

Méthodologie d'optimisation

IF-4ARCH
2010-2011

Plan

- Objectifs & introduction
- Méthodologie
- Techniques d'optimisation
- Application : détection de lignes
 - Exemples
 - Filtre de Deriche
 - Transformée de Hough

Plan (II)

- Exemple d'optimisation d'implantation -
Filtre de Deriche
- Exemple d'optimisation d'implantation -
Transformée de Hough
- Présentation de projet
 - Détection de lignes avec la transformée de Hough
 - Contraintes d'implantation
 - Opérateurs utilisés

Objectifs & Introduction

- Implantation optimisée d'algorithmes :
 - Traitement des données en respectant des contraintes de l'application et des ressources disponibles
 - Latence
 - Cadence
 - Mémoire
 - Minimisation des composants logiciels et matériels

- Applications : sécurité, contrôle industriel, imagerie médicale, compression

- Exemple : sécurité automobile
 - Application : Système d'avertissement de dépassement des lignes blanches
 - Système : DSP embarqué

Méthodologie

□ Adéquation Algorithme – Architecture

- Optimisation algorithmique
 - Nombre d'opérations (estimation du nombre d'opérations arithmétiques)
 - Dépendance de données
 - Accès au données (linéaire lignes/colonnes, aléatoire)
 - Codage de données (int, float, fixed point)
 -
 - Techniques d'optimisation de programmation
 - Rotation de registres
 - Déroulage de boucles
 - Pipeline logiciel
 - Gestion de mémoire
 - Propriétés spécifiques du système (processeur)
 - SIMD, VLIW
 - DMA
 - Hiérarchie mémoire
- } Objectif du cours
- } Appliqué en TP WinDLX
- } Abordé en cours sur les DSP et les mémoires cache

Méthodologie d'optimisation algorithmique

- Profiling : ensemble des techniques permettant collecter des informations sur des différentes parties des applications (opérateur, fonction, partie du modèle)
 - Temps d'exécution total et par bloc
 - Nombre d'instructions
 - Arithmétiques, transferts de données, branchement
 - Mémoire allouée et nombre d'accès aux données
 - Performance des mémoires cache
- Outils de profiling
 - Logiciels libres : Gprof, Valgrind, ...
 - Intégrés aux outils de développement : CCstudio, VTune
- Optimisation commence par la partie demandant le plus de calcul (pour essayer de réduire son temps d'exécution) tout en minimisant les ressources utilisées

Méthodologie d'optimisation algorithmique

- Pour un opérateur à optimiser -> identifier « le verrou technologique » d'implantation
 - Estimer le nombre d'opérations par seconde théorique de la meilleure implantation directe de l'algorithme
 - Op. arithmétiques (MOPS, GOPS) et nombre d'accès par seconde à la mémoire en octets (bande passante demandée par algorithme)
 - Analyser adéquation des ressources disponibles et des estimations obtenues
 - MOPS et bande passante disponibles
 - Considérer les spécificités de l'architecture : parallélisme de données, parallélisme des instruction, DMA, ...
 - Mettre en œuvre une optimisation algorithmique permettant de supprimer le verrou technologique :
 - En réduisant le nombre d'op. arithmétiques et/ou accès mémoire
 - Utilisant le parallélisme de données et des instructions

Application : Implantation de la détection des lignes sur le DSP

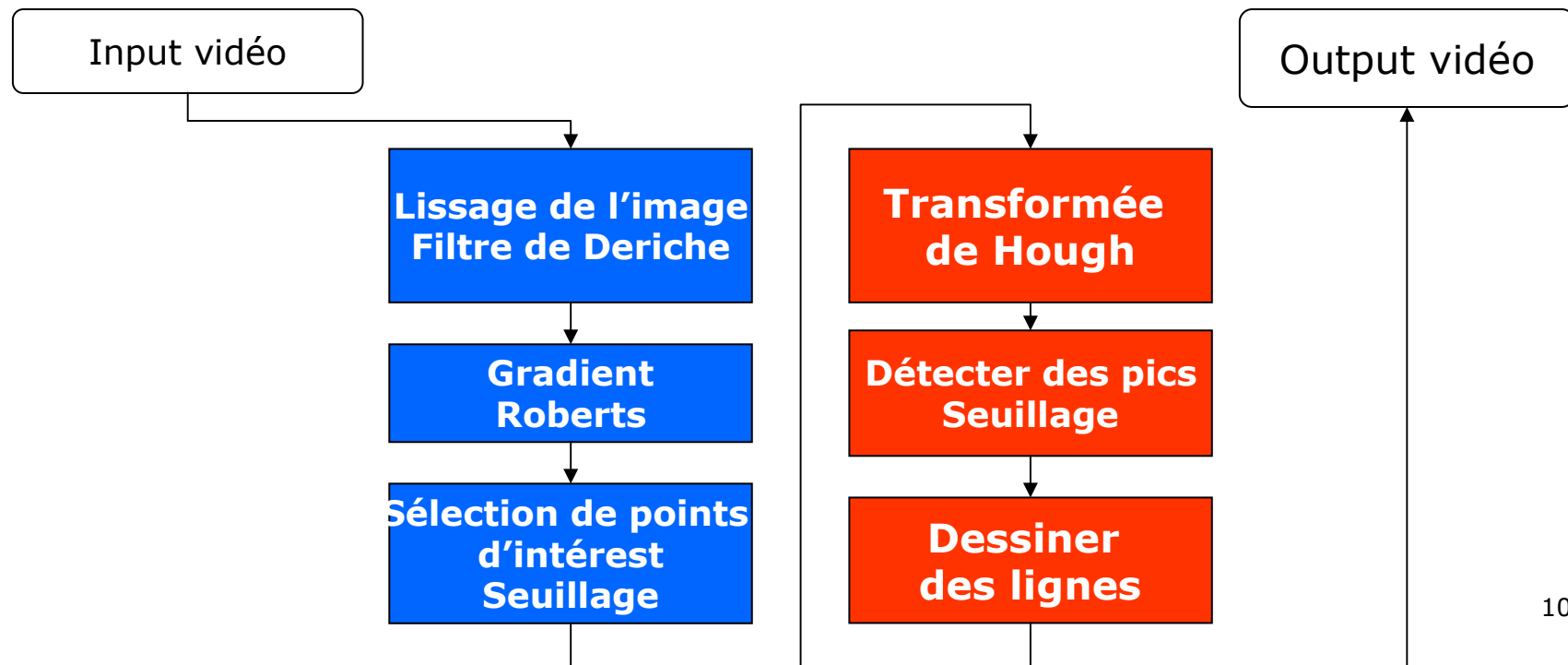
- ❑ Objectifs : implanter l'application de la détection des lignes en temps réel sur le DSP
- ❑ L'application est composée des opérateurs suivants :
 - Lissage
 - Gradient + seuil
 - Transformée de Hough
 - Détection des maxima locaux
 - Visualisation des lignes détectées
- ❑ Les étapes:
 - Calculer le nombre d'images à traiter par seconde pour satisfaire les spécifications
 - Implanter une première version fonctionnelle (sans aucune optimisation) de l'application
 - Appliquer la méthodologie et les techniques d'optimisations

Démo

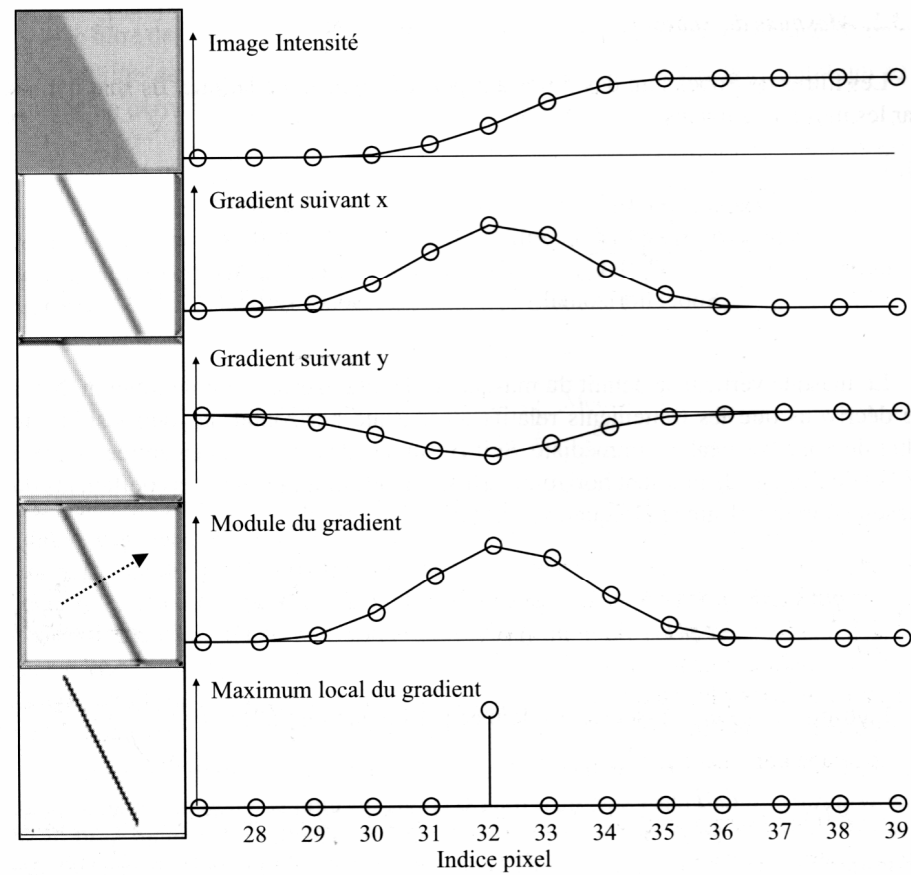
- Démo de “Lane departure warning system”
- Démo de “Détection de lignes”
 - Analyse du résultat de profiling

Présentation des opérateurs

- Objectif
 - Optimisation algorithmique



Gradient



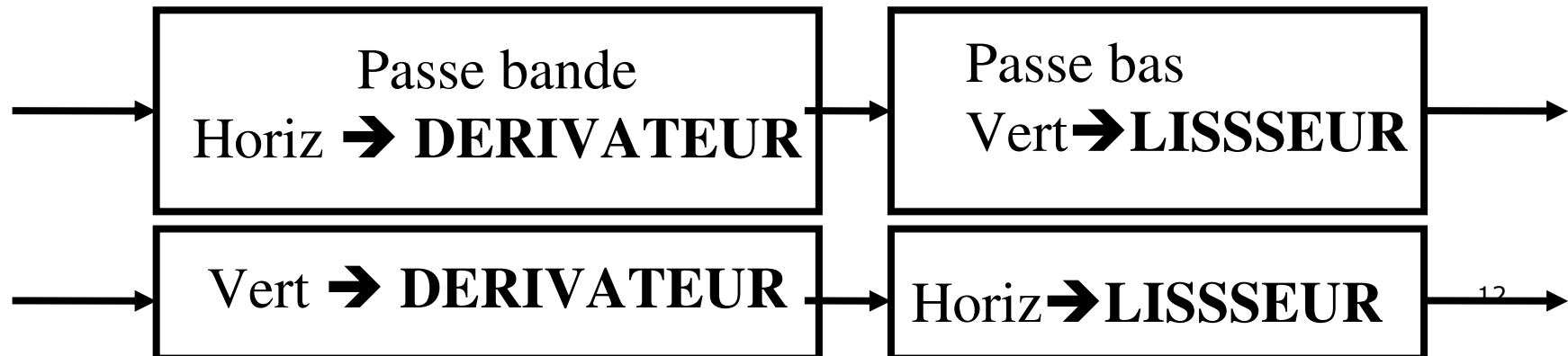
Sobel

- 2 Masques :

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
-1	-2	-1

- Horizontal : $gh(k) = p(k-N+1) - p(k-N-1) + 2p(k+1) - 2p(k-1) + p(k+N+1) - p(k+N-1)$
- $SH(z) = Gh(z) / P(Z^{-N+1} - z^{-N-1} + 2z - 2z^{-1} + z^{N+1} - z^{N-1})$
Factorisé $\rightarrow = (z - z^{-1}) (z^{-N+2} + z^N)$



Filtre de Deriche (I)

- Transformée en z de Deriche :

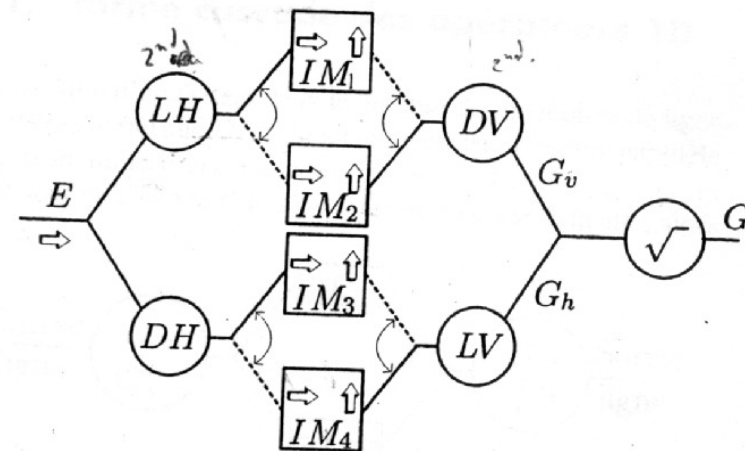
- Dérivateur

$$D(z) = k_D \left(\frac{z}{(1 - e^{-\alpha} z)^2} - \frac{z^{-1}}{(1 - e^{-\alpha} z^{-1})^2} \right)$$

- Lisseur

$$L(z) = k_L \left[\frac{(\alpha + 1)e^{-\alpha} z - e^{-2\alpha} z^2}{(1 - e^{-\alpha} z)^2} + \frac{1 + (\alpha - 1)e^{-\alpha} z^{-1}}{(1 - e^{-\alpha} z^{-1})^2} \right]$$

- Le facteur α définit **la largeur du filtre**, donc le compromis entre la détection et la localisation. Plus α est grand, plus on localise précisément le contour. Plus il est petit, plus on détecte facilement la présence des contours.



Exemples

Image originale



Image traitée avec $\alpha = 0.5$



Image traitée avec $\alpha = 1$



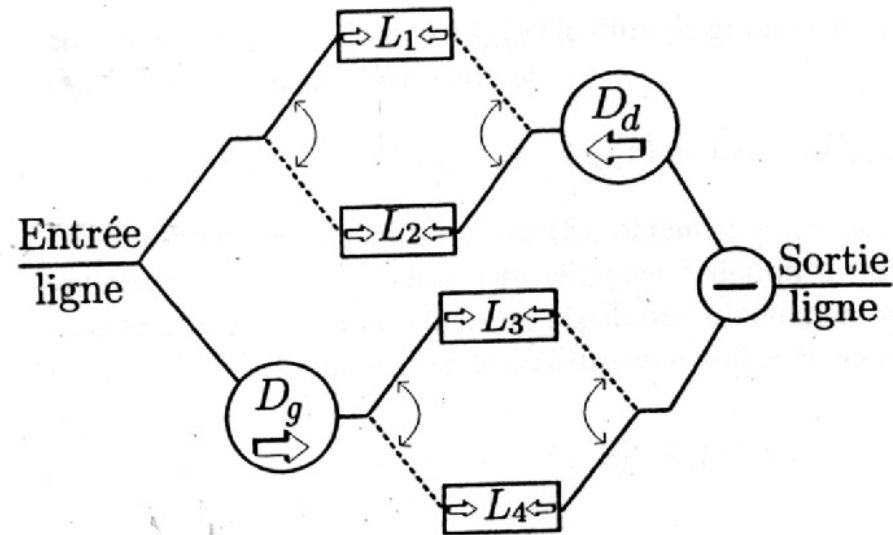
Image traitée avec $\alpha = 10$

Filtre de Deriche :

Organisation parallèle d'1 opérateur

$$D(z) = k_D \left(\underbrace{\frac{z}{(1 - e^{-\alpha}z)^2}}_{Dg} - \underbrace{\frac{z^{-1}}{(1 - e^{-\alpha}z^{-1})^2}}_{Dd} \right)$$

- 2 pôles → 2 sens de parcours : causal et anticausal



- stocker 1 ligne entière → 4 mémoire lignes en mode ping pong

Filtre de Deriche : Dérivateur (II)

- Algorithme (1 ligne, 1 direction)

Pour k de 0 à $N - 1$

$$y_m(k) = p(k - 1) + 2e^{-\alpha}y_m(k - 1) - e^{-2\alpha}y_m(k - 2)$$

Fin pour k

2 +

2 x

Pour k de $N - 1$ à 0

$$y_p(k) = p(k + 1) + 2e^{-\alpha}y_p(k + 1) - e^{-2\alpha}y_p(k + 2)$$

$$d(k) = (1 - e^{-\alpha})^2(y_p(k) - y_m(k))$$

Fin pour k

3 +

3 x

5 ADD, 5 MUL
(par pixel)

Filtre de Deriche :

Lisseur (II)

- Lisseur de même résolution que dérivateur

$$L(z) = k_L \left[\frac{(\alpha + 1)e^{-\alpha}z - e^{-2\alpha}z^2}{(1 - e^{-\alpha}z)^2} + \frac{1 + (\alpha - 1)e^{-\alpha}z^{-1}}{(1 - e^{-\alpha}z^{-1})^2} \right]$$

- Algorithme

$$y_m(-2) = y_m(-1) = p(-1) = 0$$

Pour k de 0 à $N - 1$

$$y_m(k) = p(k) + (\alpha - 1)e^{-\alpha}p(k - 1) + 2e^{-\alpha}y_m(k - 1) - e^{-2\alpha}y_m(k - 2)$$

Fin pour k

$$y_p(N + 1) = y_p(N) = a y_m(N - 1) \text{ et } p(N) = p(N + 1) = b y_m(N - 1)$$

Pour k de $N - 1$ à 0

$$y_p(k) = (\alpha + 1)e^{-\alpha}p(k + 1) - e^{-2\alpha}p(k + 2) + 2e^{-\alpha}y_p(k + 1) - e^{-2\alpha}y_p(k + 2)$$

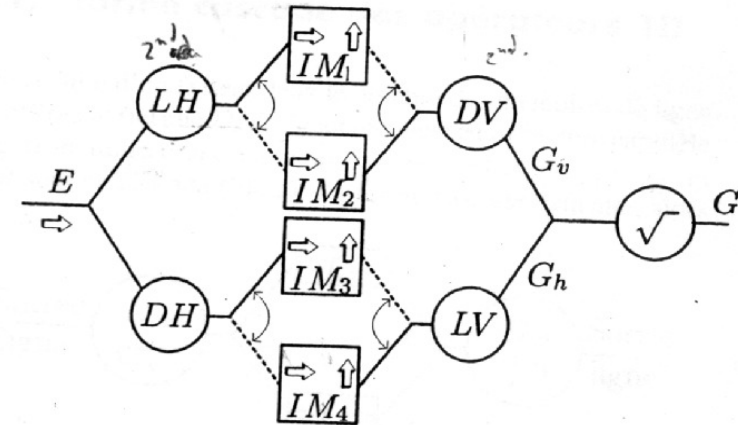
$$l(k) = \frac{(1 - e^{-\alpha})^2}{1 + 2\alpha e^{-\alpha} - e^{-2\alpha}} (y_p(k) + y_m(k))$$

Fin pour k

7 ADD, 8 MUL

Deriche 2D

- $GH(z) = D(z)L(z^N)P(z)$
- $GV(z) = D(z^N)L(z)P(z)$



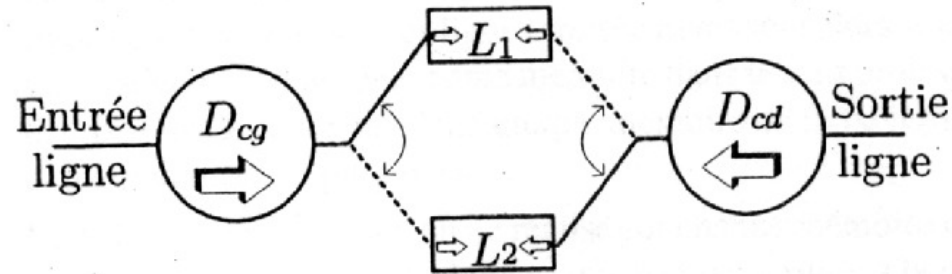
→ ordre sans importance mais important : ex dans G finir D avant L

- NB opérations par pixel : 26 MUL et 24 ADD
- Exemple :
 - 25 fps 640x480 => 384 MOPS
 - 25 fps 1920x1080 => 2,6 GOPS
- 4 mémoires images + 16 mémoires ligne

Optimisation Garcia-Lorca

Cascade des opérateurs 1D

- Objectif : réduire le nombre de mémoires de ligne par opérateur



- Applicable aussi au lisseur mais pas intéressant

Optimisation Garcia-Lorca

Cascade des opérateurs 1D (II)

- Nouvelle expression du filtre dérivateur

$$\left. \begin{aligned} D_c(z) &= D_{cg}(z) \cdot D_{cd}(z) \\ D_{cg}(z) &= \frac{(1-\gamma)^2 \cdot z^{-1}}{(1-\gamma z^{-1})^2} \\ D_{cd}(z) &= \frac{(1-\gamma^2) \cdot (z^2 - 1)}{(1-\gamma z)^2} \end{aligned} \right\} \gamma = e^{-\alpha}$$

- Nouvelle expression du filtre lisseur obtenue par la méthode des trapèze

$$\begin{aligned} L_c(z) &= L_{cg}(z) \cdot L_{cd}(z) \\ L_{cg}(z) &= \frac{(1-\gamma)^2(1+z^{-1})}{2(1-\gamma z^{-1})^2} \quad \text{et} \quad L_{cd}(z) = L_{cg}(z^{-1}) \end{aligned}$$

Garcia Lorca : optimisation

□ On obtient

$$G_h(z) = L_c(z^N) \cdot D_c(z) \quad \text{et} \quad G_v(z) = L_c(z) \cdot D_c(z^N)$$

□ Factorisation des parties non récurrentes

$$\begin{aligned} (z - z^{-1}) &= (1 - z^{-1})(1 + z) \\ (z + 2 + z^{-1}) &= (1 + z^{-1})(1 + z) \end{aligned}$$

Lisseur et
Dérivateur
Garcia Lorca

$$Dgl(z) = kgl_D (1 - z^{-1})$$

$$Lgl(z) = kgl_L (1 + z^{-1})$$

$\frac{1}{(1 - \gamma z)^2}$	$\frac{1}{(1 - \gamma z^{-1})^2}$
$\frac{1}{(1 - \gamma z)^2}$	$\frac{1}{(1 - \gamma z^{-1})^2}$
LGL(z)	

Garcia Lorca : gradient 2D

- $GH(z) = Dgl(z)Lgl(z^N)P(z)$
- $DH(z) = Dgl(z^N)Lgl(z)P(z)$

□ En posant : $LGL(z) = \frac{1}{(1-\gamma z)^2} \frac{1}{(1-\gamma z^{-1})^2} \quad kgl = \frac{(1-\gamma)^4(1-\gamma^2)^3}{2(1+\gamma^2)}$

- On obtient :

$$\begin{aligned} GH(z) &= kgl LGL(z) LGL(z^N) \begin{bmatrix} (1-z^{-1}) & (1+z^{-N}) \\ (1-z^{-N}) & (1+z^{-1}) \end{bmatrix} P(z) \\ GV(z) &= kgl LGL(z) LGL(z^N) \begin{bmatrix} (1-z^{-1}) & (1+z^{-N}) \\ (1-z^{-N}) & (1+z^{-1}) \end{bmatrix} P(z) \end{aligned}$$

□ On pose :

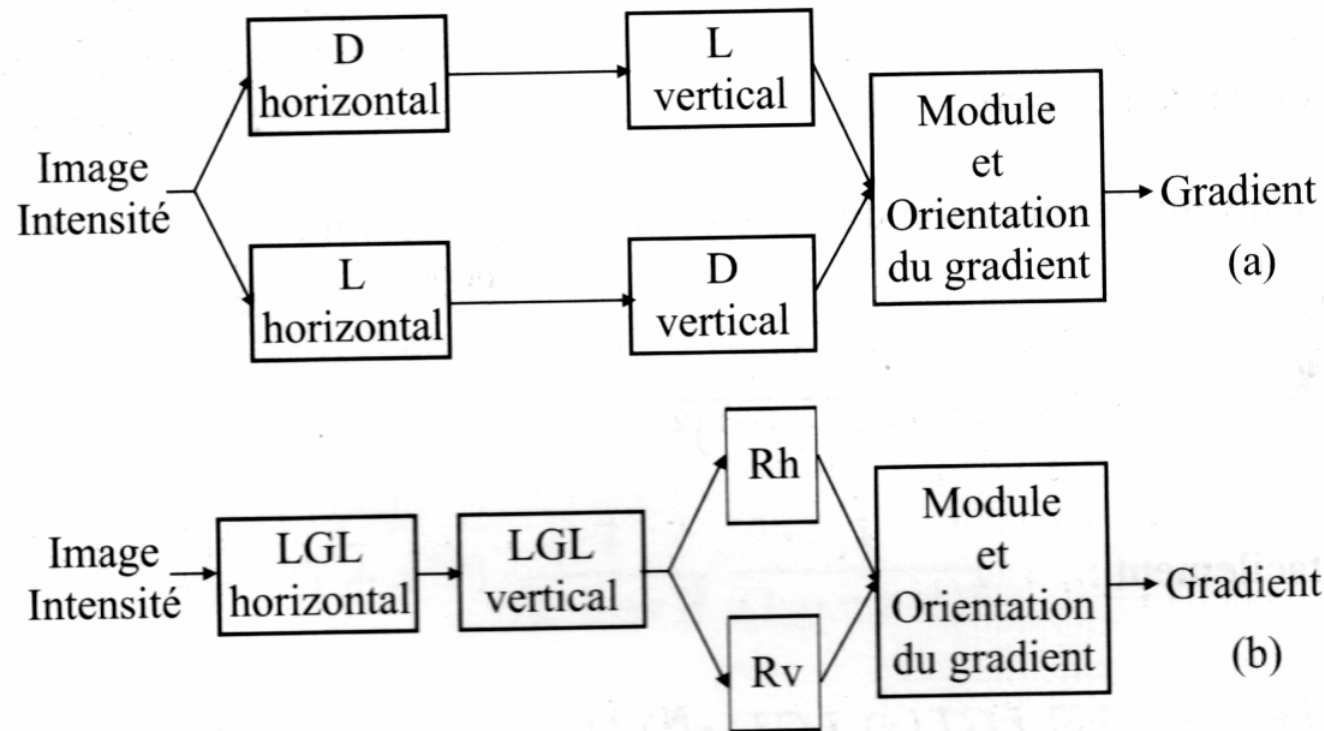
$$\begin{aligned} R_h(z) &= \begin{pmatrix} (1-z^{-1}) & (1+z^{-N}) \\ (1-z^{-N}) & (1+z^{-1}) \end{pmatrix} \\ R_v(z) &= \begin{pmatrix} (1-z^{-1}) & (1+z^{-N}) \\ (1-z^{-N}) & (1+z^{-1}) \end{pmatrix} \end{aligned}$$

- R_h et $R_v =$ filtres locaux : masques

-1	1
-1	1

-1	-1
1	1

Garcia Lorca : gradient 2D



Complexité de D et L équivalente à LGL et donc gain de 50%

Garcia Lorca : gradient 2D

Filtrage Horizontal : 4 x, 4 +

Pour i de 0 à $N - 1$ (chaque ligne)
 $lh(i, -2) = lh(i, -1) = 0$
Pour k de 0 à $N - 1$
 $lh(i, k) = p(i, k) + 2\gamma lh(i, k - 1) - \gamma^2 lh(i, k - 2)$
Fin pour k

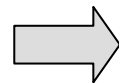
$lh(i, N + 1) = lh(i, N) = lh(i, N - 1)$
Pour k de $N - 1$ à 0
 $lh(i, k) = p(i, k) + 2\gamma lh(i, k + 1) - \gamma^2 lh(i, k + 2)$
Fin pour k
 $lh(i, -2) = lh(i, -1) = lh(i, 0)$
Pour k de 0 à $N - 1$
 $lh(i, k) = lh(i, k) + 2\gamma lh(i, k - 1) - \gamma^2 lh(i, k - 2)$
Fin pour k

Fin pour i

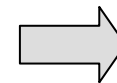
Dérivation locale : 4 +

Pour j de 1 à $N - 1$ (chaque colonne)
Pour i de 1 à $N - 1$ (chaque ligne)
 $th(i, j) = lv(i, j) - lv(i, j - 1)$
 $tv(i, j) = lv(i, j) + lv(i, j - 1)$
 $gh(i, j) = th(i, j) + th(i - 1, j)$
 $gv(i, j) = tv(i, j) - tv(i - 1, j)$
Fin pour i

Fin pour j



9 MUL
12 ADD



25 fps
640x480 =>
161 MOPS

25 fps
1920x1080 =>
1,1 GOPS

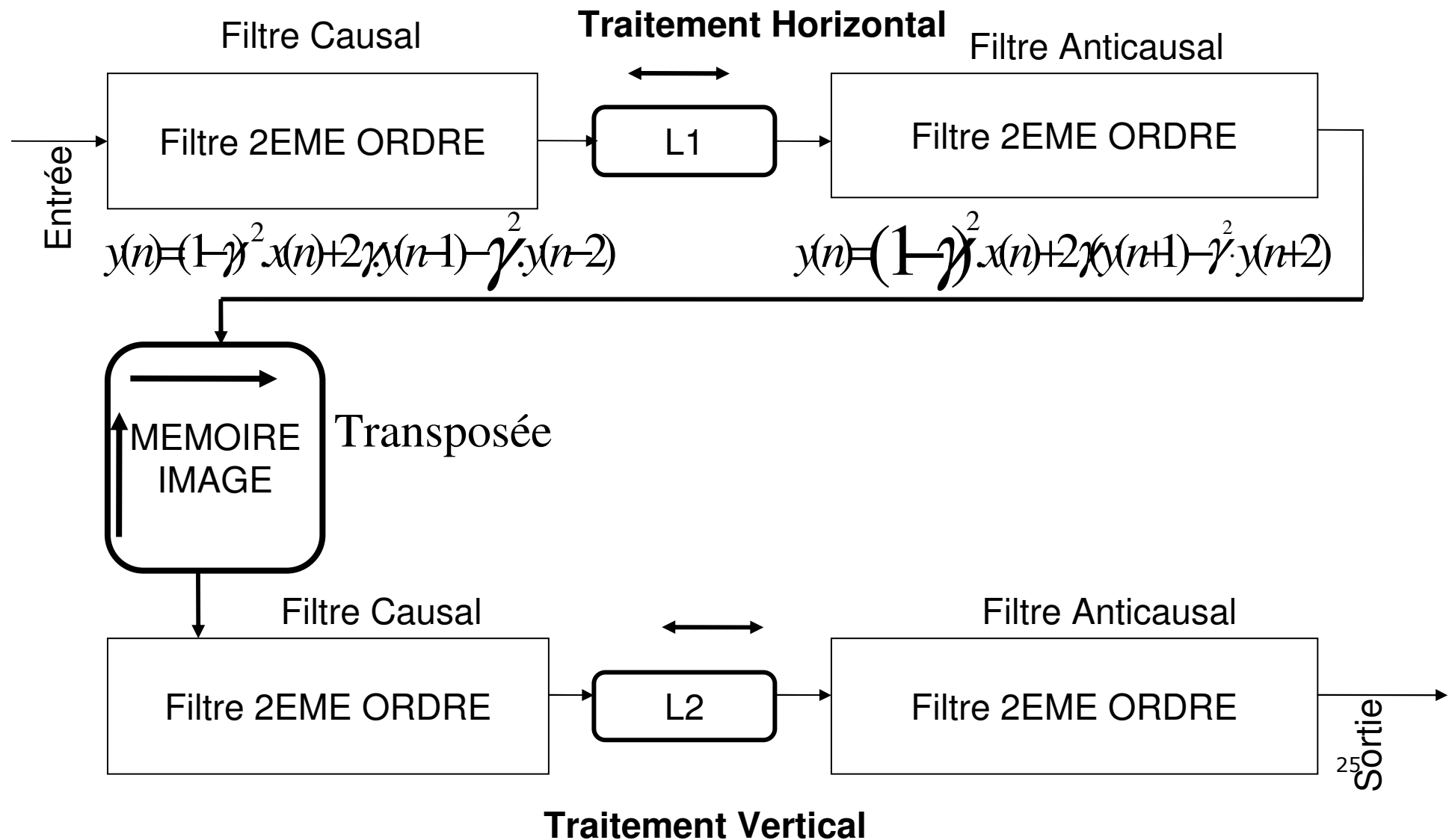
Filtrage Vertical : 5 x, 4+

Pour j de 0 à $N - 1$ (chaque colonne)
 $lv(-2) = lv(-1) = 0$
Pour k de 0 à $N - 1$
 $lv(k) = lh(k) + 2\gamma lv(k - 1) - \gamma^2 lv(k - 2)$
Fin pour k
 $lv(N + 1) = lv(N) = lv(N - 1)$

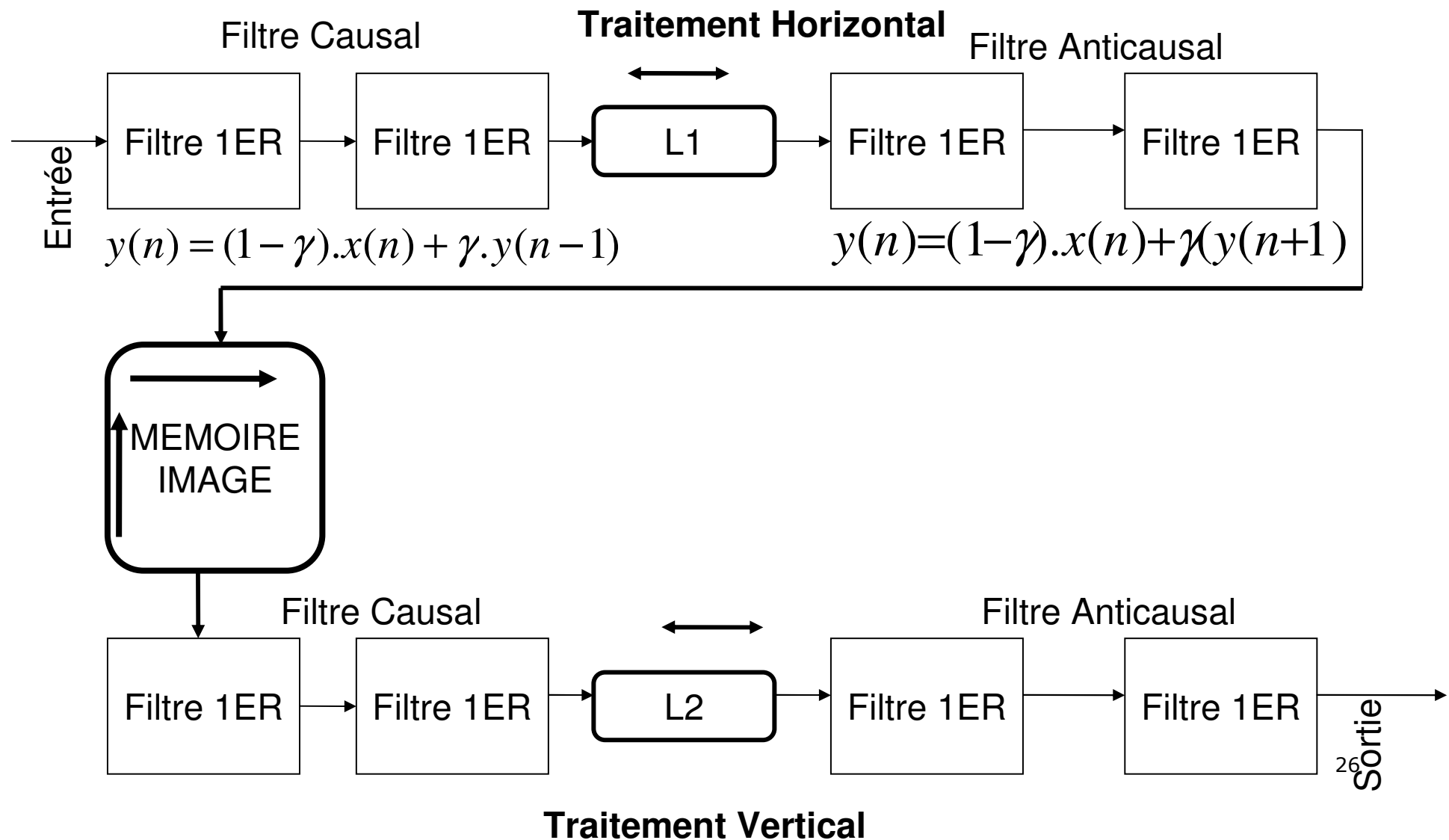
Pour k de $N - 1$ à 0
 $lv(k) = lh(k) + 2\gamma lv(k + 1) - \gamma^2 lv(k + 2)$
Fin pour k
 $lv(-2) = lv(-1) = lv(0)$
Pour k de 0 à $N - 1$
 $lv(k) = kgl lv(k) + 2\gamma lv(k - 1) - \gamma^2 lv(k - 2)$
Fin pour k

Fin pour j

Filtre de G. Lorca



Filtre de G. Lorca



Filtre de Deriche

Précision de calculs

- Filtre utilise une seule constante : γ
 - Codage des valeurs : définit le nombre de filtres différents réalisables

- pour codage sur 3 bits : $\gamma = \gamma_1 2^{-1} + \gamma_2 2^{-2} + \gamma_3 2^{-3}$
- Il est possible de réaliser 7 filtres différents

γ	α	résolution
0.125	2.08	1
0.250	1.38	2
0.375	0.98	3
0.500	0.69	4
0.625	0.47	5
0.750	0.29	8
0.875	0.13	18

- Précision de calculs

- Définit la taille des registres, des opérateurs et la largeur de la mémoire
- Pour filtre de Deriche, les calculs doivent être fait en 21 bits et le stockage en 12 bits

- Remarque :

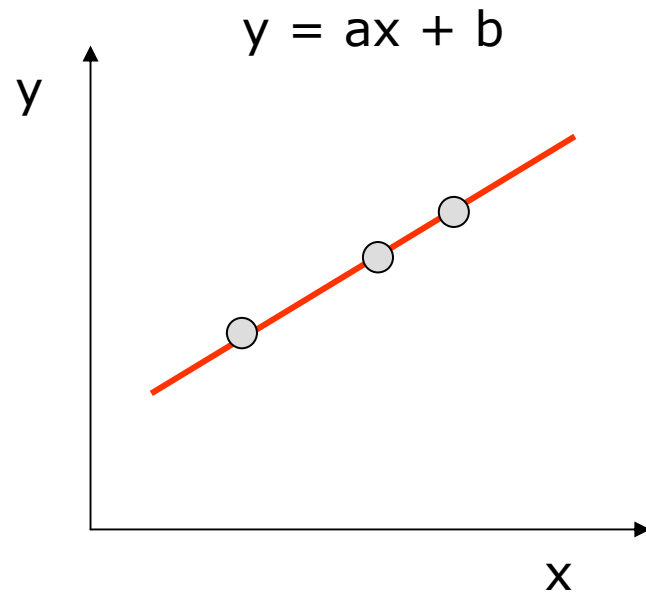
- Sur des processeurs modernes les calcul en entier de 32 bits peuvent être plus rapides que des calcul en 8 bits; les calculs en virgule flottante peut être plus rapide que en entier
- Dans le cas des architectures avec unités de calcul en entier et l'unité de calcul en flottant, les calculs en flottant peuvent permettre de décharger l'unité de calcul en entier. Ceci peut être « payé » par conversion du entier flottant si elle est nécessaire

Transformée de Hough

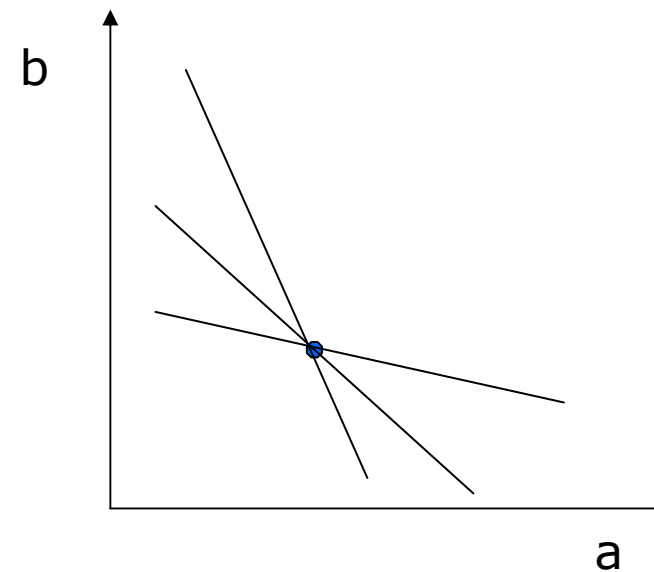
- ❑ Technique de reconnaissance de formes (1962)
- ❑ Description paramétrique: droites, cercles, ellipses
- ❑ Bonne insensibilité au bruit
- ❑ Permet de détecter des objets partiellement recouverts

Transformée de Hough : principe

□ Detection de droites

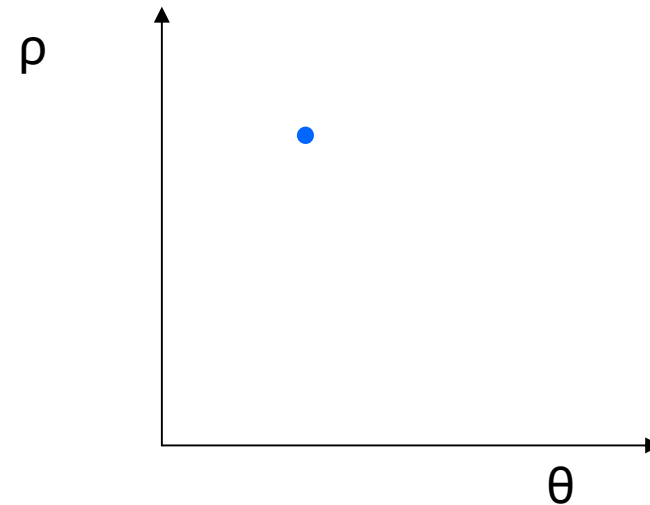
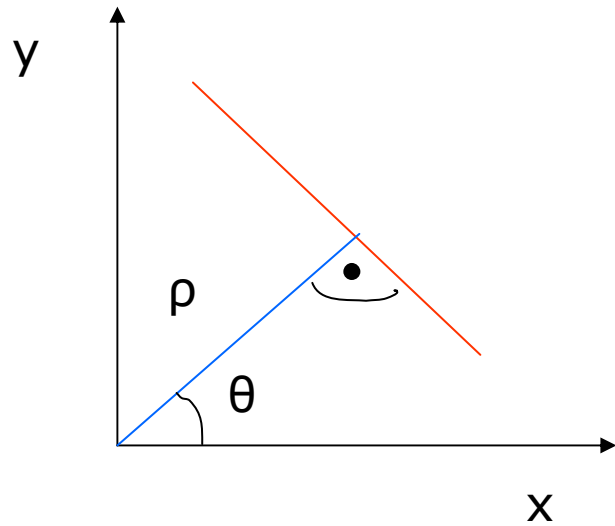


Espace "image"



Espace "paramètres"

Transformée de Hough : espace (ρ, θ)



$$x = \rho \cos(\theta)$$
$$y = \rho \sin(\theta)$$

$$x \cos(\theta) + y \sin(\theta) - \rho = 0$$

Les sinusoides correspondant aux points d'une droite se coupent au point paramétrisant cette droite

Transformée de Hough

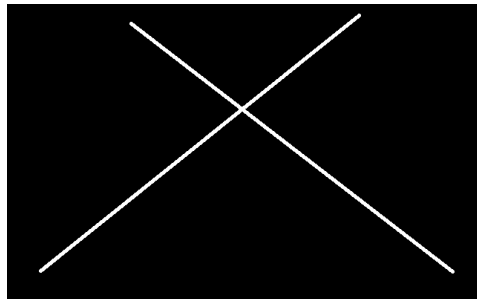
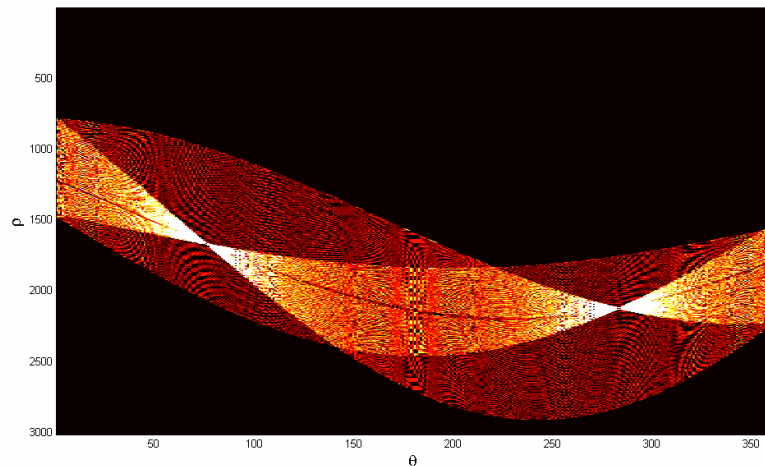
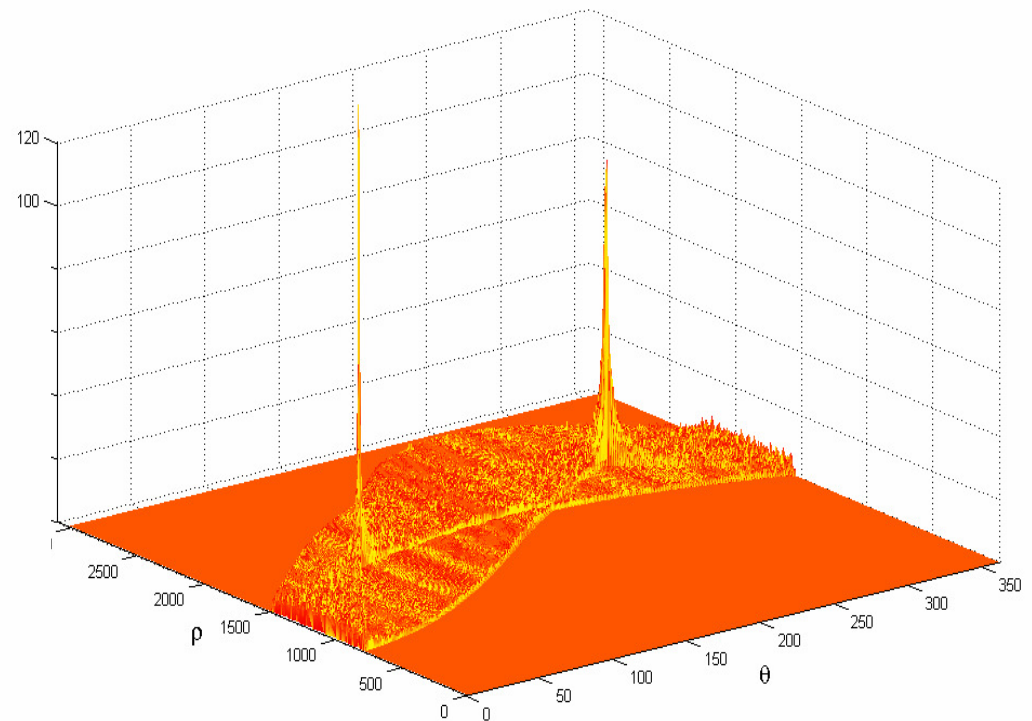


Image originale



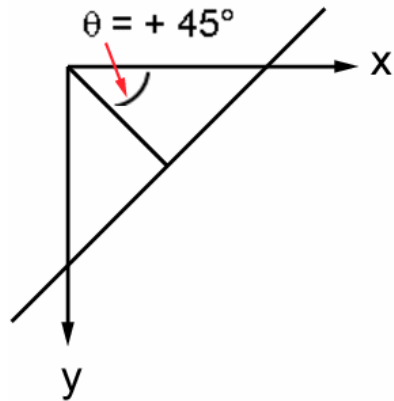
Vue 2D de
l'espace de Hough (accumulateur)



Vue 3D de l'espace de Hough

Transformée de Hough : algorithme « classique » (I)

- Discrétiser l'espace de Hough (accumulateur)
 - Discrétiser paramètre (selon l'illustration ci-dessous) :
 - $\theta \in [-90, 90)$, si l'angle exprimé en degré
 - Discrétiser paramètre
 - $\rho \in [0, d]$, où d est la taille de la diagonale



L'angle de la ligne est : $90 + \theta$

- Allouer et initialiser à zéro l'accumulateur $A(\rho, \theta)$

Transformée de Hough :

algorithme « classique » (II)

- Soit N la collection des points sur les contours, pour chaque point $p(x,y)$ faire :
 - Pour tous les $\theta \in [-90, 90)$ calculer ρ :
 - $\rho = x \cos(\theta) + y \sin(\theta)$
 - $A(\rho, \theta) = A(\rho, \theta) + 1$

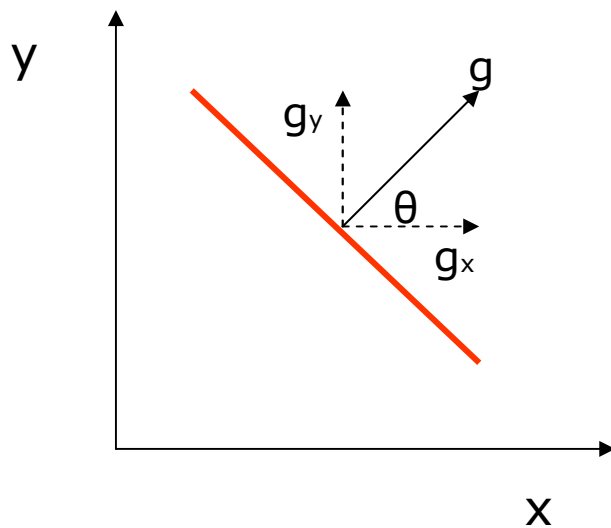
- Complexité arithmétique
 - Hypothèse : θ discrétisé avec un $\Delta = 1$

 - pour $\forall p(x,y) \in N$ calculer 180 fois $\rho = x \cos(\theta) + y \sin(\theta)$
 - si 20% pixels de l'image à calculer :
 - 25 fps 640*480 : 829 MOPS + $1,6 \times 10^6$ sin,cos
 - 25 fps 1920*1080 : 5,6 GOPS + $10,4 \times 10^6$ sin,cos

- Remarques :
 - 180 accès aléatoires à la mémoire par point traité
 - 180 calculs des fonctions trigonométriques par point traité

Transformée de Hough : algorithme de O'Gorman et Clowes (I)

- Principe : utiliser le gradient local pour réduire le nombre de « votes »
 - La direction du gradient local donne une estimation de θ



Transformée de Hough :

algorithme de O'Gorman et Clowes (II)

- Pour chaque point $p(x,y) \in N$ faire :
 - $\theta = \arctan (g_y/g_x)$, où g_x, g_y sont des éléments du gradient
 - $\rho = x \cos(\theta) + y \sin(\theta)$
 - $A(\rho, \theta) = A(\rho, \theta) + 1$

- Complexité arithmétique
 - 1 calcul du sin et cos par point
 - Mais : le calcul d'une division et d'une arctan ajoutési 20% pixels de l'image à calculer :
 - 25 fps 640*480 : 6,1 MOPS + $1,5 \times 10^6$ sin,cos,arctan
 - 25 fps 1920*1080 : 5,6 GOPS + $10,4 \times 10^6$ sin,cos,arctan

- Remarques :
 - moins de valeurs accumulées permet de rendre les pics des maxima plus « visibles »
 - 1 accès mémoire par point

Transformée de Hough : pour aller plus loin ...

- Calcul des fonctions trigonométriques
 - Utiliser des bibliothèques mathématiques
 - Ne sont pas toujours optimales en temps calcul
 - Ne sont pas adaptées à des différentes précisions de calcul

 - Autres possibilités
 - LUT
 - CORDIC

Calcul des fonctions trigonométriques

LUT

- ❑ LUT est une table de correspondance qui associe une sortie à une entrée
- ❑ LUT permet de remplacer des calculs complexes par 1 accès à une structure de données (tableau)
- ❑ Principe : précalculer la fonction dans un intervalle et avec une précision définie et les stocker dans une structure de données
 - Peut être fait à chaque fois au début du programme
 - Peut être fait une fois par un programme séparé et défini de façon « fixe » dans le programme « utilisateur ».

Calcul des fonctions trigonométriques

LUT

- Discrétiser l'intervalle $0 \leq x \leq 360$ pour obtenir la précision souhaitée
 - Si la discrétisation avec $\Delta = 1$, les degrés sont des indices du tableau
 - Trouver un compromis taille du LUT - précision
- Définir la précision des valeurs (int, float, ...)

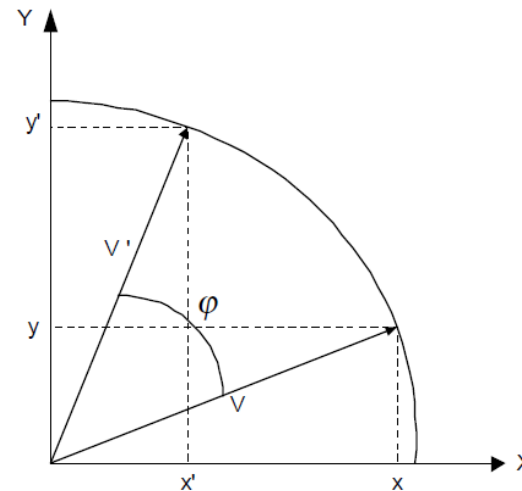
- Allouer le tableau $\text{LUT}[360/\Delta]$
- Initialiser le tableau
 - Pour $i=0, i \leq 360, i=i+\Delta$ faire
 - $\text{LUT}[i] = \sin(i)$

Calcul des fonctions trigonométriques : CORDIC

- ❑ CORDIC = COordinate Rotation DIgital Computer (1971)
 - Utilisé par les calculatrices, populaire pour une implantation FPGA
- ❑ Principe : calcul itératif, nombre d'itérations détermine la précision du résultat
 - Par les rotations successives du vecteur v , nous cherchons sa coordonnée x, y sur le cercle unité

$$\begin{aligned}x' &= \cos(\varphi) [x - y \tan(\varphi)] \\y' &= \cos(\varphi) [y + x \tan(\varphi)] \\a_{i+1} &= a_i - d_i \cdot \tan^{-1}(2^{-i})\end{aligned}$$

$$d_i = \pm 1$$



- ❑ Avantages : limite les accès aux données et minimise la mémoire allouée

Calcul des fonctions trigonométriques : CORDIC

□ Exemple (C)

```
// Initialisation des variables
a = 25; // Angle initiale
x=0.607252951;
y=0;
d2=2; // Diviseur

for(i=0; i<=10; i++)
{
    d2/=2; // Multiple de 2-i
    dx=x*d2;
    dy=y*d2;
    da=atan(d2);
    da= 180*da/PI; // Pour une valeur en degré

    if(a<0)
    {
        x += dy;
        y -= dx;
        a += da;
    }
    else
    {
        x -= dy;
        y += dx;
        a -= da;
    }
}
```

Projet

- Objectifs
- Calculez le nombre des images à traiter :
 - Vigilance = la capacité à répondre de façon adaptée à un évènement. Elle est mesurée par le temps de réaction : durée entre l'instant où le signal atteint la rétine et celui où nous faisons le geste nécessaire.
 - Le temps de réaction moyen = 1s
 - À 50km/h, 1s représente 15m parcourus
 - A 90km/h, 1s représente 27m parcourus