www.CineInformatica.com

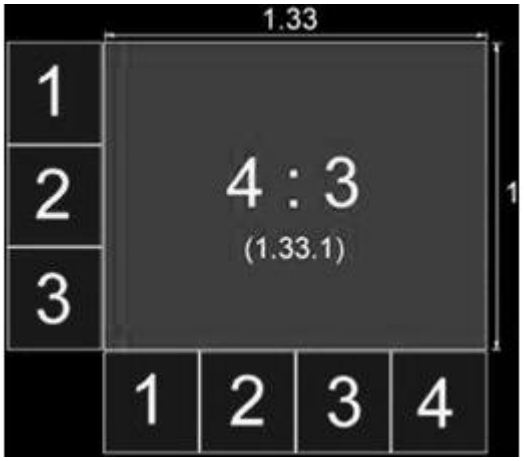
Guida BASE
Formati VIDEO

Guida BASE sui Formati VIDEO realizzata da Massimo Steri

L'Aspect Ratio

Un qualsiasi filmato è costituito da molti fotogrammi o **frames** messi in rapida sequenza. I classici televisori (ed anche i monitor per PC) hanno un rapporto larghezza/altezza dell'immagine (**aspect ratio**) 4:3. Ogni immagine trasmessa in TV è costituita da 30 (o 29.97 per i dispositivi a colori) frames al secondo (**fps**) se lo standard usato è l'***NTSC*** (America del Nord e Giappone) o 25 fps se lo standard è *PAL* (Europa, Africa Orientale, India, Australia, Cina). Ogni singolo frame è **interlacciato**, ovvero costituito da due 'mezze immagini' (*half frame* o *campi*) che appaiono e si combinano molto velocemente formando l'intero fotogramma; ogni *half frame* è costituito da righe alternate, uno contenente le righe dispari, l'altro quelle pari.

I recenti schermi panoramici hanno un rapporto larghezza/altezza 16:9 ed una più ampia superfice visiva orizzontale (1920 punti-immagine o **pixel**). Nelle TV 4:3 l'altezza dell'immagine è costituita da 480 linee visibili (*NTSC*) o 576 (*PAL*).

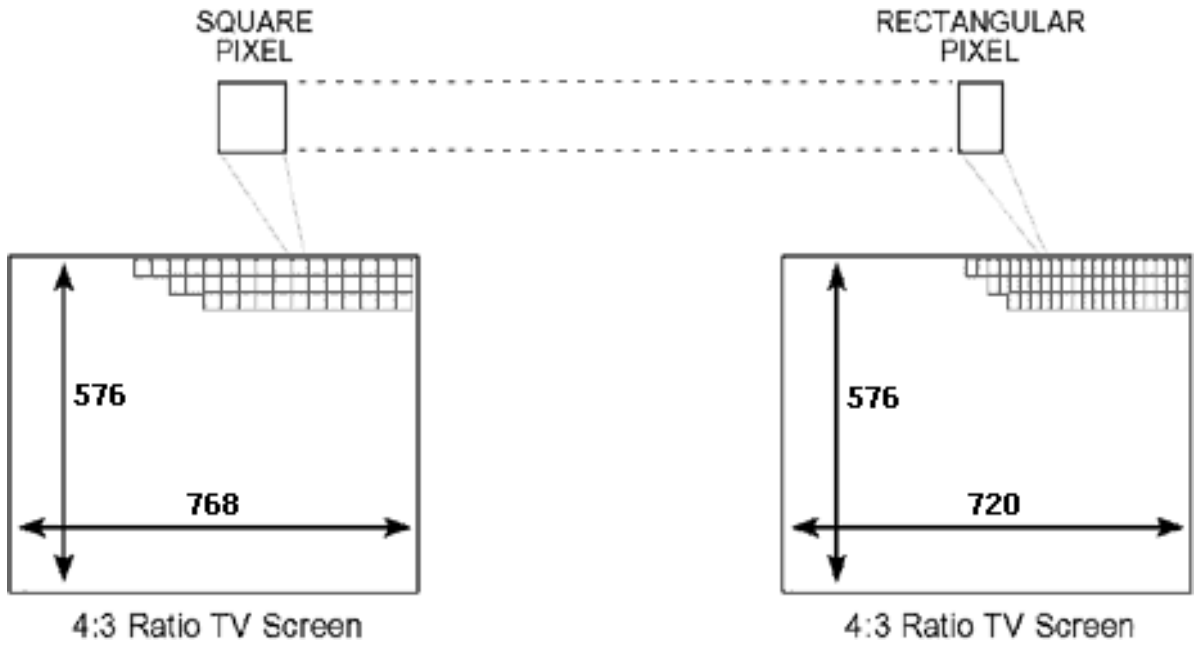


Le Pellicole Cinematografiche

Per i film sono usate pellicole da 35 mm con aspect ratio 4:3 ad altissima risoluzione, sulle quali sono impressi 24 fotogrammi al secondo. Per vedere su TV *NTSC* un film girato a 24 ***fps*** bisogna modificare il suo framerate in 30 fps. Questo processo si chiama *3:2 pulldown* e viene effettuato tramite una macchina chiamata Telecine che crea, con un particolare metodo, 4 fps aggiuntivi; per renderlo correttamente visibile su TV *PAL*, invece, il Telecine velocizza solo lievemente lo scorrimento video (*2:2 pulldown*) per raggiungere i 25 fps. Il procedimento inverso (NTSC/PAL à FILM) per creare video a 24 fps è detto Inverse Telecine. Se un filmato *NTSC* a 30 fps viene letto da un moderno dispositivo a 29.97 fps lo sfasamento induce l'Inverse Telecine a produrre un video a 23.976 fps. Nel prosieguo della nostra trattazione considereremo solo lo standard televisivo europeo PAL.

I Formati del DVD Video

Con l'avvento dei computer i mixer (miscelatori di immagini analogiche) sono stati sostituiti dai programmi di editing ed i supporti analogici, per lo più nastri, hanno lasciato il posto a CD e DVD, sui quali l'informazione video è 'scritta' in bit. Il video occupa una quantità immane di spazio in termini di *MegaByte* (1048576 bit) per cui si è pensato di creare metodi per ridurre la quantità di spazio occupata dai filmati. Il *Motion Picture Experts Group* (MPEG) ha creato sistemi di compressione video *lossy* (con perdita di dati rispetto all'originale) per inserire sempre più minuti di filmato, con qualità sempre maggiore, in spazi predefiniti come CD e DVD. Nacque così l'MPEG-1, che consentiva di mettere su un CD (650 MB) un'ora di filmato con qualità paragonabile ad una videocassetta *VHS*. Successivamente è nato MPEG-2, progettato per le trasmissioni televisive e adoperato per archiviare film su DVD (DVD Video), che offre qualità ed ingombri superiori rispetto al suo predecessore. Con i computer è risultato coerente abbandonare la macchinosa modalità interlacciata per visualizzare un filmato, ricorrendo al più pratico sistema progressivo: ogni frame è visualizzato interamente tutto in una volta; i filmati progressivi sono gestibili sia da MPEG-2 che dal suo successore MPEG-4. Il frame del DVD Video ha risoluzione (numero di pixel per unità) 720x576. Come per le pellicole cinematografiche il fotogramma può contenere video con diversi aspect ratio, prevalentemente 1,85:1 , 1,33:1 , ma soprattutto 2,35:1. Non bisogna confondere l'aspect ratio degli schermi (4:3 , 16:9) con quello dei filmati (1,85:1 , 2,35:1 …). Ogni DVD Video è ottimizzato per essere visualizzato su TV classica (4:3) o su schermo panoramico (16:9) e le modalità di disposizione dei pixel nel fotogramma 720x576 variano molto.L'aspect ratio 1,33:1 è ottimizzato ovviamente per TV 4:3 (4/3=1,33). I pixel stanno all'interno del frame esattamente come sono visti. Viene spontaneo domandarsi perché l'aspect ratio 1,33:1 viene pedissequamente e regolarmente visualizzato su TV 4:3 che ha risoluzione 720x576 (che corrisponde ad aspect ratio 5:4). La TV 4:3 ha pixel rettangolari con un rapporto larghezza/altezza pari a *0,9375:1*; teoricamente la risoluzione (con pixel quadrati) dovrebbe essere 768x576 ma in concreto i pixel orizzontali sono 786*0,9375 = 720 pixel. Ciò accade solo per le televisioni tradizionali poiché gli schermi 16:9 ed i monitor PC hanno pixel quadrati.



I formati 1,85:1 (Flat) e 2,35:1 (CinemaScope o anche Panavision) sono quelli più usati per i DVD Video (e in ambito cinematografico) e sono ottimizzati per gli schermi 16:9. Questi, con poche eccezioni, hanno una disposizione particolare dei pixel nel fotogramma, fatta con una tecnica detta anamorfosi. D'ora in poi, verosimilmente, consideriamo anamorfici tutti i film ottimizzati per 16:9. L'immagine orizzontale viene schiacciata per poter occupare tutti (o quasi) i pixel del fotogramma; ciò si traduce in una eccellente qualità in fase di lettura, quando l'immagine verrà nuovamente distesa e portata alle sue normali proporzioni.

Oltre al widescreen anamorfico esiste anche il widescreen letterbox, adoperato per visualizzare correttamente su TV 4:3 un filmato ottimizzato per schermi 16:9 : l'immagine originale anamorfica non è dilatata orizzontalmente, bensì compressa verticalmente e sono aggiunte in alto ed in basso delle bande nere (le cosiddette ***mattes***).

Un altro sistema, per mostrare film ottimizzati in 16:9 ai possessori di TV 4:3 è il **Pan & Scan ('Quadra e scorri')**, che prevede uno *zoom* (ingrandimento) dell'immagine atto a riempire verticalmente lo schermo e lo spostamento orizzontale lungo il frame per inquadrare la parte più interessante per lo spettatore.

Oltre ai formati citati ne esistono altri due poco utilizzati: 1,66:1 (European) e 2,40:1 (variante del CinemaScope). Inoltre gli aspect ratio sono indicativi, quasi mai rispettati con assoluta precisione (moltissimi film 1,85:1 sono in realtà 1,78:1).

La Compressione MPEG-4

L'MPEG-4 offre la stessa qualità di MPEG-2 ma dispone di una compressione tre volte superiore. Il processo di riduzione e compressione dei dati serve a ridurre le informazioni da memorizzare ed è per questo che alcune di esse devono necessariamente essere eliminate.

Informazioni da Eliminare:

I dati ripetuti nello stesso frame: in un fotogramma pixel vicini hanno caratteristiche di luminosità e colore simili. Il compressore sintetizzerà, quindi, queste informazioni eliminando la *ridondanza spaziale*. I dati ripetuti in fotogrammi adiacenti: in frame successivi c'è buona possibilità, tranne se non si tratta di scene differenti o particolarmente veloci, di trovare zone d'immagine con colori e luminosità simili (o uguali), per cui il compressore video accorperà questi dati, eliminando la *ridondanza temporale*. I dati di componenti del filmato non percepibili dall'occhio umano:

i valori di colore e luminosità dell'immagine (che viene elaborata scomponendola in piccoli blocchi quadrati di pixel, solitamente 16x16 o 8x8) vengono convertiti nei corrispondenti valori di frequenza video attraverso una funzione matematica, la DTC (quella inversa è detta iDTC). Poiché non sono percepibili generalmente le alte frequenze nelle scene animate (ad es. spostamento rapido di fumo, fronde, piccoli oggetti,..), queste possono essere eliminate attraverso la *quantizzazione*; una maggiore quantizzazione causerà più perdita di informazioni à peggiore qualità. Dopo queste elisioni, i dati rimasti vengono compressi con procedimenti *loseless* (senza perdita di dati).

Tipologie di Frame

I fotogrammi del filmato compresso possono essere di tipo **'I'** (Key Frame), **'P'** (Predicted Frame) e **'B'** (Frame 'Bidirectional'):

1) ***Il* key frame *è un fotogramma memorizzato integralmente e compresso, dal quale sonori creati i successivi* p-frame *o* b-frame*. I nuovi codec ne inseriscono uno ad ogni cambio di scena.***

2) ***Il* predicted frame *memorizza solamente le differenze rispetto al frame che lo precede, quindi userà poche informazioni.***

3) ***Il* b-frame *archivia le differenze rispetto al frame che lo precede e a quello che lo segue (quindi saranno utilizzati ancor meno dati rispetto al* p-frame*).***

Più key-frame in un video ne aumenteranno la qualità (e la dimensione), più *b-frame* ne ridurranno l'ingombro.

I Codec MPEG-4

Il primo codec MPEG-4 che si ricordi è stato *MS MPEG-4 Codec*, adoperato da Microsoft per la trasmissione di video attraverso la rete, non utilizzabile per l'encoding dall'utente. Questo fu modificato illegalmente e nacque il DivX 3.xx. Negli ultimi anni sono stati prodotti altri compressori MPEG-4, più o meno aderenti agli standard internazionali.

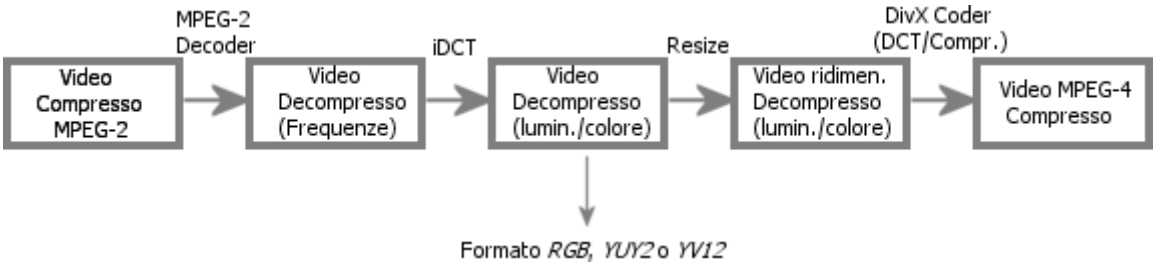
Adoperandoli tutti per periodi abbastanza lunghi, ragionevolmente si può affermare che globalmente in termini di *qualità* i codec MPEG-4 migliori sono **DivX** e **XviD**, che primeggiano anche per quantità di impostazioni. Entrambi hanno un ottimo **post-processing** (miglioramento dell'immagine in fase di decodifica) ma in termini di *velocità* DivX è più performante di XviD (che, peraltro, altera i colori del filmato originale). Ne discende che il codec MPEG-4 ideale è DivX.

La versione *4* (legalmente sviluppata) del codec DivX era gratuita; con l'avvento della versione *5* sono state create tre distribuzioni: la prima (*Free*) non permette l'uso di alcune funzioni (ad es. i *b-frame*), la seconda (*GAIN Bundle*) offre tutte le funzioni ma installa un innocuo programma-spia, la terza (*Pro*) è a pagamento. Le 'sperimentali' versioni *5.0.0* e *5.0.1* hanno condotto alla robusta e performante release *5.0.2*.

La versione *5.0.3* ha introdotto profili standard, qualche leggero miglioramento e tanti *bug* (errori); la versione *5.0.4* ha aggiunto altri bug ed un meccanismo che consente all'utente (esperto, paziente ed interessato) di manipolare accuratamente la qualità di singole parti del filmato; la release *5.0.5* corregge un importante errore della versione precedente. Per tali considerazioni è unanime l'opinione di adoperare DivX versione 5.0.2. La **DivXNetworks** corre verso la realizzazione del fantomatico DivX 6 che dovrebbe offrire, a parità d'ingombro, una qualità doppia rispetto alle release attuali.

Da DVD a DivX (Premessa)

Il Codec DivX si utilizza in tutti i casi in cui si vuole stipare filmati di buona qualità in poco spazio, per archiviarli sui diffusissimi CD o scambiarli attraverso Internet. Il processo di trasformazione di un DVD Video in DivX richiede la conoscenza delle nozioni tecniche fin qui spiegate. Il flusso video MPEG-2 viene convertito in DivX tramite delle operazioni contigue così sintetizzabili:



Come è evidente, la qualità del DivX dipenderà esclusivamente dal video originale, dalla qualità degli strumenti atti a trasformarlo (decodificatore MPEG-2, iDCT, filtro di ridimensionamento) e dai settaggi del codificatore. L'unico vincolo che ci poniamo riguarda il metodo di rappresentazione del colore: utilizzeremo la modalità **YV12**, che accelera la codifica e non compromette la qualità. L'occhio umano non percepisce le variazioni di colore così come avverte quelle di luminosità per cui, invece di adoperare 24 bit per ogni pixel (formato *RGB*), si può 'approssimare' l'informazione del colore utilizzando solo 12 bit per pixel (formato *YV12*) senza intaccare la qualità video. Considereremo quindi, tra i programmi più raffinati e funzionanti, quelli che adoperano tale modalità. Tra questi purtroppo scartiamo a priori *Vidomi 0.4* che, in modalità YV12, non consente di impostare il filtro di ridimensionamento (il *resize* e l'impostazione dei settaggi del Codec DivX sono i passaggi più importanti).

Il Decoding

I tool che decodificano il flusso video ***MPEG-2*** (previa separazione da quello audio, demux) sono, come le restanti funzioni che analizzeremo (***iDCT, resize***), per lo più compresi in programmi più complessi che si occupano di più (a volte tutte) le fasi del processo di codifica.

I ***Decoder MPEG-2*** con più alta qualità sono:

1) Quello usato da ***XMPEG 4.5*** (ottimizzazione del mitico ***FlaskMPEG 0.6***)

2) Quello usato da ***MPEG2AVI 0.1.6*** by Iker Rodriguez (incluso in tanti all-in-one)

3) MPEG2Dec3 (per Avisynth)

Ragionevolmente nel prosieguo della nostra analisi consideriamo solo ***XMPEG*** (con performance considerate senza la visualizzazione del filmato in codifica) e ***MPEG2AVI.***

La iDCT

Le ***iDCT*** usate dai due programmi sono:

E' semplice capire che con ***XMPEG*** va usato ***Optimized MMX iDCT***; come seconda scelta abbiamo due ***iDCT*** lievemente più lenti: ***MMX*** e *Miha's Fast*.

Con ***MPEG2AVI*** vanno usati ***32-bit MMX*** o ***Miha's Fast*** come ***iDCT.***

Il Resize

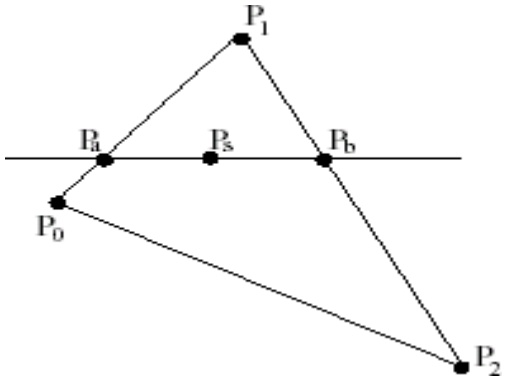
Il **resize** è quella operazione che permette di rimpicciolire (ridurre la risoluzione) una immagine o ingrandirla (aumentare la risoluzione); per ricrearla si riempie un'immagine vuota di dimensioni opportune con dei pixel ricavati per ***interpolazione*** (calcolo di un punto conoscendo le caratteristiche di quelli adiacenti) dall'immagine iniziale. Tale ***ricampionamento*** si avvale di funzioni matematiche; nel nostro caso (escludendo a priori ***nearest neighbour*** per la pessima precisione offerta) di **interpolazione bilineare** e **bicubica**.

Con un numero di punti noti noi possiamo costruire una curva interpolante passante per questi (interpolazione per punti) o una curva che si scosti poco da questi, in modo da non perdere le informazioni in essi contenute (interpolazione tra punti, *smoothing*). L'interpolazione per punti usa *funzioni lineari* e funziona bene quando conosciamo o consideriamo un numero basso di punti (fino a 9); col crescere del numero dei punti risulta molto più efficiente usare le elaborate *funzioni polinomiali a tratti*, tra cui spiccano le importanti *cubiche*. Quando i punti sono in numero elevato non è ragionevole forzare la funzione a passare esattamente per questi, ma è molto più conveniente effettuare uno *smoothing*, cioè una 'levigatura', al fine di minimizzare l'errore contenuto nei dati. E' intuitivo che gli algoritmi (insiemi di regole o direttive atte a fornire una risposta specifica ad uno o più dati in input) cubici compiono più calcoli dei lineari, quindi sono più lenti. Quando l'immagine deve essere non gradita, poiché devono essere inseriti punti completamente nuovi, l'interpolazione bicubica (che usa 16 punti) è insostituibile. Viceversa, se l'immagine deve essere ridotta moderatamente è sufficiente la perequazione dei 4 pixel usati dal resize bilineare, mentre la mediazione del resize bicubico risulta impropria, eccessiva ed altera le caratteristiche del punto interpolato. Il resize bicubico torna ad essere più qualitativo quando i rimpicciolimenti hanno *fattore di scala* inferiore al 50% poiché s'adatta meglio quando il fine è stipare più dettagli possibili in poco spazio visuale.

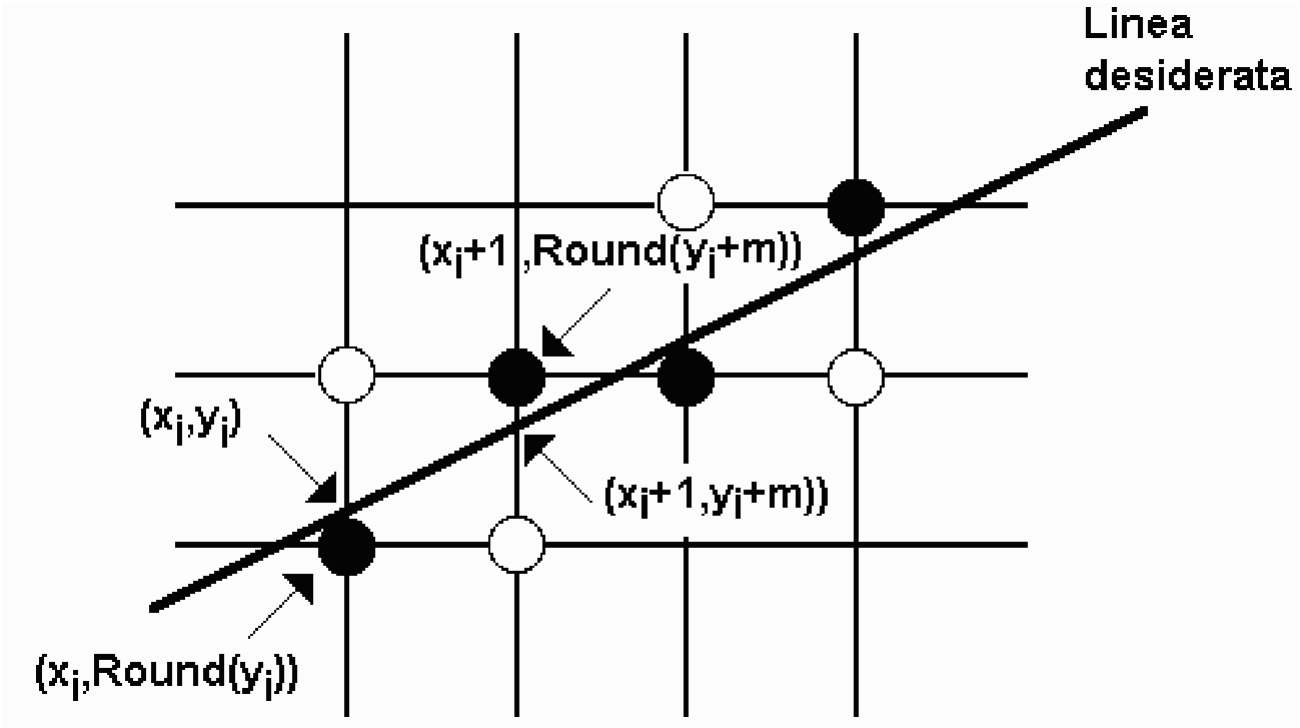
Cito - dalla guida di ***Paint Shop Pro* (*Jasc*)** - "Ricampionamento bicubico … Utilizzare questo metodo quando si aumentano le dimensioni di un'immagine.". Inoltre: "Ricampionamento bilineare… Utilizzare questo metodo quando si riducono immagini …". La teoria è suffragata dall'esperienza. Come si può notare qui in basso, l'immagine rimpicciolita moderatamente con il ricampionamento bilineare ha qualità superiore rispetto alla stessa riduzione fatta con il ricampionamento bicubico; lo stesso si verifica usando ***Adobe PhotoShop***, con differenze però inferiori. Viceversa, la riduzione forte di un frame esalta la qualità del resize bicubico.

Il metodo bilineare :

…dato un triangolo con vertici *P*0, *P*1, *P*2, calcoliamo il valore *Pa* per interpolazione lineare tra i valori *P*0 e *P*1, ed il valore *Pb* per interpolazione lineare tra i valori *P*1 e *P*2. Calcoliamo quindi il valore *Ps* per interpolazione lineare tra i valori *Pa* e *Pb*.



Per la riduzione moderata dell'immagine è più efficiente un particolare *metodo incrementale* semplificato (che usa solo aritmetica intera): ***l'algoritmo di Bresenham (o Bressenham)***. Partendo dal punto iniziale di una linea (uno dei due estremi) si incrementa ad ogni passo lo spostamento lungo la stessa fino al raggiungimento del punto finale. Dopo avere disegnato un pixel, l'algoritmo sceglie tra i suoi 8 vicini quale 'accendere' in base all'equazione della retta. La semplicità computazionale e l'uso della sola aritmetica intera garantiscono ottime prestazioni in termini di velocità ed è stata ampiamente dimostrata la sua precisione, soprattutto per la definizione di segmenti ed anche circonferenze. E' qui schematizzato il funzionamento di un generico algoritmo incrementale



Il ***resize Bressenham*** offre miglior qualità del bilineare e, come questo, è superato in qualità dal bicubico solo per fattori di scala inferiori al 50% (e ovviamente per gli ingrandimenti). Ciò è confermato anche da PX3: "Pixel-averaging is fast, and smooth for scaling factors between 0.5 and 1.0".

Per un calcolo (grossolano) del fattore di scala :

(Altezza Immagine Nuova per Larghezza Immagine Nuova) ***diviso*** ***(Altezza Immagine Origine per Larghezza Immagine Origine)***

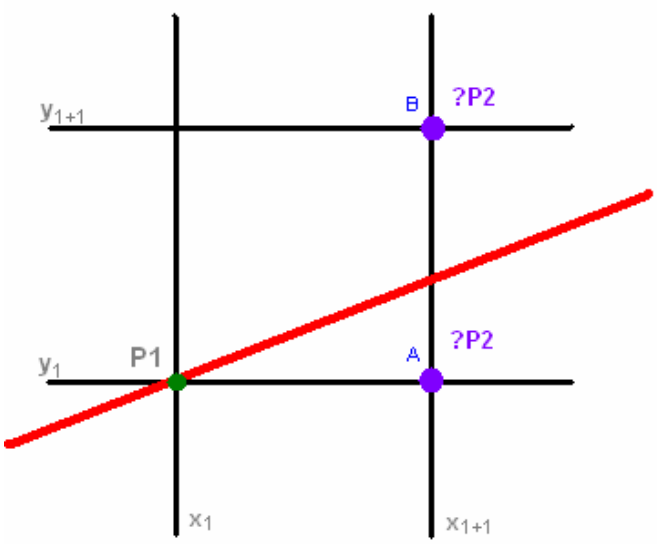
Xmpeg adopera veloci ***resize bilineari e lenti resize bicubici smoothed (levigati)***. Sono abbozzate le didascalie riguardanti la qualità di ogni singolo resize poiché, come visto, la 'very good quality' degli algoritmi bicubici esiste solo per fattori di scala inferiori al 50%, la 'very good quality' del filtro Bressenham è reale solo per riduzioni fino al 50%.

Ovviamente i resize che si prestano al nostro uso sono Bressenham per piccole/medie riduzioni, SSE Bicubic per forti rimpicciolimenti (fattore di scala < 0.5).

MPEG2AVI sfortunatamente adopera solo il resize Bressenham (con velocità e qualità quasi uguali a quello utilizzato da ***Xmpeg***), risultando inadatto per riduzioni importanti; per questo motivo, adesso, si può affermare che ***Xmpeg*** risulta essere il miglior programma per convertire ***MPEG-2*** in DivX. Elabora anche l'audio in un'unica procedura.

Resta inteso che ***MPEG2AVI*** rimane uno dei migliori programmi ***MPEG-1/2*** e ***AVI*** mai fatti. E' ottimamente implementato nel potente quanto facile all-in-one ***SimpleDivX 1.1b***.

Analizziamo ora in dettaglio l'Algoritmo della linea di Bresenham



L'**algoritmo della linea di Bresenham** (da alcuni chiamato algoritmo del punto medio) è un algoritmo di rasterizzazione di linea. Allo stato attuale è l'algoritmo più usato per la rasterizzazione di linee, soprattutto per la sua bassa richiesta di risorse computazionali.

Per capire l'algoritmo, semplifichiamo il problema assumendo m ($m = \frac{\Delta y}{\Delta x}$) sia compreso tra 0 ed 1 :

0 < m < 1.

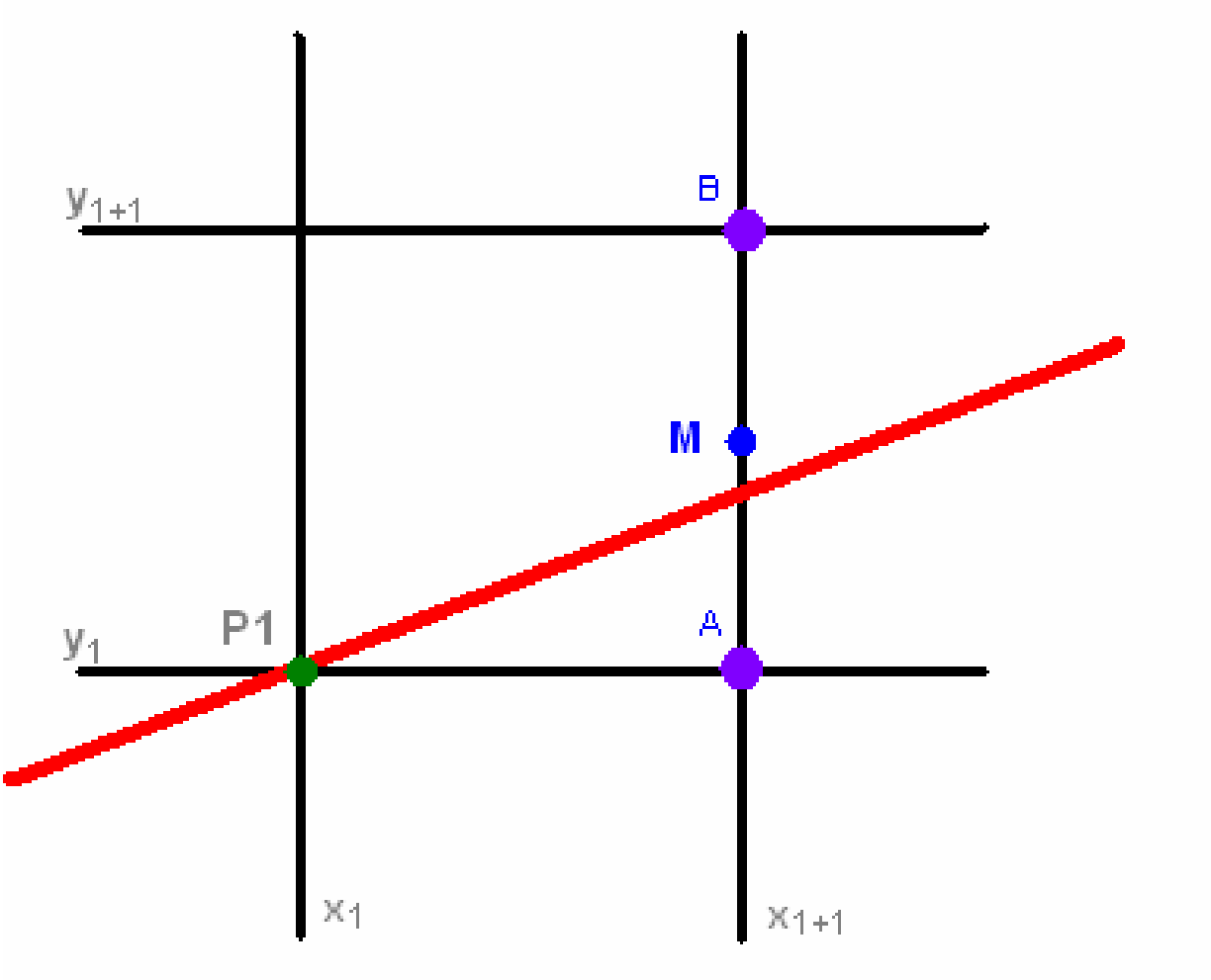
Immaginiamo di trovarci ad un passo i dell'algoritmo, ovvero abbiamo appena determinato quale pixel "accendere", per esempio $p_1(x_1,y_1)$. Ora dobbiamo determinare il prossimo pixel da "accendere", chiamiamolo ***$p_2(x_2,y_2)$***, dove ***$x_2 = x_1 + 1$.*** La situazione è quella riportata in figura 1, dove dal punto 1 (quello in verde) dobbiamo passare al punto 2 che si puo

trovare subito a destra, caso A, o in alto a destra, caso B.

Nel caso *A* abbiamo ***$y_2 = y_1$;***

Nel caso *B* abbiamo ***$y_2 = y_1 + 1$;***

Prendiamo in considerazione il punto ***$M(x_1 + 1, y_1 + 1 / 2)$***, punto medio tra A e B. Se la linea da rasterizzare passa sopra, illumineremo il pixel superiore B, altrimenti il pixel inferiore A.



Per determinare se M si trova sotto o sopra la retta, consideriamo la forma esplicita dell'equazione della retta :

$$y = \frac{\Delta y}{\Delta x} * x + q \qquad -\Delta x * y + \Delta y * x + \Delta x * q = 0$$

Tutti i punti appartenti alla retta devono verificare l'equazione. Ma questa retta divide anche due semipiani, quello composto da tutti i punti per cui la formula precedente restituisce un valore positivo e quello per cui restituisce un valore negativo.

Quindi dalla formula precedente possiamo ricavare il valore decisionale *d*, sostituendo ad x ed y le coordinate di

$$M(x_1 + 1, y_1 + 1/2) \qquad d = -\Delta x * y_M + \Delta y * x_M + \Delta x * q$$

il quale sarà:

- d = 0 Se il punto giace sulla retta; in questo caso possiamo scegliere in modo indifferente il punto A o il punto B.

- d > 0 Se il punto si trova sopra la retta; in questo caso prendiamo il punto A.
- d < 0 Se il punto si trova sotto la retta; in questo caso prendiamo il punto B.

Nell'algoritmo avremmo necessita ogni volta di sapere se *d* è positivo o negativo.

Ipotizziamo di aver scelto il punto A, in questo caso il nostro punto di partenza *p* è $p(x_1 + 1, y_1)$, e il nostro nuovo punto medio M è $M(x_1 + 2, y_1 + 1 / 2)$. Invece il nuovo valore di *d* è:

$$d_{new} = -\Delta x * y_M + \Delta y * x_M + \Delta x * q = -\Delta x * (y_1 + 1/2) + \Delta y * (x_1 + 2) + \Delta x * q$$

Proviamo a sottrarre al nuovo valore di *d* quello vecchio:

$$d_{new} - d_{old} = -\Delta x * (y_1 + 1/2) + \Delta y * (x_1 + 2) + \Delta x * q - (-\Delta x * (y_1 + 1/2) + \Delta y * (x_1 + 1) + \Delta x * q)$$

Semplificando otteniamo:

$$d_{new} - d_{old} = \Delta y$$

Quindi abbiamo modo di ricavare il nuovo valore *d* in modo più semplice dal vecchio, senza ogni volta rifare tutti i calcoli. Ora dobbiamo fare l'ipotesi per il caso in cui si sia scelto il punto B. Abbiamo i nostri nuovi punti:

$$p(x_1 + 1, y_1 + 1)$$

$$M(x_1 + 2, y_1 + 1 + 1/2) = M(x_1 + 2, y_1 + 3/2)$$

$$M(x_1 + 1, y_1 + 1/2)$$

$$d_{new} - d_{old} = -\Delta x * (y_1 + 3/2) + \Delta y * (x_1 + 2) + \Delta x * q - (-\Delta x * (y_1 + 1/2) + \Delta y * (x_1 + 1) + \Delta x * q)$$

$$d_{new} - d_{old} = -\Delta x + \Delta y$$

$$d_{i+1} = \begin{cases} d_i - \Delta x + \Delta y, & \text{se } d_i > 0 \\ d_i + \Delta y, & \text{se } d_i < 0 \end{cases}$$

Non ci rimane che conoscere il valore d_0; ricordandoci la formula per calcolare *d* e prendendo come punto *p*, $p_0(x_0, y_0)$ ovvero un estremo della retta, abbiamo:

$$d = -\Delta x * (y_0 + 1/2) + \Delta y * (x_0 + 1) + \Delta x * q = -\Delta x * y_0 + \Delta y * x_0 + \Delta x * q + (-\Delta x * 1/2 + \Delta y)$$

Nel passagio abbiamo portato fuori i valori 1/2 e +1. La prima parte della formula corrisponde all'equazione della retta applicata ad un punto della retta, quindi sappiamo che sarà uguale a zero.

$$d_0 = -\Delta x * 1/2 + \Delta y$$

Da tutte queste formule possiamo finalmente ricavare ***l'algoritmo***: Dati due punti p1 e p2, con coordinate (x_1,y_1) e (x_2,y_2):

```
DX = x2 - x1
DY = y2 - y1

//il nostro valore d0
d = - 1/2 * DX + DY

//assegna le coordinate iniziali
x = x1
y = y1
disegna_il_punto(x, y)

while x < x2 {
       if (d >= 0) {
        d = d -DX + DY;
        y = y + 1;
        x = x + 1;
       }
       else {
        d = d + DY;
        x = x + 1;
       }
       disegna_il_punto(x, y)
```

Notiamo che l'algoritmo presenta dei numeri in virgola mobile, i quali richiedono risorse computazionali, un idea per evitare questa precisione è quella di raddopiare i valori di d:

```
DX = x2 - x1
DY = y2 - y1

//il nostro valore d0
d = - DX + 2 * DY

//assegna le coordinate iniziali
x = x1
y = y1
disegna_il_punto(x, y)

while x < x2 {
       if (d >= 0) {
        d = d -2 * DX + 2 * DY;
        y = y + 1;
        x = x + 1;
       }
       else {
        d = d + 2 * DY;
        x = x + 1;
       }
       disegna_il_punto(x, y)
```

Abbiamo quindi ottenuto un algoritmo che lavora con numeri interi e semplice da implementare.