Programmare in C su Amiga

di Dario de Judicibus

decima puntata

Le tecniche di ripristino delle finestre oscurate e come definire l'immagine del mouse sono i due argomenti trattati in questa puntata. Aggiungiamo così altri due importantissimi mattoni all'insieme delle informazioni che ci permetteranno di controllare e sfruttare appieno la potenza di Intuition

In questa puntata approfondiremo le tecniche di «restauro» *[refresh]* delle finestre ed impareremo come si modifica l'immagine del puntatore del mouse e la si associa ad una finestra. Le prime servono a garantire l'integrità dell'interfaccia grafica presentata all'utente, la seconda ci permetterà di aggiungere un tocco personale ai programmi che utilizzano Intuition.

Tecniche di restauro

Le necessità di restaurare una finestra deriva dal fatto che Intuition permette la sovrapposizione totale o parziale di due o più finestre. Se una finestra parzialmente nascosta da un'altra viene spostata o portata di fronte a quest'ultima, è necessario ripristinare quella parte di finestra che era stata *oscurata*. Tale operazione si chiama «restauro» e può essere effettuata in tre modi diversi:

1. il «restauro semplice» *[simple refresh]*, in cui la responsabilità del ripristino dello schermo è affidata al programma applicativo;
2. il «restauro automatico» *[smart refresh]*, gestito da Intuition per mezzo di copie delle parti oscurate mantenute in memoria;
3. il «restauro a mappa» *[SuperBitMap]*, in cui l'area nascosta è ripristinata grazie al fatto che il contenuto della finestra è mantenuto in una parte della memoria separata da quella usata da Intuition per costruire la schermata da presentare a video (vedi il capitolo *Finestra a Mappa* nella scorsa puntata).

Vediamo ora in dettaglio le tre tecniche di restauro.

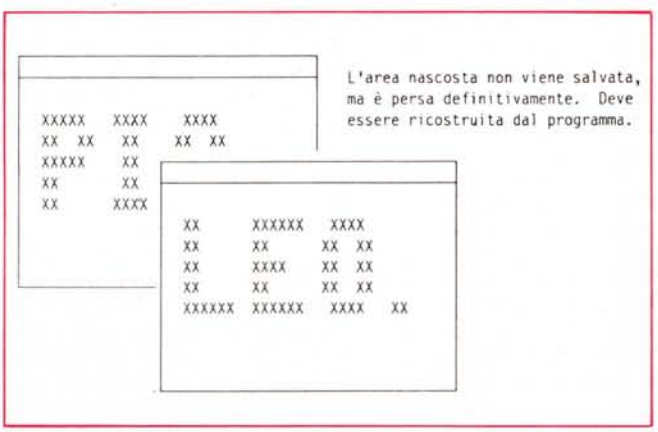


Figura 1
Tecniche di restauro: restauro semplice.

Restauro semplice

Nel caso del restauro semplice, Intuition non mantiene alcuna informazione relativa alle zone nascoste di una finestra (vedi figura 1). Se l'utente compie una qualunque operazione che scopre una parte precedentemente nascosta di quella finestra, è compito del programma che la gestisce ricostruire il contenuto della finestra andato perduto. Se viceversa la finestra è completamente visibile e l'utente si limita a spostarla qua e là, allora Intuition memorizza l'area grafica da spostare e la ridisegna là dove l'utente lascia andare il bottone del mouse. Durante lo spostamento ([1.2] e seguenti) la finestra è rappresentata da un bordo rettangolare agganciato al puntatore *[pointer]* del mouse.

Il vantaggio di questa tecnica consiste in un risparmio della memoria utilizzata, dato che Intuition utilizza solamente la memoria grafica dello schermo senza allocare ulteriore memoria per le parti nascoste. D'altra parte, dato che il restauro è completamente a carico del programma applicativo, quest'ultimo è più complesso e, in generale, il ripristino delle aree nascoste avviene in tempi maggiori. Ad esempio, se un programma ha disegnato una serie di oggetti nella finestra ed alcuni sono stati parzialmente cancellati dalla sovrapposizione di un'altra finestra, può rendersi necessario ridisegnare

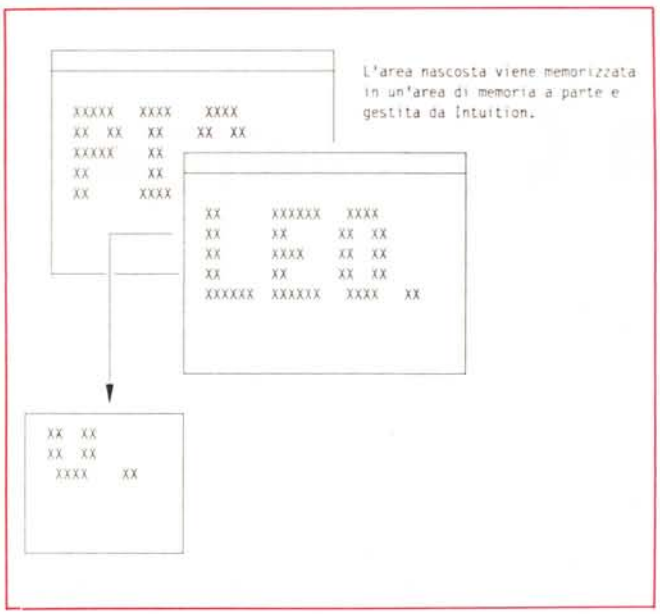


Figura 2 - Tecniche di restauro: automatico.

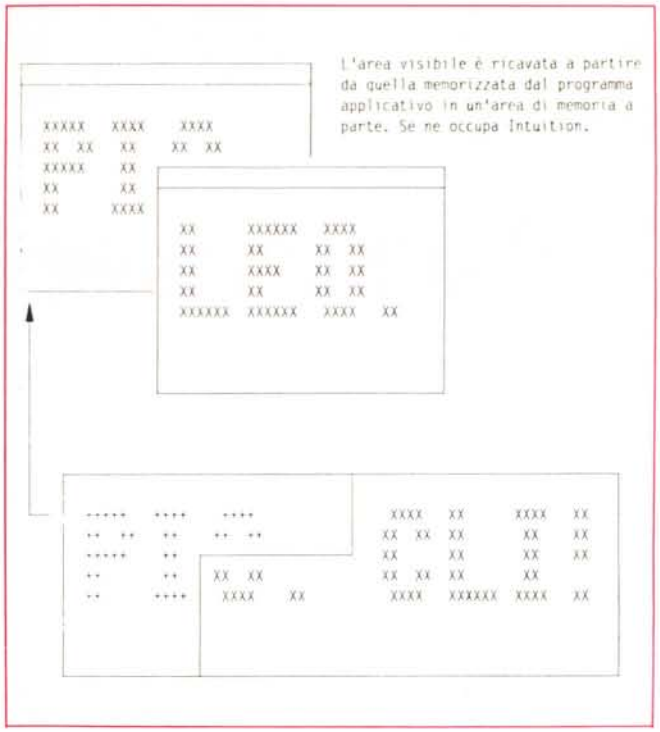


Figura 3 - Tecniche di restauro: a mappa.

gli oggetti per intero dato che il programma potrebbe non essere in grado di ricostruire solo le parti danneggiate.

Restauro automatico

Nel caso del restauro automatico, invece, Intuition si assume il compito di ripristinare tutte quelle aree che erano state nascoste per sovrapposizione. Per far questo mantiene in memoria una copia di tutte le parti oscurate (vedi figura 2) in modo da poterle ricostruire non appena l'utente sposta una finestra o la porta di fronte alle altre. Anche in questo caso, tuttavia, è possibile che il programma applicativo debba compiere alcune operazioni di restauro sulla finestra in questione. Questo avviene, ad esempio, quando la finestra viene allargata dall'utente, tramite il gadget di ridimensionamento, se disponibile, o dal programma stesso, tramite **SizeWindow()**. In questo caso Intuition non è in grado di decidere come va variato il contenuto della finestra, dato che l'area così ottenuta non esisteva precedentemente. La scelta ricade quindi sul programma applicativo che può decidere di lasciare tutto com'è, di ricostruire l'immagine nella finestra utilizzando una scala maggiore, o di aggiungere all'immagine altre parti secondo uno schema interno prefissato (vedi figura 4).

Questa tecnica ha il vantaggio di essere più rapida di quella semplice e libera il programmatore da buona parte del lavoro di ricostruzione. Tuttavia questi vantaggi si pagano con un maggior consumo di memoria per mantenere traccia delle parti nascoste. Per la parte della finestra visibile, invece, Intuition continua ad usare la memoria per lo schermo. Il consumo totale è quindi tanto maggiore quante più finestre sono aperte nello schermo e si sovrappongono fra loro.

Restauro a mappa

Terza ed ultima tecnica, quella del restauro a mappa non è in realtà solo una tecnica di ripristino delle aree nascoste, ma un vero e proprio modo alternativo di gestire il contenuto grafico di una finestra.

L'utente definisce un'area di memoria larga quanto o più delle dimensioni della finestra stessa (il massimo è 1024×1024 [1.2-1.3]) e la mette a disposizione di Intuition. Questo riempie la parte *visibile* della finestra con la corrispondente area «ritagliata» dalla mappa utente (vedi figura 3). Quando la finestra viene spostata, portata di fronte alle altre od allargata, Intuition copia dalla memoria utente quanto gli serve per ricostruire l'immagine.

Se la finestra viene viceversa parzialmente oscurata o ristretta, Intuition non fa assolutamente niente, dato che le informazioni necessarie per un eventuale successivo restauro sono comunque disponibili in un qualunque momento. Ovviamente, in questo caso, il consumo di memoria è molto elevato, dato che l'intera mappa è tutta e sempre mantenuta in memoria.

Restauro da programma

Vediamo ora cosa deve fare il programmatore nel caso debbano essere ripristinate delle aree che Intuition non può gestire. Abbiamo visto che questo può accadere sia nel caso di restauro semplice (attivato attraverso l'indicatore **SIMPLE_REFRESH**), sia in quello di restauro automatico (**SMART_REFRESH**) qualora la finestra sia allargata *lungo entrambi gli assi*. In questi casi il programma riceve da Intuition un messaggio del tipo **REFRESHWINDOW** con il quale viene avvertito della possibilità di dover effettuare un intervento di ricostruzione delle aree scoperte da una qualche azione dell'utente. Vedremo quando parleremo di IDCMP come un programma riceve o manda messaggi di questo tipo da ed ad Intuition. Spesso, se è stata utilizzata la tecnica di restauro automatico, il programma non prevede alcuna operazione di restauro diretto; se così è, basta attivare l'indicatore **NOCAREREFRESH** per segnalare ad Intuition che non si desiderano ricevere i messaggi di richiesta restauro. Nel caso invece si sia previsto un qualche intervento da parte del programma stesso, e comunque *sempre* in caso di restauro semplice, è necessario effettuare le operazioni di restauro seguendo lo schema seguente:

1. chiamare la funzione **BeginRefresh()** che individua le aree danneggiate in modo da assicurare la massima efficienza nell'opera di ricostruzione, evitando che vengano ripristinate anche quelle

parti della finestra che non ne hanno assolutamente bisogno;

2. ricostruire le aree grafiche scoperte e quelle sotto la responsabilità del programma stesso, seguendo una delle tante logiche di ricostruzione possibili (alcuni esempi sono riportati in figura 4);

3. chiamare la funzione **EndRefresh()** per ripristinare le strutture interne usate da Intuition per gestire le operazioni di restauro.

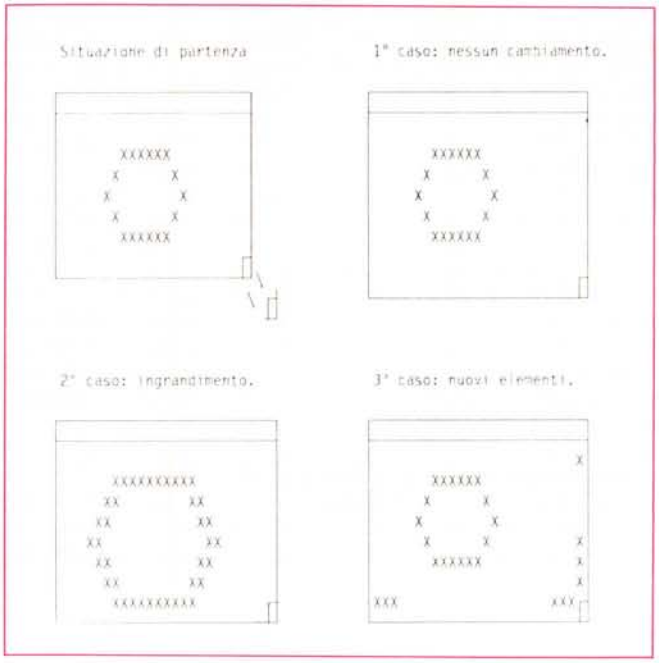
Durante la fase 2 dello schema indicato, il programma può utilizzare tutte le funzioni grafiche e quelle di Intuition specifiche per visualizzare grafica o testi nel *raster* della finestra. Sono invece assolutamente da evitare quelle che possono portare alla comparsa di un *requester* (come le funzioni dell'AmigaDOS che compiono operazioni di I/O su disco od **AutoRequest()**). Ci sono inoltre due funzioni che possono essere utilizzate al di fuori dello schema presentato e che servono a ridisegnare alcuni specifici elementi di una finestra, e precisamente:

- **RefreshWindowFrame()** [1.2]
per ridisegnare il bordo di una finestra nel caso che il programma lo abbia inavvertitamente danneggiato;
- **RefreshGadgets()**
per ridisegnare tutti i gadget appartenenti ad una finestra od ad un *requester*.

Ricordatevi che, qualora un programma riceva un messaggio di richiesta restauro per una determinata finestra, dovrà sempre e comunque effettuare almeno le operazioni minime per garantire l'integrità della finestra, e cioè chiamare in sequenza **BeginRefresh()** ed **EndRefresh()**. Vediamo perché. La funzione **BeginRefresh()** attiva una serie di indicatori e quindi riorganizza le varie parti di una finestra in modo che tutte le operazioni successive di ricostruzione avvengano solo là dove ce n'è effettivo bisogno. Supponiamo ad esempio che un utente abbia scoperto completamente una finestra che era per metà nascosta da un'altra, e che conteneva al suo interno il disegno di un cerchio. Intuition avverte il programma che gestisce la finestra che è necessario ricostruire il cerchio parzialmente danneggiato. Il programma chiama allora la **BeginRefresh()** e quindi la funzione grafica per disegnare cerchi. il fatto di aver chiamato prima la funzione **BeginRefresh()** fa sì che in realtà venga ridisegnata solo la parte mancante del cerchio, con un evidente vantaggio in *performance*. A questo punto basterà chiamare la **EndRefresh()** per azzerare gli indicatori attivati e lo stato del piano corrispondente alla finestra in questione. Chiamare in successione queste due funzioni anche quando il programma non prevede di effettuare alcuna operazione di ricostruzione, serve quindi ad annullare lo *stato di allarme* in cui Intuition si era messo, quando ha ritenuto necessario inviare il messaggio di richiesta restauro al programma.

Nel caso siano necessarie più operazioni di ricostruzione da parte di differenti task, è possibile avere più coppie di chiamate alle funzioni suddette, avendo cura di assegnare al secondo parametro della **EndRefresh()** (vedi prototipo in figura 5) il valore **FALSE**. Una volta che la finestra è stata completamente ricostruita potremo chiamare per l'ultima volta tale funzione specificando il valore **TRUE**.

Figura 4
Tecniche di restauro: ridimensionamento di una finestra.



Il puntatore del mouse

Ad ogni finestra attiva è associato un puntatore che può essere manovrato sia per mezzo della tastiera (tasto Amiga più tasti per lo spostamento del cursore), sia per mezzo di opportune periferiche quali il *mouse*, la *trackball*, il *joystick* e via dicendo. Questo puntatore serve a tirar giù i menu a tendina dalla barra orizzontale dello schermo, a selezionare oggetti (gadget, icone), a disegnare, a spostare, ad operare in generale su ciò che appare sullo schermo del vostro Amiga.

La posizione del puntatore nella finestra è mantenuta da Intuition in due cam-

```
/* ------------------------------------------------------------------------ */
/*  Funzioni di restauro                                                    */
/* ------------------------------------------------------------------------ */

void BeginRefresh(struct Window *);

void EndRefresh(struct Window *, long);

void RefreshWindowFrame(struct Window *);

void RefreshGadgets(struct Gadget *, struct Window *, struct Requester *);

/* ------------------------------------------------------------------------ */
/*  Funzioni per il puntatore                                               */
/* ------------------------------------------------------------------------ */

void ClearPointer(struct Window *);

void SetPointer(struct Window *, short *, long, long, long, long);
```

Figura 5 - Prototipi delle funzioni descritte nell'articolo.

pi della struttura **Window**, e precisamente **MouseX** e **MouseY**, entrambi relativi all'angolo superiore sinistro della finestra stessa (detto anche *origine*). Tali valori sono validi anche se la finestra in quel momento non è attiva e dipendono non solo dalla posizione relativa del cursore rispetto l'origine, ma anche dalla risoluzione utilizzata. Questo vuol dire ad esempio, che il punto centrale di una finestra di fondo senza bordi e che occupi tutto lo schermo (PAL) sarà (160,128) in bassa risoluzione e (320,256) in alta risoluzione interlacciata. Inoltre, come abbiamo già detto nella scorsa puntata, nel caso di una finestra Doppio Zero (GZZ), la posizione del puntatore relativamente alla finestra *interna* è memorizzata nei campi **GZZMouseX** e **GZZMouseY**, sempre nella struttura **Window**.

Quando parleremo di IDCMP vedremo come chiedere ad Intuition di fornirci la posizione del puntatore ogni qualvolta esso viene spostato dall'utente.

Se il programmatore non specifica altrimenti, il puntatore associato ad una finestra è lo stesso dello schermo a cui quella finestra appartiene. È tuttavia possibile associare ad una specifica finestra un altro puntatore, oppure modificare i colori di quello base. Mentre nel primo caso la modifica è limitata solo alla finestra in questione, nel secondo caso essa è estesa a tutto lo schermo, e quindi a tutte le finestre che lo compongono. Prima di vedere come si può fare tutto questo però, sarà opportuno spendere due parole sugli «spiritelli» *[sprite]* (vedi nota 1).

Gli sprite

Gli sprite sono degli oggetti larghi non più di 16 pixel in bassa risoluzione che possono essere mossi molto velocemente sullo schermo; la loro altezza invece può andare da un pixel all'altezza dello schermo su cui si muovono. Essi sono anche chiamati «sprite semplici» *[simple sprite]* per distinguerli da quelli hardware, su cui sono basati, e da quelli virtuali *[virtual sprite]*, gestiti dal **GEL** *[Graphics ELement system]*. Vedremo tutto ciò in dettaglio quando parleremo di animazione. La gestione degli sprite è basata su una funione hardware dell'Amiga, chiamata *hardware sprite system*. Uno sprite semplice è sempre visualizzato in bassa risoluzione, indipendentemente da quella del fondo in cui si muove. Al contrario i suoi colori dipendono in parte da quelli dello schermo su cui si muove, e comunque non possono essere più di quattro, di cui uno è il cosiddetto *colore trasparente*. Dato che l'Amiga ha solo otto sprite hardware, che lo sprite 0 è sempre

Questa è la "classica" sfera bianca e rossa di Amiga, dove

. è il colore trasparente
+ è il bianco
* è il rosso

```
. . . . . * * + + * . . . . .
. . . + * * + + * * + + * . . .
. . + * * + + * * * + + * * . .
. * + + * * * + + + + * * + + .
. * + * * * * + + + + * * * + .
* + + * * * + + + + + * * * + *
* * + + + + * * * * * + + + + *
* * + + + + + * * * * + + + + *
* * + + + + * * * * * + + + + *
* + + * * * * + + + + + * * * +
. * + * * * * + + + + * * * + .
. * + + * * * + + + + * * + + .
. . + * * + + * * * + + * * . .
. . . + * * + + * * + + * . . .
. . . . . * * + + * . . . . .
```

Tabella A

Colore	Piano 0	Piano 1	Reg
.	0	0	-
+	1	0	R17
*	1	1	R19

Tabella B

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Primo passo: scomposizione della figura nei due piani elementari.

Utilizzando la tabella A si ottiene:

Piano zero, o di fondo

```
0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0
```

Utilizzando la tabella B si ottiene:

Equivalente esadecimale per P0

```
0 7 E 0
1 F F 8
3 F F C
7 F F E
7 F F E
F F F F
F F F F
F F F F
F F F F
F F F F
7 F F E
7 F F E
3 F F C
1 F F 8
0 7 E 0
```

Utilizzando la tabella A si ottiene:

Piano uno, o di colore

```
0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0
0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0
0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 0
0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0
0 1 0 1 1 1 1 0 0 0 0 1 1 1 0 0
1 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0
1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1
1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1
1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1
1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1
1 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0
0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0
0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 0 0
0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0
```

Utilizzando la tabella B si ottiene:

Equivalente esadecimale per P1

```
0 3 2 0
0 C C 8
1 8 E 4
4 E 1 8
5 E 1 C
9 E 0 E
E 1 F 1
C 1 F 1
C 1 F 1
E 1 F 1
9 E 0 E
5 E 1 C
4 E 1 8
1 8 E 4
0 C C 8
0 3 2 0
```

Per cui si ottiene la seguente struttura:

```
UWORD sfera[] =
{
  0,0,
  0x07E0, 0x0320,
  0x3FFC, 0x18E4,
  0x7FFE, 0x4E18,
  0x7FFE, 0x5E1C,
  0xFFFF, 0x9E0E,
  0xFFFF, 0xE1F1,
  0xFFFF, 0xC1F1,
  0xFFFF, 0xC1F1,
  0xFFFF, 0xE1F1,
  0xFFFF, 0x9E0E,
  0x7FFE, 0x5E1C,
  0x7FFE, 0x4E18,
  0x3FFC, 0x18E4,
  0x1FF8, 0x0CC8,
  0x07E0, 0x0320,
  0,0
};
```

Nome: sfera
Larghezza: 16 pixel
Altezza: 16 linee
Colori: 3 + trasparente

NOTA: Per centrare questo sprite sulla posizione corrente del puntatore del mouse utilizzare i seguenti valori:

Delta X: -7
Delta Y: -7

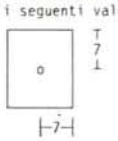


Figura 6 - Come si costruisce uno sprite.

quello associato al puntatore, e che il software che gestisce gli sprite semplici non permette il riutilizzo dello stesso sprite hardware in uno stesso passaggio del pennello elettronico sullo schermo *[video display scan]*, gli sprite semplici a disposizione del programmatore sono sette.

La struttura dati di uno sprite è una matrice formata da **n+2** coppie di parole (32 bit) dove **n** rappresenta l'altezza dello sprite in linee (o pixel). La prima e l'ultima coppia di parole sono sempre nulle (vedi figura 6) mentre le altre definiscono l'immagine dello sprite stesso. Vediamo come. Fate riferimento alla figura 6.

Innanzitutto formiamo l'immagine che vogliamo riprodurre su di un pezzo di carta quadrettata, ricordando che abbiamo a disposizione un'area larga 16 pixel al massimo ed alta quanto lo schermo. In genere anche quest'ultima è dello stesso ordine di grandezza. Nel nostro caso l'immagine da riprodurre è la famosa palla bianca e rossa, simbolo ben conosciuto dagli appassionati di Amiga. Le sue dimensioni sono 16×16. Il numero di colori a disposizione sono, come abbiamo già detto, tre più il trasparente. Nel nostro caso ne useremo solo due. Useremo il punto per indicare il colore trasparente e l'asterisco ed il segno del più per indicare gli altri due colori. Dato che per definire quattro colori bastano due piani, dovremo dividere la nostra immagine in due parti solamente. Vedremo che un modo simile di operare sarà usato anche per i gadget, i quali però possono avere molti più colori. I registri usati per i colori degli sprite (vedi nota 2) sono riportati in figura 7. Dato che il colore bianco nello schermo standard del WorkBench (vedi nota 3) è caricato nel registro 17 (che duplica il registro 1) mentre quello rosso è nel 19 (e nel 3), useremo il valore 0×01 per il bianco e 0×11 per il rosso. Il bit più basso va nel Piano 0 (o dell'immagine) mentre l'altro va nel Piano 1 (o del colore). Nel nostro caso il bit più basso è sempre «1» e questo fa sì che la prima delle due immagini riproduca un'area circolare di *uno* in una quadrata di *zeri*. L'altra invece contiene aree a scacchiera di bit 0 ed 1. Fatto questo riportiamo le due «immagini» dal formato binario a quello esadecimale. Nel nostro caso questo produce esattamente due byte per riga. Se lo sprite fosse stato più stretto, avremmo comunque dovuto riempire con zeri l'area a destra, fino a portarla a 16 bit. Ogni coppia di byte rappresenta una parola. Le due parole corrispondenti alla stessa riga dello sprite prese dalle due immagini a mappa di bit formano appunto le coppie che descrivono lo sprite nella struttura dati descritta precedentemente. Chiaro, no?

Notate che nella struttura così creata non c'è nessun riferimento a quali registri di colore vogliamo associare la nostra struttura.

Dato che intendiamo usarla per il puntatore di una finestra, e dato che tale puntatore è associato allo sprite 0, questi saranno 17 e 19, ma se avessimo usato la stessa struttura per lo sprite 5, per esempio, i colori sarebbero stati ricavati dai registri 25 e 27.

Colori	Sprite 0 ed 1	Sprite 2 e 3	Sprite 4 e 5	Sprite 6 e 7
1	17	21	25	29
2	18	22	26	30
3	19	23	27	31

Figura 7 - Registri usati per i colori degli sprite.

Come si cambia il puntatore

Ora che abbiamo il nostro sprite vediamo come utilizzarlo per associarlo alla finestra. Innanzitutto una cosa importantissima: i dati che definiscono una immagine (sprite, gadget, etc) devono assolutamente essere nella memoria di tipo **CHIP**. Essi infatti devono essere accessibili dall'hardware che gestisce la grafica e che, appunto, è in grado di accedere solo questo tipo di memoria. Per far questo ci sono tre modi (vedi nota 4):

1. Se avete la fortuna di possedere già il nuovo *Lattice C 5.0*, basterà utilizzare la dichiarativa *chip* nella definizione della struttura dati.
2. Se avete una versione precedente (3.03, 3.10, 4.00 o 4.01) dovete copiare la struttura dati in un'area di memoria CHIP precedentemente allocata, come mostrato in figura 8. Attenzione: la funzione **CopyMem()** è solo [1.2] e seguenti.
3. Se il programma non ha molti dati potete dire al compilatore di mettere *tutti* i dati in memoria di tipo CHIP utilizzando l'opzione **-ad** del comando LC o della **-cd** del comando LC2.

Un quarto modo consiste nell'usare il comando ATOM, ma non lo consiglio a meno che non siate dei buoni programmatori in Assembler 680xx.

Una raccomandazione sopra a tutte: non ignorate il fatto che certe strutture dati vanno *sempre* in memoria CHIP solo perché possedete un Amiga con solo 512K o meno, altrimenti i vostri programmi non gireranno mai su Amiga con 1M o

Note

1. Dato che il termine inglese *sprite* è molto più usato della traduzione italiana *spiritello*, useremo quello anglosassone.

2. L'Amiga ha 32 registri hardware per il colore, chiamati COLOR00, COLOR01, ..., COLOR31 con offset rispetto all'indirizzo base di *Denise* (OxDFF000) che vanno da 0×180 ad 0×1BE.

3. Ovviamente tali colori possono essere modificati tramite *Preferences*. Se si vuole essere sicuri di avere proprio un certo colore piuttosto che un altro, bisogna caricare quel colore nel registro appropriato. Attenzione però: se lo fate alterate quel colore per tutto lo schermo. Se lo schermo usa solo 16 colori questo si noterà al massimo nel puntatore associato allo schermo, se diverso da quello della finestra, ma se lo schermo usa tutti e trentadue i colori... Provare per credere.

Nello schermo standard del Work Bench i registri da 16 a 31 hanno gli stessi colori dei registri da 0 a 15.

4. Questo vale solo per il *Lattice C*. I possessori di *Atzec C* o di un altro compilatore facciano riferimento al manuale in dotazione in loro possesso. Ne approfitto per raccomandare ancora una volta i compilatori originali. Se una copia pirata di un altro prodotto può spesso essere utilizzata senza manuali, non sperate di poter usare seriamente un compilatore senza manuale o senza il supporto telefonico della software house.

5. Desidero ringraziare Michele F. e Paolo M. per il supporto tecnico nel recupero di questo articolo da un dischetto completamente distrutto, Silvia per quello morale (il supporto, non il dischetto!) e la Tecnicomp di Roma per aver riparato il mio monitor in meno di mezza giornata. Senza di loro molto probabilmente questo articolo non sarebbe mai stato pubblicato in tempo.

più. Non siate egoisti.

Una volta sicuri che **sfera[]** verrà caricata nella memoria CHIP, è necessario definire altri quattro valori da passare alla **SetPointer()**. Questi quattro campi sono:

1. l'altezza dell'immagine in linee,
2. la larghezza dell'immagine in pixel (bassa risoluzione),
3. lo slittamento orizzontale verso destra dell'origine dell'immagine rispetto alla posizione corrente del puntatore;
4. lo slittamento verticale verso il basso dell'origine dell'immagine rispetto alla posizione corrente del puntatore.

Gli ultimi due campi servono a posizionare l'immagine del puntatore su quella che Intuition considera la *posizione effettiva* del puntatore stesso. È in pratica quello che fate quando dal pannello di *Preferences* che serve a modificare il puntatore, selezionate l'opzione *Set Point* e definite un punto all'interno dell'immagine (per esempio la punta di una freccia od il centro di una crocetta). Se entrambi questi valori sono a zero, il puntatore vero e proprio sarà posizionato nell'origine dell'immagine, cioè in corrispondenza dell'angolo in alto a sinistra. Se volete che il puntatore si trovi all'interno dell'immagine, per esempio nel centro (7,7), dovrete spostare l'immagine nel senso opposto dando ad entrambi i campi il valore −7. Ricordatevi: il puntatore deve essere pensato *fisso*, è l'immagine che si sposta. Un valore positivo per entrambi i campi, quindi, allontana l'origine dell'immagine rispettivamente verso destra e verso il basso. Nell'Amiga, infatti, gli spostamenti positivi sono sempre verso destra se orizzontali, e verso il basso se verticali, nella stessa direzione quindi degli assi del sistema cartesiano dello schermo: origine in alto a sinistra, asse x orizzontale e positivo a sinistra, asse y verticale e positivo verso il basso. Nel nostro caso, per definire come nuovo puntatore la nostra sfera (16×16 pixel) con il puntatore al centro, dovremo scrivere:

```
#define ALTEZZA          16
#define LARGHEZZA        16
#define VERSODESTRA      -7
#define VERSOILBASSO     -7
SetPointer(finestra, sfera, ALTEZZA,
LARGHEZZA, VERSODESTRA, VERSOIL-
BASSO);
```

Certo una sfera a scacchi non è l'immagine più adatta ad un puntatore del mouse, ma sicuramente la vostra fantasia saprà suggerirvi un campionario di immagini vastissimo, no? Anzi, visto che ci siamo, perché non vi fate un bell'esercizio?

```
/* --- Lattice C 5.00 ------------------------------------------------ */

    UWORD chip sfera[] =
    {
      0,0,
      0x07E0, 0x0320, 0x1FF8, 0x0CC8, 0x3FFC, 0x18E4, 0x7FFE, 0x4E18,
      0x7FFE, 0x5E1C, 0xFFFF, 0x9E0E, 0xFFFF, 0xE1F1, 0xFFFF, 0xC1F1,
      0xFFFF, 0xC1F1, 0xFFFF, 0xE1F1, 0xFFFF, 0x9E0E, 0x7FFE, 0x5E1C,
      0x7FFE, 0x4E18, 0x3FFC, 0x18E4, 0x1FF8, 0x0CC8, 0x07E0, 0x0320,
      0,0
    };

/* --- Lattice C 3.xx & 4.xx ----------------------------------------- */

  #define BYTESPERSFERA 72L

  UWORD *sferachip = NULL;
  UWORD sfera[] =
  {
    0,0,
    0x07E0, 0x0320, 0x1FF8, 0x0CC8, 0x3FFC, 0x18E4, 0x7FFE, 0x4E18,
    0x7FFE, 0x5E1C, 0xFFFF, 0x9E0E, 0xFFFF, 0xE1F1, 0xFFFF, 0xC1F1,
    0xFFFF, 0xC1F1, 0xFFFF, 0xE1F1, 0xFFFF, 0x9E0E, 0x7FFE, 0x5E1C,
    0x7FFE, 0x4E18, 0x3FFC, 0x18E4, 0x1FF8, 0x0CC8, 0x07E0, 0x0320,
    0,0
  };

  void main()
  {
    .
    .
    if ((sferachip = (UWORD *)AllocChip(sfera,BYTESPERSFERA)) == NULL)
    {
      printf("Non posso allocare l'immagine del puntatore in CHIP\n");
      CloseAll();
    }
    .
    .
    if (sferachip) FreeMem(sferachip,BYTESPERSFERA);
    .
    .
  }

  APTR AllocChip(fast,size)
  UWORD *fast;
  int size;
  {
    APTR chip;
    chip = (APTR)AllocMem(size,MEMF_CHIP);
    if (chip) CopyMem(fast,chip,size);
    return (chip);
  }
```

Figura 8 - Come assicurarsi che l'immagine sia in memoria CHIP.

L'esercizio

Era da un po' troppo tempo che ve la cavavate con esercitazioni libere. Questa volta ritorniamo sul buon vecchio compito a casa. Le informazioni per portarlo a termine le avete tutte. Quello però che vi si chiede non è solo di risolvere il problema, ma di risolverlo elegantemente.

L'esercizio consiste nello scrivere un programmino che apre un certo numero di finestre. Una di queste solamente avrà il gadget di chiusura e dovrà essere alta quanto la barra del titolo, le altre verranno chiuse quando chiederete di chiudere la prima. Ogni finestra (fatele piccole) dovrà avere associato un puntatore differente, di modo che, mettendo il mouse su una certa finestra e premendo il bottone di selezione del mouse, compaia l'immagine corrispondente allo sprite definito per quella finestra. Il difficile sta nello scrivere un programma modulare e strutturato in modo da rendere estremamente semplice cambiare il numero delle finestre da aprire senza dover riscrivere il tutto. L'unica cosa da fare per aggiungere un'ulteriore finestra, dovrebbe essere la modifica di una costante e l'aggiunta di un nuovo sprite da associarvi.

Conclusione

La prossima volta vedremo come cambiare i colori del puntatore, e moltissime altre funzioni grafiche elementari per permettervi finalmente di disegnare nelle finestre che ormai dovreste saper definire, aprire e chiudere. Almeno spero!

MC

Byte Line

STAMPANTI

Dela Printer, 240 cps, 136 col.	L.	699.000
Dela Printer, 180 cps, 80 col.	L.	499.000
NEC P6 Plus 24 aghi	L.	1.298.000
NEC P7 Plus 24 aghi	L.	1.598.000
NEC P2200 24 aghi	L.	649.000
Citizen 120 D	L.	298.000
Citizen MSP - 15 E	L.	549.000
Star LC - 10	L.	399.000
Star LC - 10 color	L.	499.000
Star LC - 24 10 24 aghi	L.	649.000
Star NX - 15 136 col.	L.	699.000
Cavo IBM - Centronics	L.	13.900
Epson LQ-500	L.	649.000

COMPUTER

XT compatibile 10 MHz	da	L.	690.000
AT compatibile 12 MHz	da	L.	1.190.000
AT completo 512K HD 20Mb		L.	1.999.000

ACCESSORI

Handyscanner 105 mm	L.	450.000
IBM- Mouse	L.	79.000
GENOA SuperEGA Hires	L.	498.000
Genoa Super VGA 5200	L.	699.000
Modem 1200H interno	L.	178.000
Modem 1200C esterno	L.	239.000
Fax Murata M-1	L.	1.390.000

ordine minimo 100 dischetti 100% Errorfree

Dela Disk 5.25" 2D	L.	840
Dela Disk3.5" 2DD	L.	2.100
No Name 5.25" 2D	L.	690
No Name 3.5" 2DD	L.	1.890
No Name 5.25" 2HD 1.2 Mbyte	L.	2.100
Diskbox per 100 Floppy 5,25"	L.	14.900
Diskbox per 50 Floppy 3,5"	L.	14.900

I PREZZI SI INTENDONO
AL NETTO DI I.V.A.

VENDITA PER CORRISPONDENZA

HARD DISK

Seagate File Card	L.	799.000
ST225 21,4 MB	L.	378.000
ST251 42,8 MB	L	678.000
ST250 40 MB RLL incl. contr	L.	699.000
AMIGOS 20 MB Hard-Disk per AMIGA 500 o AMIGA 1000	L.	999.000

Prezzi suscettibili alla variazione del dollaro!!!

MONITOR

Flatscreen Dual Frequency Invers	L.	238.000
NEC Multisync II	L.	1.098.000
Mitsubishi Multisync EUM 1481 A	L.	998.000
Cavo Mitsubishi - VGA	L.	39.000

Byte Line
Via Lorenzo il Magnifico,148
00162 Roma - Tel. (06) 42.70.418