Programmiamo videogiochi (4)

di Marco Pesce

In questa puntata vi presento una routine per ottenere il tanto famoso quanto inutilizzato (soprattutto qualche tempo fa) scrolling hardware!
Passo subito ad elencarvi le sue caratteristiche tecniche: aggiornamento dello schermo in perfetto sincronismo con il raster (50 fotogrammi al secondo), assenza assoluta di sbalzi e sfarfallamenti vari, 320 per 200 pixel di area visibile in 16 colori (ampliabile senza problemi a tutto lo schermo senza limitazioni nella palette colori!) e ampiezza di 30 schermi (anch'essa ampliabile) dell'area totale

Il principio sul quale si basa questa routine è quello di utilizzare al minimo indispensabile il blitter che, seppur veloce, non permette di spostare una pagina grafica con velocità sufficiente ad aggiornare lo schermo in un 50esimo di secondo. Il trucco è quello di sfruttare il pitch fine di posizionamento dei bitplane unitamente alla possibilità di questi di avere la word di partenza ridefinibile a piacimento. Lo scrolling è solo nella direzione X, da destra verso sinistra (come richiedono la maggior parte dei videogame arcade) ma questa è una limitazione voluta per non complicare troppo la spiegazione; per ottenere scrolling anche da sinistra verso destra basta invertire alcune operazioni (vedi fine articolo), mentre per ottenere uno scrolling verticale occorre ampliare il discorso, cosa che eventualmente faremo in futuro.

Cominciamo con una introduzione teorica sul funzionamento di uno scrolling. Per realizzare tale effetto sull'Amiga c'è un sistema menzionato tra l'altro anche sull'Hardware Manual; ad esempio possiamo realizzare una pagina grafica di 1024x1024 pixel e visualizzarne solo un'area di 320x200 per poi «scrollare» liberamente in qualunque direzione, ma tale soluzione richiede un enorme spreco di memoria e ci limita ad uno scrolling relativamente ristretto. La soluzione che hanno adottato i programmatori in un primo momento era quella di non sfruttare tale metodo, ma di copiare il grossolano sistema sfruttato anche sull'Atari ST di scrolling software, realizzato oltretutto con il solo ausilio del 68000; per effettuare uno scrolling anche di un solo pixel occorreva ridisegnare tutto lo schermo spostato di un pixel e aggiungere una striscia di nuovi pixel nell'area che si «liberava». Con l'Amiga si è poi realizzato lo stesso procedimento, ma con l'aiuto del blitter e le cose andavano decisamente meglio. Si arriva così ai giorni nostri, e cominciano a farsi vivi i primi videogame dallo scrolling terrificantemente fluido (vedi ad esempio «Shadow of the Beast»). Il nostro listato ne è un esempio pratico. Veniamo al suo funzionamento. Viene aperto uno schermo di 352x230 pixel. In foto 1 c'è uno schema di come viene suddiviso tale schermo; 320x200 pixel sono adibiti all'area di gioco (visibile) vera e propria; una prima colonna verticale di 16 pixel di ampiezza orizzontale (e 200 di profondità verticale) viene nascosta alla destra dello schermo; tale colonna è già «disegnata» e mentre si effettua lo scrolling con l'ausilio del pitch (che permette uno scrolling hardware effettivo di max 16 pixel) si disegna nell'ulteriore colonna di 16 pixel di ampiezza orizzontale, in modo da completare il disegno propri al termine dei 16 pixel di scrolling effettuati tramite pitch. Terminato lo scrolling con il pitch per poter proseguire basta riazzerare il valore in esso contenuto (nella fattispecie occorre impostarlo a 15) e spostare i puntatori ai bitplane di 1 word (2 byte). A questo punto ci troviamo nella stessa situazione di partenza, con una pagina visualizzata di 320x200 pixel, una colonna di 16 pixel sulla destra già disegnata e una colonna con materiale «vecchio». Si ricomincia quindi il ciclo di scrolling tramite pitch e relativo aggiornamento della colonna più a destra. Tanto per la cronaca, tale colonna è disallineata con il resto della schermata e se per diletto facciamo in modo che non venga «aggiornata» ed eseguiamo lo scrolling come se niente fosse vedremo il vecchio materiale traslato di un pixel verso l'alto, tutto questo perché la word più in alto è stata eliminata dalla visuale in seguito all'incremento dei puntatori ai bitplane. Questo continuo incremento fa sì che al termine di 352 pixel ci troviamo «avanzati» di una striscia verticale, quindi in fondo allo schermo viene visualizzata una nuova riga. Dal momento che tali righe nascoste sono 30 si possono avere un massimo di 30 schermi di scrolling. In ogni modo basta aumentare tali righe e lo scrolling risulterà ampliato.

Lo sfondo è composto da una combinazione opportuna di alcuni riquadri base di 16x24 pixel contenuti in una seconda schermata di 320x200 pixel (foto 2); uno schermo da 320x200 pixel è composto da 160 di queste «mattonelle» per realizzare 30 schermi occorrono quindi 4800 mattonelle. La mappa della

disposizione di tali mattonelle e la schermata con le mattonelle stesse sono contenute in file esterni al programma. Il file della schermata è nello stesso formato usato per il programma di gestione del blitter, al quale questo nuovo listato va aggiunto. Infatti tale programma contiene, tra l'altro, anche le subroutine che si occuperanno dell'apertura e del caricamento degli screen. Il file della mappa dovrà contenere una successione di mattonelle, tenendo conto del fatto che tale successione verrà stampata sullo schermo partendo dalla mattonella in alto a sinistra e proseguendo con quella verticalmente successiva fino all'ottava mattonella, per poi ricominciare dall'alto, ma una posizione più a destra, e così via. Le mattonelle dello schermo che le contiene fisicamente sono numerate da 0 a 127 (seguendo la stessa successione adottata per la stampa, secondo la quale la 128esima mattonella sarebbe quella che nella foto 2 rappresenta una «pergamena»). La mappa adottata nel listato comprende solo 3200 mattonelle (quindi 20 schermi). In foto 3 vediamo un esempio di «composizione», mentre in foto 4 c'è lo scrolling in piena attività!

Come abbiamo accennato, per il funzionamento del listato è indispensabile che esso venga «aggiunto» a quello del numero di settembre scorso; se proprio non volete copiarlo tutto basterà eliminare la routine di gestione del blitter che non viene utilizzata (da «blity» alla fine del programma) e relative variabili (da «funzione» a «shift»); tutto il resto è da digitare (variabili comprese); unica accortezza è quella di sostituire la struttura dati «dati2» con quella presentata nel listato di questo numero. La parte di routine va scritta tra le vecchie routine e le vecchie variabili; le variabili nuove possono essere scritte indifferentemente prima o dopo le vecchie variabili.

E passiamo alle routine vere e proprie. La prima parte del listato si occupa dell'apertura di 2 schermi grafici, il primo di 320x200 e il secondo 352x230 (screen 6 e 5). Dalla label MAIN2 in poi troviamo la parte che effettua il caricamento dello screen 6, chiamato qui 01pic, e della tabella di 3200 mattonelle, chiamata 0data. Da notare che ogni mattonella occupa un byte ma lascia libero il bit più significativo di questo, che è bene usare per indicare se la mattonella in questione è «solida» o meno (per le collisioni); in sostanza basta porre a 1 tale bit se la mattonella è da considerarsi solida e viceversa (ciò non intaccherà il «valore» della mattonella, in quanto la routine di stampa provvede ad eliminare tale bit in fase di decodifica) per realizzare la tabella senza rischiare di finire al manicomio è bene costruirsi un editor apposito, oppure utilizzare quello contenuto nel pacchetto «Editor di videogame» disponibile su dischetto e presentato tempo fa

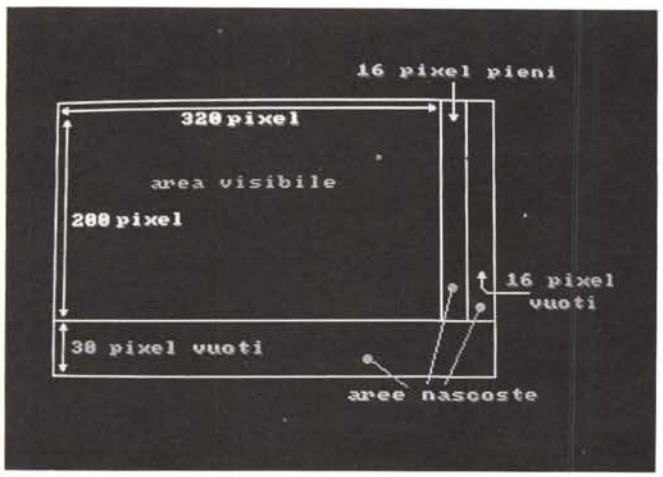

Foto 1

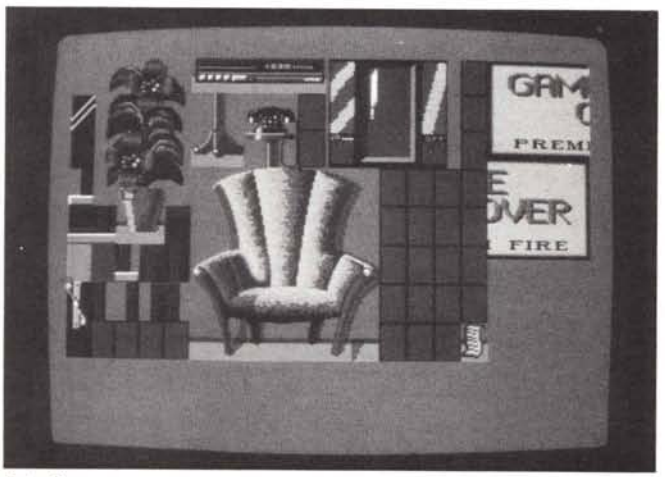

Foto 2

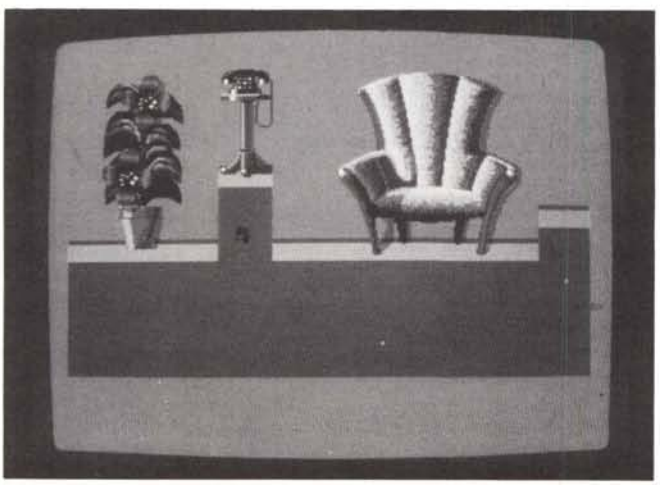

Foto 3

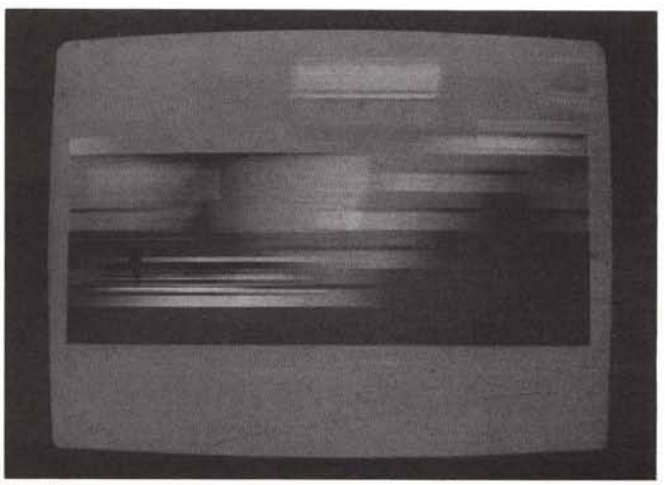

Foto 4

nella rubrica «Megagame 64». Dalla label MAIN23 in poi troviamo la routine perl'allocazione della Ram per le copperlist; si effettua tale operazione in quanto tale ram deve essere obbligatoriamente di tipo CHIP altrimenti il programma non gira su un Amiga espanso. Vengono preparate 2 copperlist, ma solo la «coplist1» è quella che ci interessa. Da MAIN3 in poi c'è la preparazione effettiva della copperlist con inserimento dei puntatori ai bitplane, quindi essa viene attivata e vengono impostate le dimensioni dell'area di schermo visibile (320x200), del «modulo» e del data FETCH, che riserva una word per effettuare lo scrolling con il pitch. Segue l'attivazione dei colori caricati in precedenza con la routine OPENFILE e memorizzati a partire dall'ottavo byte del buffer «inbuf». La parte di listato che segue la label MAIN34 si occupa della disattivazione degli sprite e dell'attivazione del DMA gnerale oltre che di quello dei bitplane. Da MAIN4 in poi c'è la routine di scrolling vero e proprio; la prima operazione è quella di azzeramento del pitch, che viene quindi impostato a 15. Da notare che il registro adibito a tale controllo ($DFF102) permette di assegnare due pitch differenti, uno per i bitplane dispari e uno per i bitplane pari; questo per avere la possibilità di scrolling differenziato nel caso si usasse la modalità «dual playfield»; nel nostro caso i due pitch devono essere impostati con lo stesso valore; i primi 4 bit sono per il primo pitch e i successivi 4 bit per il secondo. Per ottenere quest'ultimo risultato basta «shiftare» di 4 posizioni il pitch e poi aggiungere al risultato il pitch stesso (oppure moltiplicare per 5).

Dopo questa prima operazione preliminare ne segue un'altra dello stesso tipo; vengono stampate 168 mattonelle partendo dall'angolo in alto a sinistra, in modo da riempire l'area visibile e da completare anche la prima colonna di 16 pixel nascosta. Il tutto si ottiene «chiamando» per 168 volte la subroutine «STMATT». Terminato il loop si passa al blocco che verrà eseguito in un ciclo infinito (scrolling continuo). Si comincia con l'attesa di una specifica posizione del raster (la verticale $FF) quindi si controlla che il pitch non sia azzerato. Se si verifica quest'ultima ipotesi vuol dire che occorre riposizionare i puntatori ai bitplane e reimpostare a 15 il pitch. La routine che segue effettua proprio tale operazione, incrementando di due byte i 4 puntatori e ricalcolando la copperlist. Se ancora non è avvenuto l'azzeramento si salta a «ERT». Qui viene decrementato il pitch e si controlla se è il turno di stampare una mattonella nella colonna più a destra (nascosta); infatti essendo 8 le mattonelle di tale colonna ed essendo invece 16 i pixel dello scrolling occorre stamparle una volta sì e una no, in modo da restare in sincronismo sulla colonna non visibile che cambia appunto di 16 in 16 pixel. A questo punto si trasferisce il pitch in $DFF102 (secondo le stesse accortezze avute

```
*assembler

main: lea dati(pc),a0
      jsr openscreen
      move.l puntscreen,schermo6
      move.l rastport,rastport6
      move.l viewport,viewport6
      move.l bitp1,b6bitp1
      move.l bitp2,b6bitp2
      move.l bitp3,b6bitp3
      move.l bitp4,b6bitp4


      lea dati2(pc),a0
      jsr openscreen
      move.l puntscreen,schermo5
      move.l rastport,rastport5
      move.l viewport,viewport5
      move.l bitp1,b5bitp1
      move.l bitp2,b5bitp2
      move.l bitp3,b5bitp3
      move.l bitp4,b5bitp4
      jmp main2


main2
      move.l viewport6,viewport
      move.l b6bitp1,bitp1
      move.l b6bitp2,bitp2
      move.l b6bitp3,bitp3
      move.l b6bitp4,bitp4
      move.w #'01',nomepic
      jsr openfile

      move.l doslib,a6 ;caricamento tabmattonelle
      move.l #nometabella,d1
      move.l #1005,d2
      jsr -30(a6)       ;open oldfile
      move.l d0,handle
      move.l handle,d1
      move.l #tabmattonelle,d2
      move.l #3200,d3
      jsr -42(a6)
      move.l doslib,a6
      move.l handle,d1
      jsr -36(a6) ;close file
      jmp main23
 ;allochiamo la ram per le copperlist
      move.l 4,a6
      move.l #36,d0
      move.l #$10003,d1
      jsr -198(a6)
      move.l d0,coplist1
      move.l 4,a6
      move.l #36,d0
      move.l #$10003,d1
      jsr -198(a6)
      move.l d0,coplist2
      move.l #9,d0
      move.l coplist1,a0
      move.l #dcoplist1,a1
riptrip move.l (a1)+,(a0)+
      sub.b #1,d0
      bne riptrip
      move.l #9,d0
      move.l coplist2,a0
      move.l #dcoplist2,a1
rriptrip move.l (a1)+,(a0)+
      sub.b #1,d0
      bne rriptrip
      jmp main3


main3
;prepara la copper list

      move.l #b5bitp1,a1
      move.l coplist1,a2
      add.l #02,a2
loop131 move.w (a1)+,(a2)+
      add.l #02,a2
      cmp.l #b6bitp1,a1
      bne loop131
      move.l coplist2,a2
      add.l #02,a2
loop132 move.w (a1)+,(a2)+
      add.l #02,a2
      cmp.l #b7bitp1,a1
      bne loop132

loop136 move.w $dff006,d0
      and.w #$ff00,d0
      cmp.w #$a000,d0
      bne loop136
      move.l coplist1,$dff080
      move.w $dff088,d0
      move.w #$2c81,$dff08e
      move.w #$f4c1,$dff090
      move.w #$0030,$dff092; DATA FETCH orizzontale
      move.w #1,$dff108;modulo
      move.w #1,$dff10a;modulo
;attivazione colori
      move.l #inbuf+8,a0
      move.l #$dff180,a1
loop135 move (a0)+,(a1)+
      cmp.l #$dff1a0,a1
      bne loop135
      jmp main34

main34

      move.w #$0020,$dff096
      move.w #$8300,$dff096
      jmp main4
```

all'inizio della routine), quindi, eventualmente, si stampa la mattonella di turno. Si preleva il puntatore PUNTMATT che contiene il numero d'ordine della mattonella da stampare (da 0 a 3199 nel nostro caso). Tale numero ci indicherà, con il calcolo seguente, le coordinate esatte per la stampa sullo schermo. I primi 3 bit indicano la posizione verticale (in quanto il loro valore oscilla sempre tra 0 e 7) che va di 24 in 24 pixel, quindi il loro valore viene moltiplicato per 24. I restanti bit indicano la posizione X già moltiplicata per 8; dal momento che la posizione X va di 16 in 16 basta moltiplicare per 2 e anche tale coordinata è pronta. Da notare che quest'ultima dopo la 168esima mattonella va oltre il limite di 352 pixel di ampiezza dello schermo; ciò significa che la mattonella alla posizione (fittizia) 336+16 verrà stampata alla posizione x=0 ma con una y in più e questo è proprio quello che vogliamo ottenere. Per stabilie le coordinate di «prelievo» dalla screen che contiene l'insieme di tutte le mattonelle si effettua un calcolo simile; tramite il puntatore si preleva dalla tabella «TABMATTONELLE» il valore della mattonella da stampare (che oscilla tra 0 e 127) quindi il calcolo è in tutto e per tutto simile a quello precedente. Ricavate anche queste due coordinate si può procedere con la stampa. Per tale scopo si usufruisce di una routine della «graphics library», la «CLIPBLIT», routine che permette di trasferire rettangoli di schermo da una rastport ad un'altra (o anche alla stessa). Basta definire le due rastport e le coordinate di prelievo e di stampa, più l'ampiezza x e y della sezione; questa parte di listato si commenta da sola. Così come è scritto il programma effettua, come menzionato, uno scrolling in ciclo continuo, andando anche oltre le 4800 mattonelle prefissate; se vogliamo avere uno scrolling a comando basta sostituire la JMP che ritorna all'attesa della posizione raster con una RTS e «chiamare» la subroutine quando ci serve uno scrolling di un pixel. Dal momento che non viene fatto alcun controllo sull'eventuale termine dei dati della tabella (fine dei 30 schermi) occorre realizzare anche una piccola routine adibita a tale scopo; basta controllare il PUNTMATT e verificare che non sia andato oltre il limite imposto.

Il listato finisce qui. Affrontiamo ora un breve discorso sullo scrolling nella direzione opposta. Chiunque avesse bisogno anche di tale caratteristica deve effettuare delle modifiche abbastanza semplici, infatti è sufficiente invertire i seguenti processi:

— incremento di PUNTMATT (decremento)

— decremento di picht (incremento)

in più basta avere l'accortezza di riposizionare 176 mattonelle indietro il PUNTMATT quando si inverte la direzione di scrolling. Ovviamente se la direzione viene nuovamente invertita il puntatore deve essere incrementato di nuovo. E con questo abbiamo concluso. Buon lavoro. MC

```
main4
        move.w #15,pitch
        clr.l d0
        move.w pitch,d0
        muls #16,d0
        add.w pitch,d0
        move.w d0,$dff102

        move.w #168,d0
loopf   move.w d0,tampd0
        jsr stmatt
        add.w #1,puntmatt
        move.w tampd0,d0
        sub.w #1,d0
        cmp.w #0,d0
        bne loopf

;scrolling
;attesa del raster

loopw   move.w $dff006,d0
        and.w #$ff00,d0
        cmp.w #$ff00,d0
        bne loopw
        move.w pitch,d0
        bne ert
        move.w #15,pitch
        add.l #2,b5bitp1
        add.l #2,b5bitp2
        add.l #2,b5bitp3
        add.l #2,b5bitp4
        move.l #b5bitp1,a1
        move.l coplist1,a2
        add.l #02,a2
loopx1  move.w (a1)+,(a2)+
        add.l #02,a2
        cmp.l #b6bitp1,a1
        bne loopx1
        jmp mettp
ert     sub.w #1,pitch
mettp
        move.w pitch,d0
        and.w #1,d0
        beq jhk
        jsr stmatt
        add.w #1,puntmatt
jhk     clr.l d0
        move.w pitch,d0
        muls #16,d0
        add.w pitch,d0
        move.w d0,$dff102
        jmp loopw

;mattonelle

stmatt
        move.w puntmatt,d0
        and.w #7,d0
        muls #24,d0
        move.w d0,matposy
        move.w puntmatt,d0
        and.w #$fff8,d0
        muls #2,d0
        move.w d0,matposx

        move.w puntmatt,d0
        move.l #tabmattonelle,a0
        add.w d0,a0
        clr.l d0
        move.b (a0),d0
        and.b #127,d0
        and.w #7,d0
        muls #24,d0
        move.w d0,smatposy
        move.w puntmatt,d0
        move.l #tabmattonelle,a0
        add.w d0,a0
        clr.l d0
        move.b (a0),d0
        and.b #127,d0
        and.w #$fff8,d0
        muls #2,d0
        move.w d0,smatposx

;stampa mattonelle

        move.l Glib,a6
        clr.l d2
        clr.l d3
        clr.l d0
        clr.l d1
        move.l rastport6,a0   ;clipblit
        move.l rastport5,a1
        move.w smatposx,d0  ;    d0=sourcex
        move.w smatposy,d1  ;    d1=sourcey
        move.w matposx,d2   ;    d2=destx
        move.w matposy,d3   ;    d3=desty
        move.l #16,d4 ;    d4=larghezza (x) dell'area
        move.l #24,d5  ;    d5=ampiezza  (y) dell'area
        move.l #$c0,d6 ;    funzione (c0= copia)
        jsr -552(a6)   ; questa e' la funzione (-552)
        rts

tampd0 dc.w 0
smatposx dc.w 0
smatposy dc.w 0
matposx dc.w 0
matposy dc.w 0
puntmatt dc.w 0
pitch dc.w 0
dati2  dc.w 0,0,352,230,4,1,64,$10f,0,0,0,0,0,0,0,0
tabmattonelle dcb.b 3200,0
dcoplist1   dc.w  $e0,0,$e2,0,$e4,0,$e6,0,$e8,0,$ea,0,$ec,0,
$ee,0,$ffff,$fffe
coplist1 dc.l 0
coplist2 dc.l 0
dcoplist2  dc.w   $e0,0,$e2,0,$e4,0,$e6,0,$e8,0,$ea,0,$ec,0,
$ee,0,$ffff,$fffe
nometabella dc.b '0data',0
```

Programma per ottenere uno scrolling hardware.