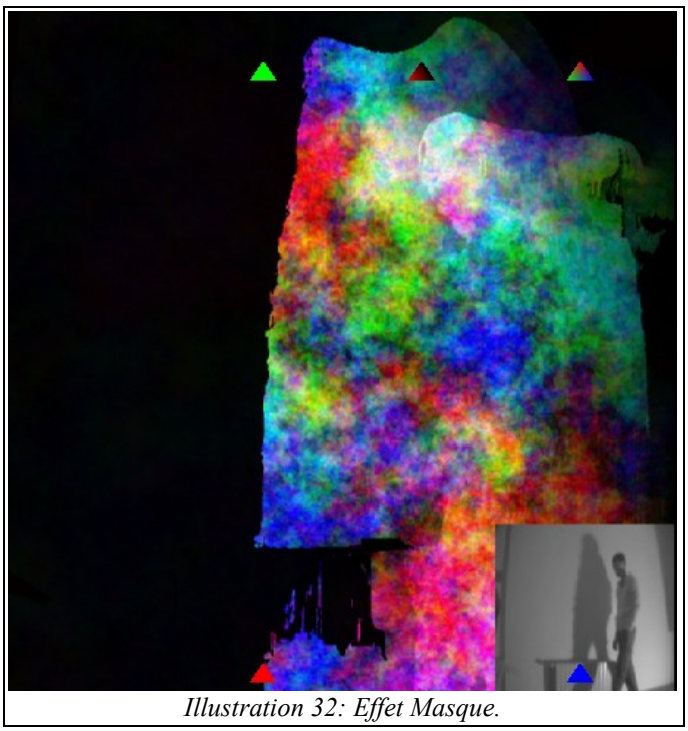

*Illustration 32: Effet Masque.*

### *h) Duplication*

**Nom texture** : « FrameBuffer_multiply_filter ».

**Fonction** : figer les éléments en mouvements.

**Nom fichier** : « filter_multiply.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-multiply.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_bypass » : 1 pour désactiver l'effet, 0 pour l'activer l'effet.
- « cg_fs_take » : figer l'objet en mouvement (en blanc).
- « cg_fs_clean » : effacer tout les l'objet en mouvement figé à un moment donné.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_dOmbre_filter » : l'image binaire résultante de la détection de l'objet en mouvement.
- « FrameBuffer_multiply_filter » : l'image résultante à l'instant précédente.

**Détail :**

L'extrait du fichier « TLLSLO-FS-multiply.cg »

```
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
float4 imageColor = texRECT(MaVideo,CoordsTmp).rgba;
float4 imageM1 =texRECT(imgM1,CoordsTmp).rgba;
if (cg_fs_bypass == 1.0){
        return  imageColor;
```

Renvoie l'image courante si « cg_fs_bypass » est à 1.

```
}else{
        if (cg_fs_clean>=0.85){
```

```
		return float4(0.,0.,0.,1.);
```

Nettoyage de l'image si « cg_fs_clean » est déclanché. Une système dans VirChor permet de tendre « cg_fs_clean » de 1 à 0 avec une multiplication de 0,95 à chaque frame. Du coup la plupart du temps « cg_fs_clean » est inférieur à 0,85. Sauf si l'utilisation remet « cg_fs_clean » à 1, dans ce cas là, « cg_fs_clean » mettra quelque instant pour être à nouveau inférieur à 0,85. C'est à ces moments là qu'on réinitialise l'image en noir.

```
	}
	if (cg_fs_take >=0.85){
		return imageColor + imageM1;
```

De la même manière que « cg_fs_clean » pour tendre de 1 à 0, « cg_fs_take » permet d'ajouter la nouvelle image sur celle qui est figée.

```
	}else{
		return imageM1;
```

Si rien se passe, alors on renvoie l'image figée.

```
	}
}
```

### *i) Halo*

**Nom texture** : « FrameBuffer_halo_filter ».

**Fonction** : création de l'effet halo voir illustration 33).

**Nom fichier** : « filter_halo.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-halo.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_halocoef ».

**Texture (s) d'entrée (s)** :

-« FrameBuffer_ssFondSobel_filter » : l'image résultante du système du soustraction de fond entre l'image contour courante, et l'image contour fond.

- « FrameBuffer_halo_filter » : l'image résultante de l'instant précédente.

**Détail :**

L'extrait du fichier « TLLSLO-FS-halo.cg »

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
float4 textureIn  = texRECT( decal , texCoord ).rgba;
float4 textM1  = texRECT( lookupTable1 , texCoord ).rgba;
```

Récupération de la valeur du pixel des 2 images «  decal » pour « FrameBuffer_ssFondSobel_filter » et « lookupTable1 » pour « FrameBuffer_halo_filter ».

```
float coef = 3,7;
float2 dir[8] = {{-coef,-coef},{0.,-coef},{coef,-coef},
{-coef,0.},{coef,0.},
{-coef,coef},{0,coef},{coef,coef}};
float w[8] = {0.1,0.15,0.1,0.15,0.15,0.1,0.15,0.1};

float4 tmp = float4 (0.,0.,0.,1.);
for (int i= 0; i <8 ; i ++){
	tmp.rgb += texRECT( lookupTable1 , texCoord - dir[i] ).rgb * w[i];
}
```

Pour donner l'effet de la propagation des ondes halo, il faut faire déplacer l'intensité lumineuse. Donc pour un pixel noir

donnée, il faut qu'il aille cherche autour de lui s'il y a des voisins donc l'intensité lumineuse n'est pas noir. C'est pour cela qu'il cherche vers les 8 directions autour de lui selon une distance de « coef ».

```
return textureIn * 0.8  + tmp *cg_fs_halocoef;
```

L'atténuation de l'image précédente est en fonction de « cg_fs_halocoef ».

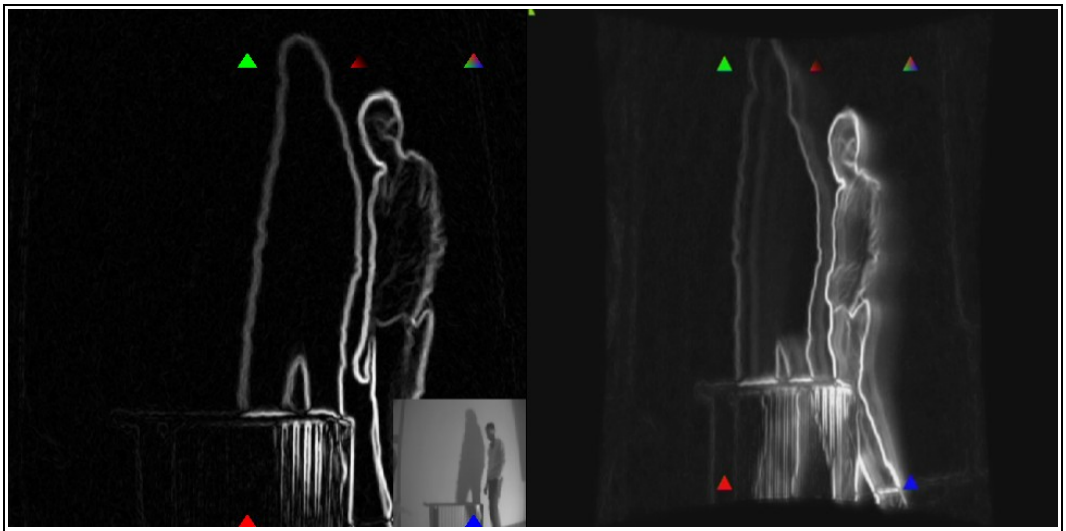

*Illustration 33: A gauche l'image de la soustraction de fond contour, et à droite l'image d'effet Halo.*

### *j) Aspirate*

Nom texture : « translation_aspirate_filter ».

**Fonction** : crée un effet de pincé et qui aspire vers le centre.

**Nom fichier** : « filter_aspirate.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-aspirate.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_aspirate ».

**Texture (s) d'entrée (s)** : « FrameBuffer_ROADBI_filter » l'image courante.

**Détail :**

L'implémentation est similaire à l'opération de calibrage « Correction radiale », d'abord il changement de repère pour que l'origine se trouve au centre de l'image au lieu d'être en bas à gauche. Puis il a passage du coordonnées cartésiens (x et y) en coordonnées polaires (angle et rayon).

Puis il y a l'application d'une fonction linéaire pour définir le nouveau rayon à partir de l'ancienne : Dans ce cas là c'est la fonction « pow ». Puis reconversion des coordonnées polaires en cartésiennes. Et récupération de l'intensité du pixel à partir des nouvelles coordonnées calculées.

### *k) Vibration*

**Nom texture** : « FrameBuffer_vibration_filter ».

**Fonction** : lors qu'on secoue la Wiimote, l'image réagit avec ses secouses.

**Nom fichier** : « filter_vibration.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-vibration.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_bypass » : 1 pour désactiver l'effet.

- « cg_fs_4fv_vibration » : Les 2 premières variables sont les décalages de l'image sur l'axe X et Y. La 3° variable z permet rétrécir ou agrandir l'image (simulation du profond). La 4° variable est l'angle de la rotation. Ces quatre valeurs sont fournis par l'accéléromètre (pour les 3 premières valeurs) et le gyroscope (l'angle de la rotation) de la Wiimote (voir « manuel d'utilisateur »).

**Texture (s) d'entrée (s)** : « FrameBuffer_ROADBI_filter » l'image courante et « noise.jpg » : la texture bruits utilisé lors des tests.

**Détail :**

L'extrait du fichier « TLLSLO-FS-vibration.cg »

```
float angle =-cg_fs_4fv_vibration.w* 3.1416 / 180.0;
float2x2 rot = {{cos(angle),sin(angle)},
{-sin(angle),cos(angle)}};
```

La matrice de rotation.

```
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
CoordsTmp-=cg_fs_2fv_size/2.;

CoordsTmp+=float2((cg_fs_4fv_vibration.xy-127.) *cg_fs_2fv_size / float2(255.) );
```

Transformation de translation en fonction de « cg_fs_4fv_vibration.xy » et de « cg_fs_4fv_vibration.y ».

```
CoordsTmp = mul(rot,(CoordsTmp));
CoordsTmp*=1+cg_fs_4fv_vibration.z/150.;
```

Transformation de rétrécissement ou agrandissement en fonction de « cg_fs_4fv_vibration.z ».

```

CoordsTmp+=cg_fs_2fv_size/2.;
float4 imageColor = texRECT(MaVideo,CoordsTmp).rgba;
if ((CoordsTmp.x > 0 && CoordsTmp.x <cg_fs_2fv_size.x)   &&(CoordsTmp.y > 0 && CoordsTmp.y
<cg_fs_2fv_size.y) ){
```

Les condition pour voir si le pixel local appartient ou pas l'image transformée ou pas. Sinon renvoie une couleur statique.

```
        return imageColor ;
}else{
        return float4(0.5,0.2,0.1,1.) ;
}
```

### *l) Fire*

**Nom texture** : « FrameBuffer_fire_filter ».

**Fonction** : effet artistique créé à partir des textures préexistantes pour donner une sensation de feu (voir illustration 35).

**Nom fichier** : « filter_fire.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-fire.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_bypass » : 1 pour désactiver l'effet et renvoie l'image de la « FrameBuffer_normalMapBlur_filter ».
- « cg_fs_4fv_poly » : inutil.
- « cg_fs_time » : le variable temporaire.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_normalMapBlur_filter » : la texture contenant le vecteur normal des pixels contour.
- « FrameBuffer_fire_filter » : l'image résultante de l'instant précédente.
- « noise.jpg » : la texture de bruits pour les valeurs aléatoires.
- « flame.jpg » : la textures pour donner à l'effet une teinte jaune/orange/rouge avec un bruit grain.

**Détail :**

L'extrait du fichier « TLLSLO-FS-fire.cg »

```
float2 CoordsTmp=decalCoords.xy*cg_fs_2fv_size;
float4 normal = texRECT(MaVideo,CoordsTmp).rgba;
float4 imageM1 = texRECT(imgM1,CoordsTmp).rgba;

float modulo = mod (cg_fs_time,9.);
float3 noiseValue = tex2D(noise,decalCoords+(cg_fs_time/9.)).rgb;
```

Récupération de 3 valeurs aléatoires à partir de la texture « noise.png ».

```
float2 dir[9] = {{0,-1},{0,-2},{-1,-1},{1,-1},{-2,-1},{2,-1},{-1,-3},{1,-3},{0,-5}};
float coef[9] = {0.25, 0.15, 0.15, 0.15, 0.1, 0.1, 0.025, 0.025, 0.04};
if (cg_fs_bypass == 1.0){
	return normal;
}else{
	normal = (normal-0.5)*2.;
```

Le changement de repère a pour but de retrouver le vecteur normal à -1 à 1.

```
	float3 flame = float3(normal.g) ;
```

C'est un masquage qui permet de travailler que sur les pixels contours verticales, où le vecteur normal a une direction vers le bas ou vers le haut.

```
	for(int i ; i<9 ; i++){
		flame += texRECT(imgM1,CoordsTmp+
		dir[i]*noiseValue.rg
		).rgb * coef[i];
	}
```

« CoordsTmp » est un tableau contenant les coordonnées de textures de pixel local, grâce au tableau « dir », qui permet d'aller chercher les valeurs de ses voisins de manière arbitraire (dans notre cas vers le bas pour ramener les valeurs d'intensité lumineuse non nul vers le haut). De plus, il y a un facteur aléatoire grâce au couple de valeur « noiseValue ».

Et « coef » affect à chaque pixel voisins un poids, une importance (et bien sure, en tant normal, plus le pixel voisin est éloigné, plus son poids est faible).

```
	return float4 (flame.r,flame.r*noiseValue.r*0.5,0,1.);
}
```

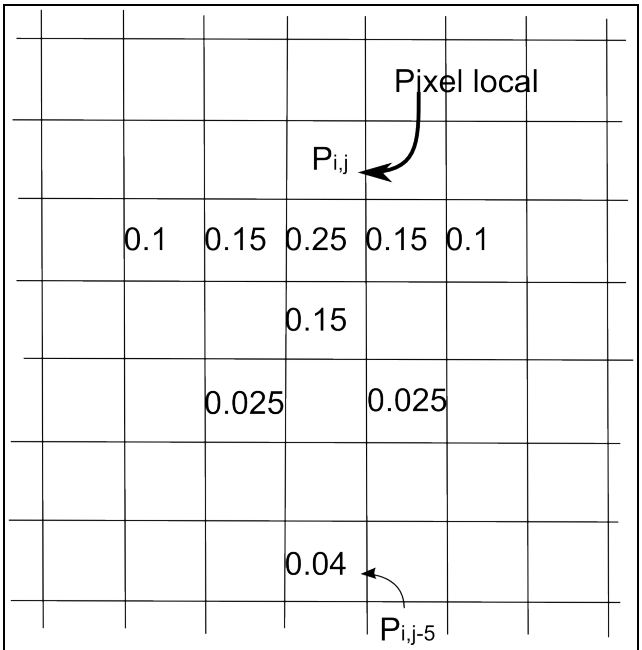


*Illustration 34: grillage de pixel avec les poids affectés aux pixels voisins.*

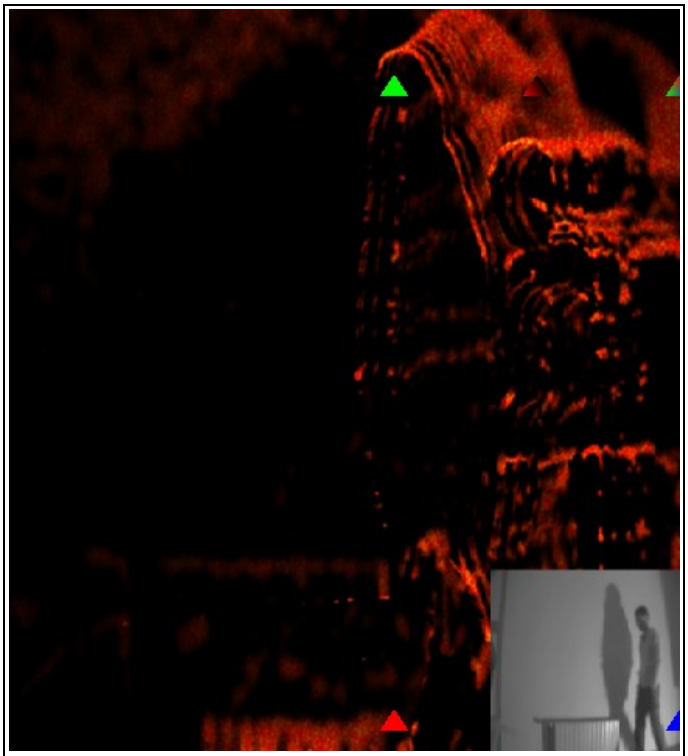

*Illustration 35: Effet Feu.*

### *m) Fire Element*

**Nom texture** : « FrameBuffer_fireElement_filter ».

**Fonction** : Un regroupement de différents petits pour en créer un qui a pour thème le feu, basé sur l'effet « fire » et l'effet « plasma » (voir illustration 36).

**Nom fichier** : « filter_fireElement.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-fireElement.cg ».

**Paramètre (s) d'entrée (s)** : aucun.
**Texture (s) d'entrée (s)** :
-« FrameBuffer_plasma_filter » :
- « FrameBuffer_fire_filter » :
- « FrameBuffer_ssFondSobel_filter » :
- « flame2.jpg » : la textures de dégradé noir au blanc passant par du rouge, orange et jaune.
Détail :
L'extrait du fichier « TLLSLO-FS-fireElement.cg »

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;

float4 plasma = texRECT( decal , texCoord ).rgba;
plasma.r = plasma.r * 0.3 + plasma.g * 0.59 + 0.11 * plasma.b;
```

Convertir l'image colorier en dégradé de gris.

```
float3 plasmaColored = tex2D( lookupTable3 , float2(saturate(plasma.x+0.005),0.1) ).rgb;
```

Récupérer la couleur flamme grâce à la texture « flame2.jpg » (« lookupTable3 ») selon la valeur « plasma.x ».

```
float4 fire = texRECT( lookupTable1 , texCoord ).rgba;
```

Récupérer ma couleur du pixel contenant dans l'effet « fire ».

```
float4 sobel = texRECT( lookupTable2 , texCoord ).rgba;
float3 sobelColored = tex2D( lookupTable3 , float2(clamp(sobel.x,0.05,1.),0.1) ).rgb;
```

Récupérer la couleur flamme grâce à la texture « flame2.jpg » (« lookupTable3 ») selon la valeur «sobel .x ».

```
return float4(plasmaColored*0.6 + fire.rgb * 0.95 + sobelColored ,1.);
```

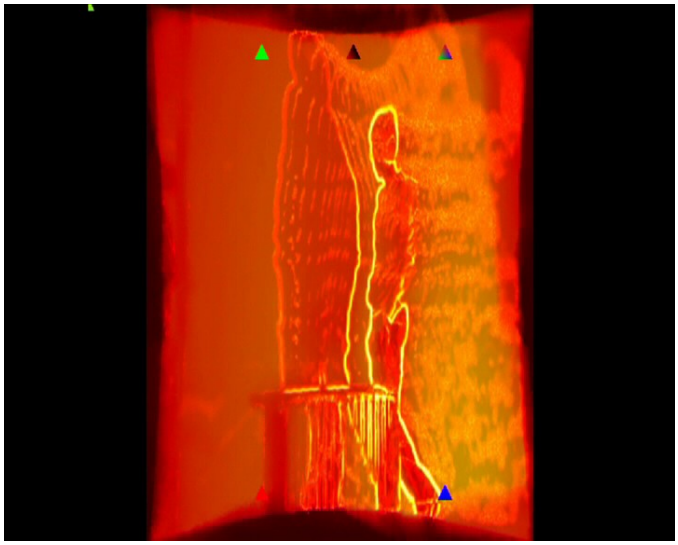

*Illustration 36: Effet élement Feu.*

## *n) Water Element*

**Nom texture** : « FrameBuffer_waterElement_filter ».
**Fonction** : Un regroupement de différents petits pour en créer un qui a pour thème l'eau, basé sur le

moteur de rendu des particules (voir illustration 37).

**Nom fichier** : « filter_waterElement.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-waterElement.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_partDistorsion » : le coefficient de déformation des coordonnées de textures en fonction de l'intensité lumineuse de la texture particules.
- « cg_fs_partColor » : 0 les particules sont invisible, 1 apparition des particules en dégradé de blanc.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_interpolROADBIDOmbre_filter » : texture qui bascule entre l'image courante et l'image binaire résultante de la détection d'objet en mouvment.
- « FrameBuffer_2particles_filter » : texture en intensité lumineuse résultante du rendu du moteur de particules.
- « water.jpg » : la textures de dégradé noir au blanc passant par du bleu foncé, bleu et cyan.

**Détail :**

L'extrait du fichier « TLLSLO-FS-waterElement.cg ».

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;

float4 particle = texRECT(lookupTable1,texCoord.xy).rgba;
float particle1 = texRECT(lookupTable1,float2(texCoord.x-1,texCoord.y)).r;
float particle2 = texRECT(lookupTable1,float2(texCoord.x+1,texCoord.y)).r;
float particle3 = texRECT(lookupTable1,float2(texCoord.x,texCoord.y-1)).r;
float particle4 = texRECT(lookupTable1,float2(texCoord.x,texCoord.y+1)).r;
```

Récupération des 5 valeurs de la texture « FrameBuffer_2particles_filter » avec le pixel local et ses 4 voisins cardinaux.

```
float3 t1 = normalize(float3(2,0,particle2-particle1));
float3 t2 = normalize(float3(0,2,particle4-particle3));
```

Création de 2 vecteurs tangentes à partir de ses voisins.

```
float3 n = cross (t1,t2);
n = normalize(n);
```

Calculer le vecteur normal.

```
float2 tmp = texCoord.xy + n.xy *1000.0 * cg_fs_partDistorsion;
```

Décaler les textures de coordonnées en fonction du vecteur normal « n » et du coefficient de distorsion « cg_fs_partDistorsion ».

```
float4 textureIn = saturate (texRECT( decal , tmp ).rgba + particle * cg_fs_partColor);
```

La texture « decal » est l'image « FrameBuffer_interpolROADBIDOmbre_filter », Le décalage de coordonnées de texture est appliqué à cette texture, Avec ou sans l'apparition des particules au niveau colorimétrique ou pas selon le paramètre « cg_fs_partColor ».

```
float3 waterColored = tex2D( lookupTable2 , float2(clamp(textureIn.x,0.05,0.99),0.1) ).rgb;
return float4(waterColored,1.);
```

Coloriser le tout avec la texture « water.png » (« lookupTable2 »).

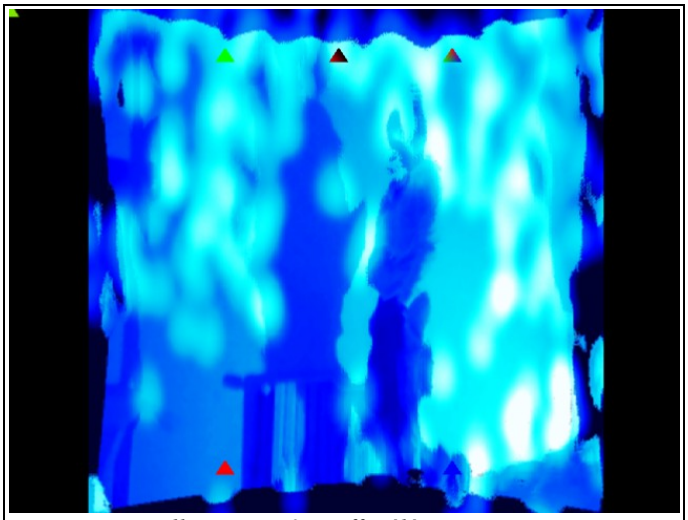

*Illustration 37: Effet élément Eau.*

### *o) Wood Element*

**Nom texture** : « FrameBuffer_woodElement_filter ».

**Fonction** : Un regroupement de différents petits pour en créer un qui a pour thème le bois, basé sur le masquage (voir illustration 39).

**Nom fichier** : « filter_woodElement.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-woodElement.cg ».

**Paramètre (s) d'entrée (s)** : aucun.

**Texture (s) d'entrée (s)** :

-« FrameBuffer_dOmbre_filter » : l'image binaire résultante de la détection des objets en mouvement.

- « woodelement.jpg » : l'image d'une barrière en bois.

- « woodfilter.jpg » : le filtre binaire de la barrière en bois.

*Illustration 38: A gauche « woodelement.jpg » et à droite « woodfilter.jpg »*

**Détail :**

L'extrait du fichier « TLLSLO-FS-woodElement.cg »

```
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;
float4 textureIn  = texRECT( decal , texCoord ).rgba;
```

récupération de la valeur d'intensité lumineuse du pixel local de l'image binaire « FrameBuffer_dOmbre_filter ».

```
decalCoords.y = 1. - decalCoords.y;
float3 texture1 = tex2D( lookupTable1 , decalCoords.xy ).rgb;
float3 texture2 = tex2D( lookupTable2 , decalCoords.xy ).rgb;
```

Effectuer une symétrie verticale, et récupérer la couleur du pixel de l'image et du filtre.

```
float4 fond = float4 (texture1 * 1-texture2, 1.);

float4 color = float4( 0.,0.,0.,1.);
color.rgb = fond.rgb + ( saturate((1-textureIn.rgb)+0.3) * texture2 * texture1);
return color ;
```

Il permet d'obtenir un masquage là l'ombre rentre un peu plus sombre les zones de la barrière et n'intervient pas sur le fond.

*Illustration 39: Effet élément bois.*

### *p) Metal Element*

**Nom texture** : « FrameBuffer_metalElement_filter ».

**Fonction** : Un regroupement de différents petits pour en créer un qui a pour thème le métal, basé sur les calcules de réflexion d'un objet en 3D (voir illustration 41).

**Nom fichier** : « filter_metalElement.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-metalElement.cg » et « TLLSLO-FS-metalElement.cg ».

**Paramètre (s) d'entrée (s)** : « cg_fs_lstrength » :

**Texture (s) d'entrée (s)** :

- « FrameBuffer_dOmbre_filter » : l'image binaire résultante de la détection des objets en mouvement.
- « metalhull.jpg » : l'image d'une plaque de métal.
- « normalmetalhull.jpg » : l'image normal map (vecteur normal) de la plaque de métal.
- « FrameBuffer_normalMapBlur_filter » : l'image contenant le vecteur normal des pixels contour.

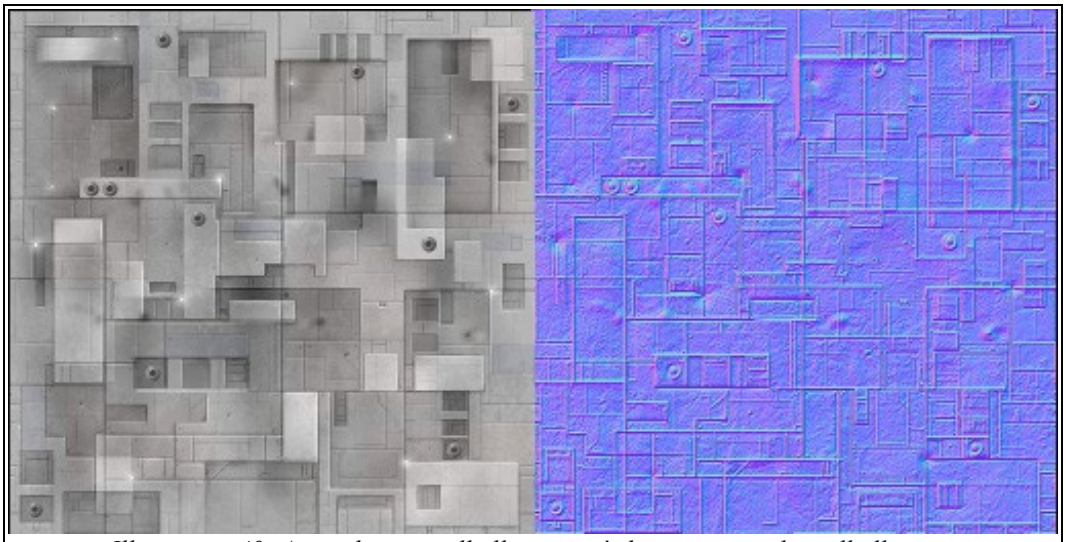

*Illustration 40: A gauche « metalhull.jpg » et à droite « normalmetalhull.jpg ».*

Détail :

L'extrait du fichier « TLLSLO-VP-metalElement.cg »

```
HPosition = mul(ModelViewProj, position);
decalCoords = texcoords;
V = mul(ModelView, position ).xyz;
L = (lightDir – V);
```

« V » est le vecteur vue, calcule la position dans le repère de l'observateur. « L » est le vecteur lumière.Ces 4 valeurs seront envoyé dans le programmer pixel shader « « TLLSLO-FS-metalElement.cg ».

L'extrait du fichier « TLLSLO-FS-metalElement.cg »

```
float2 fullScreenCorrection = cg_fs_2fv_size/float2(640,480);
float d = distance ( L, float3(decalCoords,0.) );
```

« d » est la distance entre le vecteur lumière et les coordonnées du pixel local (considérer comme plane).

```
float4 lColor;
lColor.rgb = float3(0.8,0.8,0.8) * saturate ( cg_fs_lstrength / (d * d) );
lColor.a = 1.;
```

L'intensité lumineuse dépende de la distance « d » entre la lumière et le pixel et le paramètre « cg_fs_lstrength ».

```
float2 texCoord = decalCoords.xy *cg_fs_2fv_size;
float4 shadow = texRECT(decal, texCoord).rgba;
```

Récupérer la valeur d'intensité lumineuse de la texture « FrameBuffer_dOmbre_filter ».

```
float3 nShadow = normalize( texRECT(lookupTable3, texCoord).rgb* 2.0 -1.0);
```

Récupérer le vecteur normal du pixel contour à partir de la texture « FrameBuffer_normalMapBlur_filter ».

```
float3 N = normalize( tex2D(lookupTable2, decalCoords.xy*fullScreenCorrection).xyz * 2.0 -1.0);
```

Récupérer le vecteur normal du pixel de la plaque de métal.

```
N.rg += nShadow.rg;
```

Superposer les 2 vecteurs pour renforcer l'effet de relief.

```
float NdotL = saturate ( dot(N, normalize(L) ) );
float NdotH = saturate ( dot(N, normalize(V) ) );

float4 lighting = lit(NdotL, NdotH, 0.001);
```

« dot » et « lit » sont des fonctions pour calculer l'intensité des lumière dans l'un environnement 3D (ambiant, diffuse, spéculaire) veuillez consulter la définition de ces 2 foncton ici ( http://http.developer.nvidia.com/CgTutorial/cg_tutorial_appendix_e.html).

```
float3 texture1 = tex2D( lookupTable1 , decalCoords.xy *fullScreenCorrection ).rgb;
```

Récupérer le pixel de couleur à partir de l'image de la plaque de métal.

```
float3 C = lighting.yyy * texture1 ;
```

la couleur du pixel dépends de la valeur calulée par « lighting ».

```
return float4(saturate (texture1 * 0.5 +  C * lColor)+shadow * 0.1,1.) ;
```

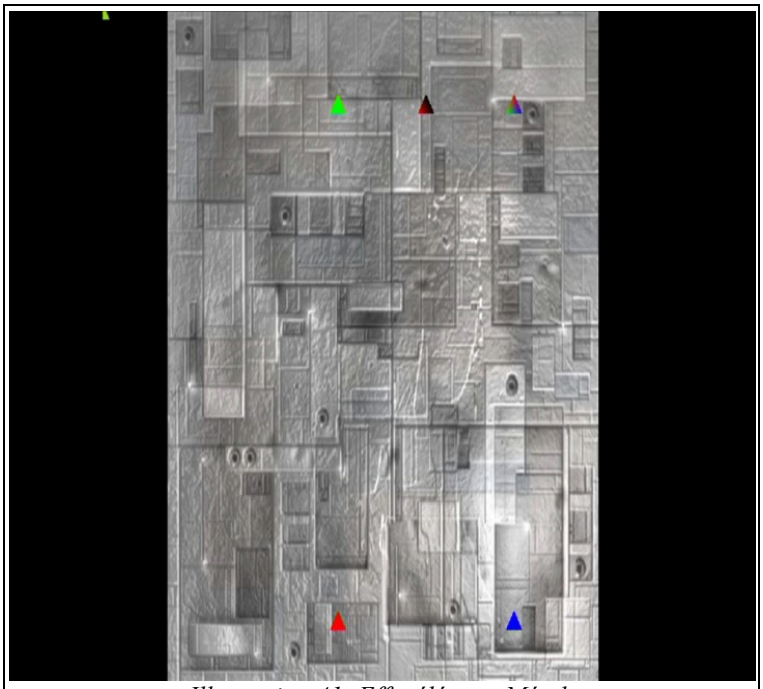

*Illustration 41: Effet élément Métal.*

## *q) Earth Element*

**Nom texture** : « FrameBuffer_earthElement_filter ».
**Fonction** : Un regroupement de différents petits pour en créer un qui a pour thème la terre.
**Nom fichier** : « filter_earthElement.xml ».
**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-earthElement.cg ».
**Paramètre (s) d'entrée (s)** : « cg_fs_2fv_v2size » : la taille de la vidéo.
**Texture (s) d'entrée (s)** :
- « FrameBuffer_dOmbre_filter » : l'image résultante de la détection de l'objet en mouvements.
- « bal.avi » : Une 2° flux vidéo, mais préenregistré.
- « raillingfilter.jpg » : le masque noir et blanc du grillages.
- « railling.jpg » : l'image du grillage.

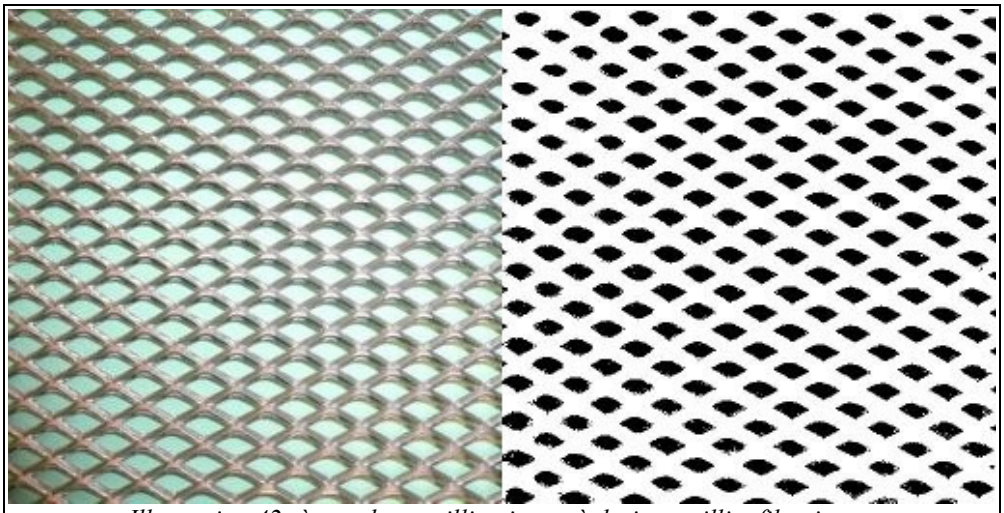

*Illustration 42: à gauche « railling.jpg », à droite« raillingfilter.jpg ».*

**Détail :**

Très similaire à l'effet bois (wood).

### *r) Warp*

**Nom texture** : « FrameBuffer_warp_filter ».

**Fonction** : déformation selon un curseur (voir illustration 43).

**Nom fichier** : « filter_warp.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-warp.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_2fv_pos1 » : la position lorsque le curseur touche l'objet « lcd » du logicielle de contre Max/MSP.
- « cg_fs_2fv_pos2 » : la position lorsque le curseur ne touche plus l'objet « lcd » du logicielle de contre Max/MSP.
- « cg_fs_2fv_postmp » : la position du curseur durant son déplacement.
- « cg_fs_warpclean » : Réinitialiser la textures de coordonnées de texture durant les instant où elle est supérieure à 0,85.

**Texture (s) d'entrée (s)** :

- « FrameBuffer_ROADBI_filter » : l'image courante propre et ajustée.
- « FrameBuffer_warpM1_filter » : la texture de coordonnées de texture l'instant précédente.

**Détail :**

la version dans VirChor est moins aboutie que celle de Max/MSP pour l'intégration exception selon la demande et les besoins. Néanmoins Le warping est Procédé qui permet la modification progressive par l'étirement d'une image. Ce procédé s'emploie généralement dans le domaine des images de synthèse.

Dans le cadre du projet TLLSLO, il y a d'abord la passe « warp » pour calculer la texture de coordonnées de texture en fonction de la positon du curseur et la texture de coordonnées de texture de l'instant précédente. Et la seconde passe est, à partir de la texture de coordonnées nouvellement calculée, d'appliquer l'image courante dessus.

### *s) Rendu de l'effet Warp*

**Nom texture** : « FrameBuffer_warprendering_filter ».

**Fonction** : récupérer la texture contenant les coordonnées de texture déformer, et l'appliquer sur l'image courante.

**Nom fichier** : « filter_warprendering.xml ».

**Nom (s) fichier(s) shader(s) : « TLLSLO-VP-final.cg » et « TLLSLO-FS-warprendering.cg ».**

**Paramètre (s) d'entrée (s)** : aucun

**Texture (s) d'entrée (s)** :

- « FrameBuffer_warp_filter » : la texture contenant les coordonnées de la texture déformée.
- « mire.jpg » : une image de calibrage pour l'expérimentation.

**Détail :**

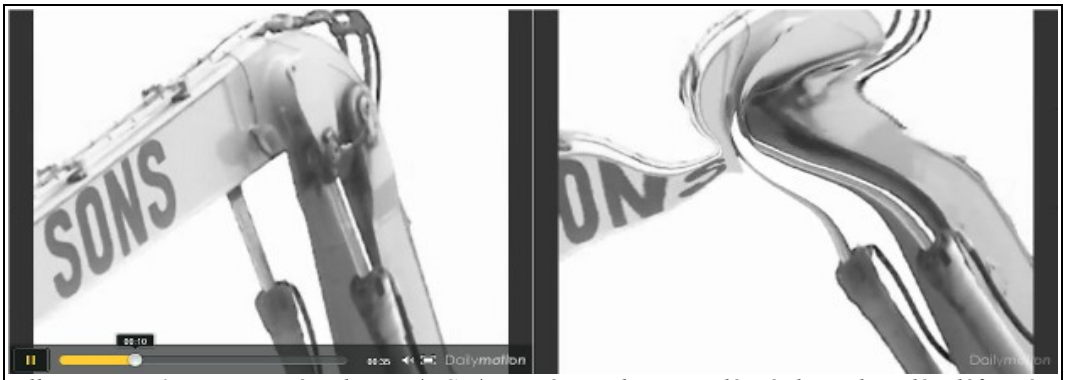


*Illustration 43: images tirées de Max/MSP/Jitter, à gauche une vidéo, à droite la vidéo déformée par effet Warping.*

# 9. Module de « compositing » et de sélection

Ce dernier module du système logicielle permet avant tout de

## *a) Sélecteur d'effets*

**Nom texture** : « FrameBuffer_compositing_filter ».

**Fonction** : rassemble 12 textures et décide la quelle de ces textures sera le flux vidéo à diffuser par le vidéo projecteur.

**Nom fichier** : « filter_compositing.xml ».

**Nom (s) fichier(s) shader(s) :** « TLLSLO-VP-final.cg » et « TLLSLO-FS-compositing.cg ».

**Paramètre (s) d'entrée (s) :**

- « cg_fs_2fv_videoSize » : contient les valeurs de la taille de la vidéo, le faite de mettre directement l'image de l'entrée à la sortir du système a pour but de vérifier le bon fonctionnement de l'acquisition vidéo.
- « cg_fs_4fv_filter_alpha1 », « cg_fs_4fv_filter_alpha2 » et « cg_fs_4fv_filter_alpha3 » : sont 3 tableaux de 4 valeurs (en tout 12 valeurs) pour sélectionner la diffusion d'un effet ou une image donnée, il fait mettre 1 au numéro de l'effet, et mettre à 0 le reste des effets pour ne pas les afficher.
- « cg_fs_imgRefOn » : 1 pour afficher l'image de référence (le fond) en bas à droit de l'image de sortie.
- « cg_fs_mire : 1 pour afficher le mire pour la calibrage.

**Texture (s) d'entrée (s)** :

- image entrelacée et capturée par la caméra grâce à au tag « vidéo » défini par VirChor.
- « FrameBuffer_colorCorrection_filter » : colorier les 2 zones de l'image binaire (noir et blanc) avec des couleurs différents.
- « FrameBuffer_ROADBI_filter » : l'image courante propre et ajusté.
- « FrameBuffer_multiply_filter » : effet figer et duplication.
- « FrameBuffer_ssFondSobel_filter » : l'image de soustraction de fond dans le domaine du coutour.
- « FrameBuffer_contrast_filter » : correction contraste luminosité des fois utile pour le calibrage entre la surface de projection et la caméra.
- « FrameBuffer_2particles_filter » : le rendu visuel des particules selon l'un des 2 modes de déplacement.
- « FrameBuffer_dOmbre_filter » : l'image binaire résultante de la passe détection de l'objet en mouvement.
- « FrameBuffer_earthElement_filter » : effet élément Terre
- « FrameBuffer_dOmbreEcho_filter » : effet artistique écho.
- « FrameBuffer_warp_filter »: la texture de coordonnées de texture warp.
- « FrameBuffer_warprendering_filter » : le rendu de l'effet warp sur une image mire.

**Détail :**

L'extrait du fichier « TLLSLO-FS-compositing.cg »

```
//original video texture coordinate
float2 videoCoord;
videoCoord = float2(decalCoords.x, 1 - decalCoords.y);
videoCoord = videoCoord*cg_fs_2fv_videoSize;
```

Le traitement du flux vidéo brute est différente des autres textures pour sa taille.

```
//buffer texture coordinate
float2 texCoord = decalCoords.xy*cg_fs_2fv_size;

//reference image coordinate
float imgRefSize = 4.0;
float tmp = (cg_fs_2fv_size.x)/imgRefSize;
float2 imgRefTexCoord = (float2(texCoord.x - (tmp*(imgRefSize-1)),texCoord.y))*imgRefSize ;
```

Placer et diminuer la taille de l'image de référence avec des transformation au niveau des coordonnées de textures.

```
float4 imageRef = texRECT( lookupTable3, imgRefTexCoord ).rgba;
float4 videoBrute = texRECT( decal, videoCoord).rgba;
float4 tex1 = texRECT( lookupTable1, texCoord).rgba;
float4 tex2 = texRECT( lookupTable2, texCoord).rgba;
float4 tex3 = texRECT( lookupTable3, texCoord).rgba;
float4 tex4 = texRECT( lookupTable4, texCoord).rgba;
float4 tex5 = texRECT( lookupTable5, texCoord).rgba;
float4 tex6 = texRECT( lookupTable6, texCoord).rgba;
float4 tex7 = texRECT( lookupTable7, texCoord).rgba;
float4 tex8 = texRECT( lookupTable8, texCoord).rgba;
float4 tex9 = texRECT( lookupTable9, texCoord).rgba;
float4 tex10 = texRECT( lookupTable10, texCoord).rgba;
float4 tex11 = texRECT( lookupTable11, texCoord).rgba;
float4 mire = tex2D( lookupTable12, float2(decalCoords.x, 1 – decalCoords.y)).rgba;
```

Récupération du pixel de couleur et d'intensité du pixel local des images ou des textures.

```
float4 color;
color = cg_fs_4fv_filter_alpha1.x * videoBrute      +
        cg_fs_4fv_filter_alpha1.y * tex1            +
        cg_fs_4fv_filter_alpha1.z * tex2            +
        cg_fs_4fv_filter_alpha1.w * tex3            +
        cg_fs_4fv_filter_alpha2.x * tex4            +
        cg_fs_4fv_filter_alpha2.y * tex5            +
        cg_fs_4fv_filter_alpha2.z * tex6            +
        cg_fs_4fv_filter_alpha2.w * tex7            +
        cg_fs_4fv_filter_alpha3.x * tex8            +
        cg_fs_4fv_filter_alpha3.y * tex9            +
        cg_fs_4fv_filter_alpha3.z * tex10           +
        cg_fs_4fv_filter_alpha3.w * tex11           ;
```

Le choix de l'effet se réalise selon la valeurs des 2 paramètres.

```
if (cg_fs_mire == 1)
        return mire;
```

Afficher ou pas l'image mire pour la calibration.

```
if (cg_fs_imgRefOn ==0)
        return float4( color );
```

Affiche ou pas l'image référence en bas à droit ou pas.

```
imgRefTexCoord = imgRefTexCoord /cg_fs_2fv_size ;
if (((decalCoords.x > 1-(1/imgRefSize))&&(decalCoords.y < 1/imgRefSize)))
        color = imageRef;
return float4( color );
```

### *b) calibrage final*

**Nom texture** : aucun.

**Fonction** : faire le calibrage entre le vidéoprojecteur et la surface de projection.

**Nom fichier** : « final_subdivision.xml ».

**Nom (s) fichier(s) shader(s)** : « TLLSLO-VP-final.cg » et « TLLSLO-FS-keystone.cg ».

**Paramètre (s) d'entrée (s)** :

- « cg_fs_bypass » : paramètre à valeur 1 pour ne pas utiliser le traitement et 0 pour l'utiliser.
- « cg_fs_2fv_coeffHG », « cg_fs_2fv_coeffHD », « cg_fs_2fv_coeffBG » et « cg_fs_2fv_coeffBD » sont 4 paires de valeurs pour la position en x et y des 4 coins de la texture (HG pour haut gauche, HD pour hautdroite, BG pour bas gauche et BD pour bas droite). Ces valeurs varient entre -4 à 4.
- « cg_fs_4fv_outColor » est un tableau de 4 variables pour la couleurs du bord de l'image (RGBA).

**Texture (s) d'entrée (s)** : « FrameBuffer_compositing_filter » :

**Détail :**

C'est le même programme shader que la calibration entre la caméra et la surface de vidéoprojection (voir la partie « keystone » du « Module de traitement d'images et de calibrage » ).

## Conclusion

La chaîne de traitement vidéo qui a commencé par le filtre désentrelacement et a fini avec cette dernière passe de calibration. L'ensemble de ses opérations regrouper en passe, puis en module constituent le cœur logicielle du dispositif TLLSLO. Les logicielles périphériques permettent avant tout de contrôler le moteur de rendu VirChor.

# V. Tableau récapitulatif des effets portés de VirChor à Max/MSP/Jitter

Ce tableau permet de récapituler les filtres expérimentés sous VirChor et portés sous Max/MSP/jitter ou pas.

| Filtre | VirChor | Max/MSP/Jitter |
|---|---|---|
| Désentrelacement | x | x |
| Correction radiale | x | x |
| Correction de la rotation et des symétries horizontale et verticale | x | |
| keystone | x | x |
| Médian spatial | x | x |
| Filtrage Bilatéral | x | x |
| Médian Temporel | x | x |
| Gaussian | x | x |
| Fond | x | x |
| Détection de contour Sobel | x | x |
| Détection de contour Sobel de l'image fond | x | x |
| Soustraction de fond en Détection de contour « ssFondPlusSobel » | x | x |
| Soustraction de fond dans le domaine logarithmique « SsFondLOG » | x | x |
| Rétrécissement | x | x |
| Homogénéité | x | x |
| Aggrandissement | x | x |
| Filtrage avec soustraction de fond et homogénéïté | x | x |
| Détection d'Ombre | x | x |
| Fonctionnement du système de tracking dans VirChor | x | o |
| Calcul de la boîte englobante de l'objet en mouvement | x | o |
| FrameBuffer_normalMap_filter | x | x |
| FrameBuffer_normalMapBlur_filter | x | x |
| Echo de la détection d'Ombre | x | x |
| Moteur physique de particules | x | X gravité seulement |
| Filtre d’intensité lumineuse des particules en goutte | x | x |
| Filtre d’intensité lumineuse des particules en mots | x | o |
| Mode de vision du rendu | x | o |
| Compensation colorimétrique | x | o |
| Effet flash | x | x |

| | | |
|---|---|---|
| Contraste et lluminosité | x | x |
| Bascule entre l'image d'ombre et l'image courante | x | o |
| Plasma | x | x |
| Echo | x | o |
| Masque | x | x |
| Duplication | x | o |
| Halo | x | o |
| Aspirate | x | o |
| Vibration | x | o |
| Fire | x | x |
| Fire Element | x | o |
| Water Element | x | o |
| Wood Element | x | o |
| Metal Element | x | o |
| Earth Element | x | o |
| Warp | x | X finaliser |
| Rendu de l'effet Warp | x | o |
| Sélecteur d'effets | x | o |
| calibrage final | x | o |

# VI. Référence

[1] Garnett Roman; Huegerich Timothy; Chui Charles; Wenjie He. A universal noise removal algorithm with an impulse detector. *IEEE transactions on image processing ISSN 1057-7149*

[2] Chris Stauffer W.E.L Grimson Adaptive background mixture models for real-time tracking The Artificial Intelligence Laboratory Massachusetts Institute of Technology Cambridge,M A 02139

[3] A. Leone, C. Distante. Shadow detection for moving objects based on texture analysis. *Istituto per la Microelettronica e Microsistemi IMM-CNR*.

[4] Quen-Zong Wu, Bor-Shenn Jeng. Background subtraction based on logarithmic intensities. *Chunghwa Telecommunication Laboratories*.

D'autres filtres, ont été implémentés et testés (ex : le filtre trilatéral, qui est plus performant que la bilatéral, mais plus gourmande, ou encore les filtres morphologies mathématiques). Ils ont pas été inclut dans la chaîne de traitement
De même que pour la documentation, les articles cités dans ce document ne sont pas la liste exhaustive de la documentation, vous pouvez accéder à cette liste via cette adresse : http://vida.limsi.fr/index.php/TLLSLO_EdL_Scientifique_R%C3%A9f%C3%A9rences_scientifiques