Virgule fixe

S. Mancini

Plan

- ✖ Rappels de représentation des nombres
- ❑ Opérateurs arithmétiques fondamentaux
- ❑ Virgule fixe et bruit de calcul
- ❑ TD: 3D Z-buffer

Nombres entiers et Virgule flottante

- Nombre entiers:

$$a = \sum_{i=0}^{n-1} a_i 2^i$$

- Nombres en virgule flottante:

$$a = (-1)^s .1, M_a .2^{(E_a - offset)}$$

Norme IEEE 754:

	Encodage	Signe	Exposant	Mantisse	Valeur d'un nombre
Simple précision	32 bits	1 bit	8 bits	23 bits	$(-1)^S \times 1, M \times 2^{(E-127)}$
Double précision	64 bits	1 bit	11 bits	52 bits	$(-1)^S \times 1, M \times 2^{(E-1023)}$

Le cours d'Alain Guyot est souvent repris in-extenso !

Plan

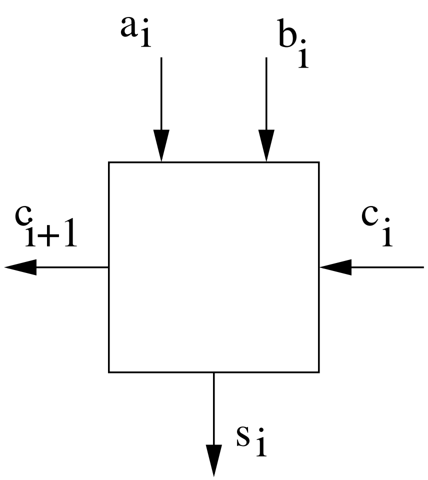
- ❑ Rappels de représentation des nombres
- ✖ Opérateurs arithmétiques fondamentaux
- ❑ Virgule fixe et bruit de calcul
- ❑ TD: 3D Z-buffer

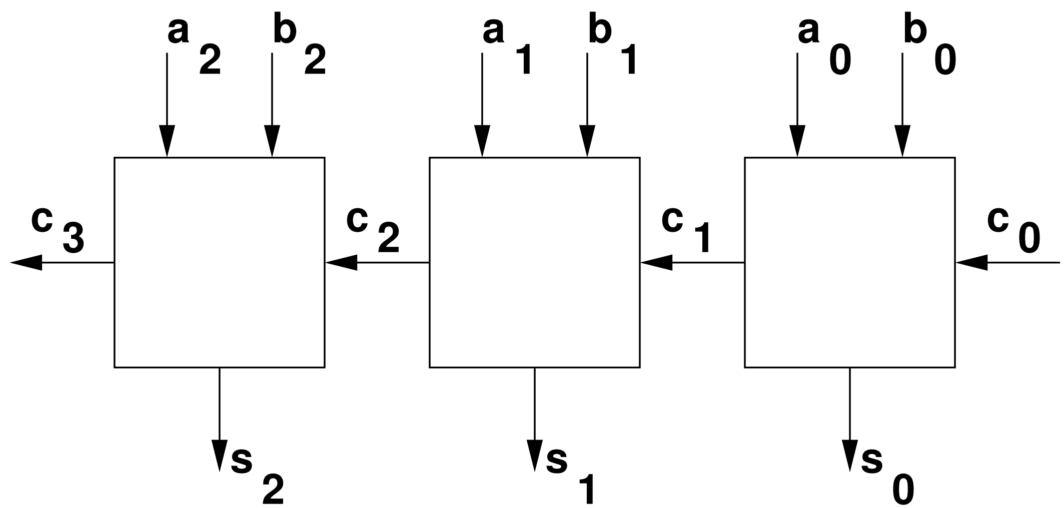
Addition - Rappel

$$
\begin{aligned}
s &= a + b \\
&= \sum_{i=0}^{n-1} (a_i + b_i) 2^i
\end{aligned}
$$

Implémentation naïve par propagation de retenue:

$$
\begin{aligned}
s_i &= a_i \oplus b_i \oplus c_{i-1} \\
c_i &= (a_i . b_i) + (a_i . c_{i-1}) + (c_{i-1} . b)
\end{aligned}
$$





Addition - compléments

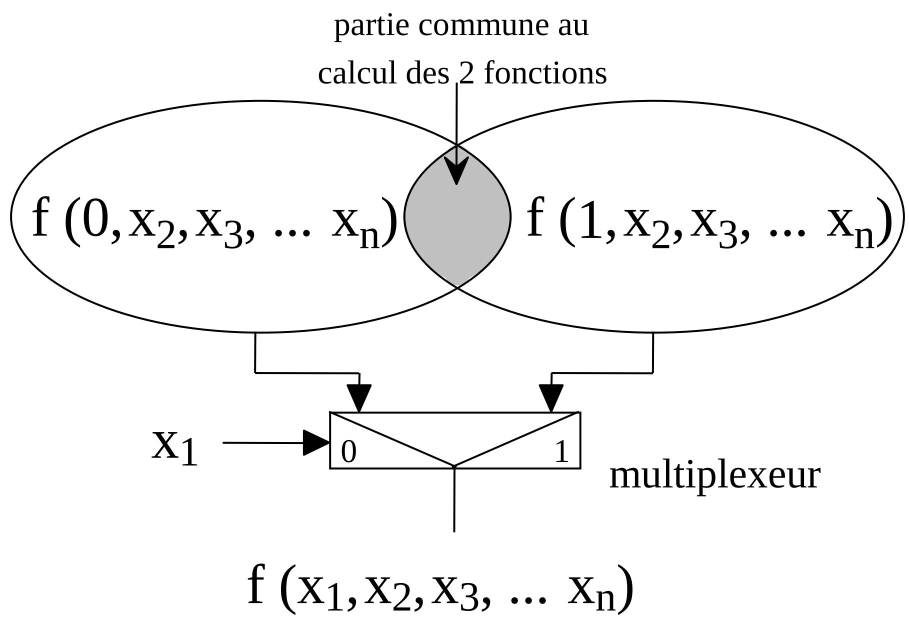
S. Mancini

Écriture de Shannon

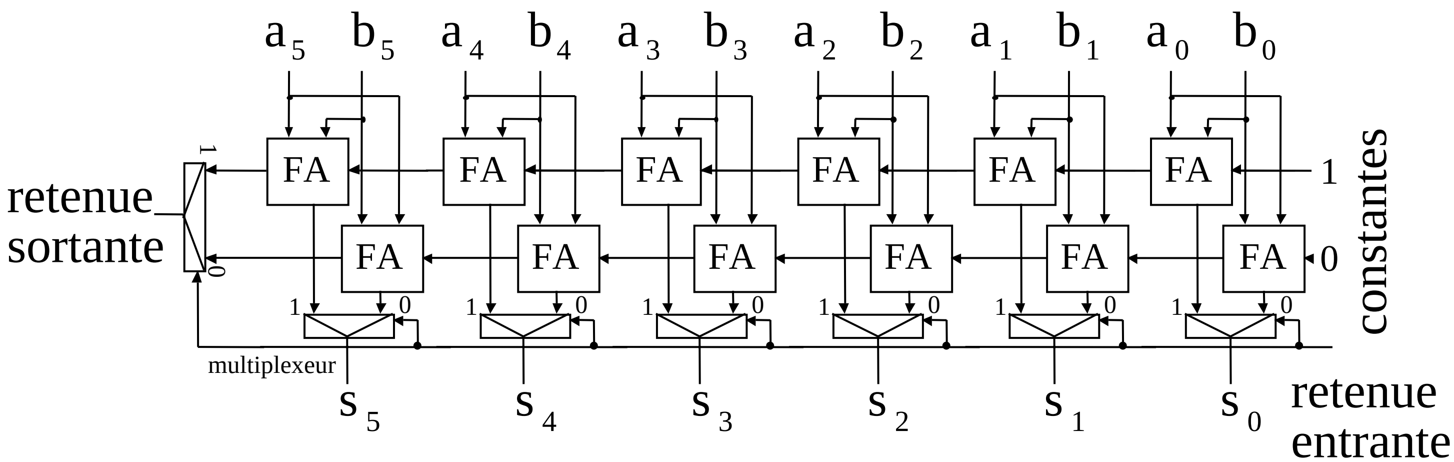
On a à réaliser $f\,(x_1, x_2, x_3, \ldots\ x_n)$

on veut disposer de temps pour calculer x_1

$\Rightarrow$ on précalcule $f\,(0, x_2, x_3, \ldots\ x_n)$ et $f\,(1, x_2, x_3, \ldots\ x_n)$



“Carry select adder”

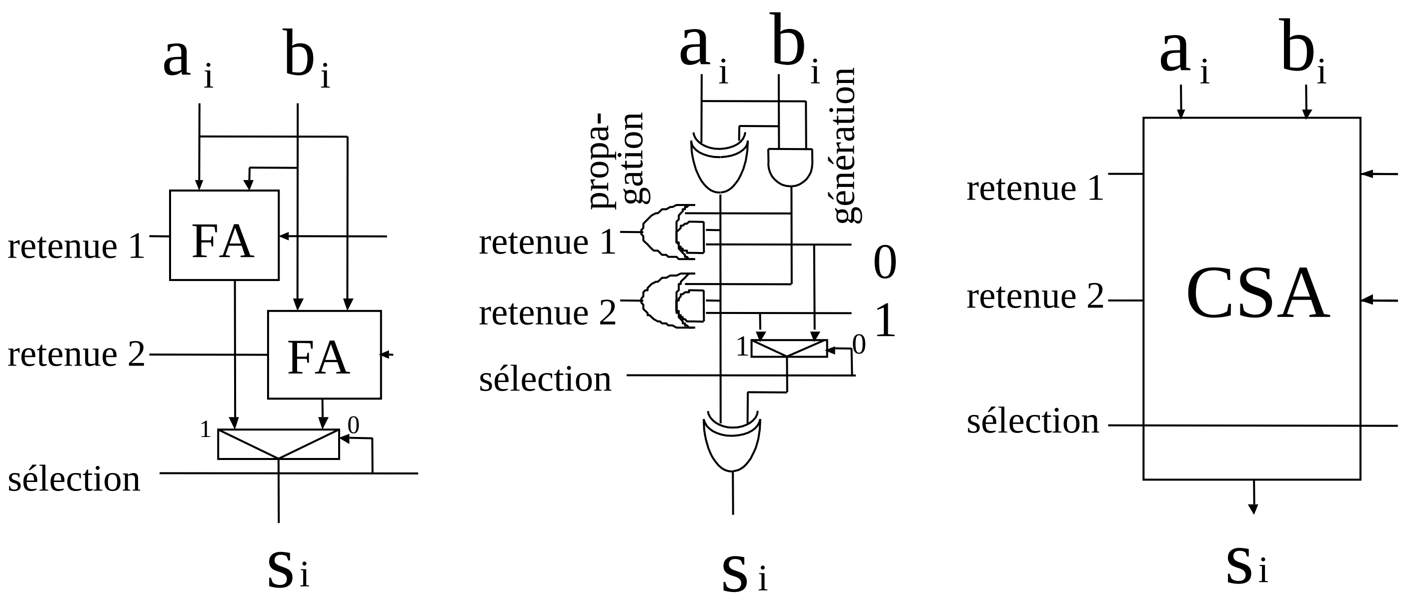


La retenue entrante ne se propage pas à travers les FA

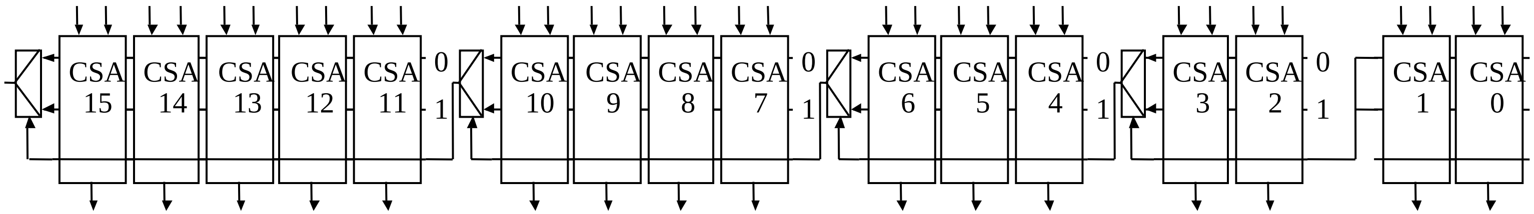
⇒ on dispose de temps pour la calculer

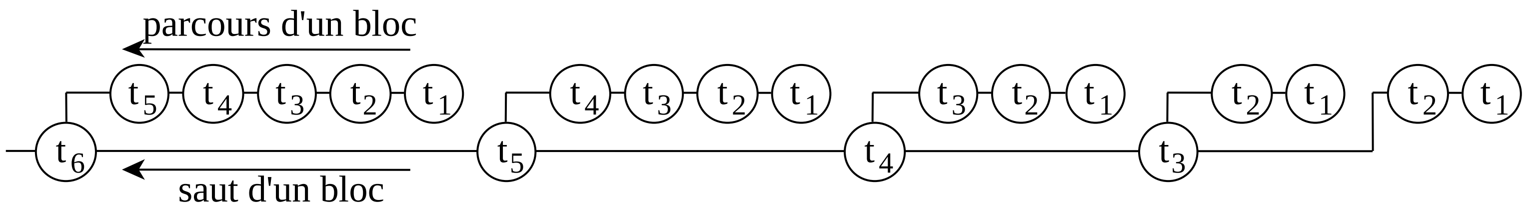
⇒ à mettre en poids forts la où la retenue est en retard

Cellule de carry select adder



Additionneur en temps $\sqrt{n}$





$$n = \sum_{i=0}^{\tau-2} i = \frac{\tau(\tau+1)}{2} + 1 \qquad 2n = \tau^2 + \tau + 2 \Rightarrow \tau = \frac{\sqrt{8n-7}-1}{2} \approx \sqrt{2n} \approx 1{,}4\sqrt{n}$$

Addition - Performance

Type d'addition	# de Δ-cells	Délai (Δ -cell)	Max. fan-out	Exemple n = 32 bits		
Propagation	$n-1$	$n-1$	2	31	31	2
2-level carry select	$\lceil 2n - \sqrt{2} \rceil$	$\lceil \sqrt{2n} \rceil$	$\lceil \sqrt{2n} \rceil$	54	8	6
3-level carry select	$3n??$	$\lceil \sqrt[3]{6n} \rceil$	$\lceil \sqrt[3]{6n} \rceil$	66	6	9

Multiplication - Rappel

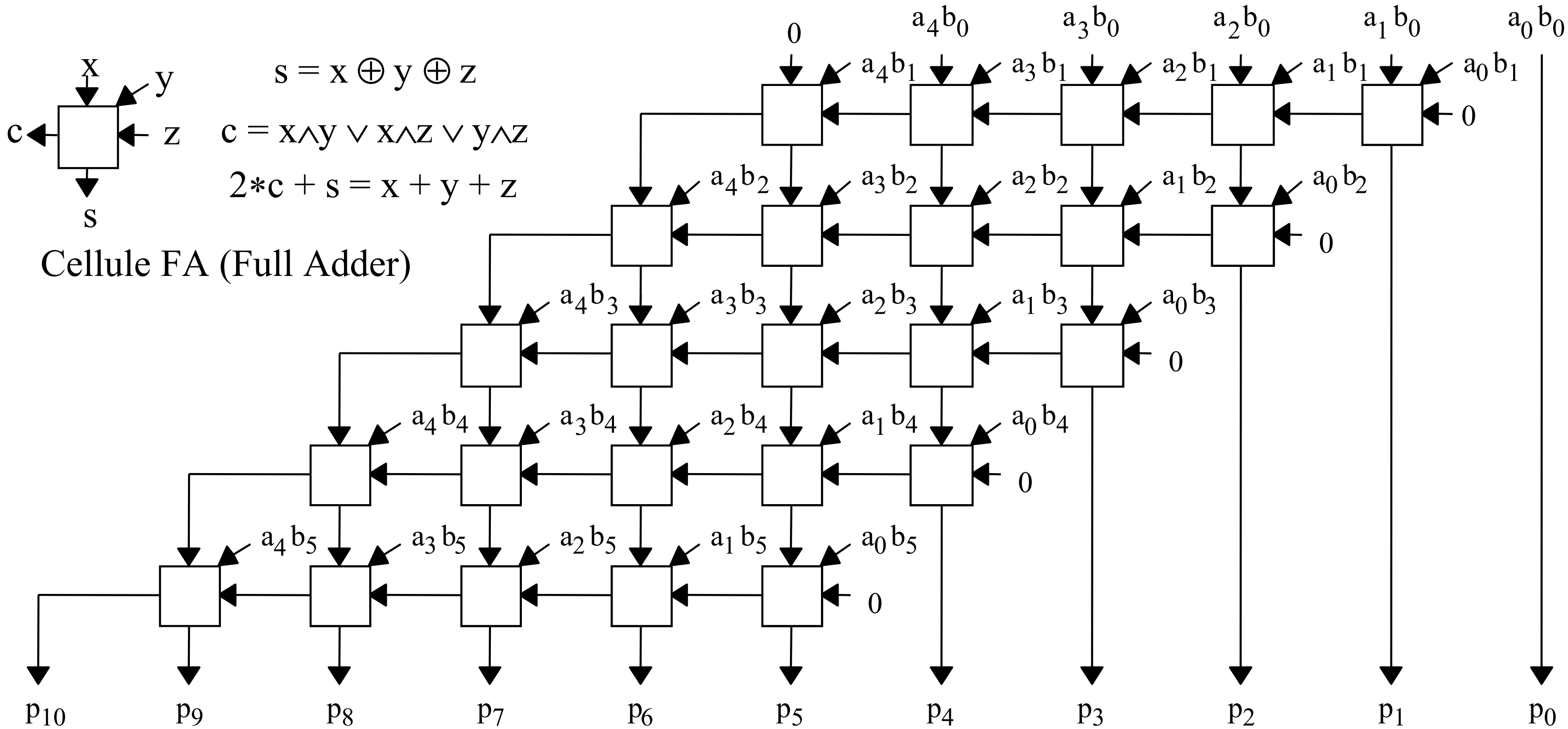
$$A = \sum_{i=0}^{4} a_i 2^i \qquad B = \sum_{i=0}^{5} b_i 2^i \qquad P = A*B = \sum_{i=0}^{10} p_i 2^i$$

$$s = x \oplus y \oplus z$$

$$c = x \wedge y \vee x \wedge z \vee y \wedge z$$

$$2*c + s = x + y + z$$

Cellule FA (Full Adder)



Multiplication - performance

Multiplieur "naïf" d'entiers	$1,5n$
Multiplieur "carry save" d'entiers	n
Multiplieur "naïf" d'entiers relatifs	$1,5n$
Multiplieur "carry save" d'entiers relatifs	n
Multiplieur	$\log_{3/2} n, log_2 n$

Division

Généralités / Plan

Algorithmes de Division

- Multiplicatives
  - Goldschmidt
  - Newton-Raphson
- Soustractives (récurrentes)
  - À restauration
  - Sans restauration
  - Sans propagation
  - Avec réduction

$$Q_{j+1} := Q_j - \frac{f(Q_j)}{f'(Q_j)} \qquad Q_{j+1} := Q_j * (2 - A * Q_j)$$

La division de deux entier n'est en général pas un entier,
en conséquence on introduit les rationnels (virgule fixe)

$$A = \sum_{i=0}^{n-1} a_i 2^{-i} = a_0 , a_1 a_2 a_3 a_4 \ldots a_{n-2} a_{n-1} \qquad -A = \overline{A} + 2^{-n+1}$$

Division récurrente: principes

On veut calculer $Q = \frac{A}{D}$.

On va construire une suite $Q_0, Q_1, Q_2, \dots Q_n$ et une suite $R_0, R_1, R_2, \dots R_n$ telles que l'invariant $A = Q_j * D + R_j$ soit respecté $\forall$ j.

La récurrence est : $Q_{j+1} = Q_j + q_{j+1} * 2^{-j-1}$

$R_{j+1} = R_j - q_{j+1} * D * 2^{-j-1}$

avec comme état initial: $Q_0 = 0$

$R_0 = A$

Quand on s'arrête, on a $Q_n = \sum_{i=0}^{n} q_i * 2^{-i}$

On impose que le choix des q_j soit tel que $R_j \to 0$ quand $j \to \infty$.

On aura donc une approximation Q_n de Q telle que $Q_n = \frac{A - R_n}{D}$ avec R_n petit.

Comme la valeur de Q est bornée par l'implémentation ($|Q| < 2$),
il faut que $-2 * D < A < 2 * D$ (D *doit être suffisamment grand*)

Exemple de division récurrente en décimal

On veut calculer $Q = \frac{22}{7}$.

Calcul des restes	Valeurs	Calcul des quotients	Valeurs
$R_0 := 22$	22	$Q_0 := 0$	
$R_1 := R_0 - 3 * D$	01,0	$Q_1 := Q_0 + 3$	3
$R_2 := R_1 - 0{,}1 * D$	00,30	$Q_2 := Q_1 + 0{,}1$	3,1
$R_3 := R_2 - 0{,}04 * D$	00,020	$Q_3 := Q_2 + 0{,}04$	3,14
$R_4 := R_3 - 0{,}002 * D$	00,0060	$Q_4 := Q_3 + 0{,}002$	3,142
$R_5 := R_4 - 0{,}0008 * D$	00,00040	$Q_5 := Q_4 + 0{,}0008$	3,1428
$R_6 := R_5 - 0{,}00005 * D$	00,000050	$Q_6 := Q_5 + 0{,}00005$	3,14285

On vérifie que

$$\begin{aligned}
22 - 0{,}3 &= 21{,}7 = 7 * 3{,}1 \\
22 - 0{,}02 &= 21{,}98 = 7 * 3{,}14 \\
22 - 0{,}006 &= 21{,}994 = 7 * 3{,}142 \\
22 - 0{,}0004 &= 21{,}9996 = 7 * 3{,}1428 \\
22 - 0{,}00005 &= 21{,}99995 = 7 * 3{,}14285
\end{aligned}$$

Exemple de division récurrente en binaire

On veut calculer $Q = \frac{10110}{111}$.

L'algorithme de division en base 2 est une transposition de l'algorithme en base 10.

Calcul des restes	Valeurs reste	Calcul des quotients	Valeurs du quotient	Valeurs du quotient
$R_0 := 10110$	10110	$Q_0 := 0$		
$R_1 := R_0 - 10 * D$	01000	$Q_1 := Q_0 + 10$	10	*2*
$R_2 := R_1 - 1 * D$	00001	$Q_2 := Q_1 + 1$	11	3
$R_3 := R_2 - 0{,}0 * D$	00001,0	$Q_3 := Q_2 + 0{,}0$	11,0	3
$R_4 := R_3 - 0{,}00 * D$	00001,00	$Q_4 := Q_3 + 0{,}00$	11,00	3
$R_5 := R_4 - 0{,}001 * D$	00000,001	$Q_5 := Q_4 + 0{,}001$	11,001	3,1*25*
$R_6 := R_5 - 0{,}0000 * D$	00000,0010	$Q_6 := Q_5 + 0{,}0000$	11,0010	3,1*25*
$R_7 := R_6 - 0{,}00000 * D$	00000,00100	$Q_7 := Q_6 + 0{,}00000$	11,00100	3,1*25*
$R_8 := R_7 - 0{,}000001 * D$	00000,000001	$Q_8 := Q_7 + 0{,}000001$	11,001001	3,14*0625*

3,142 *578125*

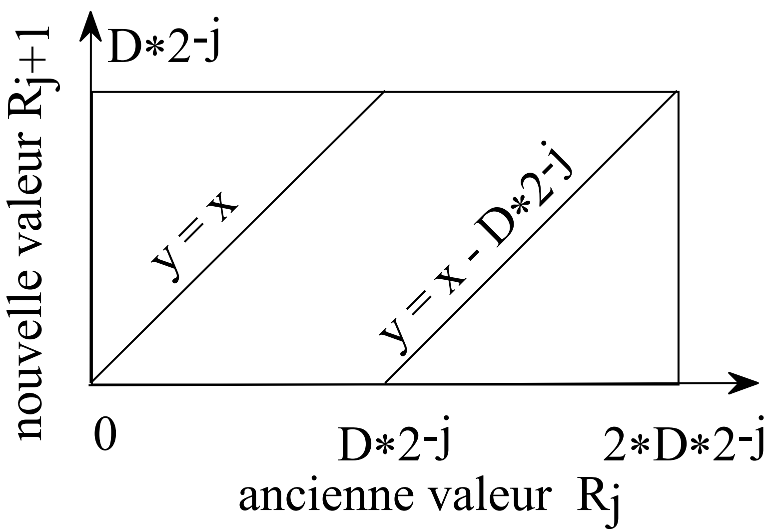
3,1428 *22265*

3,14285 *2783*

Diagramme de « Robertson »

(diviseur naïf ou à restauration)

$\boxed{\text{Invariant: } 0 \le R_j \le 2*D*2^{-j}}$



$\Longleftrightarrow$

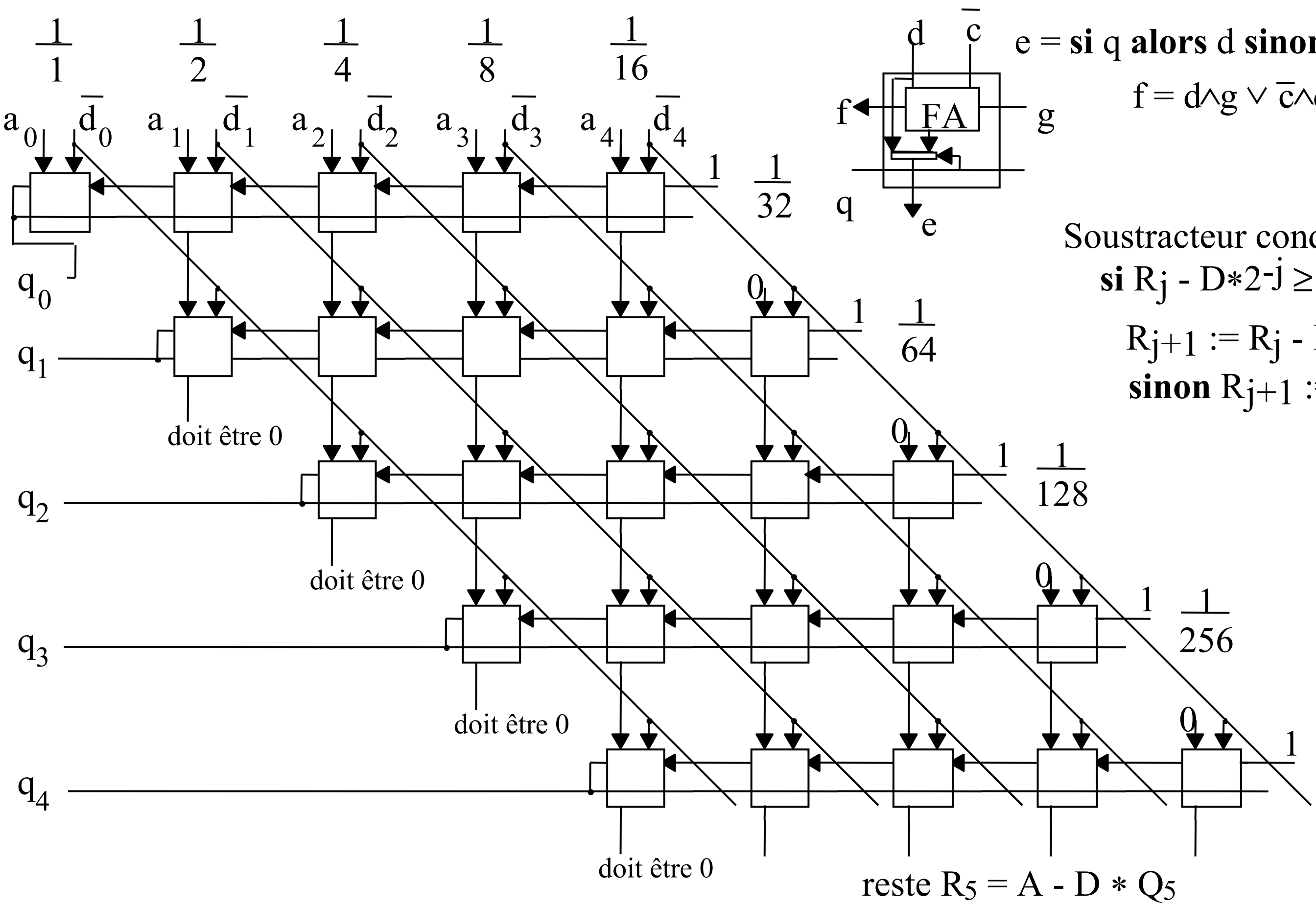
si $R_j \ge D*2^{-j}$ **alors**

$R_{j+1} := R_j - D*2^{-j}$

sinon $R_{j+1} := R_j$;

Récurrence

Diviseur naïf (à restauration)



$e =$ **si** q **alors** d **sinon** $d \oplus \bar{c} \oplus g$

$f = d \wedge g \vee \bar{c} \wedge d \vee \bar{c} \wedge g$

Soustracteur conditionnel

si $R_j - D * 2^{-j} \geq 0$ **alors**

$R_{j+1} := R_j - D * 2^{-j}$

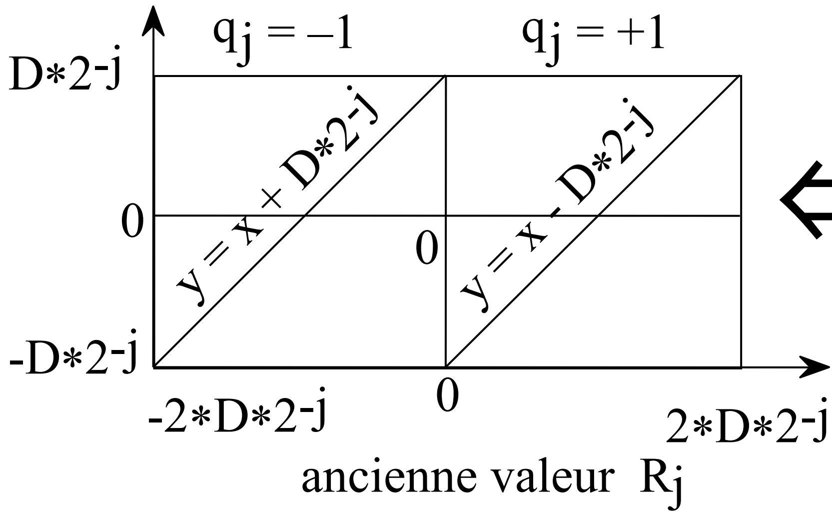
sinon $R_{j+1} := R_j$;

Exemple de division récurrente sans restauration

Calcul des restes	Valeurs du reste (signé)		Calcul des quotients	Valeurs
$R_0 := 10110$	10110	22	$Q_0 := 0$	
$R_1 := R_0 - 10 * D$	$\bar{0}1000$	8	$Q_1 := Q_0 + 10$	10
$R_2 := R_1 - 1 * D$	$0\bar{0}001$	1	$Q_2 := Q_1 + 1$	11
$R_3 := R_2 - 0{,}1 * D$	$00\bar{1}01{,}1$	-2,5	$Q_3 := Q_2 + 0{,}1$	11,1
$R_4 := R_3 + 0{,}01 * D$	$000\bar{1}1{,}01$	-0,75	$Q_4 := Q_3 - 0{,}01$	11,01
$R_5 := R_4 + 0{,}001 * D$	$0000\bar{0}{,}001$	0,125	$Q_5 := Q_4 - 0{,}001$	11,001
$R_6 := R_5 - 0{,}0001 * D$	$00000{,}\bar{1}011$	-0,3125	$Q_6 := Q_5 + 0{,}0001$	11,0011
$R_7 := R_6 + 0{,}00001 * D$	$00000{,}0\bar{1}101$	-0,09375	$Q_7 := Q_6 - 0{,}00001$	11,00101
$R_8 := R_7 + 0{,}000001 * D$	$00000{,}00\bar{0}001$	0,015625	$Q_8 := Q_7 - 0{,}000001$	11,001001

Division sans restauration

$$\boxed{\text{Invariant: } -2*D*2^{-j} \leq R_j \leq 2*D*2^{-j}}$$



$\Leftrightarrow$

si $R_j \geq 0$ **alors**

$R_{j+1} := R_j - D*2^{-j}$

sinon

$R_{j+1} := R_j + D*2^{-j}$;

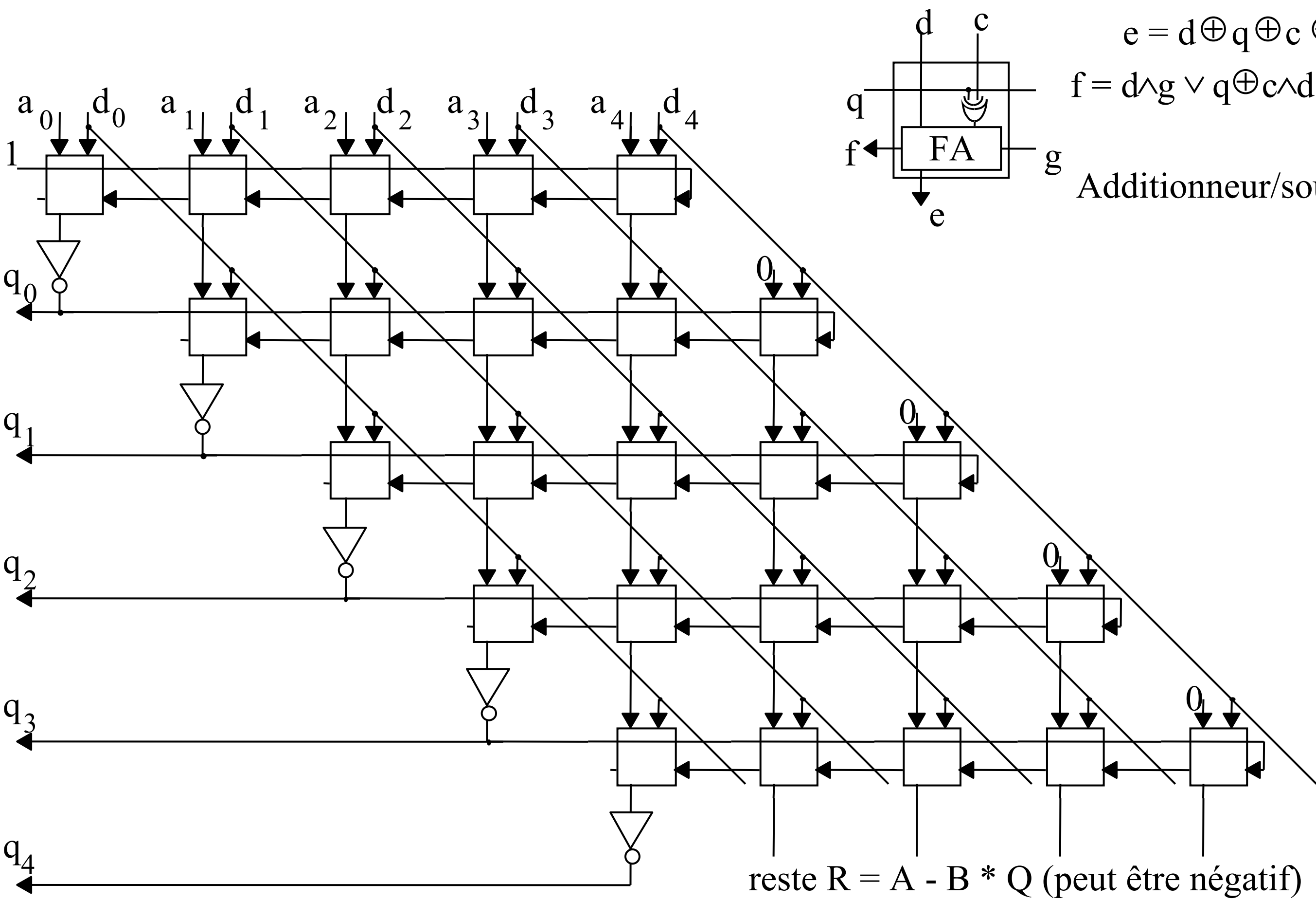
Récurence

Diviseur sans restauration (2)

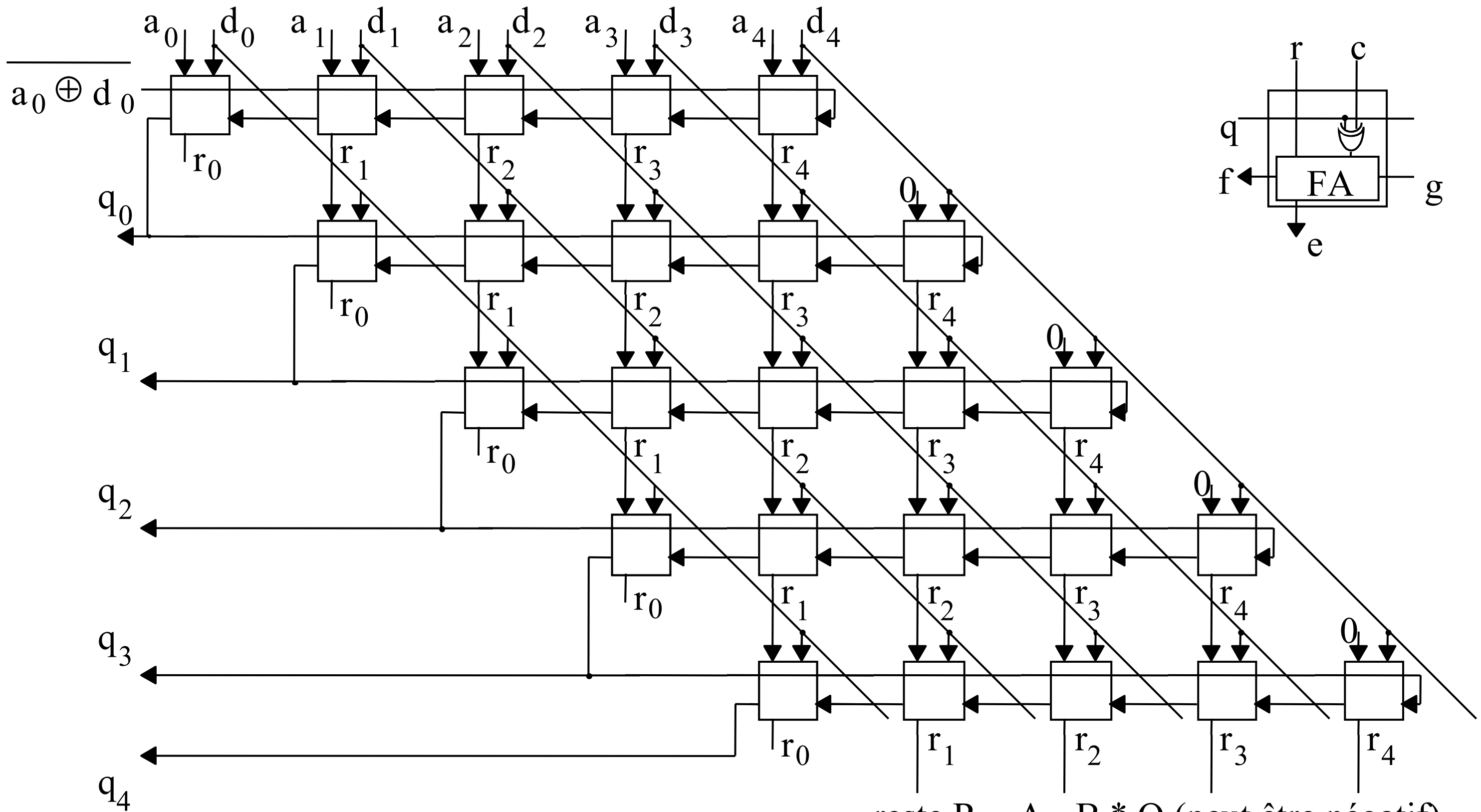
$e = d \oplus q \oplus c \oplus g$

$f = d \wedge g \vee q \oplus c \wedge d \vee q \oplus c \wedge g$

Additionneur/soustracteur



Division signée



reste R = A - B * Q (peut être négatif)

Conversion du quotient

Le quotient Q s'écrit $Q = \sum_{i=0}^{n-1} q_i * 2^{-i}$ avec $q_i \in \{-1,+1\}$. Pour le convertir on le réécrit:

$$Q = \sum_{i=0}^{n-1} q_i * 2^{-i} = \sum_{i=0}^{n-1} q_i * 2^{-i} + \sum_{i=0}^{n-1} 2^{-i} - \sum_{i=0}^{n-1} 2^{-i}$$

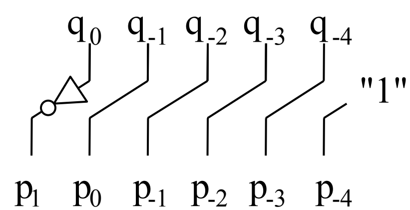
$$'' \qquad = \sum_{i=0}^{n-1} (q_i + 1) * 2^{-i} - \sum_{i=0}^{n-1} 2^{-i}$$

$$'' \qquad = 2 * \sum_{i=0}^{n-1} p_i * 2^{-i} - 2 + 2^{-n}$$

$$'' \qquad = 2 * \left(p_0 - 1 + \sum_{i=1}^{n-1} p_i * 2^{-i} + 2^{-n-1} \right)$$

$$'' \qquad = 2 * \left(\underbrace{-\overline{p_0} * 2^0 + \sum_{i=1}^{n-1} p_i * 2^{-i} + 2^{-n-1}} \right)$$

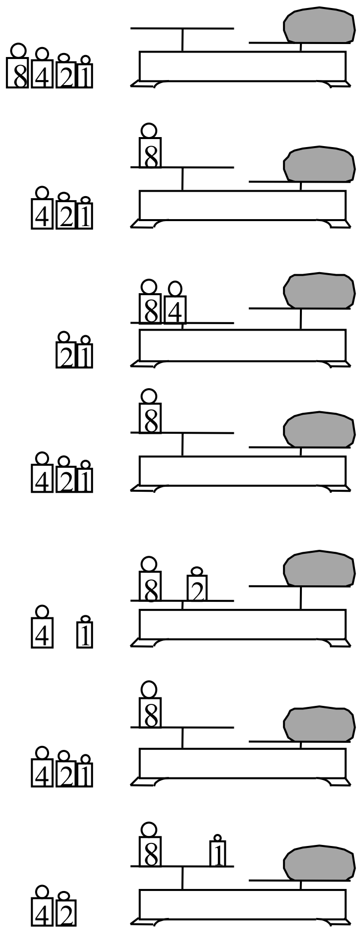
Ceci est la notation en complément à 2 $\qquad p_i \in \{0,1\}$



La conversion ne change pas la <u>valeur de Q</u> mais seulement sa <u>représentation</u> sous forme de chaîne de bits

Remarque: si on connaît a priori le signe du résultat, l'inverseur n'est même pas nécessaire

Pesée à restauration



On ajoute 8,
ça va

On ajoute 4,
c’est trop

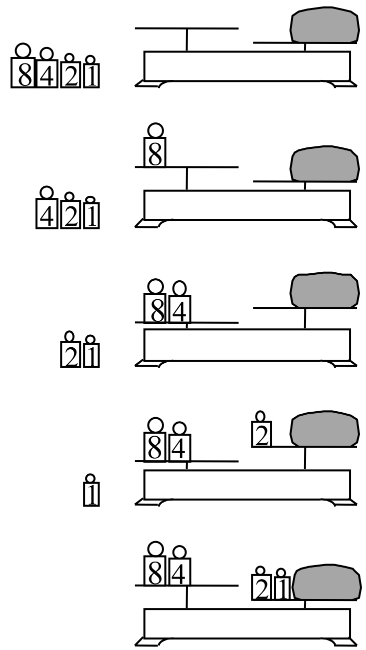
On enlève 4
(restauration)

On ajoute 2,
c’est trop

On enlève 2
(restauration)

On essaye 1,
ça va

Pesée sans restauration



On ajoute 8
sur le plus haut

On ajoute 4
sur le plus haut

On ajoute 2
sur le plus haut

On ajoute 1
sur le plus haut

On se sert de tous les poids
On pourrait peser un poids “négatif”

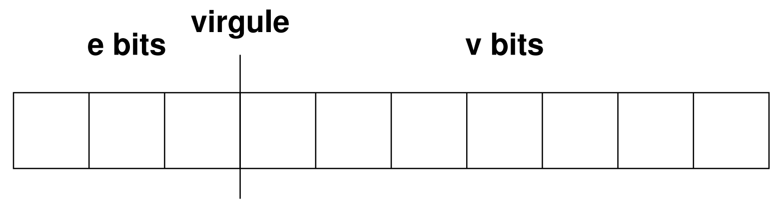
Plan

- ❑ Rappels de représentation des nombres
- ❑ Opérateurs arithmétiques fondamentaux
- ✖ Virgule fixe et bruit de calcul
- ❑ TD: 3D Z-buffer

Représentation virgule fixe

Représenter des nombres avec une partie fractionnaire:

$$a = \sum_{i=-v}^{e-1} a_i 2^i$$



avec :

- ☆ la partie entière e : nombre de bits à gauche de la virgule
- ☆ la partie fractionnaire v : nombre de bits à droite de la virgule

En non-signé, on peut représenter des nombres de 0 à $2^e - 1$ avec un pas de 2^{-v}.

Exemple : représenter $3,64$ avec le format ci-dessus

Comparaison virgule fixe/flottante

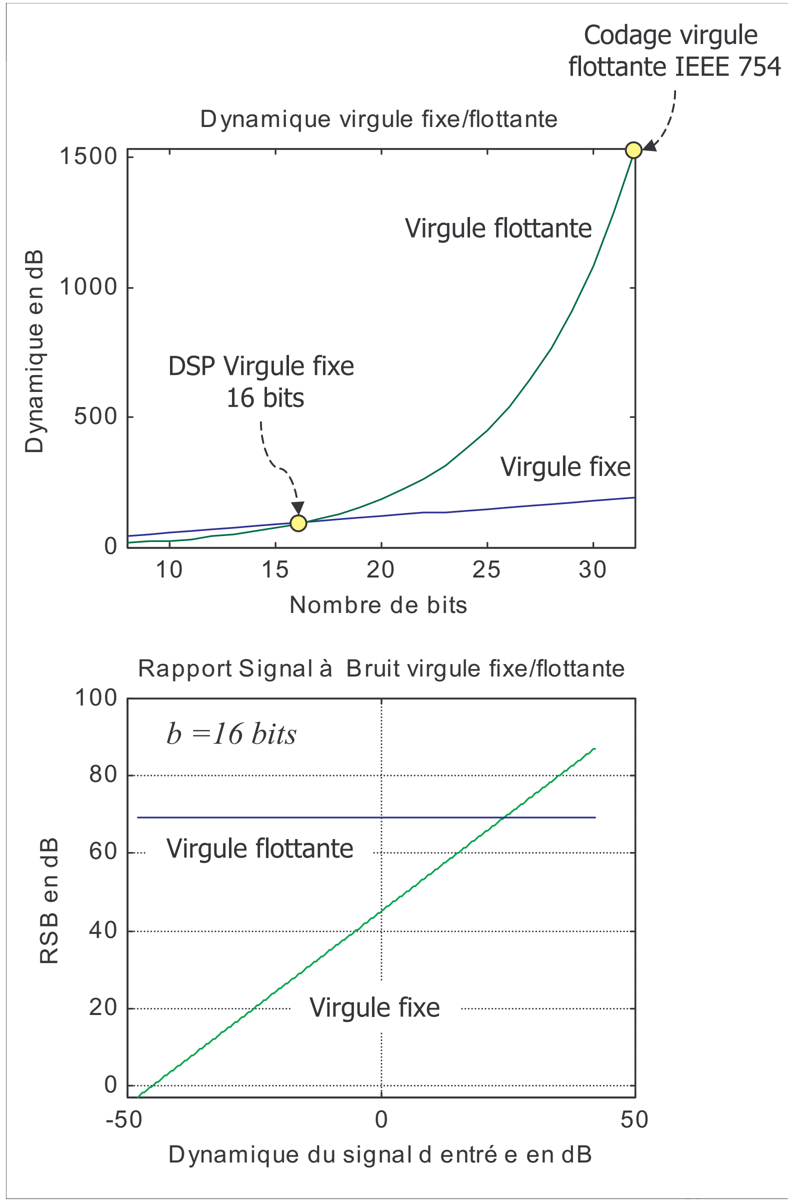
- Niveau de dynamique

$$D_{N(dB)} = 20\log\frac{\max|x|}{\min|x|}$$

- Rapport Signal à Bruit de Quantification

$$\rho_{dB} = 10\log\frac{P_s}{P_e}$$

c.f. D. Menard



Usages

☆ Arithmétique virgule flottante

- Largeur des données stockées en mémoire : 32 bits (IEEE 754)
- Opérateurs complexes (gestion de la mantisse et de l'exposant)
- L'utilisateur ne doit pas coder les données ➩ Temps de développement faible

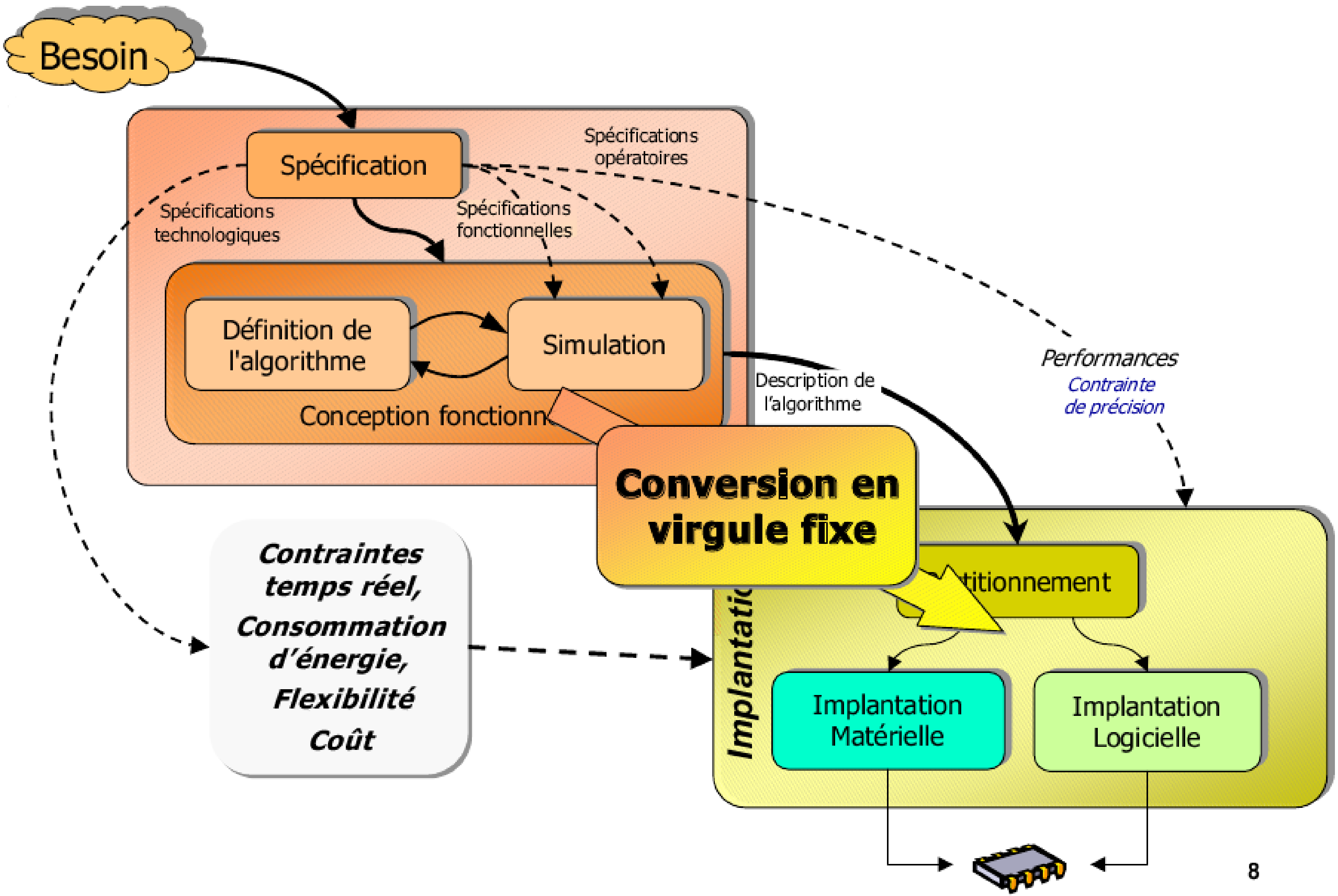
Applications spécifiques : audionumérique, faible volume

☆ Arithmétique virgule fixe

- Opérateurs plus simples ➩ Processeur plus rapide
- Largeur des données stockées en mémoire : environ 16 bits
  - ➩ Consommation moins importante
  - ➩ Processeur moins cher ➩ Applications grand public

DSP virgule fixe : 95% des ventes

Cycle de développement (D. Menard)



Arrondi , troncature & saturation

- Troncature : supprimer des bits de poids faible ou fort

$$a_{e',v'} = \lfloor a_{e,v} \rfloor_{e,v'}$$

- Arrondi : prendre la valeur la plus proche

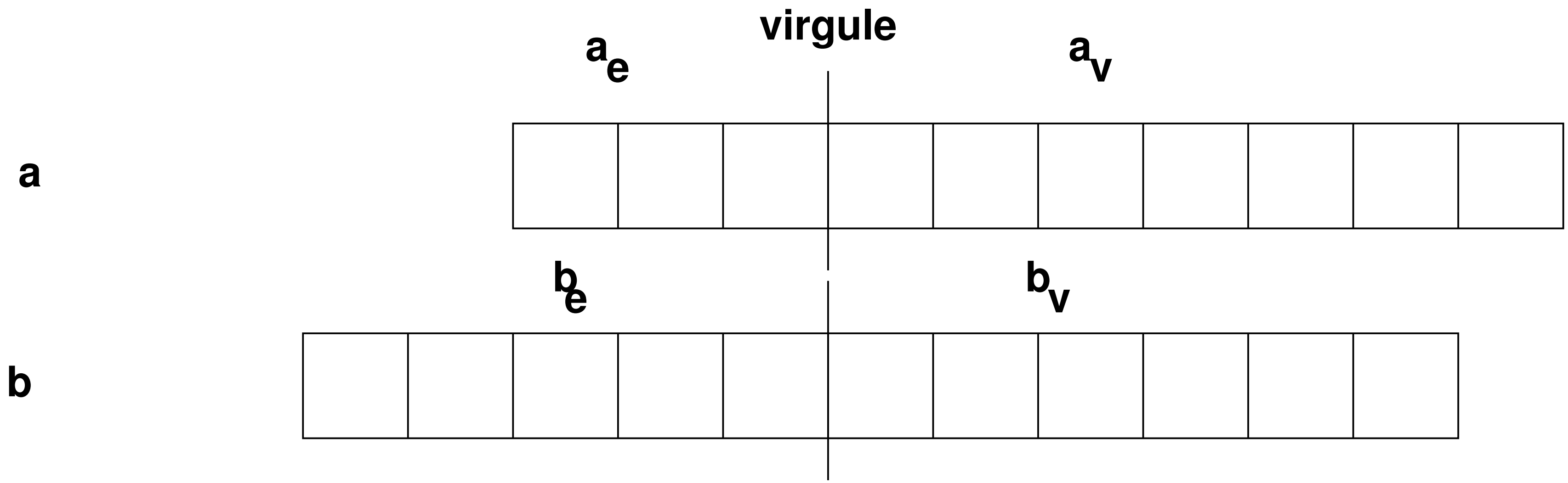
$$a_{e,v'} = \lfloor a_{e,v} + 2^{-v'-1} \rfloor_{e,v'}$$

- Saturation: supprimer des bits de poids fort sans “modulo”

$$a_{e',v'} = \lfloor \max(2^{e'} - 1, a_{e,v}) \rfloor_{e',v'}$$

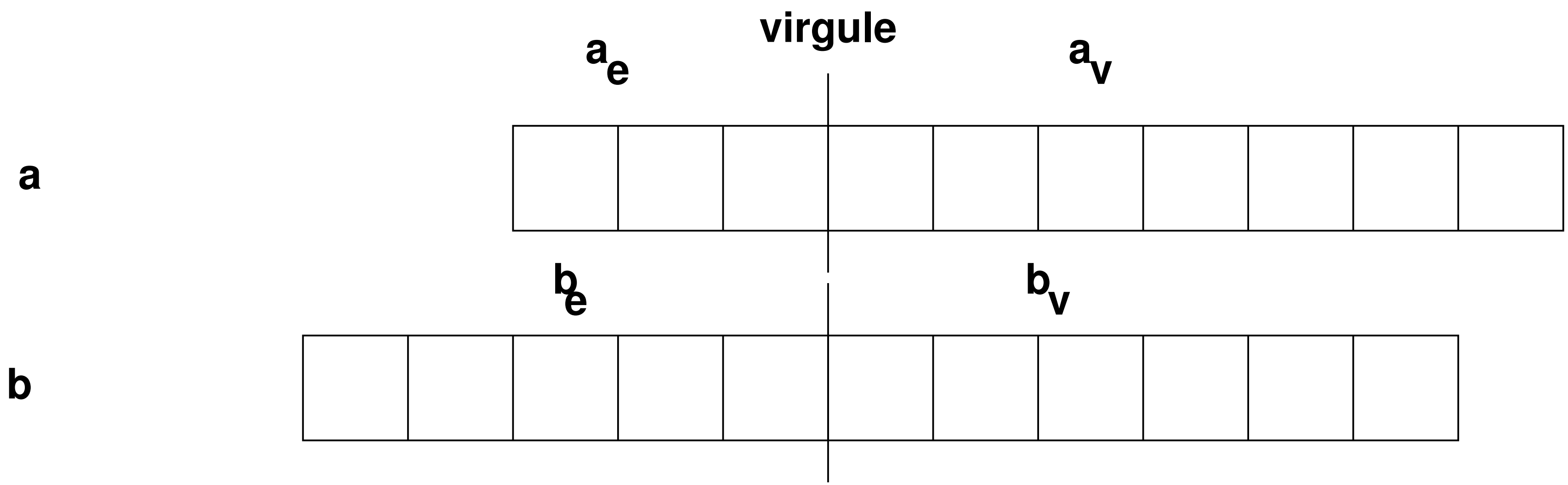
S. Mancini

Addition



?

Multiplication



?

Notion de bruit de calcul

Lorsqu'un calcul nécessite plusieurs additions/multiplications successsives, il n'est pas toujours possible de conserver tous les bits: les résultats intermédiaires doivent être arrondi/tronqué/saturé.

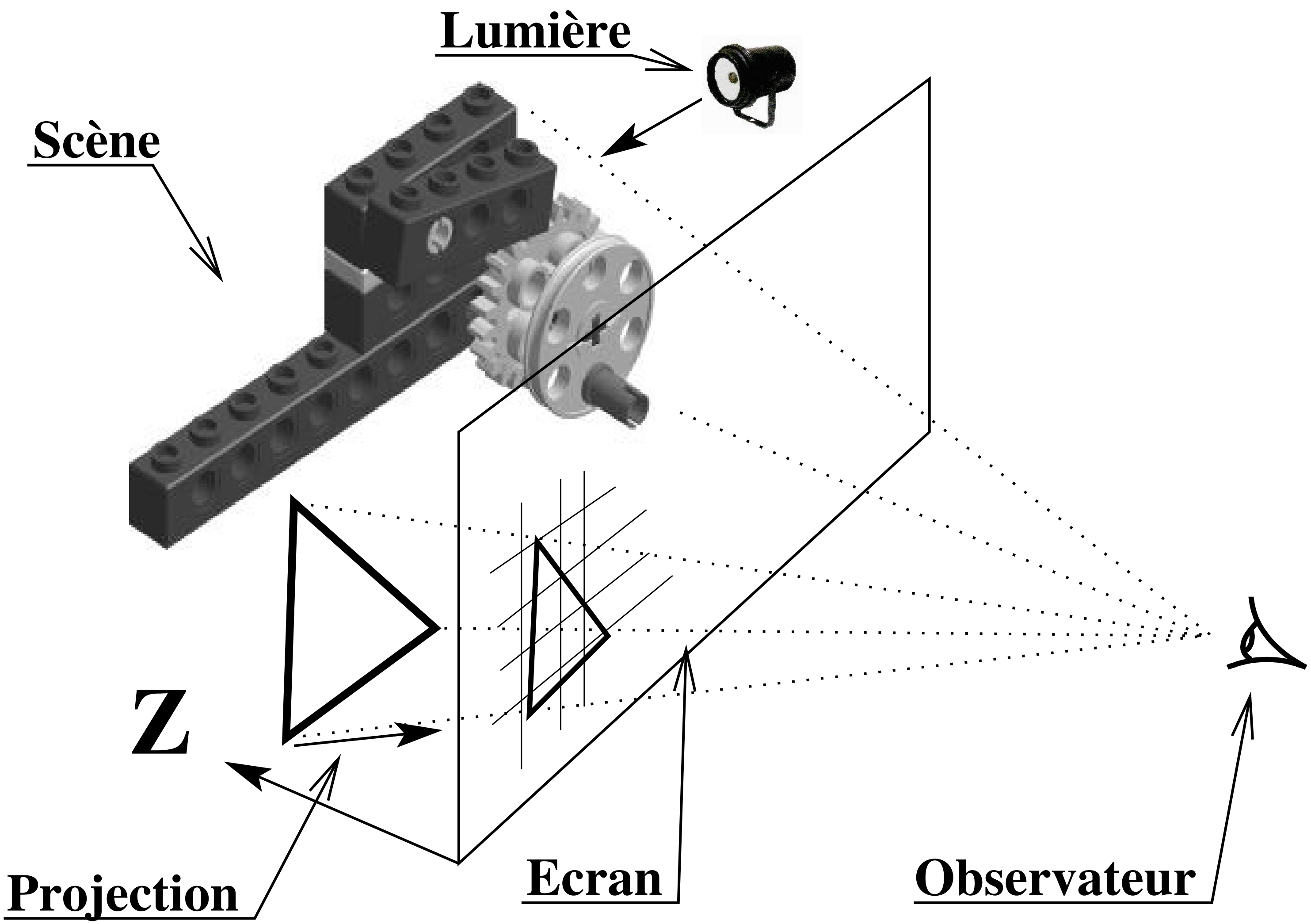
L'arrondi/troncature a le même effet qu'un bruit ajouté aux données !

☞☞☞☞ Il est nécessaire de calculer ou mesurer l'effet du passage à la virgule fixe par rapport à une application de référence !

Plan

- Rappels de représentation des nombres
- Opérateurs arithmétiques fondamentaux
- Virgule fixe et bruit de calcul
- ✖ TD: 3D Z-buffer
  - Tracé de droites
  - Tracé de triangle
  - Calcul d'éclairement et couleur
  - Application de texture

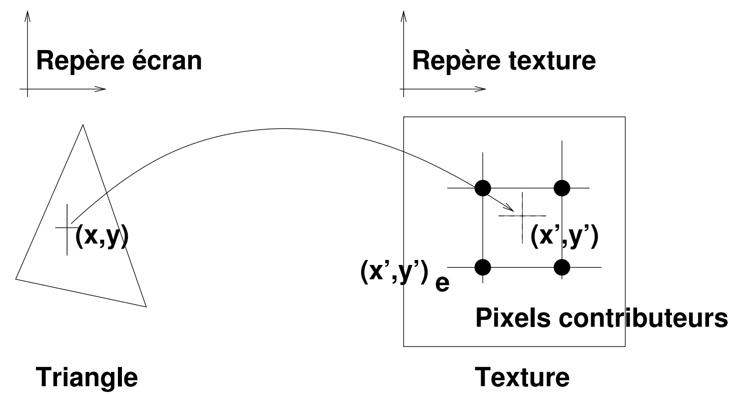
Objectif: Tracer des triangles pour le rendu 3D



Application de texture

Appliquer une texture consiste à attribuer les pixels d'une photo aux pixels de l'image produite. Les coordonnées des pixels de la texture sont obtenues par transformation perspective:

$$x' = \frac{ax+by+c}{gx+hy+i}, y' = \frac{dx+ey+f}{gx+hy+i}$$



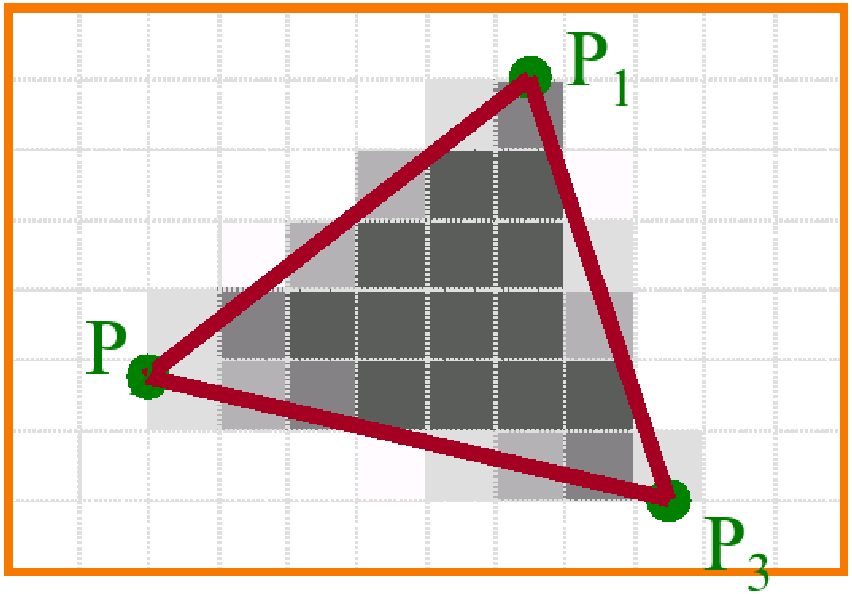
Comme les coordonnées (x', y') ne sont pas entières, il est nécessaire de faire une interpolation linéaire de la couleur:

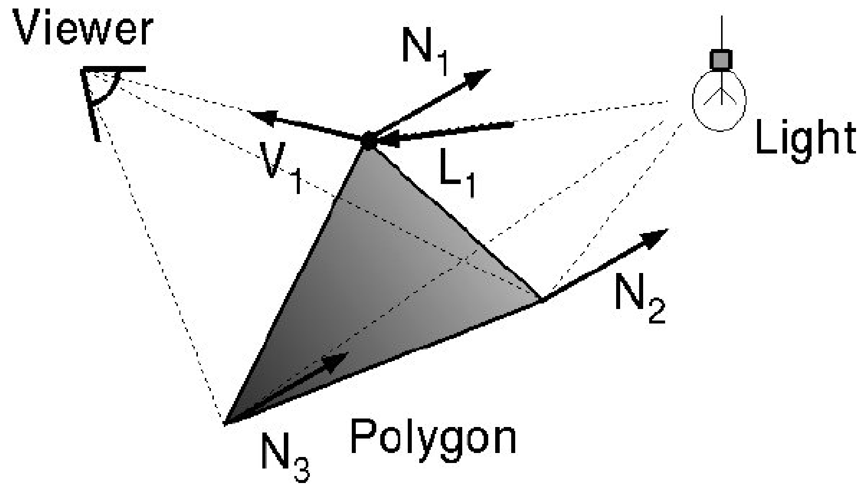
$$x' = x'_e + x'_d, \quad x'_e \in \mathcal{N}$$

$$y' = y'_e + y'_d, \quad y'_e \in \mathcal{N}$$

$$\begin{aligned} C_{x',y'} = \ & (1-x'_d)(1-y'_d)T_{x'_e,y'_e} + (1-x'_d)y'_d T_{x'_e,y'_e+1} + \\ & x'_d(1-y'_d)T_{x'_e+1,y'_e} \quad + x'_d y'_d T_{x'_e+1,y'_e+1} \end{aligned}$$

Questions





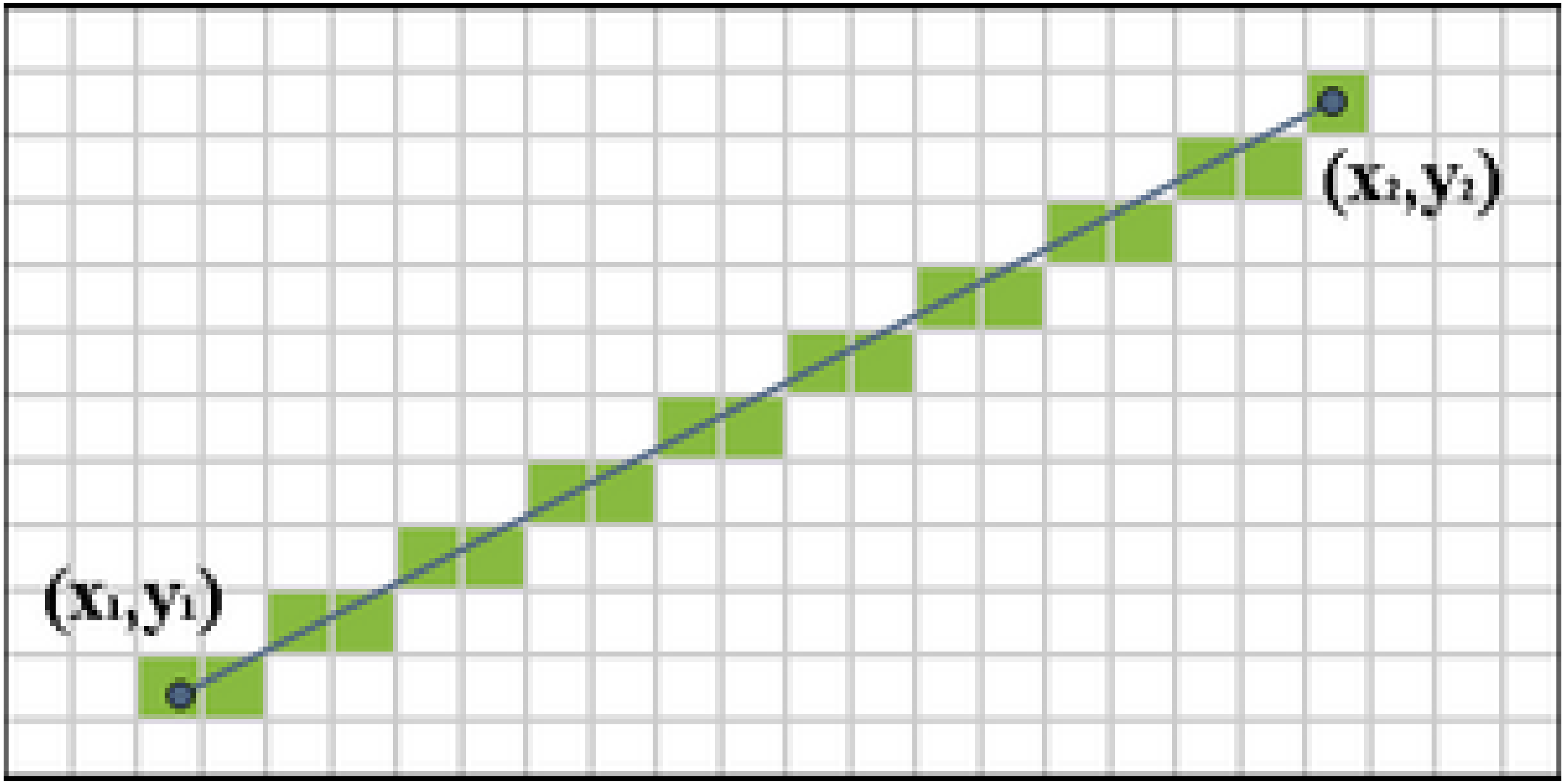
Comment remplir des triangles en attribuant des caractéristiques (couleur, profondeur, coordonnées de texture, etc) à chaque pixel, sous les contraintes suivantes:

- En utilisant le moins de mémoire possible
- En arithmétique entière

Plan

- ❑ Rappels de représentation des nombres
- ❑ Opérateurs arithmétiques fondamentaux
- ❑ Virgule fixe et bruit de calcul
- ✖ TD: 3D Z-buffer
  - ✔ Tracé de droites
  - ❑ Tracé de triangle
  - ❑ Calcul d'éclairement et couleur
  - ❑ Application de texture

Tracé de droite



Algorithme de Bresenham

Bresenham en nombres entiers

Plan

- ❑ Rappels de représentation des nombres
- ❑ Opérateurs arithmétiques fondamentaux
- ❑ Virgule fixe et bruit de calcul
- ✖ TD: 3D Z-buffer
  - ❑ Tracé de droites
  - ✔ Tracé de triangle
  - ❑ Calcul d'éclairement et couleur
  - ❑ Application de texture

Principe

Architecture

Plan

- ❑ Rappels de représentation des nombres
- ❑ Opérateurs arithmétiques fondamentaux
- ❑ Virgule fixe et bruit de calcul
- ✖ TD: 3D Z-buffer
  - ❑ Tracé de droites
  - ❑ Tracé de triangle
  - ✔ Calcul d'éclairement et couleur
  - ❑ Application de texture

Simplification des calculs

- Est il possible de simplifier les calculs des valeurs d'éclairement et couleurs?

Précision des calculs

? Quelle doit être la précision des variables pour réaliser les calculs d'éclairement et de couleur?

S. Mancini

Plan

- ❑ Rappels de représentation des nombres
- ❑ Opérateurs arithmétiques fondamentaux
- ❑ Virgule fixe et bruit de calcul
- ✖ TD: 3D Z-buffer
  - ❑ Tracé de droites
  - ❑ Tracé de triangle
  - ❑ Calcul d'éclairement et couleur
  - ✔ Application de texture

Approximation

? Est il possible d'approcher le calcul de transformation perspective par un autre, plus simple? Dans quelles conditions?

Complexité

Si on veut faire tout le calcul, quel est le nombre minimal d'opérations qu'il est possible de faire?

S. Mancini

Adéquation Algorithme Architecture

Virgule fixe

S. Mancini

Plan Détaillé