Capítol 4

Millora del temps de processament

Hem vist en el capítol anterior que el mètode de detecció de simetria proposat precisa d'una primera etapa consistent en un processament de imatge previ, dirigit a obtenir els segments rectilinis de la imatge. El mètode pròpiament dit parteix de la llista de segments rectilinis, caracteritzats per els punts inicials i finals (Fig.3.9) i consta de tres processos diferenciats:

- L'obtenció dels segments de contribució a la simetria $\overline{SCSL}$ (veure ap. 3.6)

- L'obtenció del mapa d'acumulació de les superposicions dels segments de contribució $\overline{SCSL}$ (veure ap. 3.7)
- L'extracció d'eixos locals ponderats (veure ap. 3.8)

També hem vist que el procés d'obtenció dels segments de contribució $\overline{SCSL}$ té un cost de computació d'ordre $O(n^2)$, on n és el nombre de segments rectilinis considerats més significatius de la imatge. Aquest nombre no acostuma a sobrepassar un ordre de magnitud de 10^2, en la majoria de imatges, si fixem la longitud mínima dels segments a considerar com a significatius per sobre de 5 píxels (veure taula 3.2).

Per altra banda, el procés d'obtenció del mapa d'acumulació té un cost de computació d'ordre lineal $O(n)$, però aquí n és el nombre de píxels que composen els segments de

contribució $\overline{SCSL}$. Si tenim en compte que aquest nombre pot arribar fàcilment a tenir dos ordres de magnitud superior, respecte al nombre de segments rectilinis detectats en el procés anterior (veure taula 3.2), podem concloure que el cost de computació d'aquest procés s'aproxima al del cas anterior.

A títol d'exemple, una imatge amb relativament pocs segments rectilinis detectats, (Cas 1 de la taula 3.2, 343 segments amb $n^o\ píxels > 5$) comporta, si considerem com a llargada mitjana dels segments de contribució $\overline{SCSL}$ 50 píxels, un total de $6.394\ X\ 50 = 319.700$ càlculs per tal d'obtenir la matriu d'acumulació mentre, que l'obtenció dels segments de contribució representa $(343 \times 342)/2 = 58481$ càlculs segons l'equació {3.3}, tot i que aquí els càlculs són més complexos que en el cas anterior en el qual es tracta d'efectuar una simple suma.

Finalment, l'etapa d'extracció d'eixos locals consta dels processos d'acotació i ponderació dels eixos virtuals que són tasques d'ordre lineal respecte aquests últims. Aquest processos treballen sobre el subconjunt d'eixos virtuals més significatius, (els eixos virtuals que han obtingut més superposicions segons em vist en el apartat 3.8.1). Aquest fet fa que el nombre d'elements a tractar, sigui aproximadament dos ordres de magnitud inferior al nombre de segments de contribució. En conseqüència el cost dels processos d'acotació i ponderació, és molt inferior al de l'obtenció del segments de contribució o al de la generació del mapa d'acumulació

El fet que l'obtenció del mapa d'acumulació en el mètode de detecció de simetria presentat sigui un dels processos amb més alt cost de computació i la possibilitat que el coneixement de la superposició dels píxels que formen segments rectilinis, pugui ser d'utilitat en processos gràfics o de visió per computador, conjuntament amb el fet que una part de la generació del mapa coincideix amb el problema gràfic de traçat de línies amb alta velocitat (aplicacions interactives, veure ap.3.7.1), fa que la tasca de reduir el temps de processament d'aquesta etapa, resulti molt més interessant que intentar-ho en els altres processos.

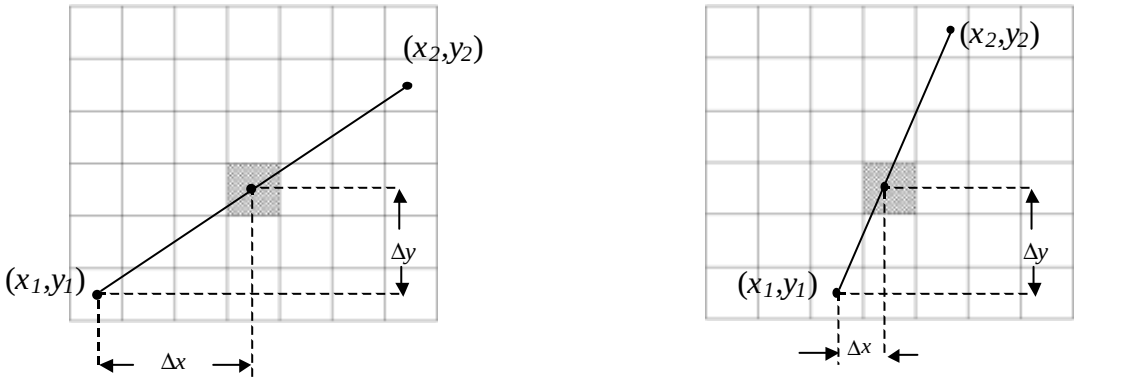
La diferència entre el problema del traçat de línies i el nostre rau, en que nosaltres a part d'obtenir els píxels que composen el segments rectilinis, entre els seus punts inicial i

final, que és la tasca dels algorismes de traçat de línies, memoritzem també un valor d'acumulació quan els segments es superposen tal com es va exposar en el apartat 3.7

4.1 Traçat de segments rectilinis

Una de les tasques més comunes en la generació gràfica de dibuixos per computador consisteix en el traçat de segments rectilinis en el pla discret. Per citar alguns exemples trobem aquest problema en diagrames de blocs, gràfics de barres, dibuixos d'enginyeria civil, plànol d'enginyeria mecànica i arquitectura, diagrames lògics, etc. A més a més, sovint les corbes es poden aproximar a partir de segments rectilinis molt petits.

El problema del traçat de línies consisteix en obtenir aquells píxels que proporcionen la millor aproximació d'un segment rectilini caracteritzat per els seus punts inicial i final. Depenent del tipus d'aplicació, la millor aproximació no és sempre la que proporciona segments amb aparença més rectilínia. Així, quan volem obtenir formes poligonals els punts terminal del segment són els més importants per tal de tancar la figura geomètrica, mentre que en altres casos la propietat desitjada és la densitat de píxels constant respecte la llargada i l'angle. En les aplicacions interactives on s'han de redibuixar constantment les figures a diferents escales i perspectives, la velocitat de traçat és vital a causa de l'elevat nombre de segments amb que s'ha de tractar, per unitat de temps.



$P = \frac{\Delta x}{\Delta y} = n^o\ pixels\ en\ x\ per\ cada\ pixel\ en\ y$ (a)

$m = \frac{\Delta y}{\Delta x} = n^o\ pixels\ en\ y\ per\ cada\ pixel\ en\ x$ (b)

Figura 4.1: Variació de les coordenades d'un segment. (a) La coordenada x es la que varia més ràpidament. (b)) La coordenada y es la que varia més ràpidament.

G. Casiola a [Casiola 88] planteja un conjunt d'idees per tal d'accelerar els algorismes del traçat de línies.

La generació del mapa d'acumulació anterior s'inscriu dins problemàtica del traçat de línies amb alta velocitat, doncs tal com hem vist en el capítol anterior per acomplir aquest objectiu s'ha de tractar amb un nombre elevat de segments, els segments de contribució a la simetria $\overline{SCSL}$.

El problema del traçat de línies ha estat abastament estudiat en els nostres dies J.E Bresenham [Bresenham 96] planteja les idees fonamentals per tal de tractar-lo. En l'actualitat es disposa d'una sèrie d'algorismes coneguts amb el nom genèric "d'Algorismes de traçat de línies" tot i que alguns problemes estan per resoldre, com mostra [Pitteway 97].

La majoria d'aquests algorismes, utilitzen les anomenades tècniques incrementals, que obtenen de forma iterativa les coordenades de cada un dels punt que aproximen el segment rectilini des del punt inicial al final, en el pla discret. Aquestes tècniques consisteixen en incrementar seqüencialment en una unitat la coordenada de més variació a cada iteració, partint d'un punt extrem del segment, i utilitzant un criteri per tal de decidir si també s'incrementa la coordenada de menys variació (Fig 4.1). La pendent *m* i la seva inversa *p* determinen la raó de variació per els dos casos existents, és a dir:

- la coordenada *x* evoluciona més ràpidament que la *y* (Fig 4.1 (a))

- la coordenada *y* evoluciona més ràpidament que la x (Fig 4.1 (b))

si el criteri utilitzat per determinar si cal incrementar la coordenada de menys variació, és suficientment acurat, les coordenades amb valors sencers que es generen seqüencialment, són suficientment pròxims als valor reals. A [Newman 88] es poden trobar els diferents criteris que utilitzen aquest algorismes.

Els dos algorismes més utilitzats són el "*Digital Differencial Analizer*" (DDA) [Newman 88] i el Algorisme de Bresenham [Bresenham 65]. Aquest últim utilitza

l'error acumulat en els píxels obtinguts en la iteració anterior, respecte del valor del segment real, per tal de decidir si incrementa la coordenada de menys variació. Aquest algorisme és un dels més utilitzats pel fet que es pot implementar utilitzant únicament sumes i restes, de forma que proporciona un bon compromís entre l'aparença rectilínia del segment a traçar i el cost de computació.
Cal fer nota que aquests algorismes incrementals són seqüencials i que tot i que en reduï m el cos, aquest sempre serà proporcional al nombre de píxels del segment rectilini a traçar. En les aplicacions interactives, on el nombre de segments que es tracten és molt elevat, aquest funcionament seqüencial és un problema.

En el nostre cas, un algorisme seqüencial no és tampoc una bona solució alhora de generar el mapa d'acumulació, ja que s'ha de calcular el valor d'acumulació (superposició) del cada un dels píxels que composen el conjunt de segments de contribució a la simetria $\overline{SCSL}$ obtinguts, amb el mínim de temps possible.

Alguns autors han proposat accelerar aquest procés utilitzant màquines de processament massiu paral·lel SIMD [Pang 90], [Laurent 93] o MIMD [Wright 90], però apareix el problema del cost de la distribució de tasques i la comunicació entre processadors, en funció del nombre utilitzat.

S'han desenvolupat també processadors gràfics, que generen seqüencialment les adreces de la memòria de visualització dels píxels que aproximen el segment rectilini que ha de ser traçat. Aquests processadors, tot i que realitzen les operacions de càlcul molt més ràpidament, mantenen la problemàtica d'un cost proporcional al nombre de píxels del segment a traçar, a causa del funcionament seqüencial. [SGS 91]

Amb un altre punt de vista alguns autors han proposat implementar memòries lògiques millorades per tal d'accelerar els processos gràfics. D'acord amb aquesta idea Fuchs i Poulton [Fuchs 81], proposen una eficient arquitectura gràfica coneguda amb el nom de "*Pixel Planes*". Aquesta arquitectura consta d'un petit processador de píxel i d'un multiplicador en arbre binari capaç de tractar simultàniament, per tots el píxels, expressions lineals de la forma $F(x,y) = Ax + By + C$. El processador de píxel consta d'un processador d'un bit amb memòria, capaç de realitzar seqüencialment operacions locals. En aquesta arquitectura es poden implementar una gran varietat d'algorismes

gràfics, si es descriuen les operacions de píxel en forma d'expressions lineals. [Fuchs 85]. Aquesta metodologia representa una gran millora en el temps de processament i en l'ampla de banda de la memòria de visualització.

Aquesta arquitectura és, òbviament, capaç de traçar segments rectilinis però les operacions que comporta la identificació del coeficients *A, B, C,* de l'expressió lineal anterior, a cada processador, són seqüencials pel fet que el processador de píxel és d'un bit. L'augment necessari en la llargada de la paraula del processador de píxels, que eliminaria el càlcul seqüencial, conduiria a un implementació que ultrapassa les possibilitats tecnològiques actuals, ja que a l'arquitectura s'han de replicar tants processadors com píxels tingui la imatge.

4.1.1. Solució proposada al traçat de segments rectilinis

Per tal d'augmentar la velocitat en el traçat de segments rectilinis, que permetrà accelerar el procés d'obtenció del mapa d'acumulació dels segments de contribució a la simetria $\overline{SCSL}$, proposem:

- Obtenir un algorisme de traçat de línies no incremental, capaç de generar els píxels que aproximen un segment rectilini de forma simultània i que permeti un implementació hardware.

- Dissenyar i verificar un ASIC implementable, que utilitzant l'algorisme anterior, sigui capaç de calcular i memoritzar simultàniament el valor d'acumulació dels píxels que aproximen els segments de contribució a la simetria ($\overline{SCSL}$) (veure ap. 3.7)

4.1.2. Algorisme de traçat de línies no incremental

A continuació descriurem un algorisme de traçat de línies, capaç d'activar simultàniament els píxels que aproximen un segment rectilini especificat pels seus punts inicial i final.

$$\overline{P_{in}P_{fi}} = ((x_{in}, y_{in}),(x_{fi}, y_{fi}))$$

L'algorisme es dissenya amb la idea de permetre una implementació hardware i, de tal manera que es pugui complementar amb la capacitat d'acumular i memoritzar simultàniament el valor d'acumulació, quan es donin superposicions dels píxels dels segments traçats (veure procés d'acumulació Apart. 3.7.2)

Explicitarem una sèrie de càlculs del tot innecessaris si es vol obtenir un algorisme seqüencial, però que permeten veure la possibilitat d'una implementació paral lela del traçat de línies.

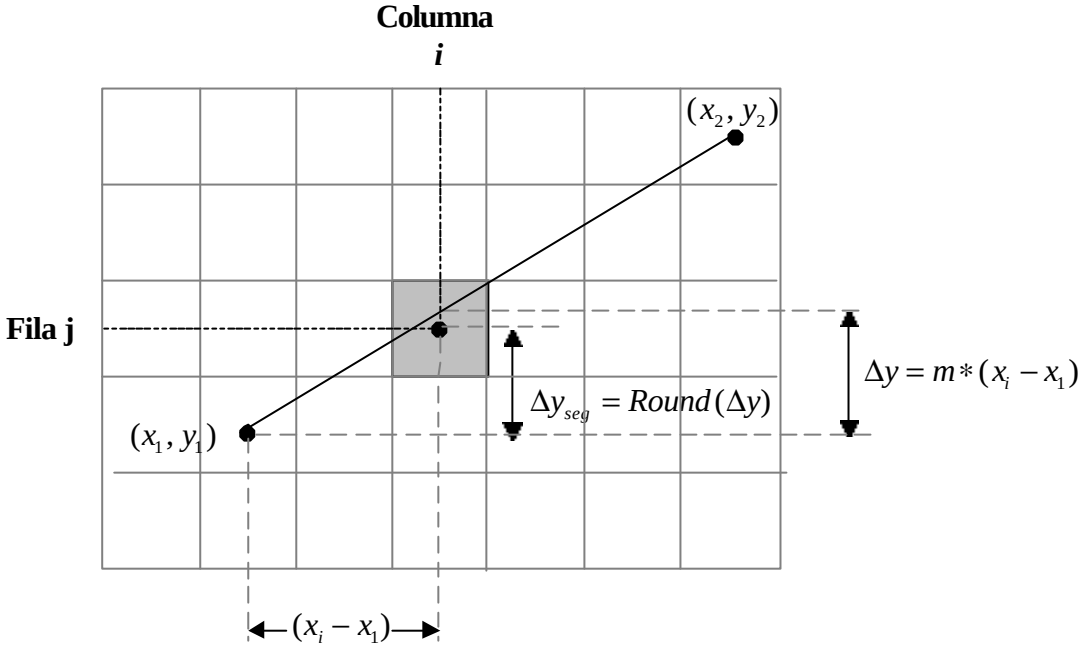


Figura 4.2: Segment amb pendent positiva on la coordenada x és la que varia més rapidament

4.1.2.1. Modelat del traçat de línies en paral lel

Mostrarem els raonaments necessaris que porten a l'obtenció de l'algorisme, considerant inicialment un segment amb pendent positiva $0 \le m \le 1$, amb la x com a coordenada de més variació, tal com es mostra a la Fig.4.2. Més endavant generalitzarem els resultats a tots els casos.

La equació de la recta que inclou el segment bé donada per :

$$\frac{y_i - y_1}{x_i - x_1} = \frac{y_2 - y_1}{x_2 - x_1} = m$$

que es pot escriure com:

$$\Delta y = m * (x_i - x_1) \qquad \{4.1\}$$

On m és la pendent, $(x_i - x_1)$ i Δy són els increments en x i y des del punt inicial fins un punt qualsevol (x_i, y_i) del segment.

En el pla discret, per tal de trobar els píxels que aproximen el segment podem utilitzar la equació {4.1} modificada de la següent manera:

$$\Delta y_{seg} = Round\left(m * (x_i - x_1)\right) \qquad \{4.2\}$$

D'aquesta forma Δy_{seg} es l'increment, comptabilitzat en píxels, sobre l'eix de coordenades y, des del punt inicial fins la coordenada y de un punt qualsevol (x_i, y_i) del segment, en pla discret (veure Fig.4.2). Així, aquesta expressió permet implementar un test per determinar si un determinat píxel (x_i, y_i), a qualsevol posició del pla discret, pertany al conjunt de píxels que aproximen un determinat segment rectilini, caracteritzat pel seus punts inicial i final $(x_1, y_1), (x_2, y_2)$ i amb pendent $0 \le m \le 1$. El test consisteix en les següents comparacions (veure Fig 4.3)

Un píxel qualsevol (x_i, y_i) $\in$ a un segment amb pendent $0 \le m \le 1$ definit pels seus punts inicial i final $(x_1, y_1), (x_2, y_2)$

$$\text{SI} \quad \Delta y_{Pix} = \Delta y_{Seg}$$

$$\text{on} \quad \Delta y_{Pix} = (y_i - y_1) \quad \text{i} \quad \Delta y_{Seg} = Round(m * (x_i - x_1)$$

$$\text{amb} \quad y_1 \le y_i \le y_2 \quad \text{i} \quad x_1 \le x_i \le x_2$$

Òbviament aquesta metodologia és absolutament ineficient si volem implementar un algorisme incremental o un processador seqüencial. No obstant, el test anterior té una gran força basada en la seva senzillesa i es pot aplicar en qualsevol moment, sense cap càlcul previ i sobretot dóna via a l'implementació, tan d'un algorisme paral·lel, com d'una arquitectura paral·lela, com veurem en l'apartat següent.

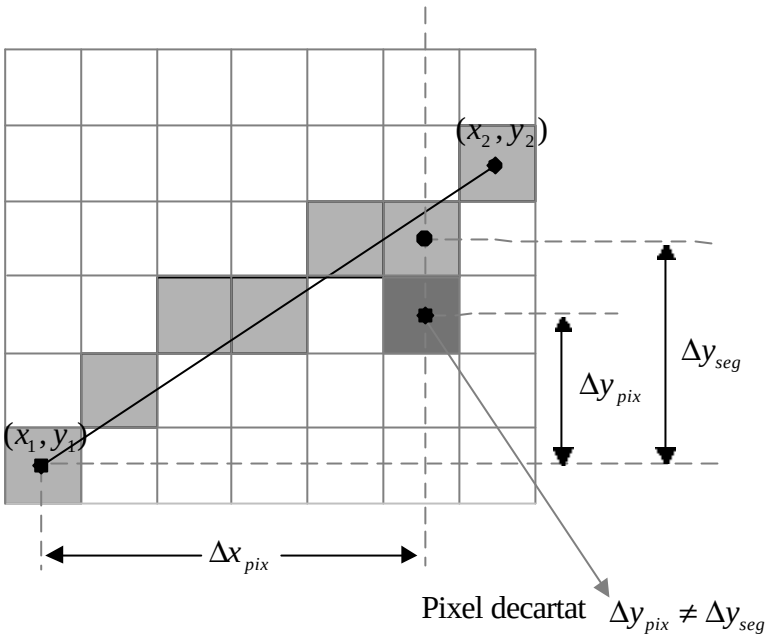


Figura 4.3: Conjumt de píxels que compleixen $\Delta y_{pix}=\Delta y_{seg}$ i que aproximen el segment

Si la pendent és negativa entre $-1 \leq m \leq 0$, la coordenada x continua sent la de més variació i el test anterior continua sent vàlid. Contràriament per pendents compreses entre $-\infty \leq m \leq -1$ i $1 \leq m \leq \infty$, la coordenada y passa a ser la de més variació. Això introdueix un nou cas que resolem de forma simètrica a l'anterior:

Un píxel qualsevol $(x_i, y_j) \in$ a un segment amb pendent compreses entre $-\infty \leq m \leq -1$ i $-\infty \leq m \leq -1$ i definit pels seus punts inicial i final $(x_1, y_1), (x_2, y_2)$

$$\text{SI} \quad \Delta x_{Pix} = \Delta x_{Seg}$$

$$\text{on} \quad \Delta x_{Pix} = (x_i - x_1)) \qquad \text{i} \qquad \Delta x_{Seg} = Round\left(\frac{y_i - y_1}{m}\right)$$

$$\text{amb} \qquad x_1 \leq x_i \leq x_2 \qquad \text{i} \qquad y_1 \leq y_i \leq y_2$$

4.1.2.2. Error que es comet en la selecció dels píxels

El mètode anterior selecciona el conjunt de píxels que aproximen un segment en funció del valor Δy_{seg} o Δx_{seg}, calculats a partir de la pendent del segment a dibuixar.

Els errors màxims que cometem en aquest càlcul són els

$$Max(Error\_\Delta y_{seg}) = Max(\Delta y - \Delta y_{seg}) = Max\left((m(x_i - x_1) - Round\left(m(x_i - x_1)\right)\right) = \pm\frac{1}{2}Pixel$$

$$Max(Error\_\Delta x_{seg}) = Max(\Delta x - \Delta x_{seg}) = Max\left(\frac{(y_i - y_1)}{m} - Round\left(\frac{(y_i - y_1)}{m}\right)\right) = \pm\frac{1}{2}pixel$$

On aquest resultat s'obté per definició, de la funció arrodoniment $Round(x)$ i es pot comprovar intuï tivament a la Fig 4.3

Ara, el nostre test, tria per cada columna *x* o fila *y* el píxels que compleixen:

$$per\ cada\ columa\ x \quad \Delta y_{pix} = \Delta y_{seg} \quad amb\ -1 \le m \le 1$$

$$per\ cada\ fila\ y \quad \Delta x_{pix} = \Delta x_{seg} \quad per\ la\ resta\ de\ pendents$$

Per tant, el error comès en la tria del píxel és el mateix que el comès en el càlcul de $\Delta y_{seg}, \Delta x_{seg}$, és a dir:

$$Error_{max}(x) = Max(Error\_\Delta y_{seg}) = \pm\frac{1}{2}Píxel \quad ; x = 1,2,3...N$$

$$Error_{max}(y) = Max(Error\_\Delta x_{seg}) = \pm\frac{1}{2}Píxel \quad ; y = 1,2,3...M$$

podem afirmar doncs que els píxels triats aproximen els punt, de la columna *x* o fila *y*, del segment real amb un error màxim de ½píxel (veure Fig.4.4). Aquest error representa

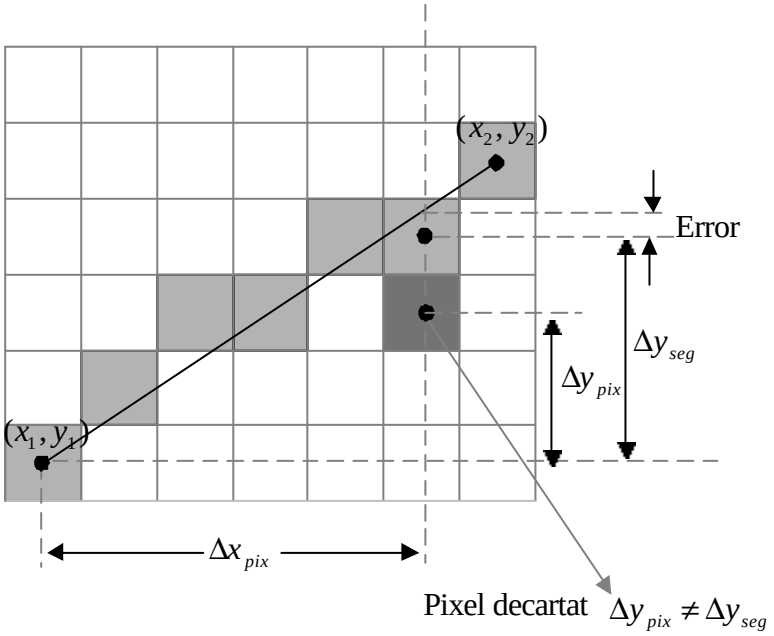


Figura 4.4: Error en la tria de píxels que aproximen el segment

una bona aproximació, de forma que l'aparença rectilínia del segment aproximat dependrà sobretot, de la pròpia discretització, és a dir, de les dimensions del píxel.

4.1.2.3. Formulació de l'algorisme paral·lel

En aquest apartat presentem la idea anterior formulada en un algorisme que inclou tots els casos que es poden donar en el traçat de línies.

Començarem pel cas de pendents $-1 \le m \le 1$. Hem dit anteriorment, com mostra gràficament la Fig.4.3, que el mètode consisteix en comparar l'increment en y (Δy_{pix}), des del punt inicial del segment fins la coordenada y del píxel considerat, amb l'increment calculat a partir de la pendent del segment (Δy_{seg}). És a dir, comparem l'increment en y del píxel considerat amb l'increment en y que ha de tenir el píxel que aproxima el segment.

Cal fer notar que aquesta comparació s'ha de fer per tots els píxels del pla discret i que només es avantatjós fer-ho d'aquesta forma, si disposem de la possibilitat d'un processament en paral·lel, és a dir, que la comparació anterior es faci simultàniament, per tots els píxels, en diferents processadors

Per tal de simplificar l'algorisme cal tenir en compte que per cada columna, només existeix un píxel que aproxima el segment, que és el representat per $\Delta y_{Seg} = Round\,(m*(x_i - x_1))$ tal com es pot veure en la Fig.4.3. pel cas $-1 \le m \le 1$, i que aquest valor només caldrà calcular-lo un cop per columna. A partir d'aquí ens referirem a aquest valor com a l'increment de columna C_{inc}.

De la mateixa manera tots el píxels de la mateixa fila tenen el mateix increment en y des del punt inicial $\Delta y_{Pix} = (y_j - y_1)$, per tant, també aquí, només caldrà calcular aquest valor un cop per fila. A partir d'aquí ens referirem a aquest valor com a l'increment de fila F_{inc}.

De forma similar per el altre cas, representat per les pendents $\infty \ge m \ge 1,\ -1 \ge m \ge -\infty$, es podem aplicar els mateixos raonaments canviant els increments $\Delta y_{seg}, \Delta y_{pix}$ per Δx_{seg} i Δx_{pix}. D'aquesta manera podem escriure un algorisme que activi simultàniament tots els píxels que aproximen un segment rectilini caracteritzat pel seus punts inicial i

final, en base a uns càlculs inicial previs que només es realitzen un cop per fila i un cop per columna, per cada cas. A continuació presentem un algorisme formal paral·lel amb pseudocodi:

```
Proc Non_incr_ld (x1, y1, x2, y2 : integer);
var
  Cinc[N], Rinc[M], i, j : integer;
  m : double;
begin
  m := (y2 − y1) / (x2 − x1);   // Slope
  if (−1 ≤ m ≤ 1) then begin
    for all Column i in [0..N]   // Columns tasks
      Cinc[i] := Round(m * (i − x1));
    for all Row j in [0..M]   // Rows tasks
      Finc[j] = j − y1;
  end
  else begin
    for all Column i in [0..N]   // Columns tasks
      Cinc[i] = i − x1;
    for all Row j in [0..M]   // Rows tasks
      Finc[j] = Round((j − y1) / m);
  end;
  wait_tasks();   // Waiting end of Columns / Rows tasks
  for all Pixels (i, j) in [0..N][0..M]   // Pixels tasks
    if (Cinc[i] == Finc[j]) then begin
      SetPixel(i, j);
    end;
end;
```

4.1.2.4. Comparació amb l'algorisme clàssic de Bresenham

Hem vist que en l'apartat 4.1.2.2 que l'error que cometem alhora de triar els píxels que millor aproximen un segment caracteritzat per els punts inicial i final, és com a màxim de ½ píxel. Es pot afirmar per tant que l'aproximació és valida i que l'aparença rectilínia dels segments obtinguts serà acceptable, sempre que les dimensions del píxel

siguin suficientment petites. Per altra banda, l'algorisme respecte per definició els punts inicial i final, ja que estan definits en el pla discret.

Hem obtingut un resultat en el traçat de línies molt semblant al que proporciona l'algorisme clàssic més utilitzat, el de Bresenham amb l'avantatge que es poden calcular tots els píxels que aproximen el segment de forma independent a partir del punt inicial i final, mentre que l'algorisme clàssic és incremental i necessita conèixer el píxel anterior per calcular el següent.

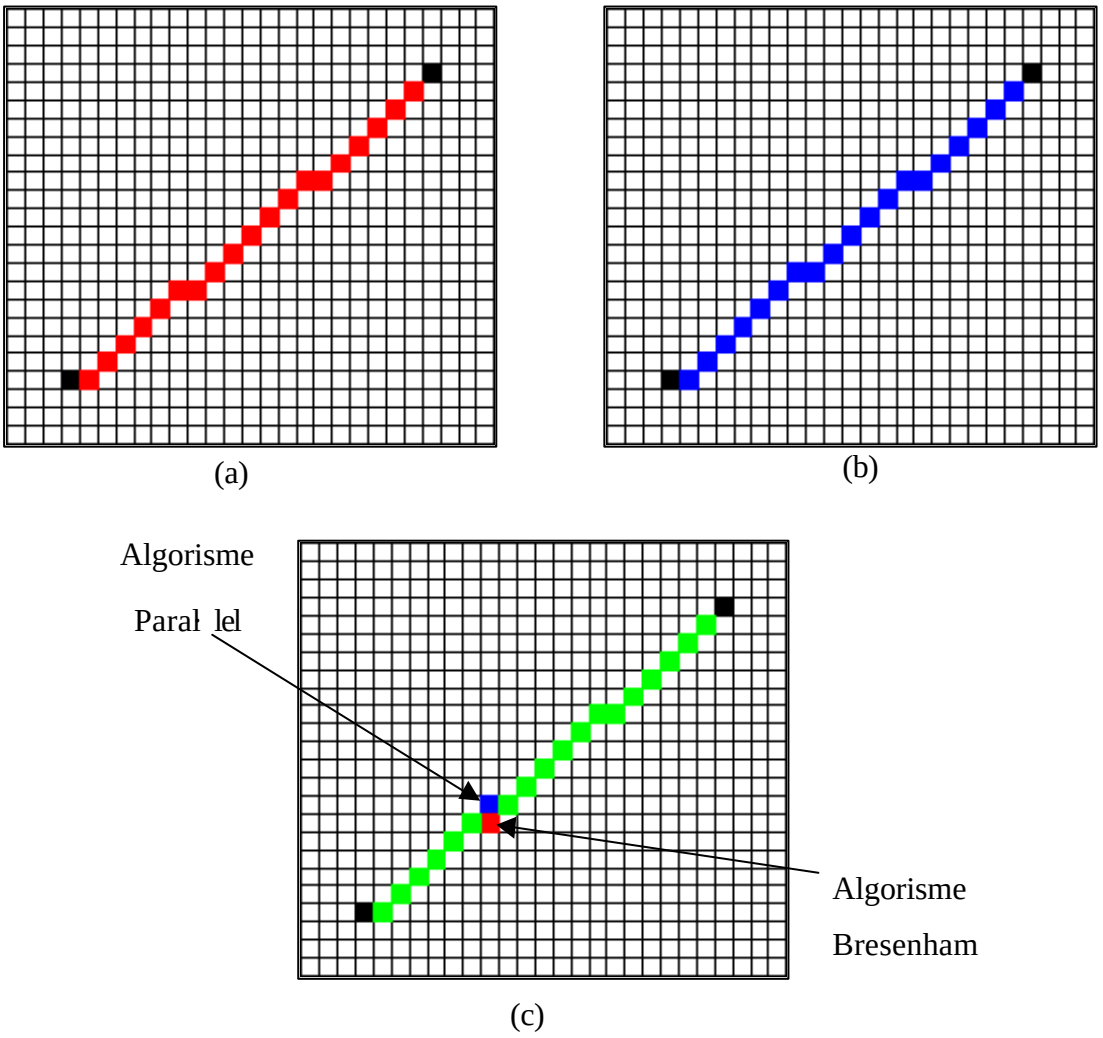


Figura 4.5: Comparació de l'algorisme paral·lel amb l'algorisme clàssic de Bresenham per le segment $P_{in}(0,0)$, $P_{fi}(20,17)$. (a) Algorisme de Bresenham. (b) Algorisme paral·lel. (c) Comparació entre els dos algorismes.

S'han obert dues possibilitats, per un costat, com a conseqüència de la independència del càlcul de cada píxel, s'obté un processament paral·lel molt eficient, tal com es dedueix de l'algorisme formal que hem presentat. Per l'altra costat, la seva senzillesa proporciona la possibilitat d'obtenir un processador dedicat molt eficient, com mostrarem en el apartat següent.

Les operacions efectuades per els dos algorismes, per tal d'obtenir els píxels que aproximen el segment, són del tot diferent. Mentre que l'algorisme de Bresenham decideix el salt de la coordenada de menys variació a partir de l'error acumulat, l'algorisme paral·lel, que aquí hem presentat, tria per cada columna el píxel que millor aproxima la pendent. És doncs evident, que tot i que l'aparença del segments traçat serà molt similar, els salts dels píxels es donaran en llocs diferents.

Per tal de visualitzar aquest fenomen hem generat una simulació dels dos algorismes. A la Fig 3.5 mostrem un cas per un segment caracteritzat per el punts $P_{in}(0,0)$, $P_{fi}(20,17)$, on es pot veure que els dos algorismes trien els mateixos píxels, a excepció de la coordenada $x=6$ on l'algorisme paral·lel tria $y=7$ mentre que Bresenham tria $y=6$.

En general el resultat de la simulació mostra que els píxels concedeixen totalment per les pendents ± 1 i les diferencies augmenten lleugerament a mesura que ens allunyem d'aquesta pendent.

4.2 Arquitectura paral·lela dedicada al càlcul del mapa d'acumulació

Segons tot el que s'ha dit fins ara un processador específic que calculi en paral·lel el mapa d'acumulació (veure apartat 3.7), ha de complir:

- Disposar d'un registre associat a cada píxel ($N \times M$ registres de píxel)

- Ser capaç de seleccionar simultàniament els píxels que aproximen el segment rectilini que presentem a l'arquitectura, especificat pels punts inicial i final

- Ser capaç d'activar simultàniament els registres dels píxels seleccionats

- Ser capaç d'incrementar simultàniament els registres del píxels seleccionats, amb el valor de ponderació P associat al segment de contribució a la simetria $\overline{SCSL}$ que presentem a l'arquitectura, per tal d'acumular, de forma ponderada, les superposicions de segments (veure ap. 3.4.3 i 3.7.2).

Però hem de fer notar que aquestes condicions es compleixen amb la implementació hardware de l'algorisme paral·lel del apartat anterior. La única modificació consisteix que, enlloc d'activar simplement els registres dels píxels que aproximen els segments que presentem a l'arquitectura, hem de disposar d'un mecanisme addicional que incrementi aquest registres amb el valor de ponderació P associat als segments de contribució. A nivell de l'algorisme anterior aquesta modificació consisteix en substituir la sentència $setPixel(i, j)$ per $Pixel(i,j) = Pixel(i,j) + P_s$. D'aquesta forma, després de presentar a l'arquitectura tots els segments de contribució a la simetria $\overline{SCSL}$, els registres de píxel contindran un valor d'acumulació, relacionat amb el nombre de superposicions ponderades que ha tingut cada píxels,

Per altra banda, segons l'algorisme anterior, hem mostrat que si disposem dels increments de fila F_{inc} i columna C_{inc}, es pot determinar si un píxel qualsevol pertany al conjunt de píxels que aproximen el segment rectilini, especificat pels seus punts inicial i final. Així doncs, per cada píxel, a part de disposar del registre d'acumulació citat anteriorment, es necessita una certa capacitat de processament per tal de comparar els increments anteriors i sumar el contingut dels registres de píxel amb els valors de ponderació del segment que s'està tractant. Cal fer notar que aquest procediment només serà justificable si la cel·la de píxel resultant és suficientment petita, per tal de ser replicada $N \times M$ cops, on aquest nombre depèn de la resolució de la imatge que es vol tractar.

4.2.1. Cel·la Processadora de píxel

A la Fig. 4.6 mostrem la primera aproximació de la cel·la processadora del *i*-esim píxel que implementa les funcions que hem comentat en l'apartat anterior deduïdes de l'algorisme paral·lel de traçat de línies. Com es pot veure la cel·la consisteix, per un

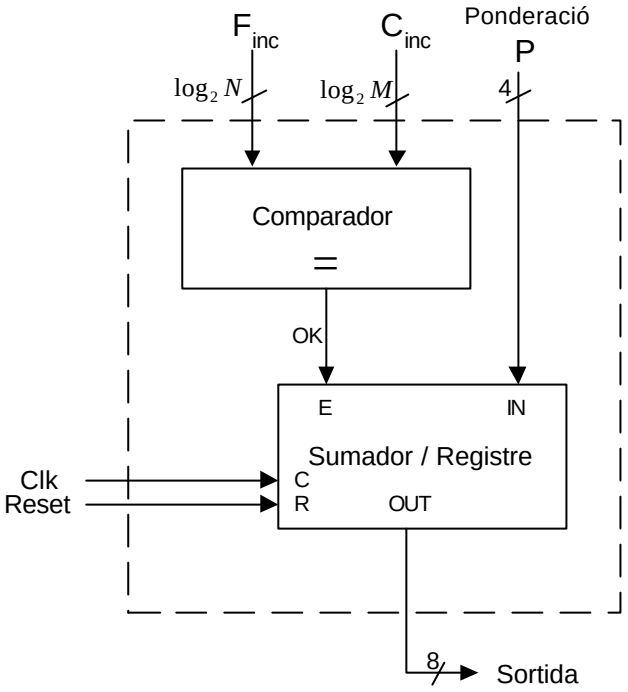


Figura 4.6: Cel· la processadora de píxel

costat, en un comparador que verifica la igualtat entre els increments de fila F_{inc} i de columna C_{inc}. Si el píxel associat a la cel·la processadora pertany al segment que s'ha presentat a la arquitectura, els increments anteriors seran iguals i el comparador activarà la senyal *OK*. Per un altre costat, un sumador amb registre acumula i memoritza la ponderació dels diferents segments de contribució que es presenten a l'arquitectura, sempre que el píxel pertanyi al segment. (activació del senyal d'igualtat dels increments *OK).*

Com hem dit a la introducció d'aquest apartat, la cel·la processadora s'ha de replicar $N \times M$ cops i aquest nombre depèn de la resolució de la imatge que es vol tractar. Per petita que sigui aquesta resolució el nombre serà molt elevat. Cal doncs analitzar les dimensions d'aquesta cel·la.

Pel que fa al comparador, tenint en compta que ens movem en el pla discret els

increments de fila F_{inc} i columna C_{inc} estan acotats següent manera:

$$0 \le F_{inc} \le N$$
$$0 \le C_{inc} \le M$$

S'han de comparar doncs, $log_2 N$ bits amb $log_2 M$ bits tal com es mostra a la Fig 4.6, això significa que per una imatge 256×256, el comparador haurà de ser de 16 bits, la qual cosa és assumible.

La unitat de suma i registre sembla més complexa. Però s'ha de tenim en compte que :

- La ponderació esta acotada entre $1 \le P \le 10$ i es pot representar per 4 bits: (veure apartat 3.4.3)

- El valor d'acumulació es pot acotar entre $0 \le valor\_acumul \le 255$ i es pot representar per 8 bits. Això representa que acotem també el nombre de superposicions de segments de contribució. En el pitjor dels casos, on tots els segments tinguessin una ponderació màxima, tindríem:

$$\max(\text{nº sup } erposicions) = \frac{\max(valor\_acumul)}{\max(p)} = \frac{255}{10} = 25$$

Aquest nombre pot semblar insuficient però cal tenim en compte que les imatges que tractem (construccions artificials o entorns estructurats), on les línies rectes són predominats, moltes parelles de segments són paral·leles o formen angles petits entre ells de forma que generen segments de contribució amb ponderacions pròximes a la mínima ($P \approx 1$) o en tot cas molt inferiors a les ponderació màximes ($P < 5$). Aquest fet fa que el nombre màxim de superposicions possibles, creixi considerablement. Recordem també que a la pràctica existeix una dispersió inevitable en la posició dels segments de contribució que fa que el nombre de superposicions sigui inferior al esperat. Segons les consideracions anteriors podem concloure que l'acotació anterior no significa una limitació real.

Per tant, l'amplada del registre de la cel·la de píxel, així com el bus de sortida queden fixats a 8 bits per tal de tractar el màxim valor d'acumulació.
Per altra banda, el fet que el valor d'acumulació s'incrementi cada cop amb un valor d'amplada màxima de 4 bits, pot ser aprofitat per obtenir un sumador òptim alhora de la implementació VLSI. Segons això la cel·la processadora obtinguda ha d'admetre la replicació de $N \times M$ cops, però les dimensions màximes de les imatges que podrem tractar dependrà de la tecnologia VLSI emprada i dels resultats del disseny i simulació que veurem en el capítol 6

4.2.2. Unitats de càlcul de fila i de columna

Hem vist que el processador de píxel es basa en la comparació dels increments de fila i columna. Per tant, s'haurà de disposar d'una unitat de càlcul d'aquests increments, per cada fila i columna de píxels existents a la imatge que tractem. Això vol dir que en funció de la resolució tindrem $N + M$ unitats de càlcul.

Segons l'algorisme de l'apartat 4.1.2.3 aquestes unitats han d'implementar els càlculs dels increments de fila i columna, per els dos casos simètrics, en funció de les diferents pendents dels segments a tractar, expressats per les equacions :

per el cas $-1 \le m \le 1$:	*pel cas* $-\infty \le m \le -1, 1 \le m \le \infty$:
$C_{inc}[i] := Round(m*(i - x_1));$	$C_{inc}[i] = i - x_1;$
$F_{inc}[j] = j - y_1;$	$F_{inc}[j] = Round\left(\frac{(j - y_1)}{m}\right);$

A la Fig.4.7 mostrem el disseny de les unitats de càlcul de la columna *i* (verticals) i de les unitats de càlcul de fila *j* (horitzontals). Es pot veure que hem aprofitat el fet que el valor de la columna *j* i la fila *i* són coneguts a cada unitat, per reduir l'arquitectura. Un multiplexor 2:1 controlat pel senyal m_1 / m_2 selecciona els dos casos anteriors. Per altra banda la forma més òptima i a la vegada més ràpida d'obtenir el càlcul de les funcions $Round(m*(i - x_1)), Round((j - y_1)/m)$ és mitjançant una tabulació LUT. Finalment

per completar el càlcul de les equacions, $i - x_1$ i $j - y_1$ no queda més que afegir un restador a cada unitat.

No obstant cal tenir en compte que la capacitat de les LUT poden comprometre el disseny ja que s'han de replicar $N + M$ cops. La primera problemàtica a resoldre és que la pendent m és una variable real i contínua. Una discretització en coma flotant ens portaria a LUTs molt grans. No obstant, cal fer notar que per una imatge discreta de resolució $N \times M$ píxels només existeixen $2*(N+M)$ pendents diferents. De forma que

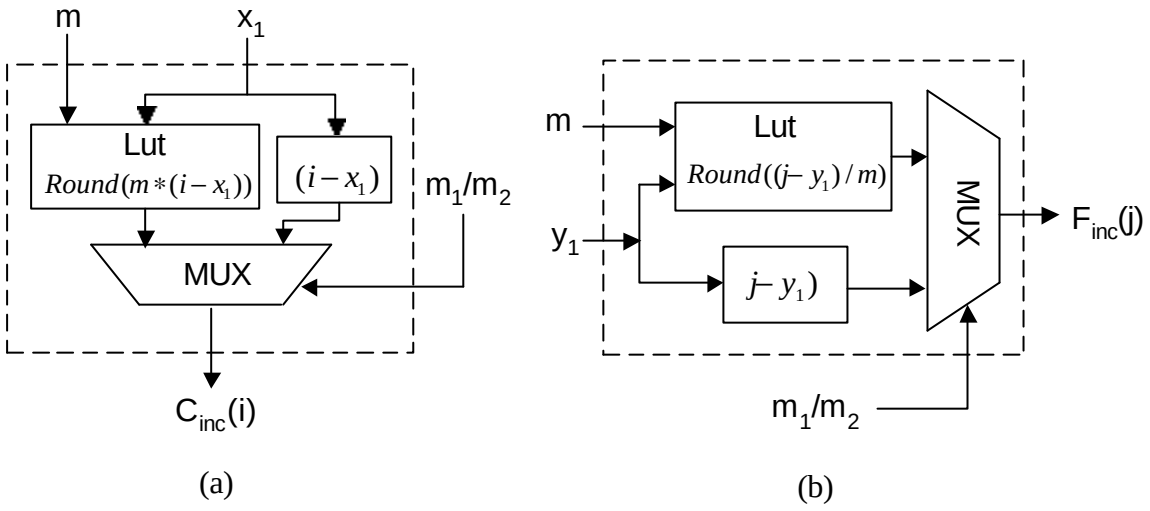


Figura 4.7: Unitats de càlcul. (a) Càlcul de la columna i. (b) Càlcul de la fila j

es necessiten $\log_2(2*(N+M))$ bits per definir la pendent.

Pel que fa a l'entrada de la coordenada inicial del segment x_1 o y_1, es necessiten $\log_2(N)$ bits més. Això vol dir que per una simple imatge de *256x256* calen *10* bits per la pendent *m* i 8 bits per x_1, y_1, és a dir, *256K* paraules. Tenint en compte que s'haurien de replicar *512* cops, queda clar que d'aquesta forma el disseny és impracticable.

4.2.3. Simplificació de les unitats de càlcul

Per tal de reduir les dimensions de les LUT's analitzem la unitat de càlcul de l'increment de columna per el cas $-1 \le m \le 1$, en el ben entès que la simetria de l'algorisme farà extensibles els raonaments a l'altra cas. Podem escriure l'equació de l'increment de la següent manera:

$$C_{inc}[i] = Round(m*(i - x_1)) = Round(m*i - m*x_1)$$

On es pot observar que el producte $m*x_1$ és independent de la columna *i*, en conseqüència es podria fer aquest càlcul comú per totes les columnes, fora de les unitats de càlcul, evitant així la replicació, si no fos per què la funció arrodoniment no té la propietat distributiva. Tot i així, per definició de la mateixa funció, l'error màxim que cometríem, és d'una unitat:

$$C_{inc}[i] = Round(m*i) - Round(m*x_1) + er$$

$$on \quad \max(er) = \begin{cases} 1 & 1^{er}\ arrodoniment\ \ per\ excés\ 2^{on}\ per\ defecte \\ -1 & 1^{er}\ arrodoniment\ \ per\ defete\ 2^{on}\ per\ excés \end{cases}$$

Això significa que l'algorisme decidirà incrementar la coordenada de menys variació un píxel abans o després del que ho faria amb el càlcul sense la descomposició anterior. Com que l'error no és acumulatiu només pot canviar l'aspecte d'algunes rectes en concret però els dos tipus de càlculs donen aproximacions que són valides perquè respecten la pendent.

A la Fig 4.8 es pot veure una comparació entre els càlculs amb i sense aproximació, per un cas simulat on hem escollit una recta amb $P_{in}(2,6)\ i\ m = 23$ que tria píxels molt diferents per els dos casos. Es pot observar que tot i així l'aparença de les rectes és molt similar, respecte la pendent i el punt inicial. A la Fig 4.8 (c) es mostra els píxels que coincideixen (color verd) i els que difereixen a les dues rectes obtingudes.

Aplicarem doncs aquesta simplificació a les unitats de càlcul. La Fig. 4.9 (a) mostra la unitat de càlcul comú resultant per totes les columnes, CC i la fig. 4.9 (b) mostra com queden les unitats de càlcul dedicades de cada columna, CD_i. Com es pot veure s'ha obtingut una simplificació de la LUT de columna, pel fet de reduir les entrades a només la pendent *m*, i s'ha modificat l'arquitectura per tal de no afegir cap nou restador. Per adreçar la nova LUT només caldran $log_2(2*(N+M))$ bits, que per una imatge de

$256x256$ signifiquen *10* bit o *1k* paraules, que representa una reduccio de més de dos ordres de magnitud, fent viable la replicació i per tant el disseny que ens proposem.

Els mateixos càlculs i raonaments es poden aplicar, per tal d'obtenir les mateixes simplificacions a les unitats de càlcul de files, obtenint la unitat de càlcul comú a totes les files, FC (Fig. 4.10 (a)) i les unitats de càlcul dedicades de cada fila, FD_j Fig. 4.10 (b).

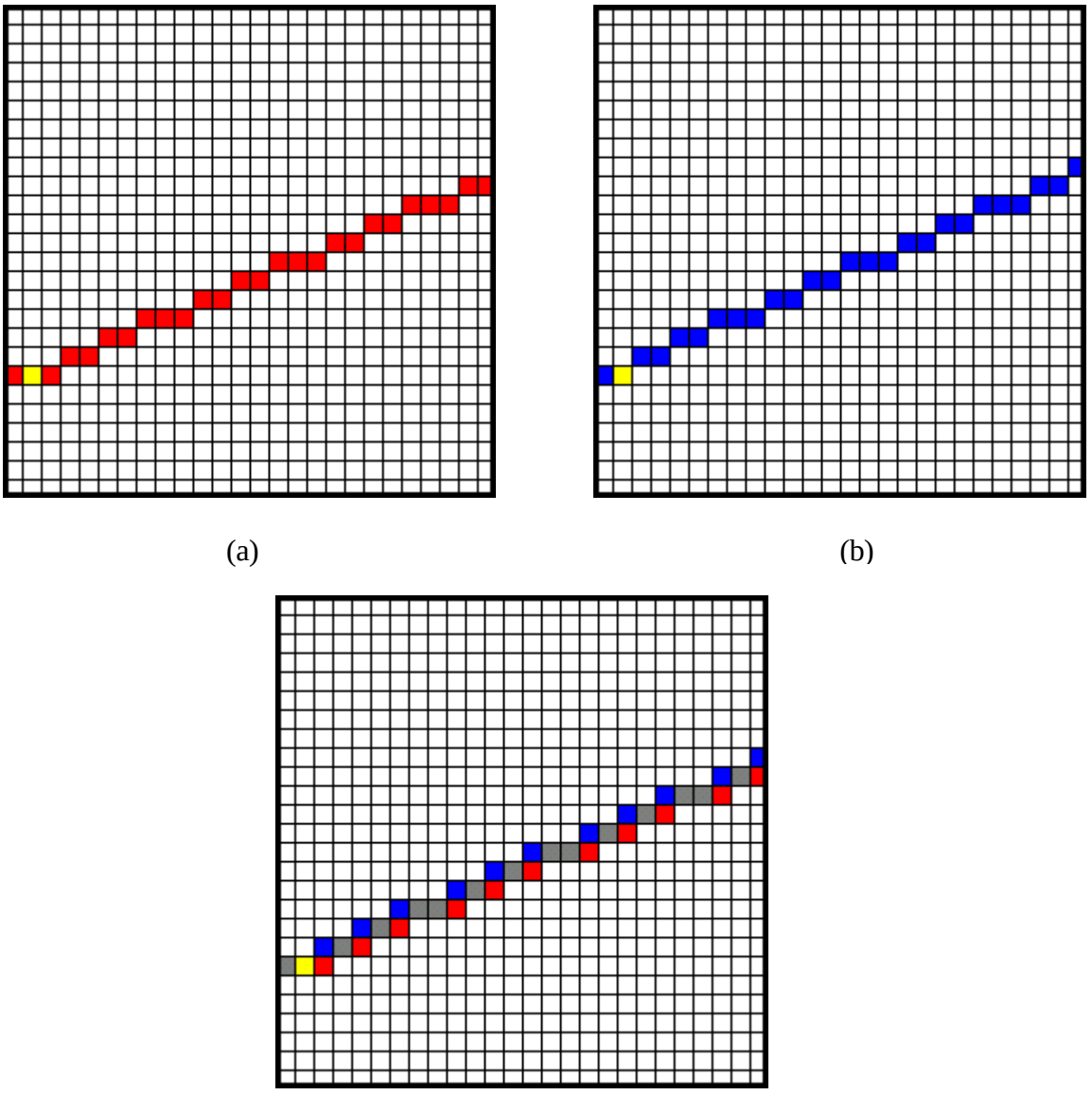


Figura 4.8: Un cas simulat de comparació entre el càlcul de l'arrodoniment de la diferencia i l'aproximació de la diferencia d'arrodoniments. (a) Recta sense aproximació. (b) Recta amb aproximació. (c) Coincidències i diferències

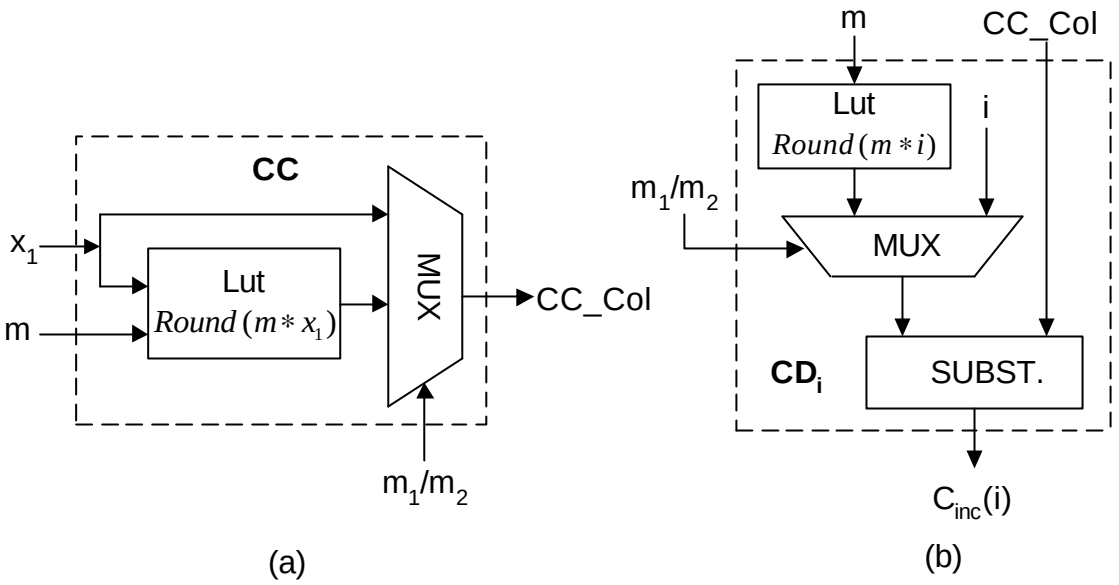


Figura4.9: Simplificació de les unitats de càlcul de columna. (a) CC càlcul comú per totes les columnes. (b) CD_i càlcul dedicat a la columna i

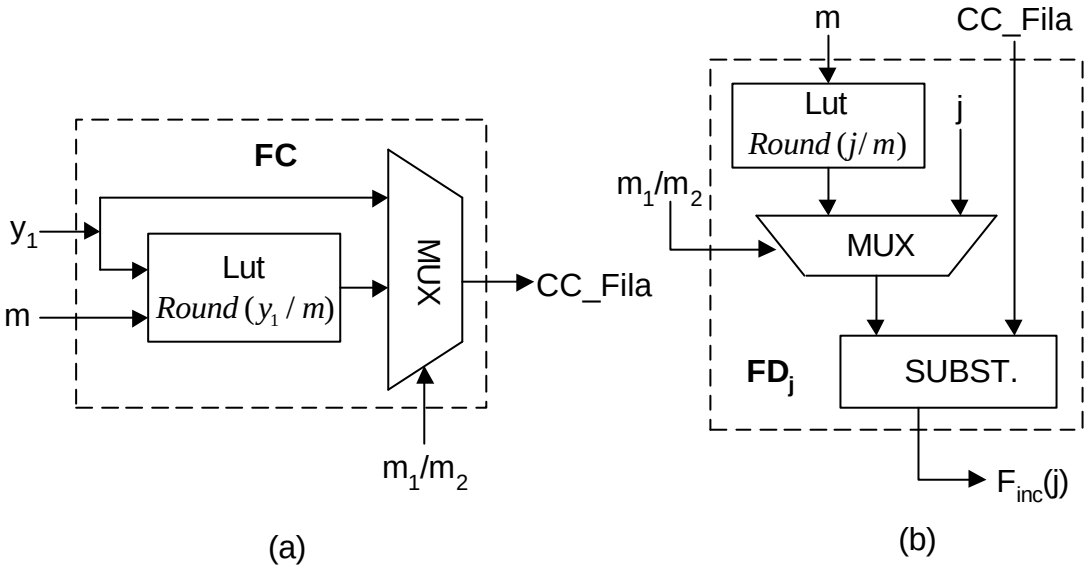


Figura 4.10: Simplificació de les unitats de càlcul de fila. (a) CC calcul comú per totes les files. (b) FD_j càlcul dedicat a la fila j

4.2.4. Matriu de cel·les processadores de píxel

Com es pot veure a la Fig. 4.11 l'arquitectura que hem desenvolupat fins aquí, consta d'una matriu de $N \times M$ cel·les processadores de píxels (CPP), de N unitats de càlcul de fila (FD_i) i de M unitats de càlcul de columna (CDi) i dos unitats de càlculs comuns, la unitat comuna de fila (FC) i la unitat comuna de columna (CC) .

Com hem vist en l'apartat anterior aquestes unitats proporcionen simultàniament a les cel·les processadores, els increments de fila F_{inc} i de columna C_{inc}, a partir dels càlculs comuns de fila i columna. Les cel·les processadores, a partir d'aquest increments, determinen simultàniament si el registre del píxel, que tenen associat, s'incrementa amb el valor de ponderació del segment de contribució a la simetria, que hem presentat a l'arquitectura.

Si presentem a l'arquitectura el conjunt de segments de contribució a la simetria $\overline{SCSL}$ (veure apart. 3.4.2), els registres de les cel·les processadores s'aniran incrementant en funció de les superposicions dels segments i dels valors de ponderació associats, de forma que al final del procés, la matriu de registres constituirà el mapa d'acumulació dels segments de contribució a la simetria, que es va definir en el capítol anterior en l'apartat 3.7.

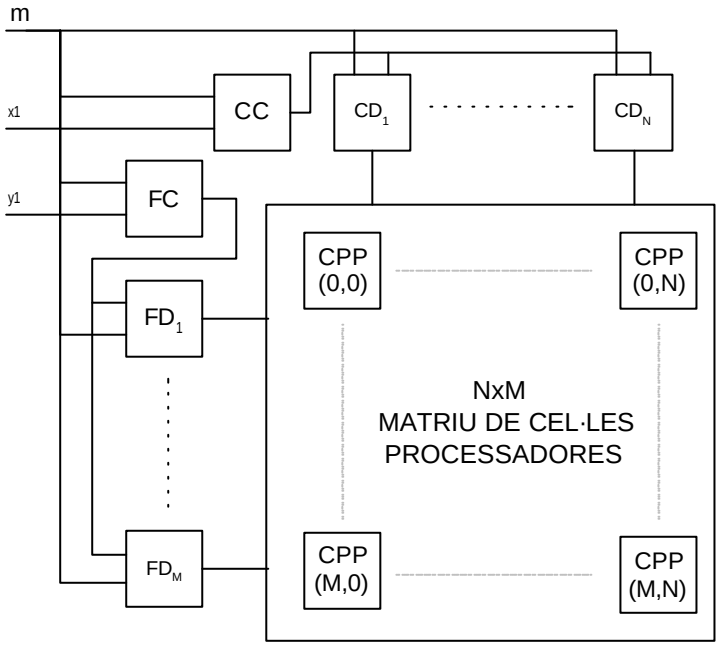


Figura 4.11: Interconnexió de la matriu de cel· les processadores amb les unitats de càlcul

4.2.5. Accés a la matriu de cel·les processadores de píxel

Fins aquí, l'arquitectura que tenim realitza de forma automàtica operacions d'escriptura al registres de la cel·la de píxels, com a resultat del processament de segments rectilinis, a partir del senyal *OK* del comparador de la cel·la de píxel (Fig.4.6). Ara hem de dotar l'arquitectura de la possibilitat de llegir els valors d'acumulació dels píxels,

emmagatzemats en el registres associats. Per acomplir aquest objectiu s'ha de decidir el tipus d'accés del qual volem dotar l'arquitectura, és a dir, bàsicament si l'accés a de ser aleatori o seqüencial. No obstant aquesta decisió està condicionada per l'ús que fa, l'algorisme de detecció de simetria, del mapa d'acumulació. Recordem (veure "Extracció dels eixos de simetria" en l'apartat 3.8 del capítol anterior) que s'ha d'identificar en el mapa d'acumulació les alineacions rectilínies amb valors d'acumulació més elevats i això implica la possibilitat d'un accés aleatori en el mapa. Segons això s'ha de dotar l'arquitectura de les següents funcions:

- Selecció aleatòria de la cel·la processadora que ha de ser accedida
- Accés al registre de la cel·la processadora seleccionada
- Portar a la sortida el contingut del registre de la cel·la processadora seleccionada

El primer punt es resol de forma clàssica, és a dir, disposant una unitat de descodificació de fila i columna, que dotarà l'arquitectura de línies de descodificació verticals (Column-i) i horitzontals (Fila-j) (Fig. 4.12). A la Fig. 4.14 es poden veure les unitats que descodifiquen l'adreça del bus extern de fila i columna, de la cel·la a la qual es vol accedir i que generen les línies de descodificació de fila i columna (Ci, Fj).

Respecte la cel·la processadora de píxel, la Fig.4.12 mostra les modificacions necessàries per tal de dotar-la de les prestacions citades. Hem incorporat el senyal típic de lectura $\overline{RD}$ que, combinat amb l'adreçament proporcionat per les línies de descodificació, habiliten la connexió de la sortida de tres estats al bus de sortida, comú a totes les cel·les. D'aquesta forma la cel·la processadora és accedida de dos formes diferents :

- Escriptura automàtica, com a resultat del processament de segments rectilinis
- Lectura aleatòria del contingut del registre.

Cal fer notar que aquests dos funcionaments són mútuament exclusius pel fet que els accessos de lectura no tenen sentit fins que el procés d'acumulació ha estat completat.

Per altra banda hem de dir que aquesta arquitectura s'assembla molt a l'estructura d'una memòria d'accés aleatori amb la diferència que l'arquitectura que aquí presentem té, a més a més, la capacitat de processar segments rectilinis de forma paral·lela i acumular les seves superposicions.

Finalment haurem de dissenyar una etapa d'entrada/sortida que serà la interfície amb el bus extern. Abans, però analitzarem els busos interns de l'arquitectura.

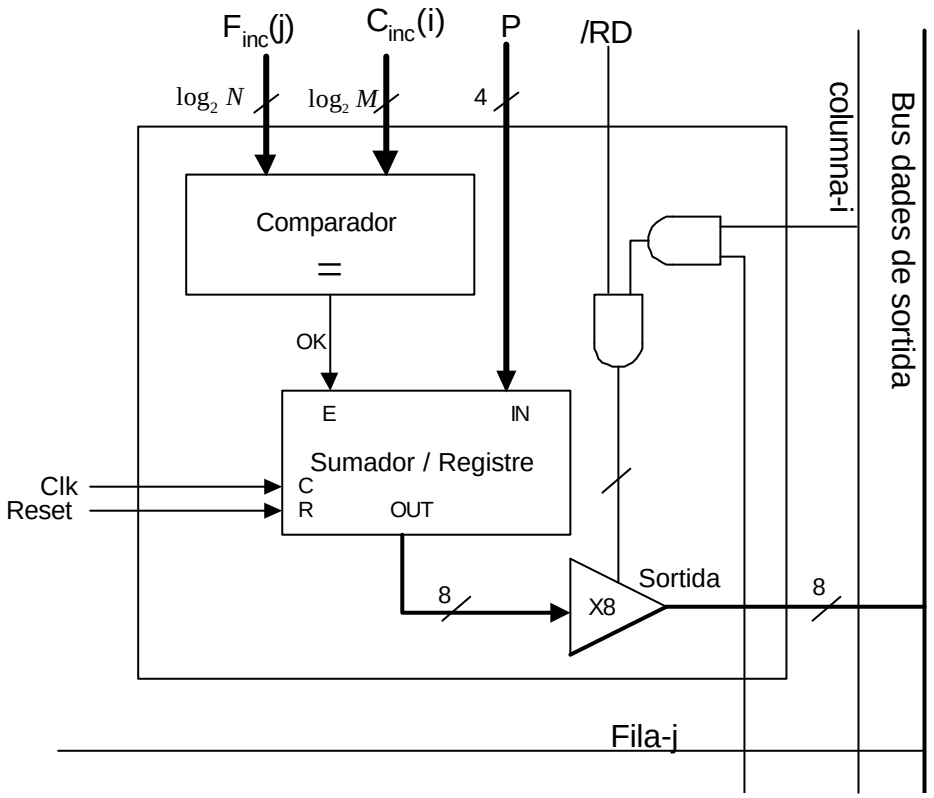


Figura 4.12: Cel· la Processadora de píxel amb lògica d'accés

4.2.6. Busos de l'arquitectura interns

A la Fig.4.14 es pot veure la distribució dels busos principals dins i fora de la matriu de cel·les processadores de píxel. Així, per tal de distribuir els resultats de les unitats de càlcul hem de dotar el disseny de *N* busos horitzontals d'amplada $log_2(N)$ i *M* busos

verticals d'amplada $log_2(M)$, que entren a la matriu i es distribueixen a totes les cel·les de píxel .

Les sortides de les cel·les de píxel s'uneixen en un bus de 8 bits que constitueix els bus de dades de sortida, el qual recull el valor d'acumulació de la cel·la seleccionada en les operacions de lectura i el porta a l'etapa de sortida.
Les dades d'entrada necessàries per el processament de segments rectilínies, punt inicial i final $x_{1,}y_{2,}x_{1,}y_{2,}$ i la ponderació P constitueixen el bus de dades d'entrada. Tenint en compte que ens movem en el pla discret necessitarem $2 \times log_2(N)$ bits pel punt inicial, $2 \times log_2(M)$ bits pel punt final i *4* bits per la ponderació associada. Tot i que, depenen de la resolució de la imatge, l'amplada d'aquest bus pot resultar considerable s'ha de tenir en compte que només els *4* bits de la ponderació entren a la matriu de cel·les de píxel, mentre que la resta es distribueixen a les *N* unitats de càlcul de columna i a les *M* unitats de càlcul de fila (veure Fig 4.14).

Per altra banda, com les operacions de processament de segments rectilinis i els accessos de lectura són mútuament exclusius, tenim l'opció de multiplexar en el temps el bus de dades de sortida i convertir-lo en un bus bidireccional d'entrada/sortida de manera que podem distribuir el valor de ponderació *P* per el mateix bus. Aquesta opció planteja la problemàtica que s'hauria de dotar la cel·la processadora d'un multiplexor que fes la selecció entrada/sortida que representa un augment de $N \times M$ unitats més.

La pendent *m,* opcionalment, la pot calcular l'arquitectura a partir dels punts inicial i final. Aquestes dues opcions, que acabem d'esmentar, s'han de decidir en el disseny *VLSI* de l'arquitectura, en funció de la tecnologia escollida i de l'àrea de silici disponible. En el capítol 6 (Disseny i verificació VLSI del processador dedicat) podem donar la resposta.

Els senyals d'entrada $\overline{RD}$, *Reset* i *clk* constitueixen el bus de control que entra a la matriu i es distribueix per totes les cel·les de píxel. Aquest bus no s'ha explicitat en la Fig.4.11. per qüestions de claredat en el dibuix

4.2.7. Acotadors de longitud

Respecte el processament de segments rectilinis encara no complim les especificacions inicials ja que el disseny actual no delimita el segment definit pels seus punts inicial i final, sinó que tracta tots els píxels de la recta que passen per ell. Per tal de delimitar la longitud del segment incorporem unes noves unitats que anomenarem acotadors.

Per facilitat de disseny hem disposat una unitat acotadora de files i una altre de columnes. Donades les coordenades inicials i finals del segment a tractar, s'implementa, per cada fila o columna, una funció combinatòria que deshabita les cel·les de píxel que no estiguin compreses entre les columnes i files dels punts inicial i final, impedint que els seus registres es carreguin amb al valor de ponderació.

Com que les operacions d'accés aleatori i la de processament de segments rectilinis són exclusives en el temps utilitzarem les línies de selecció de fila i columna per aquesta funció. Conseqüentment les unitats d'acotació i descodificació compartiran aquestes línies multiplexades en el temps com es mostra a la Fig 4.14

4.2.8. Interfície al bus extern

La interfície de bus extern ha de resoldre la problemàtica del flux de dades d'entrada/sortida. L'arquitectura que presentem realitza dos tipus d'operacions que hem multiplexat en el temps.

Per un costat, una escriptura automàtica als registres de píxel com a resultat del processament de segments rectilinis, que precisa d'un flux de dades d'entrada consistent en els punts inicial i final (x_1,y_1) (x_2,y_2) i la ponderació *P* del segment de contribució.

Per l'altre costat, una lectura d'accés aleatori, que requereix de l'entrada, en els descodificadors de fila i columna, de l'adreça de la cel·la de píxel que volem llegir, la qual genera un flux de sortida consistent en el contingut del registre de píxel.
Aquests dos fluxos de dades es canalitzaran per un mateix bus de dades bidireccionals ja que estan multiplexats en el temps. Tal com es veu a la Fig 4.14, com en el cas de les memòries d'escriptura/ lectura, la interfície al bus extern disposa d'una unitat que

connecta el bus d'entrada o de sortida intern en funció del tipus d'operació, controlada per el senyal *RD*.

A la Fig 4.13 mostrem la connexió al bus extern i a la taula 4.1 es poden veure les diferents operacions amb les dades que apareixen a cada bus.

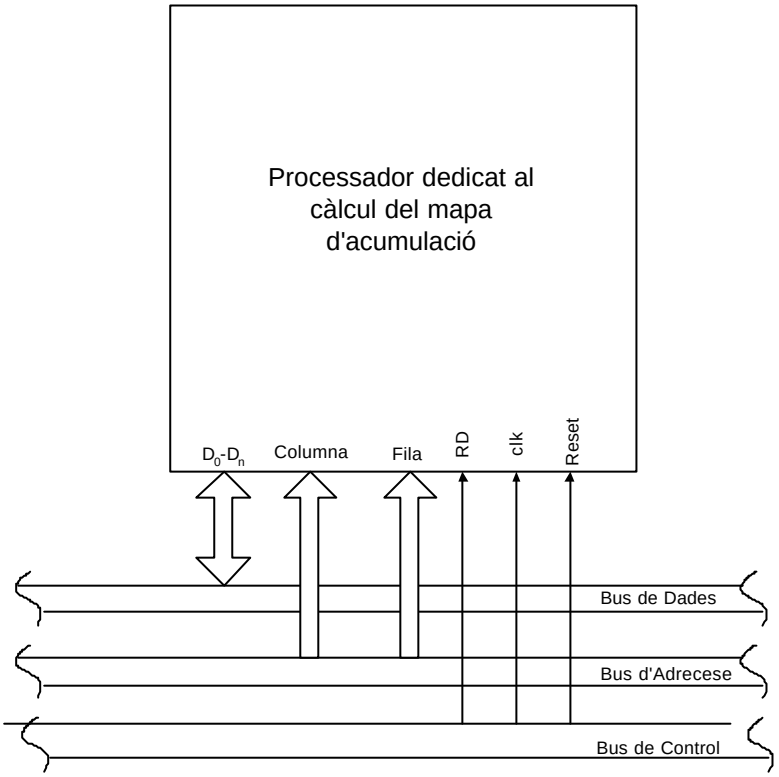


Figura 4.13: Connexió al bus extern

RD	Columna	Fila	D_0-D_n	Descripció
0	X	X	x_1, y_1, x_2 y_2, p	Processament de segments
1	@Columna	@Fila	Valor d'acumulació	Lectura Registre

Taula 4.1. Modes d'operació

CCP_i=Cel·la processadora de píxel
CC=Unitat de càlcul de columna comuna
FC=Unitat de càlcul de Fila comuna
CDi=Unitat de càlcul de la columna i
FDj=Unitat de càlcul de la fila j

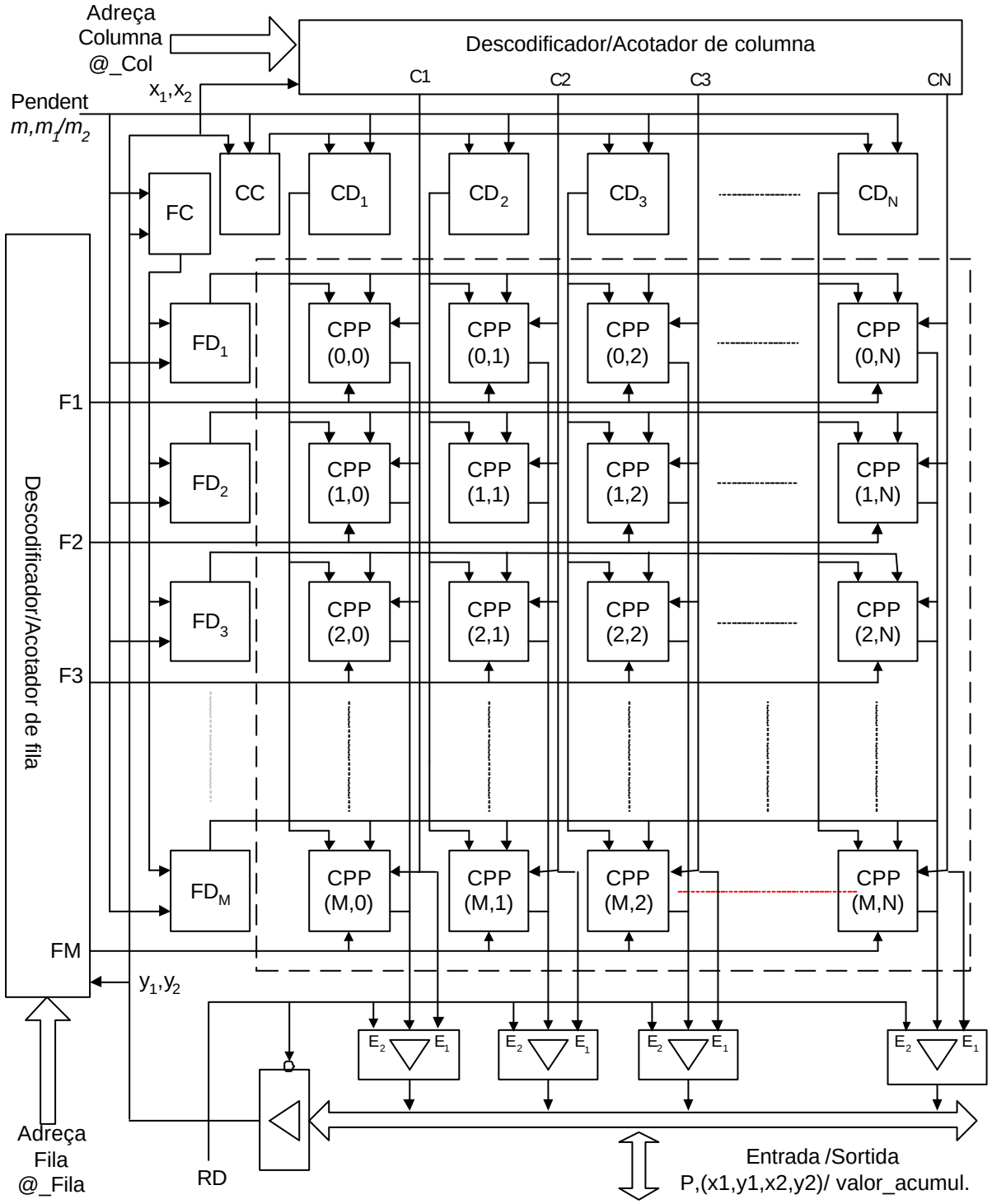


Figura 4.14: Arquitectura del processador dedicat al càlcul del mapa d'acumulació

4.3 Aplicacions gràfiques de l'arquitectura

L'arquitectura que hem obtingut, per tal d'obtenir el mapa d'acumulació, identifica en el pla discret els píxels que aproximen els segments rectlinis caracteritzats per les seus punts inicial i final, a partir d'un algorisme de traçat de linies no incremenal que hem presentat al principi del capítol i que hem concretat a partir de pseudocodi en el apartat 4.3.3. Queda clar per tant, que una de les parts importants del processador que hem dissenyat concideix amb la tascas del traçat de linies rectes.

Per altra banda, es pot intuir que amb petites modificacions de l'arquitectura bàsica, es poden donar altres aplicacions gràfiques al processador, tal com el traçat i ompliment de diferents figures geomètriques en paral·lel, sobretot en el cas de triangles considerant les aplicacions gràfiques que es deriven.

Presentem a continuació les modificacions del disseny per fer operativa la funció del traçat de línies rectes. L'arquitectura és molt eficient en aquesta tasca doncs té la capacitat d'obtenir en paral·lel els píxels que aproximen, en el pla discret, uns segments rectlinis caracteritzats per les seus punts inicial i final, amb un únic període de rellotge. Pel que fa a l'estudi i adaptació a les altres possibles aplicacions gràfiques, considerem que estan fora dels objectius d'aquest tesi, tot i que pot representar una interessant línia de treballs de recerca futurs.

4.3.1. Traçat de línies rectes en paral·lel

La modificació de l'arquitectura per dotar-la de la capacitat bàsica d'identificar i emmagatzemar els píxels que millor aproximen un determinat segments rectilini, és en realitat una simplificació del disseny que hem presentat. La modificació només afecta a la cel·la de píxel, mentre que les unitats de càlcul i la resta d'unitats queden intactes, ja que han de realitzar la mateixa funció que en el cas del càlcul del mapa d'acumulació.

Com mostra la Fig.4.15 la simplificació respecte la cel·la que intervenia en el càlcul del mapa d'acumulació Fig. 4.12, consisteix en alliberar la cel·la de píxel de la tasca de sumar la ponderació amb els valors d'acumulació, emmagatzemats en els registres de la cel·la

del segment de contribució a la simetria que s'està tractant. Per tant, la cel·la ha de complir simplement dos funcions. La primera, una comparació dels increments de fila i columna, proporcionats per les unitats de càlcul . Com hem vist en l'algorisme paral·lel que hem presentat en apartat 4.1.2.3, aquesta comparació determina si el píxel associat a la cel·la pertany al conjunt dels píxels que aproximen el segment rectilini que s'està tractant. La segona, emmagatzemar els píxels actius en el traçat de línies amb la seva paraula de color.

Així doncs com mostra la Fig.4.15 la cel·la processadora de píxel, per tal de detectar els píxels que aproximen, en el pla discret, un determinat segment rectilini disposa simplement d'un comparador i per tal emmagatzemar la paraula de color disposa d'un

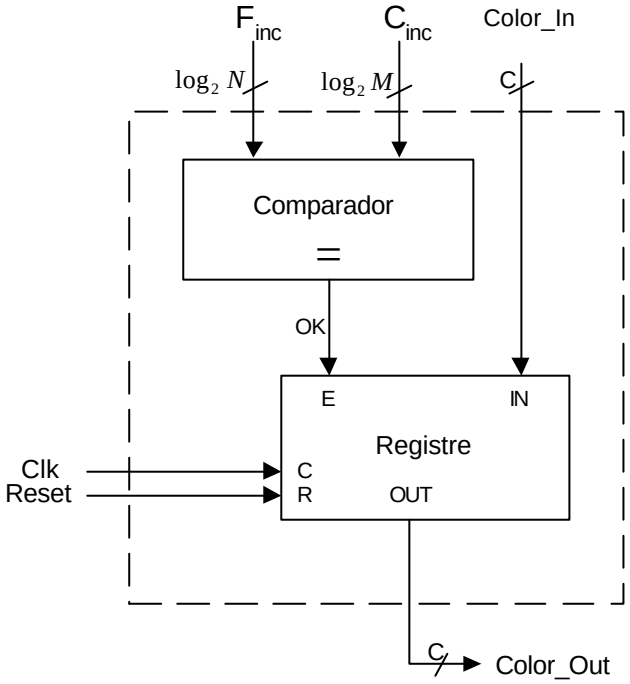


Figura 4.15: Cel· la processadora de píxel per la tasca de traçat de línies

registre, l'amplada del qual dependrà del tractament de color que es vulgui aplicar. Constatem doncs que la cel·la processadora de píxel per tal d'acomplir la tasca del traçat de línies rectes és significativament més reduï da.

Tenint en compte l'elevada replicació d'aquesta unitat, la simplificació anterior reportarà una disminució molt considerable de l'àrea de silici necessària per la seva implementació i fa pensar que les resolucions que podrem tractar en un sol circuit seran d'ordre superior al cas del càlcul del mapa d'acumulació.

Per altra banda, la potència de càlcul serà també superior per el fet que en el cas que tractem la cel·la de píxel no ha de fer cap càlcul, sinó que únicament ha d'activar el registre de píxel, si les unitats de càlcul així ho indiquen i això representa un estalvi de temps.

4.3.2. Processador gràfic amb memòria integrada

L'aplicació del traçat de línies, a diferencia de l'obtenció del mapa d'acumulació, té com objectiu final la visualització i l'aproximació en el pla discret del les línies rectes que es presenten l'arquitectura, caracteritzades per els seus punts inicial i final. Lògicament és tant important identificar i emmagatzemar els píxels que aproximen la línia recta, com facilitar la visualització d'aquestes dades.

Modificarem el disseny per tal de configurar una sortida de dades automàtica i independent del processament de línies del tipus "*Raster*", és a dir, visualització seqüencial dels píxels per files, donat que els dispositius de visualització d'aquest tipus són els més utilitzats.

Per altra banda, per tal de facilitar la interacció d'altres tractaments gràfics amb l'arquitectura, mantindrem l'accés de lectura aleatori multiplexat que la cel·la ja disposava i afegirem la possibilitat d'escriptura.

La Fig.4.16 mostra com queda la cel·la de píxel un cop dotada de les prestacions descrites anteriorment. Aquest resultat es pot comparar respecte la cel·la de partida de la Fig.4.12.
Per un costat s'ha afegit una sortida independent que configura el bus Color_Out/Raster que subministrarà les dades a l'etapa de sortida de forma seqüencial i automàtica ("Raster"). Per tal d'habilitar la sortida de la paraula de color de la fila de píxels, el bus anterior s'ha de dotar de línies de selecció de fila que hem anomenat Raster_fila_j.
Per un altra costat, hem afegit l'operació d'escriptura aleatòria multiplexada amb el processament de línies ja que utilitzen el mateix bus entrada/sortida Color_In/Out i les mateixes línies de selecció de cel·la Fila-j i Columna-i. Les dos funcions es seleccionen a partir d'una nova senyal que hem anomenat Mode.

Finalment l'etapa de sortida com mostra la Fig.4.17 consisteix en un conjunt de registres desplaçament, capaços de carregar, cada un d'ells, un bit de la paraula de color de cada una de les ceł les processadores de píxel que composen una fila. Així aquest bits són p serialitzats de forma síncrona i seqüencial, configurant d'aquesta manera la sortida "*Raster* " esmentada anteriorment. Segons això es necessiten tants registres de desplaçament com bits tingui la paraula de color que tractem. Això pot semblar un inconvenient important però no ho és per el fet que l'arquitectura és escalable respecta

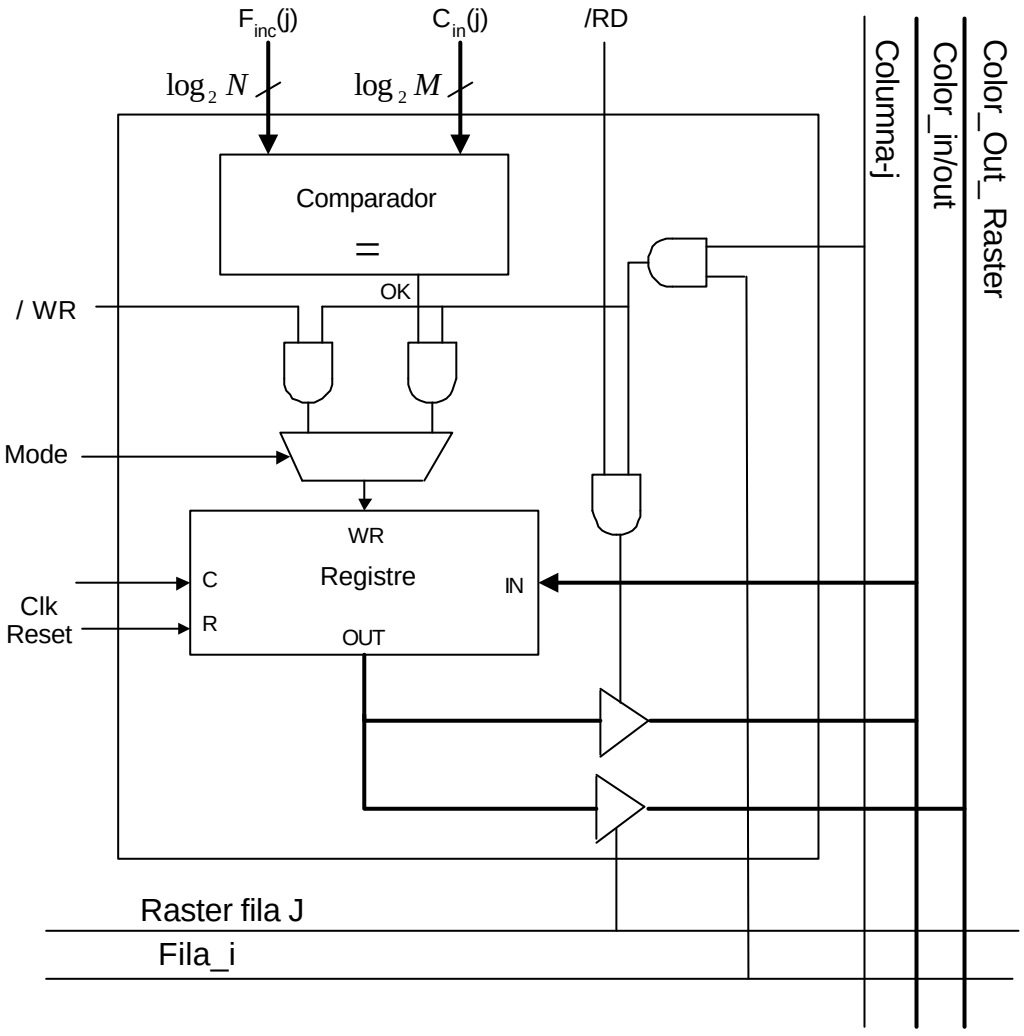


Figura 4.16: Cel· la de píxel amb accés lectura/escriptura aleatori i sortida seqüencial

l'amplada de la paraula de color, és a dir, que es poden apilar circuits idèntics on cada un d'ells treballa amb una part de la paraula de color. Això significa, per exemple, que per tractar amb color real, hauríem de disposar tres circuits amb registres de ceł la de 8 bits d'amplada i 8 registres de desplaçament a la sortida. .

La Fig.4.17 mostra circuit complet on s'observa que en realitat hem obtingut un processador gràfic especialitzat en el traçat de línies rectes amb memòria incorporada i sortida directa per dispositiu de visualització *"raster"* .

A l a Fig.4.18 mostrem la interfície a bus extern del processador gràfic i a la taula 4.2 els modes de funcionament memòria.

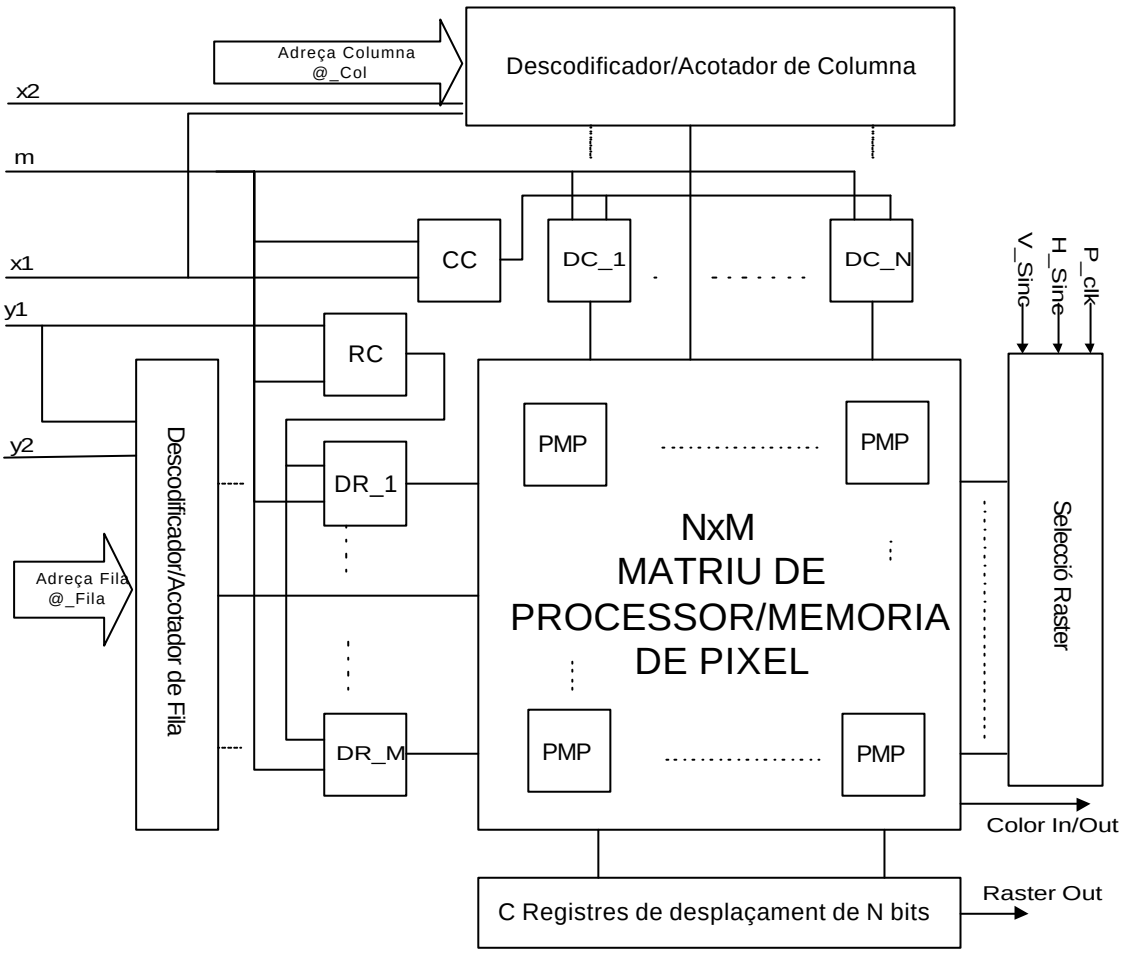


Figura 4.17: Processador gràfic amb memòria, orientat al traçat de línies rectes en paral· lel

Mode	RD	WR	Adreça	D_0-D_n	Funció
0	1	0	@_cel· la	Color_Out	Lectura paraula de color
0	0	1	@_cel· la	Color_In	Escriptura paraula de color
1	1	0	X	X	Reservat
1	0	1	X	x1,y1,x2,y2, Color	Traçat de línia

Taula 4.2 Modes d'operació del processador/memòria gràfic

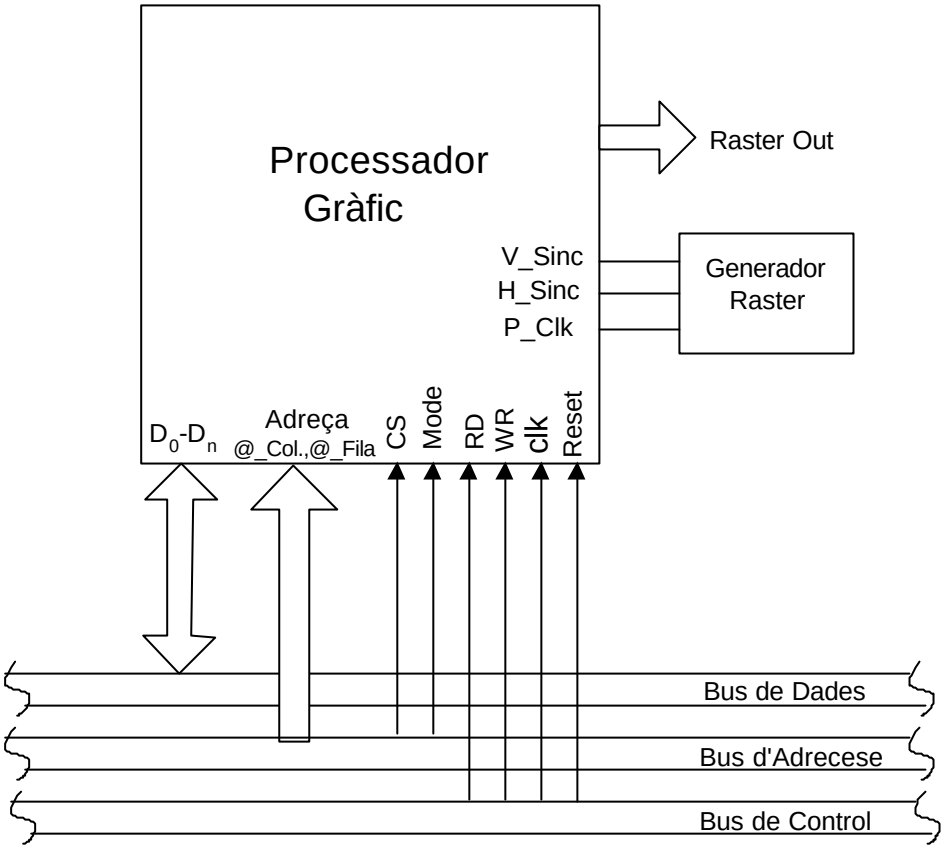


Figura 4.18: Connexió del processador gràfic al bus extern