Programmare in C su Amiga

di Dario de Judicibus

Punti, linee, archi e cerchi: con questa puntata entriamo nel cuore della libreria grafica dell'Amiga. «Noi forniamo gli strumenti, il resto è immaginazione ed estro...»

Dato che la scorsa puntata è stata dedicata completamente al programma di utilità *LMK*, questa volta torneremo a parlare di programmazione in C e precisamente delle funzioni della libreria grafica, o **graphics.library.**

Introduzione

Nell'undicesima puntata, abbiamo riportato una lista completa delle funzioni della libreria grafica [1.3], sotto forma di prototipi. In figura 3 riportiamo, per comodità, quelle di cui ci occuperemo in questa puntata.

Sempre per comodità ricordiamo che, per poter usare tali funzioni, bisogna sempre aprire la libreria grafica (vedi figura 1 - sezione 1) ed in genere, è opportuno definire uno o più puntatori alle strutture di tipo **RastPort** corrispondenti ai *raster* su cui si vuole operare. Nel caso di finestre, in particolare, tali puntatori si ottengono da quelli delle corrispondenti strutture **Window** come riportato in figura 1 - sezione 2.

```
/*
**  *** Sezione 1 ***  Libreria grafica ***
*/

 #define GNAME "graphics.library"
 #define GVERS 0          /* o la versione che REALMENTE vi serve */
 struct GfxBase *GfxBase; /* Non cambiate MAI il nome del puntatore BASE! */
 ...
 GfxBase = (struct GfxBase *)OpenLibrary(GNAME,GVERS);
 if (GfxBase == NULL) Error(NOGRAPHICSLIBRARY); /* o quello che volete voi */
 ...

/*
**  *** Sezione 2 ***  Puntatori ai raster ***
*/

 struct Window *wa, *wb, ... , *wn;
 struct RastPort *rpa;
 struct RastPort *rpb;
 ...
 struct RastPort *rpn;
 ...
 rpa = wa->RPort;
 rpb = wb->RPort;
 ...
 rpn = wn->RPort;
 ...
```

Figura 1 - Codice di inizializzazione.

Punti, linee, e tanti colori...

Supponiamo di aver aperto la nostra libreria grafica, quella di Intuition, visto che per ora lavoreremo con delle finestre, ed una finestra di tipo **GimmeZeroZero**, tanto per non doverci per adesso preoccupare dei bordi della finestra. Se a questo punto non avete capito il perché di quanto or ora affermato, andatevi a rileggere la nona puntata, quella pubblicata sul numero 82 di MC (febbraio 1989). In seguito vedremo che non è necessario usare una finestra GZZ purché si adottino alcune precauzioni, e che, di fatto, non è neppure necessario usare proprio una finestra (e di conseguenza Intuition).

Immaginiamo di avere una penna, e di dover definire un insieme di funzioni per manovrare questa penna un po' particolare e molto potente. Per prima cosa dovremo decidere come scrivere con questa penna. Tali modi, si dicono appunto *modi grafici*. Vediamo quindi per primi i diversi modi grafici che la **graphics.library** ci mette a disposizione (vedi nota 1). Per far questo, dobbiamo prima però definire due termini: la maschera sorgente [*source pattern*] e l'area bersaglio [*target area*].

La maschera sorgente altro non è che il modello binario dell'oggetto grafico da riportare nel *raster*. Ad esempio: la maschera sorgente di un punto è un singolo bit di tipo **«1»** un un'area di tipo **«0»**; quella di una linea è una sequenza di **«1»**; una linea tratteggiata regolare è formata da un certo numero di **«1»** seguiti da uno stesso (o differente) numero di **«0»** e così via, all'infinito.

Analogamente, si può definire una maschera sorgente per delle aree, le cosiddette *tassellature [area pattern]*. Alcuni esempi sono riportati in figura 2.

Un'area bersaglio, invece, è quella porzione del *raster* su cui si opera con le varie funzioni grafiche.

La differenza fondamentale tra le due aree (maschera sorgente ed area bersaglio), è che, mentre la prima è in realtà una mascherina a due colori, come quelle che si usano per dipingere sui camion militari le scritte con la vernice a spruzzo, la seconda è realmente un

pezzo della finestra, cioè una superficie formata da un certo numero di pixel che possono avere tanti colori quanti sono quelli ammessi per il numero di piani allocali per quella determinata area grafica. Dato che la maschera sorgente serve a definire come vanno modificati i colori dei pixel dell'area bersaglio, a seconda del modo grafico, ne risulta che è possibile utilizzare nello stesso momento due colori diversi, come se cioè la nostra penna, invece di essere simile al pennino di un plotter, fosse un po' come quelle penne bicolori che contengono una cartuccia blu ed una rossa nello stesso stilo. Questo ovviamente non vuol dire che non è possibile disegnare con più di due colori, dato che il numero di colori dipende solamente dal numero di piani sui quali si sta operando; bensì significa che è possibile con lo stesso comando, associando ai bit di tipo **«1»** un colore, ed a quelli di tipo **«0»** un altro, disegnando linee, aree, figure geometriche elementari con due colori piuttosto che con uno solamente.

Torniamo ora ai modi grafici disponibili:

JAM1
utilizza il colore primario per disegnare; cioè, tutti i pixel dell'area bersaglio che corrispondono ai bit di tipo **«1»** della maschera sorgente, sono colorati utilizzando il colore primario.

JAM2
utilizza sia il colore primario che il secondario per disegnare; cioè, tutti i pixel dell'area bersaglio che corrispondono ai bit di tipo **«1»** della maschera sorgente, sono colorati utilizzando il colore primario, quelli in corrispondenza dei bit di tipo **«O»**, invece, vengono colorati con il secondario.

COMPLEMENT
in corrispondenza dei bit di tipo **«1»** della maschera sorgente, cambia il colore dei pixel dell'area bersaglio con il suo complemento. Ad esempio, se un certo pixel ha il colore definito dal registro 0×0110, esso viene modificato in 0×1001. Naturalmente questi sono i valori dei registri di colore. Al contrario di quanto succede in fotografia, ad esempio, dove ogni colore ha il suo complemento ben definito, qui il complemento di un colore dipende dalla tavolozza caricata nei registri di colore, e non è definito a priori.

INVERSID
viene utilizzato in combinazione con **JAM1** e **JAM2** per invertire fra di loro il colore di fondo [*background*] e quello del tratto [*foreground*], per ottenere la cosiddetta «immagine inversa» [*reverse image*]. Particolarmente usato con i testi.

Il modo grafico viene impostato utilizzando la funzione **SetDrMd()** (vedi il prototipo in figura 3).

```
          00000
Punto:    00100
          00000


Linee:    1111111111111111111111111111111111111111111111111   continua

          1111000011110000111100001111000011110000111110   tratteggiata

          1111100100111110010011111001001111100100111111   linea-punto

          1010101010101010101010101010101010101010101010   puntini


Aree:     1110001110001110001 <-- Scacchiera    1000100010001000100
          1110001110001110001                   0100010001000100010
          0001110001110001110                   0010001000100010001
          0001110001110001110                   0001000100010001000
          1110001110001110001                   1000100010001000100
          1110001110001110001                   0100010001000100010
          0001110001110001110                   0010001000100010001
          0001110001110001110                   0001000100010001000
          1110001110001110001                   1000100010001000100
          1110001110001110001     Diagonali --> 0100010001000100010


          1111111000001111111 <-- Piastrelle    0111000111000111000
          1000001111111000000                   1000111000111000111
          1111111000001111111                   0111000111000111000
          1000001111111000000                   1000111000111000111
          1111111000001111111                   0111000111000111000
          1000001111111000000                   1000111000111000111
          1111111000001111111                   0111000111000111000
          1000001111111000000                   1000111000111000111
          1111111000001111111     Cerchietti -> 0111000111000111000
          1000001111111000000                   1000111000111000111
```

Figura 2
Mascherine e tassellature.

Figura 3 ▶
Le funzioni grafiche dell'Amiga trattate in questo articolo.

```
/*
** Prototipi delle funzioni grafiche: Versione 1.3
*/

/*
** *** COLORI ***
*/
 void  SetDrMd    (struct RastPort *, long);
 void  SetAPen    (struct RastPort *, long);
 void  SetBPen    (struct RastPort *, long);

/*
** *** PUNTI ***
*/
 void  Move       (struct RastPort *, long, long);
 long  ReadPixel  (struct RastPort *, long, long);
 void  WritePixel (struct RastPort *, long, long);

/*
** *** LINEE ***
*/
 void  Draw       (struct RastPort *, long, long);
 void  PolyDraw   (struct RastPort *, long, short *);

/*
** *** AREE ***
*/
 void  RectFill   (struct RastPort *, long, long, long, long);
```

Figura 4 ▶
Le macro grafiche dell'Amiga trattate in questo articolo.

```
/*
** Definizioni delle macro grafiche: Versione 1.3
**
** PER USARLE, AGGIUNGERE: #include "graphics/gfxmacros.h"
*/

/*
** *** COLORI ***
*/
#define SetOPen(r,c)     {(r)->AOlPen = c;(r)->Flags |= AREAOUTLINE;}
#define SetDrPt(r,p)     {(r)->LinePtrn = p;(r)->Flags |= FRST_DOT;
                          (r)->linpatcnt=15;}
#define SetAfPt(r,p,n)   {(r)->AreaPtrn = p;(r)->AreaPtSz = n;}

#define BNDRYOFF(r)      {(r)->Flags &= AREAOUTLINE;}
```

A questo punto è necessario definire i colori da usare per la penna. In realtà, dato che una finestra è qualcosa di più di un semplice *raster*, non è proprio obbligatorio definire il colore primario e quello secondario. Infatti una finestra ha già definiti di *default* tali colori. Se infatti aprite una finestra e, utilizzando il puntatore alla struttura **RastPort** ad essa associata, stampate a *console* i valori che i campi di tale struttura, puntatori esclusi, hanno all'inizio, otterrete pressappoco il listato in figura 5 (vedi nota 2). Come si può facilmente vedere, il modo grafico iniziale (**DrawMode**) è posto ad 1, cioè **JAM1**. I due colori della penna, quello primario e quello secondario, sono memorizzati rispettivamente in **FgPen** e **BgPen**. Mentre quest'ultimo è a 0, colore corrispondente al colore di fondo della finestra in questione, il primo è posto a –1. Non sono riuscito a trovare informazioni sul significato di tale valore, ma, tracciando delle linee *senza* specificare un differente colore primario, ho ottenuto come colore quello che nel WorkBench è il 2. Provate a vedere cosa succede su uno schermo utente a, rispettivamente 2, 4 e 8 colori (compreso quello di fondo). In ogni caso, il colore primario di *default* sembra non corrispondere a quello di *foreground* della finestra.

Tornando alle nostre funzioni grafiche, le procedure per cambiare i colori della penna sono **SetAPen()** per il primario e **SetBPen()** per il secondario (prototipi in figura 3).

Naturalmente, se utilizzate una qualunque delle funzioni appena viste, e la maggior parte di quelle che vedremo in seguito, anche i rispettivi campi nella struttura **RastPort** associata verranno aggiornati. Tali valori sono quindi sempre a disposizione per ricavare tutte le informazioni correnti per il *raster* su cui si sta operando. Un esempio di come utilizzare tali informazioni è riportato in figura 6.

A questo punto vi sarete forse chiesti cosa è quell'altra penna che compare sia in figura 5 che in figura 6, cioè **A01Pen**. Essa è utilizzata nei seguenti casi, come vedremo in seguito, quando parleremo di *operazioni di riempimento [filling]*:

- nel caso di generazioni di aree piene [*area fill*], definisce il colore del bordo che va disegnato intorno all'area in questione;
- nel caso di riempimento di aree [*flood fill*], definisce il colore *contro* il quale il riempimento deve cessare.

Chi ha lavorato qualche volta con *DeLuxe Paint* non dovrebbe aver problemi a riconoscere quest'ultimo caso. In questo prodotto (uno dei primi per Amiga e, a mio parere, ancora ben difficile da eguagliare per potenza e facilità d'uso) infatti, una volta definita una superficie chiusa di un certo colore, se si va a riempirla con un colore differente il riempimento si esaurisce là dove i due colori entrano in contatto.

Per specificare il colore della penna di contorno [*outline pen*], si usa questa volta una macro C, e precisamente **Set-0Pen()**, la cui definizione è riportata in figura 4. In effetti, dal punto di vista del programmatore, questa macro si usa in modo analogo alle due funzioni già viste, tuttavia, essendo appunto una macro, è necessario includere il file appropriato, e precisamente **graphics/gfxmacros.h**, il quale, a sua volta, include il file **graphics/rastport.h** che contiene tra l'altro le definizioni delle costanti usate dalle macro stesse. Il primo parametro da passare alla macro è, come al solito, il puntatore alla struttura **RastPort** su cui si opera, il secondo è, in linea con i prototipi già visti, il colore da utilizzare.

Nel caso si voglia sospendere anche temporaneamente la *bordatura* delle aree piene, si può utilizzare la macro **BNDRYOFF().** Per ripristinarla si può chiamare di nuovo la **Set0Pen()**, fornendogli lo stesso registro di colore, oppu-

```
*  Valori iniziali per  campi della struttura RastPort associata ad una
*  finestra, esclusi gli indirizzi ai puntatori ad altre strutture.

   struct RastPort

   UBYTE   Mask          :    255
   BYTE    FgPen         :     -1
   BYTE    BgPen         :      0
   BYTE    AOlPen        :     -1
   BYTE    DrawMode      :      1
   BYTE    AreaPtSz      :      0
   BYTE    linpatcnt     :      0
   USHORT  Flags         : 0x0000
   USHORT  LinePtrn      : 0xFFFF
   SHORT   cp_x, cp_y    :      0 , 0
   UBYTE   minterms[0]   :   0xCA
   UBYTE   minterms[1]   :   0xCA
   UBYTE   minterms[2]   :   0xCA
   UBYTE   minterms[3]   :   0xCA
   UBYTE   minterms[4]   :   0xCA
   UBYTE   minterms[5]   :   0xCA
   UBYTE   minterms[6]   :   0xCA
   UBYTE   minterms[7]   :   0xCA
   UBYTE   minterms[8]   :   0x00
   SHORT   PenWidth      :      0
   SHORT   PenHeight     :      0
   UBYTE   AlgoStyle     :      0
   UBYTE   TxFlags       :      0
   UWORD   TxHeight      :      8
   UWORD   TxWidth       :      8
   UWORD   TxBaseline    :      6
   WORD    TxSpacing     :      0
```

Figura 5 - RastPort all'apertura di una finestra.

```
/*
**  Queste sono alcune semplici macro che mostrano come utilizzare
**  le informazioni contenute in una struttura RastPort passando
**  il puntatore a tale struttura...
*/

#define  GetDrawMode(r) ((r)->DrawMode)
#define  GetAPen(r)     ((r)->FgPen)
#define  GetBPen(r)     ((r)->BgPen)
#define  GetOPen(r)     ((r)->AOlPen)
#define  GetxPos(r)     ((r)->cp_x)
#define  GetyPos(r)     ((r)->cp_y)
#define  GetDrPt(r)     ((r)->LinePtrn)

/*
**  ...oppure direttamente quello della finestra su cui si sta operando
*/

#define  WinDrawMode(w) (((w)->RPort)->DrawMode)
#define  WinAPen(w)     (((w)->RPort)->FgPen)
#define  WinBPen(w)     (((w)->RPort)->BgPen)
#define  WinOPen(w)     (((w)->RPort)->AOlPen)
#define  WinxPos(w)     (((w)->RPort)->cp_x)
#define  WinyPos(w)     (((w)->RPort)->cp_y)
#define  WinDrPt(w)     (((w)->RPort)->LinePtrn)

/*
**  Le parentesi intorno al puntatore passato servono ad evitare
**  effetti secondari nell'uso delle macro stesse
*/
```

Figura 6 - Come utilizzare le informazioni in RastPort.

◀ *Figura 7*
Esempio: Poly Draw()

```
#define XY_N       5L
#define POLYCOLOR  3

short xy_coords[XY_N] =
{
     25, 158,
    221,  78,
     98,  34,
    137, 200,
      1,  45
};

SetAPen (rp, POLYCOLOR);
PolyDraw (rp, XY_N, xy_coords);
```

```
/*                                          v   v   v   v
**  Alcune mascherine:    Linea Continua    1111111111111111    0xFFFF
**                        Trattini          1111000011110000    0xF0F0
**                        Tratto Punto      1110010011100100    0xE4E4
**                        Puntini           1100110011001100    0xCCCC
**                        Asimmetrico       1011001010110010    0xB2B2
**                        Fantasia          1011001110001100    0xB38C
**
*/

 #define LINEA       0xFFFF
 #define TRATTINI    0xF0F0
 #define TRAPUNTO    0xE4E4
 #define PUNTINI     0xE4E4
 #define ASIMM       0xB2B2
 #define FANTASIA    0xB38C

 void DrawLine(r,xa,ya,xb,yb,c,p)
   struct RastPort *r;
   long xa, ya, xb, yb, c;
   USHORT p;
 {
 /*
 ** ---------------- *
 ** oc = GetAPen(r); *  Togli il commento a queste righe per salvare
 ** op = GetDrPt(r); *  i valori attuali della penna: posizione (x,y),
 ** xo = GetxPos(c); *  colore primario, mascherina di linea.
 ** yo = GetyPos(c); *
 ** ---------------- *
 */
   SetAPen(r,c);    /*
   SetDrPt(r,p);     *  Questo blocco traccia il segmento tra gli estremi
   Move(r,xa,ya);    *  dati, nel colore e tratteggio specificati.
   Draw(r,xb,yb);    */
 /*
 ** ---------------- *
 **  SetAPen(r,oc);  *  Togli il commento a queste righe per ripristinare
 **  SetDrPt(r,op);  *  i valori attuali della penna: posizione (x,y),
 **  Move(r,xo,yo);  *  colore primario, mascherina di linea.
 ** ---------------- *
 **
 */
 }

/* === ESEMPIO DI USO ================================================ */
/*
**  Triangolo con i lati: ROSSO CONTINUO, NERO A TRATTINI, BIANCO FANTASIA.
*/
 #define ROSSO  3
 #define NERO   2
 #define BIANCO 1

 DrawLine (r,  50,  50, 150,  50, ROSSO,  LINEA);
 DrawLine (r, 150,  50, 100, 125, NERO,   TRATTINI);
 DrawLine (r, 100, 125,  50,  50, BIANCO, LINEA);
```

Figura 8 - Esempio: Mascherine lineari.

```
/*
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0    M A S C H E R I N A
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F        A   D U E
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F    D I M E N S I O N I
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0      . . . . . .
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0
**   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0     0xF0F0
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F    S C A C C H I E R A
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F
**   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1     0x0F0F
*/

UWORD scacchiera[] =
{
  0xf0f0, 0xf0f0, 0xf0f0, 0xf0f0,  /* prima fila di riquadri   *,
  0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f,  /* seconda fila di riquadri */
  0xf0f0, 0xf0f0, 0xf0f0, 0xf0f0,  /* terza fila di riquadri   */
  0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f   /* quarta fila di riquadri  */
};

SetAfPt(rp, scacchiera, 4);  /* dato che 2 alla 4a = 16 */
```

```
/*
**  Piano 0:       Piano 1:       Piano 2:          3 piani == 8 colori
**
**                                                  3 immagini, una per
**                                ////////          piano, ognuna larga
**                                ////////          16 bit ed alta 16
**                                ////////          righe.
**
**  scacchiera     cornice        venature
*/

UWORD sca_marmo[] =
{
  /*  Piano 0: scacchiera     */

  0xf0f0, 0xf0f0, 0xf0f0, 0xf0f0,  /* prima fila di riquadri   */
  0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f,  /* seconda fila di riquadri */
  0xf0f0, 0xf0f0, 0xf0f0, 0xf0f0,  /* terza fila di riquadri   */
  0x0f0f, 0x0f0f, 0x0f0f, 0x0f0f,  /* quarta fila di riquadri  */

  /*  Piano 1: cornice        */

  0xffff, 0x8001, 0x8001, 0x8001,  /* prima fila (superiore)   */
  0x8001, 0x8001, 0x8001, 0x8001,  /* seconda fila             */
  0x8001, 0x8001, 0x8001, 0x8001,  /* terza fila               */
  0x8001, 0x8001, 0x8001, 0xffff,  /* quarta fila (inferiore)  */

  /*  Piano 2: venature       */

  0x9249, 0x2492, 0x4924, 0x9249,  /* prima fila               */
  0x2492, 0x4924, 0x9249, 0x2492,  /* seconda fila             */
  0x4924, 0x9249, 0x2492, 0x4924,  /* terza fila               */
  0x9249, 0x2492, 0x4924, 0x9249,  /* quarta fila              */
};

SetAPen(rp, 255);            /* si vuole disegnare su TUTTI i piani      */
SetBPen(rp, 0);              /* questo è obbligatorio                    */
SetDrMd(rp, JAM2);           /* questo è il modo corretto da usare       */
SetAfPt(rp, sca_marmo, -4);  /* dato che 2 alla 4a = 16 ed è multicolore */
```

Figura 9 - Esempio: Mascherine bidimensionali.

re, più semplicemente, la macro
#define BNDRYON(r) {(r)->Flags | = AREAOUTLINE;}

Attenzione però: quest'ultima non è definita da nessuna parte! Dovete cioè definirla voi, tenendo presente che non è una macro ufficiale dell'Amiga e che non verifica se precedentemente era stato o meno definito un colore di contorno. A questo punto siamo pronti a... muovere i primi passi. E difatti la funzione successiva serve proprio a spostare la penna nel punto specificato dalle coordinate **x** e **y**. Tali coordinate sono relative ad un piano cartesiano con origine nell'angolo in alto a sinistra del *raster* su cui siamo operando, ascisse orizzontali positive verso destra ed ordinate verticali positive verso il basso. Nel nostro caso, lavorando con una finestra GZZ, esse corrispondono anche a quelle della finestra interna. Il nome di tale funzione, neanche a dirlo, è **Move().** Tale funzione, oltre a spostare la penna nella nuova posizione, aggiorna opportunamente le variabili **cp_x** e **cp_y** contenute nella struttura **RastPort**. All'inizio la penna è posizionata nell'origine.

Supponiamo adesso di voler tracciare una linea dalla posizione corrente ad un'altra. In questo caso useremo la funzione **Draw()**. Ovviamente, se invece volessimo tracciare tale segmento tra altri due punti qualsiasi, dovremmo prima muovere in uno dei due estremi, e poi utilizzare la **Draw()** per arrivare all'altro estremo. Naturalmente, nel caso fosse richiesto di tornare al punto di partenza, è necessario salvare da qualche parte le coordinate iniziali, come segue: ▼

```
                                /* * QUESTE LE HO DEFINITE PRIMA! * */
x_adesso = GetxPos(rp);         /*   GetxPos(r)   è  ((r)->cp_x)    */
y_adesso = GetyPos(rp);         /*   GetyPos(r)   è  ((r)->cp_y)    */
Move(rp, x_A, y_A);             /* Muovi all'estremo A del segmento */
Draw(rp, x_B, y_B);             /* Disegna ->estremo B del segmento */
Move(rp, x_adesso, y_adesso);   /* Torna al punto di partenza       */
```

In molti programmi grafici è necessario tuttavia colorare un singolo pixel e, a volte, rileggerne più tardi il colore. Questo è molto comune, ad esempio, nella generazione di frattali.

Per far questo ci tornano utili due funzioni: **WritePixel()** e **ReadPixel()**. La prima colora il pixel specificato con il colore primario, la seconda restituisce un valore da 0 a 255, oppure −1 nel caso non sia stato possibile leggere il pixel alle coordinate specificate. Una macro da me inventata, e che può tornare utile quando si devono colorare molti pixel in modo casuale, è la seguente: ▼

```
#define WriteDot(r,x,y,c)   {SetAPen(r,c);WritePixel(r,x,y);}
```

Note

1. *To jam* in inglese vuol dire *premere, comprimere. To jam a color* vuol dire quindi schiacciare, premere un colore sul piano di lavoro, un po' come un timbro imprime l'inchiostro sul foglio sul quale è premuto.
2. Il listato mostrato in figura è stato ottenuto a fronte di una finestra di tipo GZZ aperta sullo schermo del WorkBench. Dato che tali informazioni non sono documentate, almeno per quello che ne so io, è possibile che tali valori non siano applicabili in altri casi specifici. Se vi interessa, provate a lanciare lo stesso tipo di programma su vari tipi di finestre e schermi. I campi della struttura **RastPort** si possono ottenere dal *ROM Kernel Manual - Volume 2*, oppure dallo stesso listato presentato in questo articolo, aggiungendo **rp->** di fronte ad ogni campo, dove **rp = w->RPort** e **w** è il puntatore alla struttura **Window** che si ottiene all'apertura della finestra.

Un'altra funzione molto utile è **Poly Draw()**. Essa permette di tracciare una poligonale fornendo una tabella di coordinate ed il numero di punti che ne compongono vertici ed estremi (nel caso di poligonale aperta). Un esempio è riportato in figura 7.

Mascherine, aree e riempimenti

Abbiamo visto in precedenza, quando abbiamo parlato di modi grafici, che gli elementi base grafici (punti, linee) possono essere rappresentati da modelli binari chiamati mascherine o tassellature. In effetti questa rappresentazione può essere utilizzata per definire oggetti grafici ben più complessi. Prendiamo una linea, ad esempio, e supponiamo di dover disegnare la piantina di un appartamento.
Chiunque abbia fatto disegno tecnico sa perfettamente che a tale scopo si usano diversi tipi di tratteggi, a seconda del tipo di perimetro che si vuole riportare. Ecco allora tratti continui, tratteggio linea-linea e linea-punto, tratti a puntini, e via dicendo. In Amiga possiamo definire una mascherina che verrà utilizzata da **Draw()** per tracciare vari tipi di linee. Essa è formata da sedici bit (due byte) ed inizialmente assume il valore 0xFFFF, cioè, come è facile intuire, un tratto continuo. Se mettiamo alcuni bit a zero, il tratto risulterà differente, sempre ricordando però che deve essere tenuto in considerazione anche il modo grafico. Ecco allora che una maschera del tipo 0xE187 (cioè 11100001 10000111), rappresenterà una linea tratto lungo - tratto corto in JAM1, mentre in JAM2 verrà visualizzata come un tratto lungo ed un tratto corto nel colore primario, separati da tratti di lunghezza media in quello secondario. Un esempio di come utilizzare le mascherine di linea è riportato in figura 8. La funzione usata è **SetDrPt().**

Un discorso analogo può essere fatto per le mascherine a due dimensioni.

Una mascherina bidimensionale (o tassellatura) è larga come quella lineare 16 bit, ma può essere alta quanto vi pare, purché sempre di un numero di righe uguale ad una potenza di due (2, 4, 8 e così via). Ad esempio, la tassellatura in figura 9 è una scacchiera 16×16, cioè un vettore di sedici parole (2 byte). Per selezionarla, si usa un'altra macro C chiamata **SetAfPt()**. Ricordatevi che, oltre all'ormai consueto puntatore al *raster* e, ovviamente, a quello della mascherina che si vuole utilizzare (nel nostro caso **scacchiera**), bisogna fornire anche l'altezza della tassellatura *in potenze di due*. Fate attenzione quindi: nel nostro caso è 4 e non 16 (che rappresenta le reali dimensioni del vettore in questione).

La tassellatura così definita sarà utilizzata ogniqualvolta verrà chiamata una funzione di riempimento, come **RectFill().** Ovviamente, se il modo grafico selezionato è JAM1, la scacchera avrà i riquadri di tipo «**0**» di colore trasparente, mentre, nel caso sia stato specificato JAM2, i riquadri saranno resi nei due colori di penna, il primario e il secondario.
È anche possibile definire delle mascherine multicolore. La struttura della matrice che rappresenta tali mascherine ricorda quella che abbiamo già visto un po' di tempo fa per gli sprite. In pratica si tratta di disegnare una tasellatura per ognuno dei piani utilizzati. Se, ad esempio, lo schermo su cui stiamo operando non è quello del WorkBench ma uno schermo a 8 colori (cioè 3 piani), dovremo preparare tre mascherine bidimensionali, convertirle in una sequenza di parole e dichiararle nel nostro programma come un unico vettore (vedi ancora figura 9). A questo punto, la sequenza di chiamate è un po' differente, e precisamente:

- il colore primario va messo a 255, per indicare che si vuole disegnare su tutti i piani disponibili;
- quello secondario deve essere obbligatoriamente 0;
- il modo grafico deve essere posto a JAM2;
- la potenza di due che indica le di-

mensioni *di una singola immagine* (e non quindi del vettore finale risultante), deve essere *negativa*.

L'ultima funzione, per questa puntata, permette di disegnare dei rettangoli vuoti o pieni, con bordo o senza. L'abbiamo già nominata: **RectFill()**. I parametri da passare sono:

1. il puntatore alla struttura **RastPort**;
2. le coordinate (x,y) dell'angolo superiore sinistro del rettangolo da disegnare;
3. le coordinate (x,y) dell'angolo inferiore destro del rettangolo da disegnare.

Ovviamente le seconde devono essere più grandi delle prime.

E allora? — direte voi — Tutti quei discorsi sul pieno ed il vuoto, sul bordo, e via dicendo, come li specifichiamo?

Semplice: lo avete già fatto! **Rect Fill()**, come tutte le funzioni di disegno della libreria grafica, si comporta diversamente a seconda di come sono stati selezionati i modi grafici, i colori della penna primaria, secondaria e di contorno, ed eventuali mascherine lineari e/o bidimensionali.

Bene, per questa volta è tutto. Continueremo nelle prossime puntate con le funzioni poligonali, le tavolozze associate agli schermi, i caratteri ed i testi, e molto altro ancora.

L'esercizio

E rieccoci con l'esercizio. Pensavate di esservela cavata un'altra volta, non è vero? E invece niente: non ci sono scuse. Oramai ne sapete abbastanza su Intuition e sulle funzioni grafiche per affrontare un esercizio semplice semplice.

Vi ricordate del buon vecchio WorkBench 1.1? No? Ebbene, nel dischetto che veniva allora venduto insieme all'Amiga, c'era un cassetto contenente delle semplici *demo* grafiche. Erano delle finestrelle in cui venivano disegnate, ad una buona velocità, in una, linee, in un'altra, cerchi, e così via. Il tutto per dimostrare come Amiga fosse in grado di far girare più programmi contemporaneamente. Chi è partito subito con un A500 con 1.2, per esempio, potrebbe sorridere vedendo quei programmini in esecuzione, ma all'epoca (e già, oramai il nostro Amiga ha i suoi anni!) faceva una certa impressione a chi, come me, veniva da un ambiente MS-DOS monotasking...

Comunque, ricordi a parte, l'esercizio di questo mese consiste appunto nell'aprire una finestra (usate un GZZ se non ve la sentite di gestire il problema dei bordi da programma), e di disegnare, seguendo una sequenza casuale di coordinate e colori, una serie di linee che si incrociano a caso una sull'altra. Questa volta usate l'istruzione

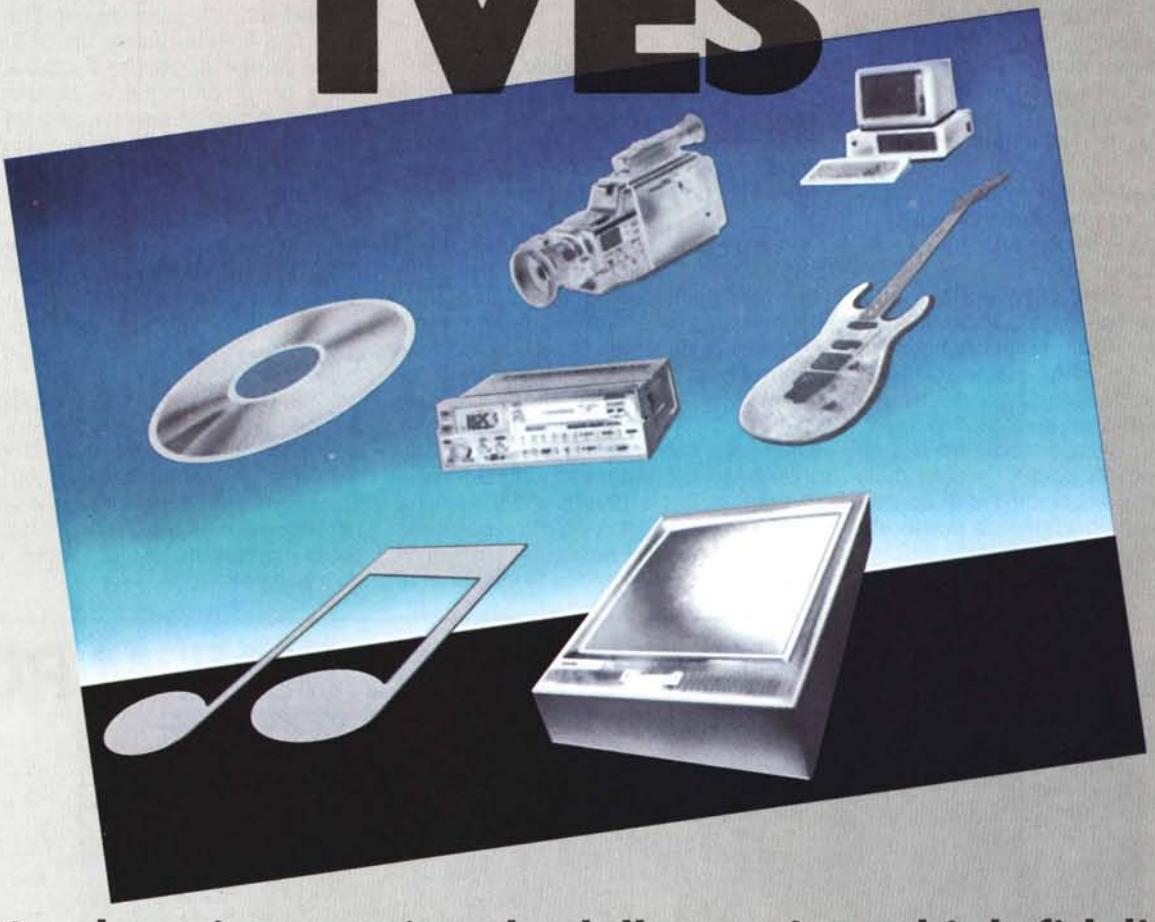
WaitPort(w->UserPort);

per intercettare il segnale di chiusura della finestra e terminare così il programma. Facile, no?

Conclusione

Nelle due ultime puntate abbiamo iniziato ad affrontare due argomenti estremamente interessanti: quello del programma di utilità *LMK* e quello della libreria grafica. Nella prossima puntata riprenderemo entrambi, in parallelo. Al prossimo numero di MC, quindi, e buone vacanze! MC

SIM-HI·FI IVES



23º salone internazionale della musica e high fidelity international video and consumer electronics show

14-18 settembre 1989
Fiera Milano

STRUMENTI MUSICALI,
ALTA FEDELTÀ, HOME VIDEO,
HI-FI CAR, CAR ALARM SYSTEMS,
PERSONAL COMPUTER, TV,
VIDEOREGISTRAZIONE,
ELETTRONICA DI CONSUMO.

SIM HI·FI IVES

un grande Sim!

HOME VIDEO

VIVA i giovani

Ingressi: Piazza Carlo Magno - Via Gattamelata - **Orario:** 9.00-18.00
Aperto al pubblico: 14•15•16•17 - **Giornata professionale:** lunedì 18 settembre

ASSOEXPO

ASSOCIATO ASSOMUSICA

FIERA MILANO

Segreteria Generale SIM-HI•FI-IVES: Via Domenichino, 11 - 20149 Milano - Tel.: 02-4815541 - Telex: 313627 - Fax 02-4980330