Institut
Mines-Télécom

Le composeur d'images . . .

Composition par tuiles, liste graphique, pipeline graphique

Yves Mathieu

Plan

Le composeur vidéo

Caractéristiques

- Composition d'objets (vidéo, icones, curseurs),temps réel
- Temps presque réel : transformation des objets
- Filtrage bilinéaire (à la demande)
- Opacité réglable (à la demande)

Le programme de référence "appli.c"

Positionner un objet

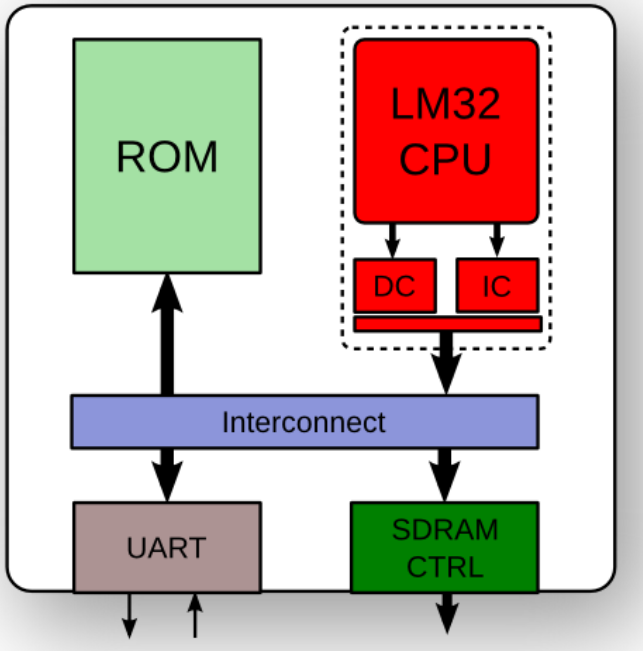
- Une version "logicielle" des traitements à executer
- Base pour l'application devant tourner sur le CPU
- Composition confinée dans "composeur.c"

```
// On "place" les objets dans la scène
// rappel des appels de fonction:
// obj_place_corners(obj_num, x_top_left,y_top_left, x_bottom_right, y_bottom_right)
// obj_place(obj_num, x_top_left,y_top_left, zoom_factor)

// on place la texture correspondant à la video
obj_place_corners(0,-100,300,700,100) ;
// on place le logo TPT avec interpolation
obj_place(1,300,100,to_LG_fp32(2.0)) ;
// on place le logo TPT sans interpolation
obj_place(2,120,300,to_LG_fp32(1.0)) ;
// on place le curseur avec interpolation et transparence
obj_place(4,50,100,to_LG_fp32(4.0)) ;
// on place le curseur sans interpolation
obj_place(3,400,300,to_LG_fp32(4.5)) ;
// On met à jour les listes chaînées dans les tuiles
maj_scene() ;
// On lance l'opération de composition
composeur(p_scene) ;
```

Le Soc de base

Caractéristiques

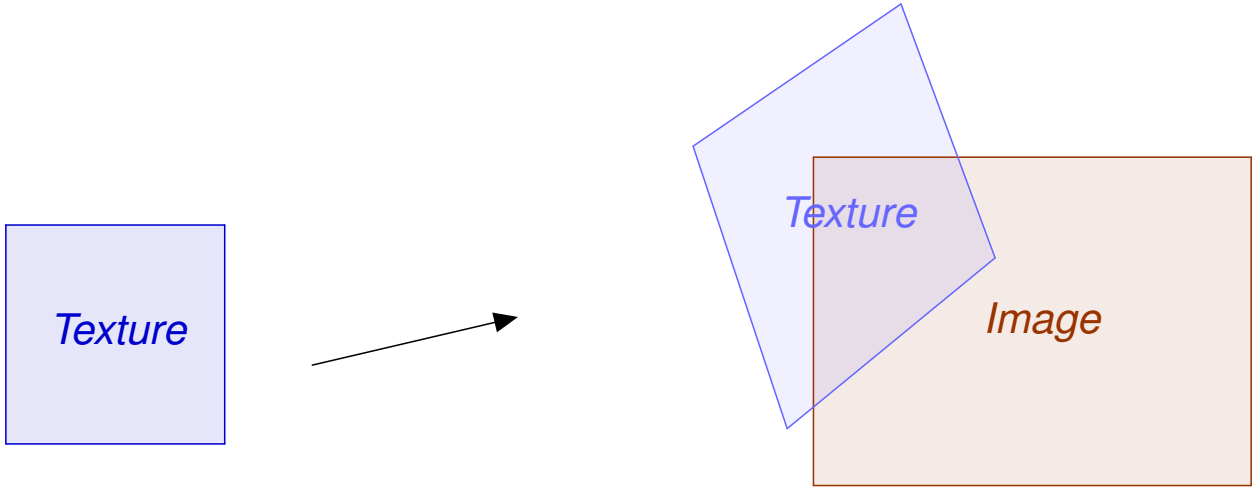


- Insertion d'un bloc entrée vidéo
- Insertion d'un bloc sortie vidéo
- Insertion d'un composeur d'images

Plan

La transformation directe

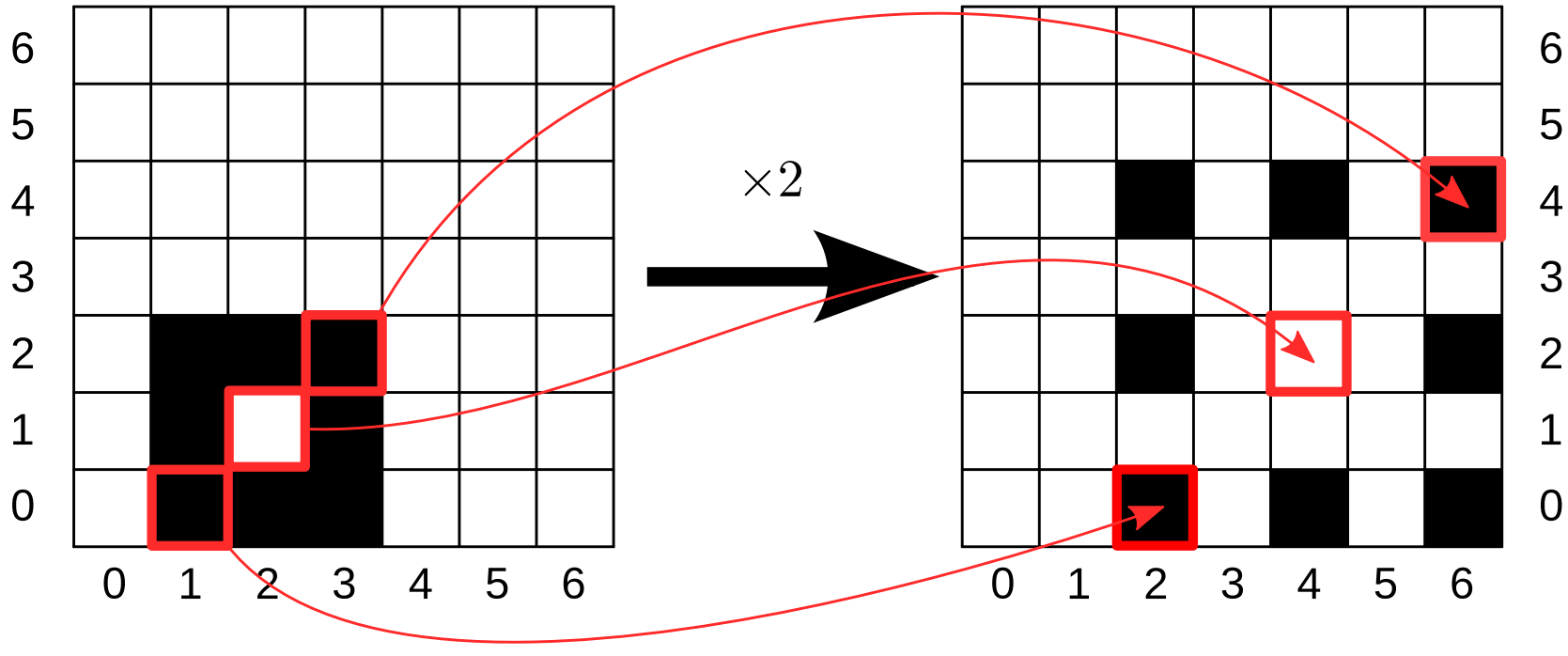
Ce que spécifie l'application



- Un objet est une texture rectangulaire
- L'application spécifie la transformation affine de l'espace "texture" vers l'espace "image"

La transformation directe

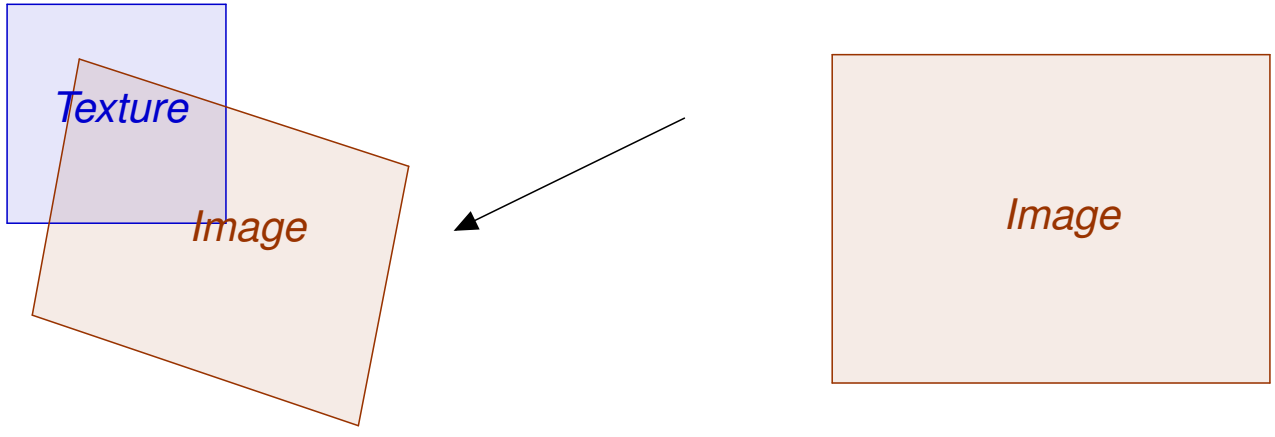
Exemple : zoom $\times 2$



- Difficilement utilisable pour le rendu effectif : les pixels sont projetés à des positions arbitraires dans l'image

La transformation inverse

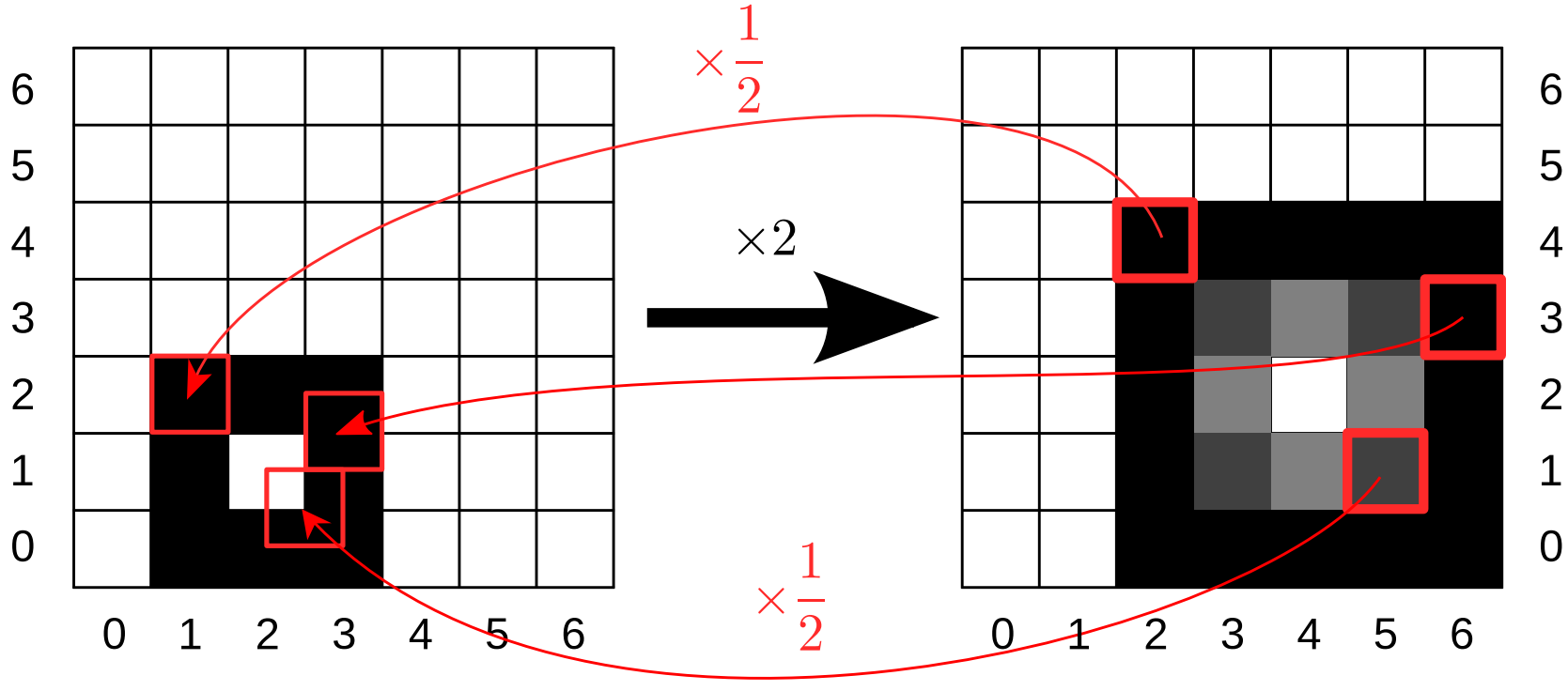
Ce que font les composeurs d'images



- L'image est parcourue ligne à ligne, pixel à pixel
- La transformation affine inverse donne la position dans la texture du pixel

La transformation inverse

Exemple : zoom $\times 2$



- On échantillonne le pixel le plus proche dans la texture
- ... ou on filtre un voisinage de pixels dans la texture

Plan

La composition d'images

Pseudo algorithme

```
Initialiser l'image a la couleur de fond dans la mémoire externe
Pour chaque texture ayant une intersection non vide avec l'image:
   Lire l'image courante
   Lire la texture
   Composer la texture avec l'image courante
   Ecrire l'image courante
FinPour
```

- Problème 1 : 2 lectures + 1 ecriture en mémoire par pixel
- Problème 2 : La SDRAM n'est efficace que par paquets
- Problème 3 : 4 pixels à lire pour l'interpolation bilinéaire
- Problème 4 : Initialisation : écriture complète d'une image
- Problème 5 : Cache statistique inefficace (taux de réutilisation, taille des textures)

La composition d'images par tuiles

Pseudo algorithme

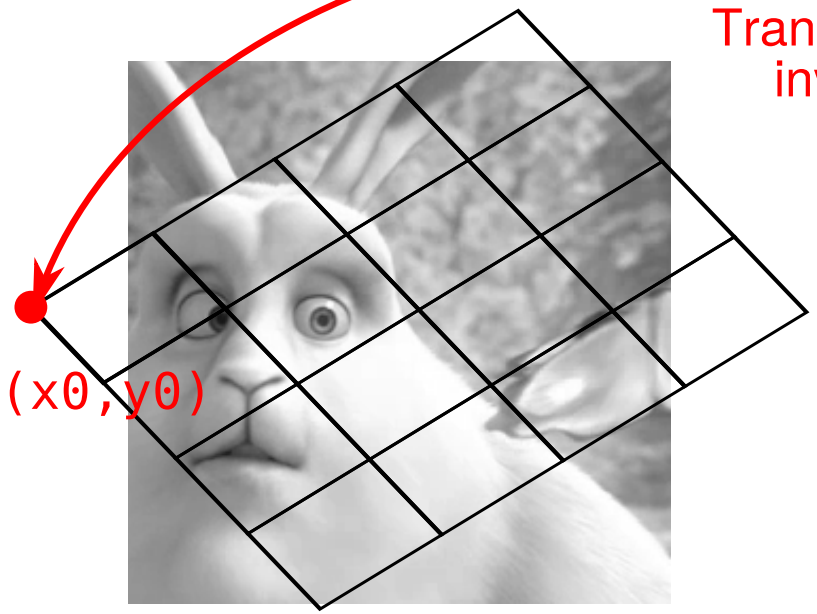
- Découper l'image en petits blocs carrés (tuiles) de 32x32 pixels

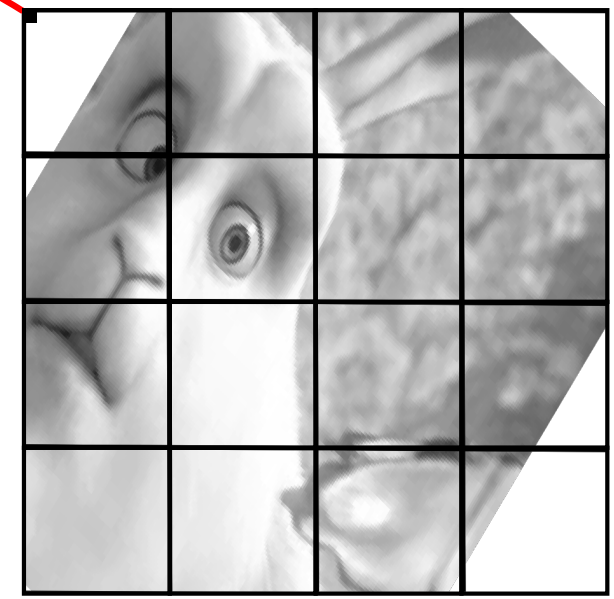
```
Pour chaque tuile de l'image
  Initialiser une tuile avec la couleur de fond en mémoire locale
  Pour chaque texture interceptant la tuile courante
      Lire la portion de texture interceptant la tuile
      Composer la portion de texture avec la tuile
  FinPour
  Sauver la tuile en mémoire externe
FinPour
```

- Gain 1 : une lecture en mémoire par pixel
- Gain 2 : Une seule écriture finale
- Gain 3 : Pixels pour l'interpolation en mémoire locale
- Gain 4 : Initialisation en mémoire locale.

Image composée

Transformée inverse





Texture

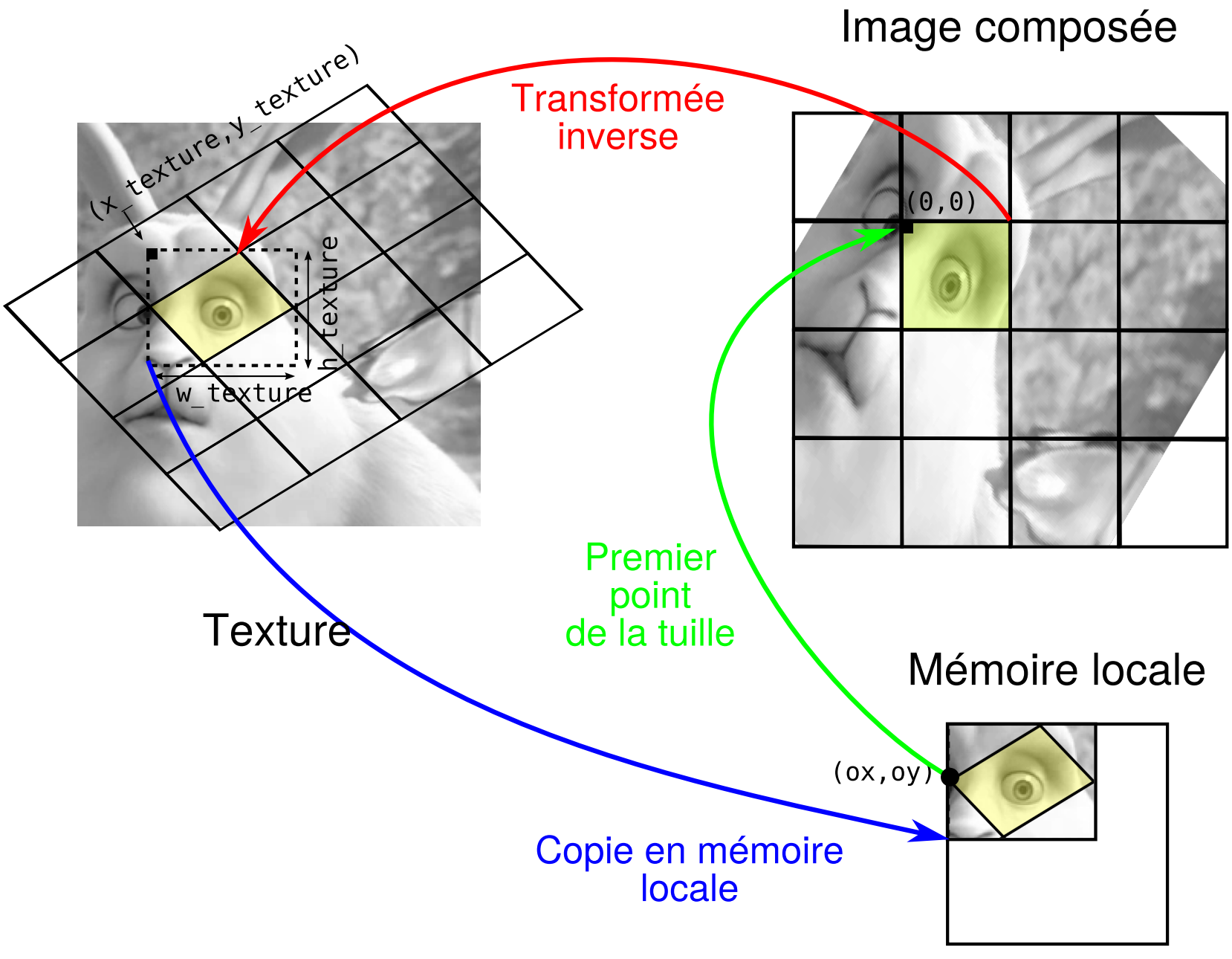


Image composée
Transformée inverse
(x_texture,y_texture)
h_texture
w_texture
(0,0)
Premier point de la tuille
Texture
Mémoire locale
(ox,oy)
Copie en mémoire locale

Plan

Principes de la liste graphique

Partitionnement Logiciel / Matériel

- Le Composeur est un esclave du micro-processeur
- Le Composeur est accède à une structure de données appelée "liste graphique"
- La liste graphique contient toutes les informations nécessaires au traitement de toutes les tuiles de l'image.
- La liste graphique est générée par le micro-processeur

Principes de la liste graphique

Partitionnement Logiciel / Matériel

- Le Composeur ne communique qu'avec la mémoire SDRAM qui contient la liste graphique, les textures et l'image finale
- Chaque tuile est traitée de manière indépendante des autres tuiles.
- Le micro-processeur ordonne au Composeur de démarrer le traitement d'une liste graphique
- Le Composeur informe le micro-processeur de la fin du traitement

Structures de données

La scène

- Le micro-processeur transmet au Composeur un pointeur en mémoire vers un descripteur de "scène"

```
typedef struct {
   LGTuile_t*       p_liste_de_tuiles ;  // pointeur vers la liste de tuiles
   LGPixel_t*       p_image;             // Addresse de l'image dans la mémoire
   unsigned short   largeur ;            // largeur de l'image
   LGPixel_t        ng_fond;             // le niveau de gris du fond par défaut
   unsigned char    vide[1];             // bourrage (alignement sur 32 bits)

} LGScene_t;
```

Structures de données

La liste de de descripteurs de tuiles

- Liste chaînée des tuiles.
- Le processeur peut limiter le renouvellement à un nombre arbitraire de tuiles.
- L'ordre de traitement est quelconque

```
typedef struct LGTuile_struct LGTuile_t ;
struct LGTuile_struct {
   LGTuile_t*  p_tuile_suivante;          // pointeur vers la tuile suivante
   LGSurface_t*  p_liste_de_surfaces;     // pointeur vers la liste des surfaces
   unsigned char row  ;                   // rangée de la tuile dans la scène
   unsigned char col  ;                   // la colonne de la tuile dans la scène
   unsigned char vide[2];                 // bourrage
} ;
```

Structures de données

Une "surface"

- Intersection entre un objet et la tuile courante (dans l'espace de la texture)
- Paramètres nécessaires à la récupération de la portion de texture correspondante

```
typedef struct LGSurface_struct LGSurface_t ;
struct LGSurface_struct {
   LGSurface_t*          p_surface_suivante; // pointeur vers la surface  suivante
   LGObj_t*              p_obj;              // pointer vers l'objet correspondant
   LGAttributs_t*        p_attributs;        // pointeur vers les attributs de l'objet
   LG_fp16               ox ;                // abcisse dans la texture locale de la
                                             // l'image du point (0,0) de la tuile
   LG_fp16               oy ;                // ordonnée dans la texture locale de la
                                             // l'image du point (0,0) de la tuile
   signed short          x_texture ;         // abcisse de la boite entourante
                                             // de la surface dans la texture
   signed short          y_texture ;         // ordonnée de la boite entourante
                                             // de la surface dans la texture
   signed short          w_texture ;         // largeur de la boite entourante de la
                                             // surface dans la texture
   signed short          h_texture ;         // hauteur de la boite entourante de la
                                             // surface dans la texture
} ;
```

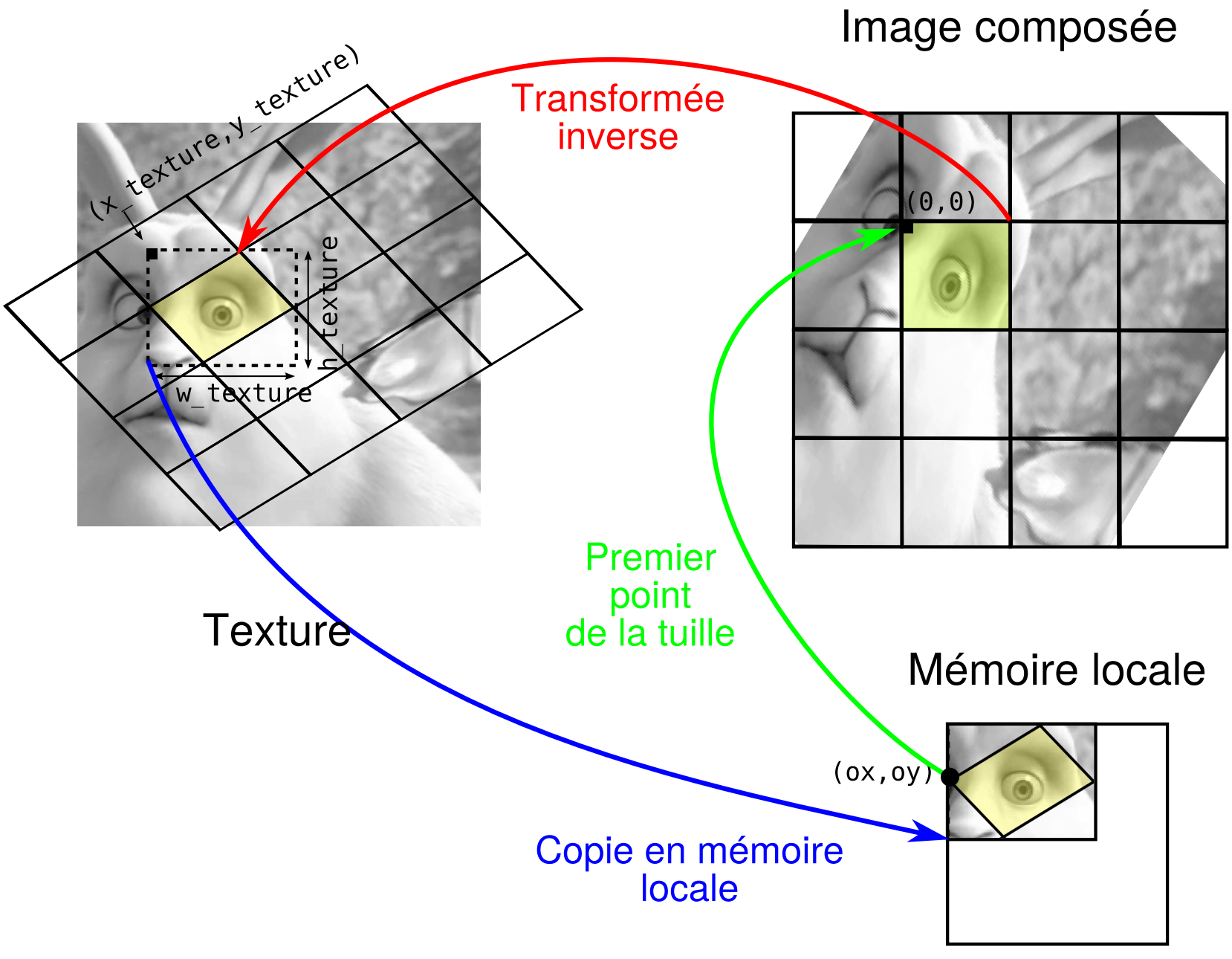


Image composée
Transformée inverse
(x_texture,y_texture)
h_texture
w_texture
(0,0)
Premier point de la tuille
Texture
Mémoire locale
(ox,oy)
Copie en mémoire locale

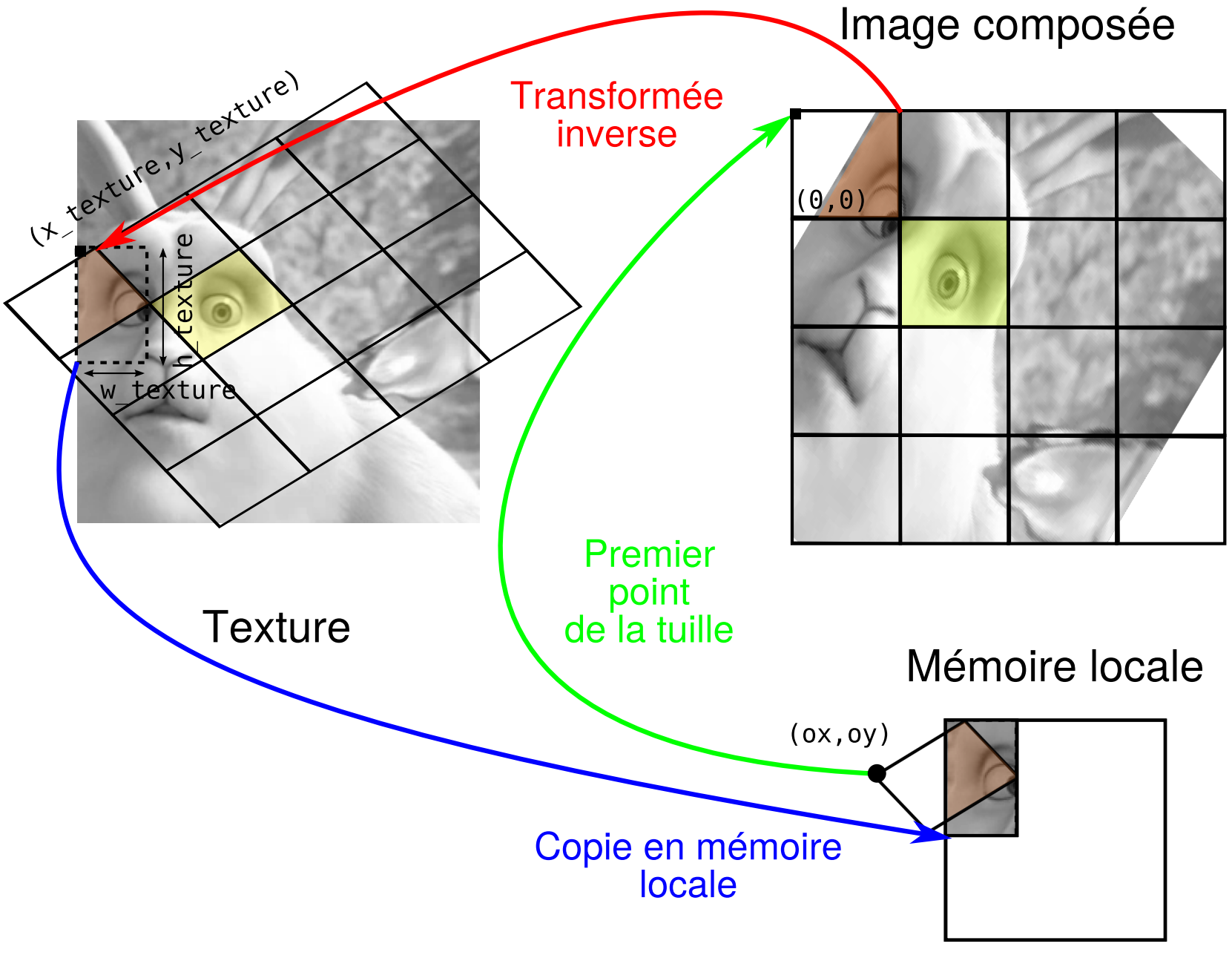


Image composée
Transformée
inverse
(x_texture,y_texture)
h_texture
w_texture
(0,0)
Texture
Premier
point
de la tuille
Mémoire locale
(ox,oy)
Copie en mémoire
locale

Structures de données

Un objet

- Un objet fait référence à une texture et éventuellement un masque

```
typedef struct {
   LGPixel_t *       p_texture ; // addresse de la texture associée
   unsigned int  *   p_masque  ; // addresse du masque binaire de forme
   unsigned short    largeur   ; // largeur de la texture en pixels
   unsigned short    hauteur   ; // hauteur de la texture en pixels
} LGObj_t;
```

Structures de données

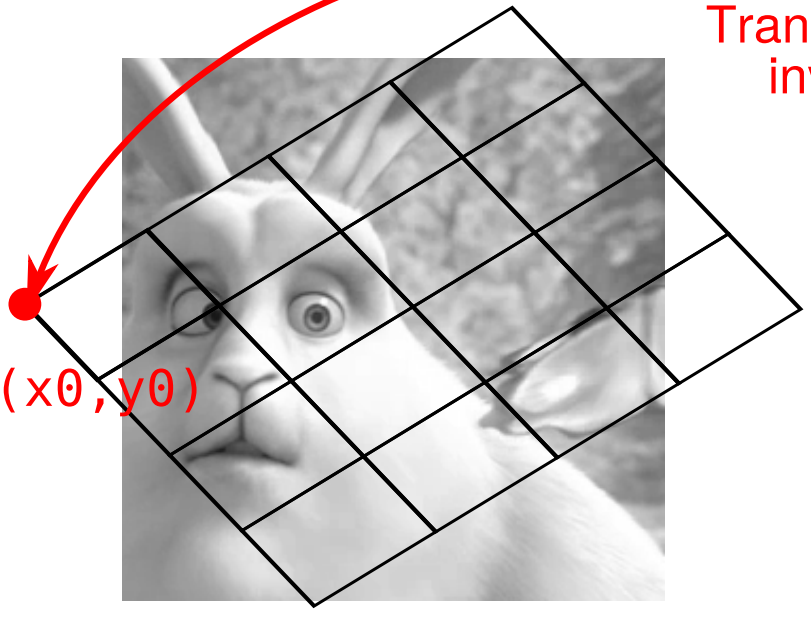
Attributs d'un objet

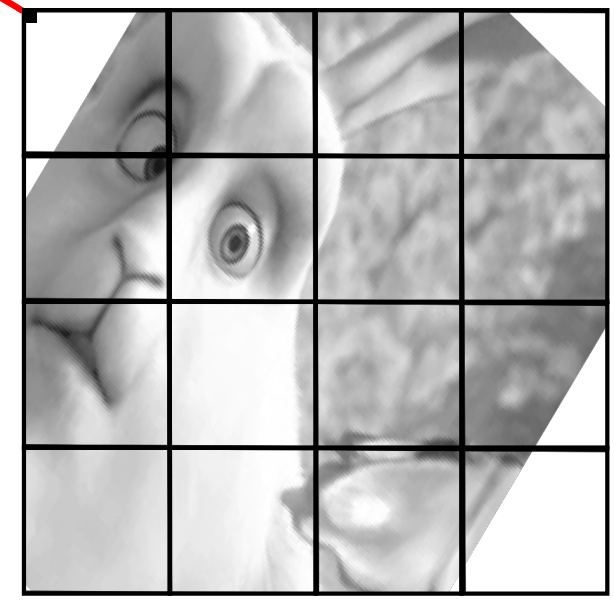
- Les attributs concernent la transformation de la texture et son mode de rendu

```
/** @brief descripteur d'attributs */
typedef struct {
                                    // coodonnées du point (0,0) de l'image après
                                    // transformée inverse
   LG_fp32           x0  ;          // abcisse dans la texture globale
   LG_fp32           y0  ;          // ordonnée dans la texture globale
                                    // coefficients de la transformée inverse
   LG_fp32           a_x ;          // pour l'abcisse  selon l'indice i
   LG_fp32           b_x ;          // pour l'abcisse  selon l'indice j
   LG_fp32           a_y ;          // pour l'ordonnée selon l'indice i
   LG_fp32           b_y ;          // pour l'ordonnée selon l'indice j
   unsigned char     alpha ;        // Opacité de l'objet
   unsigned char     interpolation ;// choix de l'interpolation
   unsigned char vide[2];           // bourrage
} LGAttributs_t;
```

Image composée

Transformée inverse





Texture

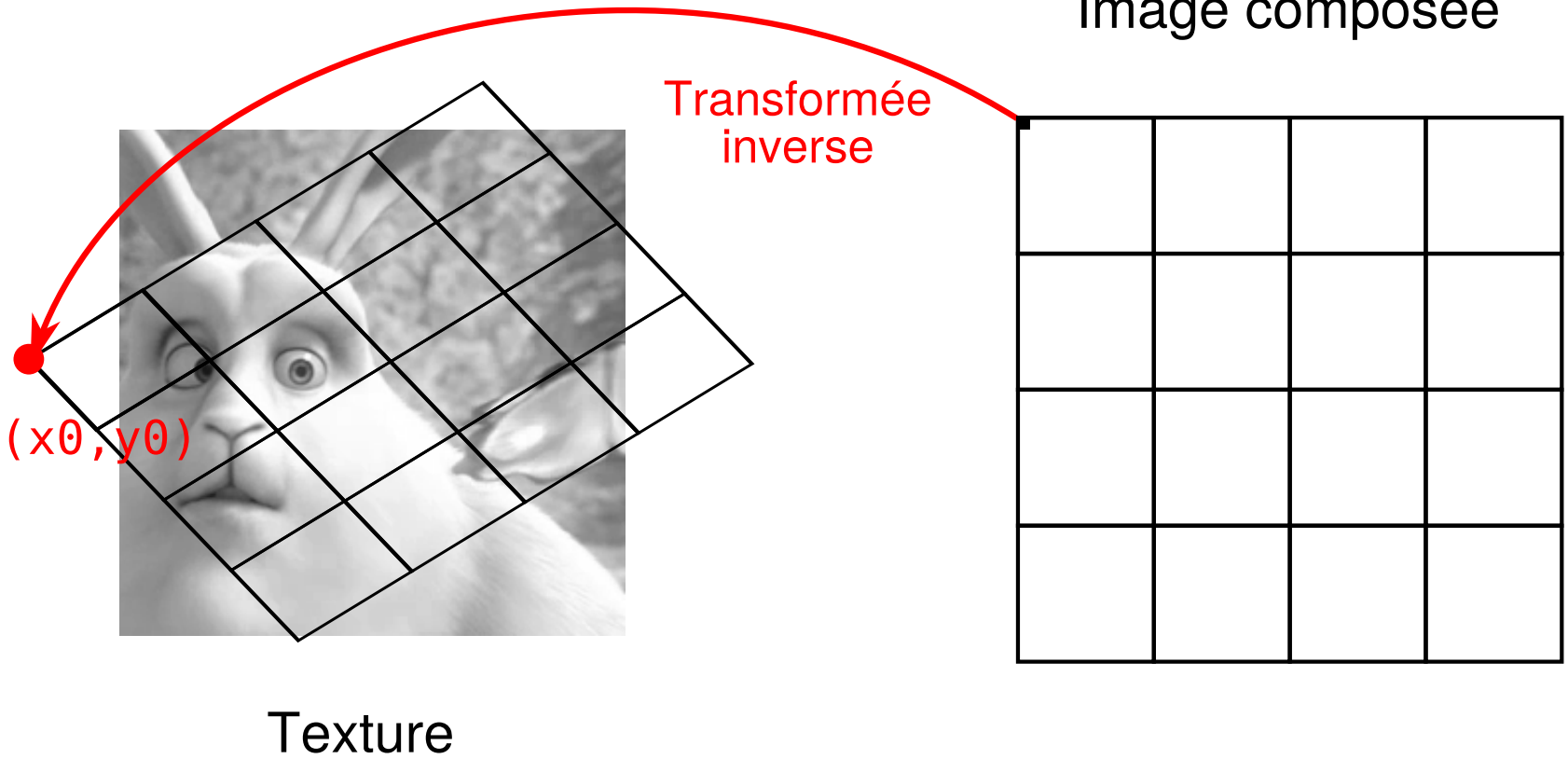


Image composée
Transformée
inverse
(x0,y0)
Texture

Types de données

Calculs en virgule fixe

- Coordonnées et coefficients affines
- Coordonnées 0,0 en haut/gauche des textures et images.
- Eviter les calculs flottants
- Deux représentations "implicites" définies par des types

```
// Un nombre en virgule fixe sur 16 bits
// et 8 bits de partie fractionnaire */
typedef signed short LG_fp16 ;

// Un nombre en virgule fixe sur 32 bits
// et 16 bits de partie fractionnaire */
typedef signed int   LG_fp32 ;

// Un pixel
typedef unsigned  char LGPixel_t ;
```

Types de données

Pixels, textures et masques

- Pixels codés sur 8 bits, 4 pixels par mot.
- Masque binaire stocké à raison de 32 pixels par mot.
- Largeurs de textures multiples de 32 pixels.
- Stockage linéaire de la texture en mémoire.

```
// Un nombre en virgule fixe sur 16 bits
// et 8 bits de partie fractionnaire */
typedef signed short LG_fp16 ;

// Un nombre en virgule fixe sur 32 bits
// et 16 bits de partie fractionnaire */
typedef signed int   LG_fp32 ;

// Un pixel
typedef unsigned  char LGPixel_t ;
```

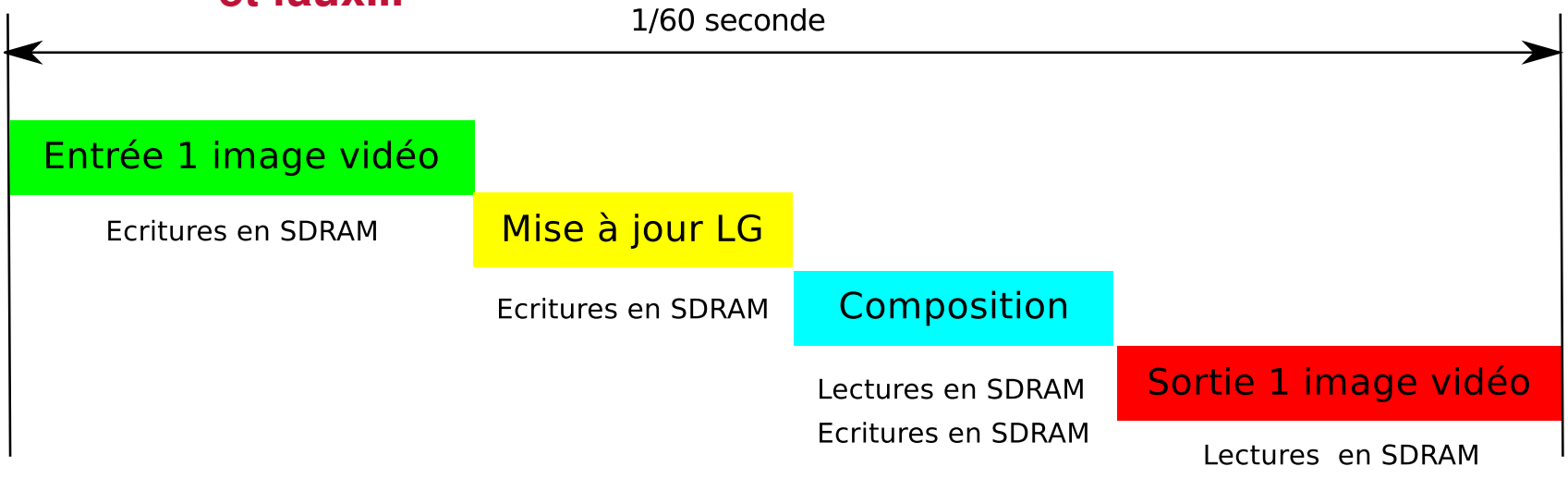
Plan

Considérations générales

- Nous voulons du temps réél
- Premier cas : liste graphique statique, flux video
- Deuxième cas : liste graphique dynamique, flux video
- Temps d'accès aux données (lecture / écriture)
- Temps de traitement
- granularité du pipeline

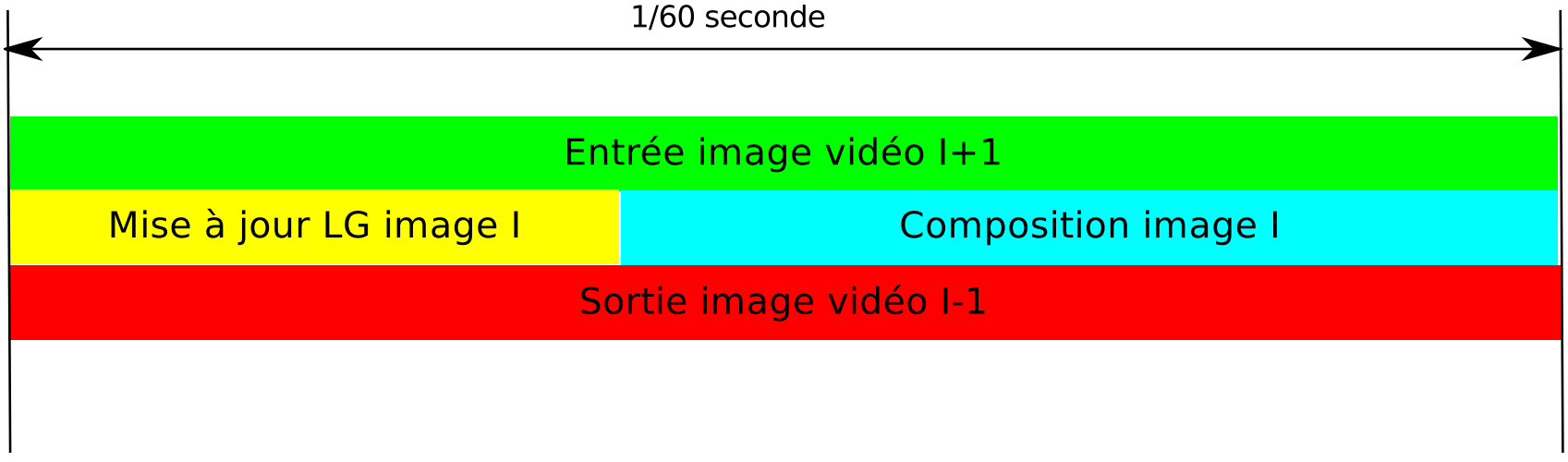
Un pipeline simpliste

et faux...



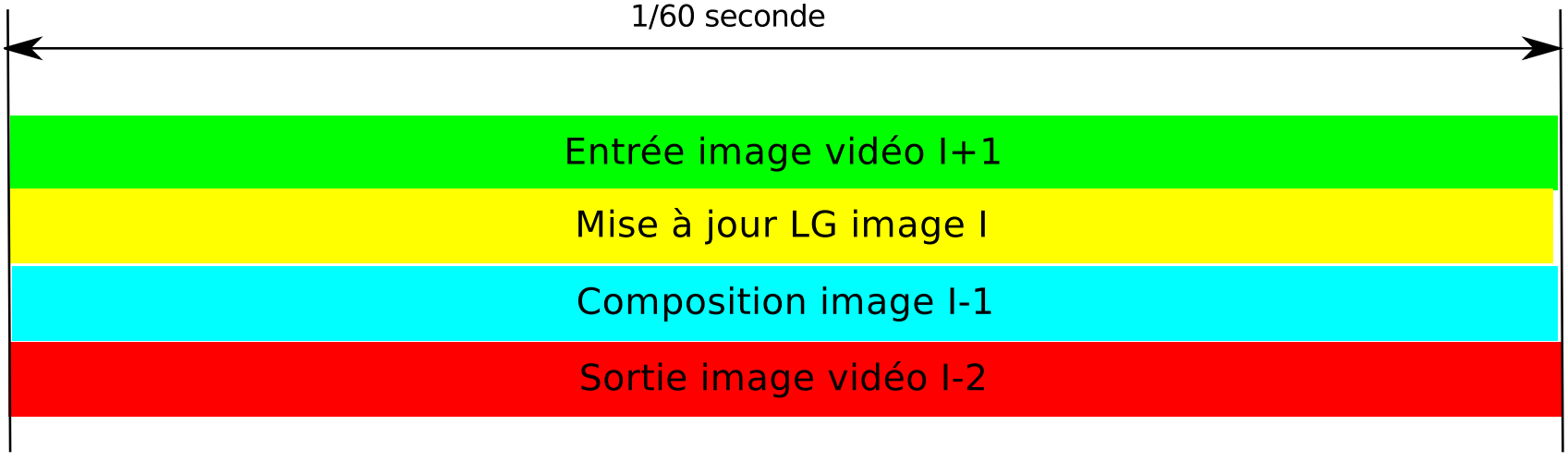
- Les images sont générées à 60Hz
- Le temps total dépend du nombre d'entrées sorties sur la SDRAM
- Le temps total dépend du cumul des temps (IO,CPU, Composition)
- **Trop simpliste : les images vidéo doivent être reçues et générées à rythme régulier**

Un pipeline simple



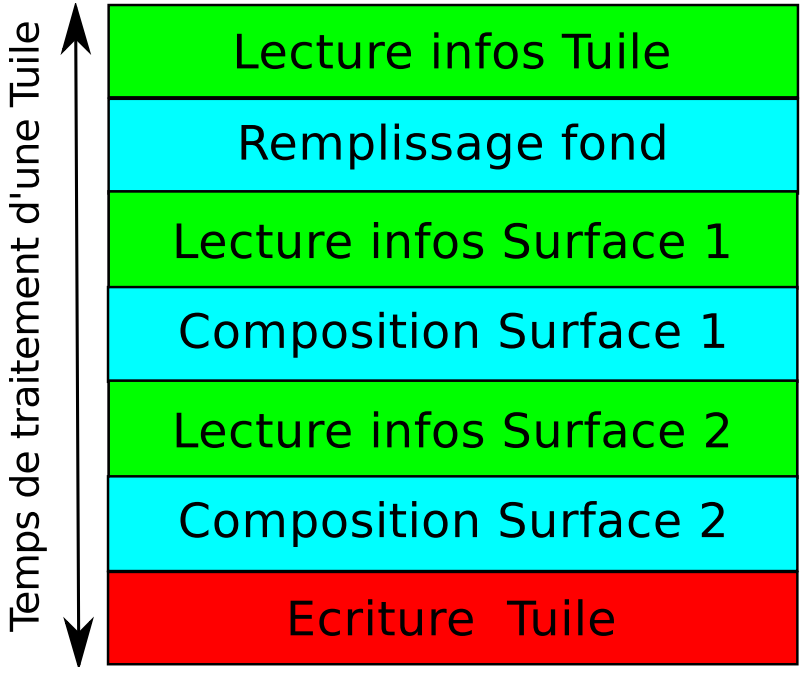
- Paralléliser les IO sur la SDRAM
- La SDRAM doit pouvoir contenir 3 images
- Si on ne tient pas le temps réel :
  - Répéter les images affichées
  - Sauter des images entrantes

Un pipeline complet



- Paralléliser les mise à jour de listes graphiques avec la composition
- La SDRAM doit pouvoir contenir 4 images
- La SDRAM doit contenir 2 listes graphiques

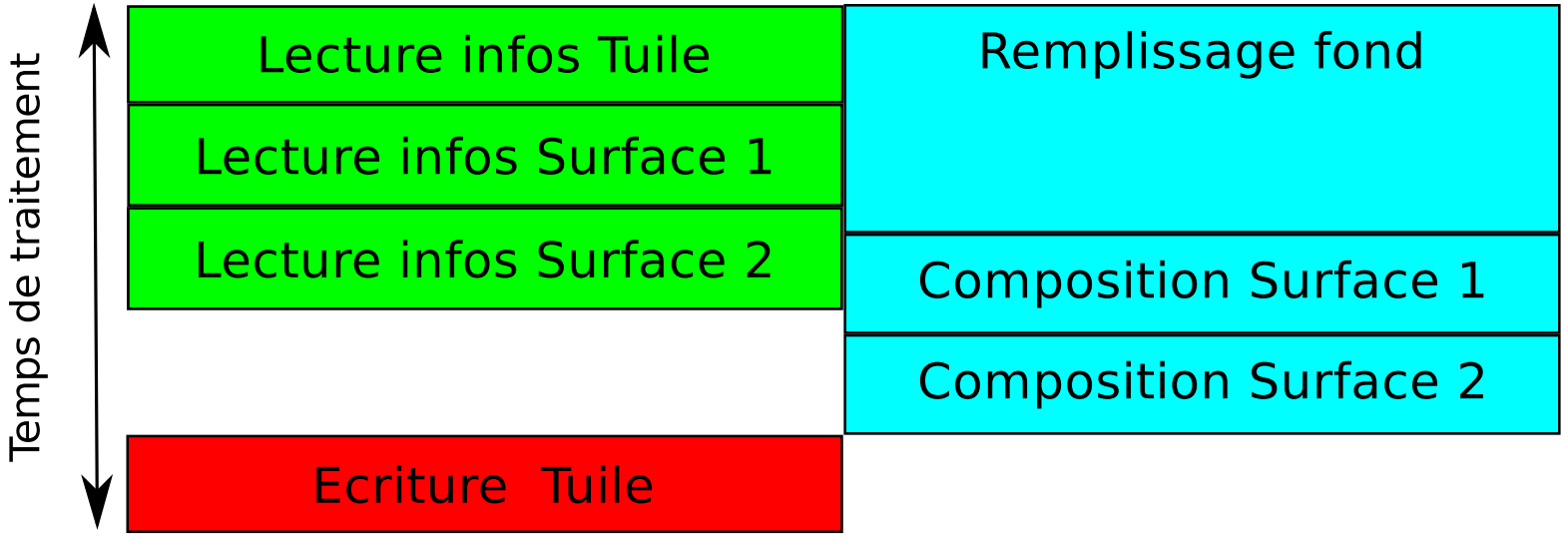
Pipeline interne du composeur



- Succession de lectures / traitements
- Peuvent être parallélisés

Pipeline interne du composeur

Parallélisation I/O et calculs



- Paralléliser les I/O avec les calculs
- Pipeline possible de "surfaces" ou de "tuiles"
- Nécessite de dupliquer les mémoires locales de surfaces ou de tuiles

Plan

Conclusions

- Estimer quantités de données transférées pour chaque image
- Estimer les débits sur les bus
- Estimer les temps de calcul nécessaires (micro-processeur, Composeur)
- Choisir un niveau de pipeline raisonnable cohérent avec ces estimations
- En déduire une architecture générale, avec une estimation des blocs mémoires nécessaires et de leur taille.