I rapporti tra font e QuickDraw

Diversi mesi fa abbandonai questa rubrica, un poco per motivi di spazio sulle pagine della rivista, un poco perché pensavo che, allo stato attuale dell'arte, con database capaci di creare applicazioni a dir poco inimmaginabili qualche anno fa, con fogli elettronici addirittura compilabili, con wp capaci di creare documenti autoeseguibili, ci fosse rimasto ben poco spazio per la nobile quanto pionieristica arte della programmazione

Raffaello De Masi

Una volta si era soliti dividere i linguaggi informatici in due grandi famiglie; ad alto e basso livello. Questi ultimi erano poi ridotti all'assembler o poco più mentre gli altri brillavano di luce propria per essere più o meno detentori pressoché esclusivi di questa o quella prerogativa programmatoria: ricursione, modularità, riutilizzabilità delle routine, facile leggibilità e trasportabilità, possesso di un numero maggiore o minore di keyword. Fu merito del «C», quando passò dalle alte sfere dei grandi sistemi al giornaliero mondo della microinformatica, fare piazza pulita di una gran quantità di retaggi e di caratteristiche pseudoesclusive, e molti dei linguaggi (ivi comprteso il Forth, mio primo amore informatico) che passarono come meteore nel firmamento informatico di qualche anno fa, oggi sono praticamente spariti o vivono in qualche oscura nicchia specialistica.

Oggi la situazione è notevolmente cambiata; la comparsa di package programmabili ha shiftato la situazione trasformando linguaggi di base come Pascal, Basic, Fortran in tool non certo utilizzabili a cuor leggero. Se teniamo conto che esistono pacchetti, come ProGraph, che permettono di disegnare, con tecniche simili a quelle di un package di grafica, praticamente tutto un programma, e di affidare a un tool il compito di sviluppare addirittura un sorgente ben strutturato e costruito; se teniamo conto che lo stesso permette di creare moduli, in perfetto stile OO, utilizzabili tal quali in altri ambienti di programmazione; se consideriamo, sempre per continuare nell'area dei tool di programmazione, che esiste un package che costruisce interfacce perfettamente integrate in ambiente Mac e restituisce un listato in Basic perfettamente leggibile e strutturato, o, volendo, crea automaticamente una risorsa immediatamente utilizzabile, senza perdere la testa ad imparare alcunché dell'ambiente e del funzionamento di Resource Maker; ebbene i linguaggi «puri», con le loro lunghe ore (se non giorni) di estenuante e faticoso debug, stanno divenendo sempre più di basso livello, per lasciare il loro posto a questi novelli tool multiuso, dalle caratteristiche così accattivanti e dalla potenza tanto impressionante.

Sebbene, quindi la rubrica mi fosse particolarmente cara, non fosse altro perché mi permetteva di parlare di tutto il sottobosco pressoché sconosciuto del sistema operativo Mac, mi era parso a un certo punto di essere una voce nel deserto e smisi, diversi mesi fa, appunto, di scrivere sull'argomento. Ma voce nel deserto non era, poiché nel frattempo ho ricevuto numerose richieste (sotto forma epistolare e come messaggi in MC-link) di recuperare la rubrica e i suoi contenuti. A farmi decidere di riprendere a scrivere di queste note fu un messaggio inviato alla redazione da un gruppo di studenti di ingegneria informatica di Padova (una lista di nomi che finisce con quello di Margherita Rigoni; perdonate ragazzi non posso nominarvi tutti) che mi chiedevano di riprendere a parlare di «arte» programmatoria su queste pagine. E così eccomi qua di nuovo, a parlare di listati, risorse, Toolbox, QuickDraw e così via.

Da che punto partire?

Il punto di partenza, proprio per non scoraggiare chi comincia adesso a vedere per la prima volta questa rubrica, sarà facile facile e parte da un'altra richiesta, che ho ricevuto per lettera proprio due settimane fa. Un lettore, tal Giussano della provincia di Novara, mi chiedeva di parlare di font e in particolare di come possono essere maneggiate con un linguaggio. La cosa mi era particolarmente cara e gradita non tanto per specificare come si fa a scegliere un carattere durante la redazione di un listato (dal più semplice Pascal o Basic al più forzuto «C»), si tratta in fondo di eseguire una chiamata a una routine di Toolbox, come ad esempio: ▼

```
_carattere = 4 ' carattere Monaco
_grandezza = 9
_caratteristiche_tipografiche = 2 ' corsivo
_bit_copy = 1
'
CALL TEXFONT _carattere
CALL TEXSIZE _grandezza
CALL TEXFACE _caratteristiche_tipografiche
CALL TEXTMODE _bit_copy
```

o, ancora più semplicemente: ▼

```
TEXT _carattere, _grandezza, _caratteristiche_tipografiche,
_bit_copy
```

Figura a - Le diverse rappresentazioni del glifo A.

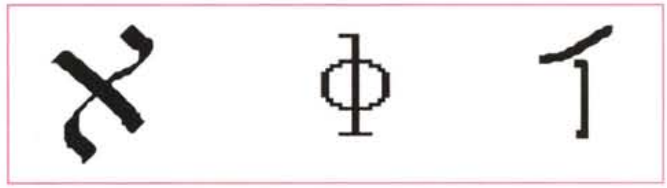

Figura b - L'ALEPH ebraico, la A cirillica e l'ALIF arabo.

quanto per parlare di quello che c'è alle spalle di tutto questo lavorio, e per chiarire, alla fine, le cose, effettivamente, come stanno.

Sebbene qualunque programmatore si affidi, durante la stesura bruta di un programma, al font di default (Geneva o Monaco, nella maggioranza dei casi, se non si è giocherellato con ResEdit o con una delle innumerevoli utility che permettono di modificare i default di sistema), le successive operazioni di finitura passano prima o poi attraverso la modifica dei caratteri dei successivi step di programma, sia a video che in stampa. Ad esempio, uno splash screen rende certamente molto meglio se costruito con più font.

In ambiente Mac i caratteri sono maneggiati dal Font Manager, ma il lavoro bruto e finale di tracciare gli stessi sullo schermo è affidato a QuickDraw. E la cosa si complica se si tiene conto che spesso, nella nostra applicazione, i font saranno maneggiati da un menu. Questo ci porta inevitabilmente a parlare di QuickDraw e MenuManager, tanto per complicare un poco le cose.

Il carattere dei caratteri

Quando un gruppo di persone decide concordemente di usare un simbolo per rappresentare qualcosa che è comunemente noto, il simbolo diviene un «glifo». Ad esempio, un segnale di divieto di transito o quello di allacciare le cinture è un glifo. Per forza di cose (e di definizione) anche le lettere dell'alfabeto sono glifi.

Perché le stesse persone che obbediscono alla convenzione appena detta usino lo stesso metro di interpretazione, è necessario, ovviamente, che gli stessi glifi siano distinguibili tra loro. Così, perché si possa distinguere una A da una C è necessario che la prima e la seconda lettera abbiano caratteristiche uniche e inequivocabili, che permettano a chiunque aderisca alla convenzione (dell'alfabeto latino) di stabilire che la «A» sia proprio A, la «F» sia proprio F e così via. Devono poi esserci alcune convenzioni e regole ben chiare che permettano di distinguere la «A» anche se questa è scritta in diverso modo, come nella figura a).

La stessa cosa vale, ovviamente, per altri alfabeti, come il cirillico, l'arabo o l'ebraico (in questi ultimi due casi, fig. b cambia anche il nome, rispettivamente 'alif' e 'aleph').

Per ritornare all'argomento dell'articolo, dunque, se un insieme di glifi può essere raggruppato in una «categoria» omogenea, essi rappresentano un set di caratteri, per dirla in gergo informatico, un 'typeface'. Così divengono un typeface l'insieme dei segnali stradali, dei simboli matematici, dei caratteri alfabetici o delle «casette-alberghi» della guida Michelin.

Ogni «collezione», chiamiamola così, può poi essere diversificata in alcune sue caratteristiche non pregiudiziali per la comprensione. Nel caso dei caratteri di stampa del Mac avremo così il Times, l'AvantGarde, il Chicago, e così via, che si diversificheranno per grandezza, contorno, spessore.

La grandezza è misurata in «punti» (molto simili, dimensionalmente, al punto tipografico), pari a 1/72 di pollice, e la cosa è molto utile, nel nostro caso, in quanto lo schermo del Mac ha, appunto, una risoluzione proprio di 72 punti per pollice.

I font sono conservati in file di risorsa del tipo «FONT», e possono essere indirizzati in base al loro nome o a un numero unico di identificazione. Ecco quindi giustificate le due righe di programma precedente (il numero identificativo di Monaco è 4)

Scegliamo il nostro carattere da sottoporre ad analisi; potrebbe essere ad esempio, l'Helvetica (#21). Tanto per amore di precisione potremmo affermare che la precedente chiamata:

```
CALL TEXFONT _carattere 'dove _carattere
= 21
```

Chicago grassetto 12 pt.
Courier corsivo 17 pt.
Helvetica ombreggiato corsivo 24 pt.
Monaco sottolineato 18 pt.

Figura c - Diversi tipi di font, a differente grandezza e caratteristiche.

valore	codice
0	piano
1	**grassetto**
2	*corsivo*
4	sottolineato
8	contornato
16	ombreggiato
32	condensato
64	esteso

Figura d.

può essere convertita in

```
CALL TEXFONT 21
```

ottenendo lo stesso risultato.

TEXTFACE, la seconda chiamata, cambia l'impronta, se così si può dire, della stampa. Il sistema operativo usa una semplice convenzione per individuare l'aspetto del carattere stesso. In figura d vediamo la corrispondenza tra valori e forma, ma è possibile anche combinare tra di loro i parametri semplicemente addizionandoli. Così

```
CALL TEXTFACE 15
```

scriverà un carattere in neretto, corsivo, sottolineato e contornato. TEXTSIZE, d'altro canto permetterà di settare, come era intuitivo, la dimensione dei caratteri (es TEXTSIZE 36).

La scrittura dei font

Escludendo dal discorso alcuni particolari che qui risulterebbero sovrabbondanti (quali la differenza tra caratteri monospaziati e proporzionali, come pure il comando TEXTMODE) QuickDraw traccerà il font, sullo schermo o sulla stampante, da una locazione di penna che è chiamata origine del glifo o, in gergo tipografico, linea di base (baseline). È possibile usare un'ulteriore chiamata al Toolbox (MOVETO) per posizionare la penna in un punto qualsiasi dello schermo (e in teoria anche fuori), al momento di inizio della scrittura. Così in:

```
CALL MOVETO 100,150
```

150 diverrà la linea di base da cui partirà la scrittura del carattere. Ma la cosa non è certo finita qui: le caratteristiche del font che QuickDraw usa per «scrivere» è custodita, nella stessa risorsa del carattere, in una struttura di record chiamata FontInfo: si tratta di quattro valori accessori che permettono di identificare il «field» in cui viene disegnato ogni carattere del font. Un'opportuna chiamata al Toolbox consente di ricavare questi valori; essi vengono depositati in un vettore grande quattro posti. La cosa diviene più chiara leggendo queste righe:

```
DIM _informazioni_font(3) ' quattro posti con option base 0
CALL GetFontInfo _informazioni_font(0)
'
' la chiamata a GetFontInfo deposita i quattro valori
caratteristici nel vettore
'
PRINT " ascendente = "; _informazioni_font(0)
PRINT " discendente = "; _informazioni_font(1)
PRINT " larghezza = "; _informazioni_font(2)
PRINT " interlinea = "; _informazioni_font(3)
```

Figura f - Un font bitmapped (a destra) a confronto con uno TrueType.

Il significato dei valori è facilmente deducibile dalla figura allegata; diremo solo che la «larghezza» rappresenta il massimo spazio, in punti, di cui la penna si muove nel tracciare i caratteri (spazio che varia nei font proporzionali, mentre è fisso in quelli, appunto, monospaziati), l'ascendente e il discendente rappresentano la posizione delle linee immaginarie che toccano rispettivamente il «tetto» della più alta maiuscola e il «piede» dei caratteri discendenti, come la [g] o la [q], e, infine, l'interlinea è, ancora, il numero dei punti che passano tra il discendente di una riga e l'ascendente della riga successiva. Così, la quantità di spazio di cui si muove la penna per passare alla riga successiva è rappresentato dalla somma dell'ascendente, del discendente e dell'interlinea.

In altri termini, quando, ad esempio, anche nel più banale dei wp, come TeachText, diamo un carattere di [Return], FontManager, attraverso QuickDraw esegue due comandi del tipo:

```
_altezza_della_riga = _informazioni_font(0) +
_informazioni_font(1) + _informazioni_font(3)
CALL MOVETO (x, y + _altezza_della_riga
```

Il lavoro del FontManager non si ferma qui; esso utilizza, contemporaneamente, un'altra routine, StringWidth, che tiene una ordinata conta della lunghezza totale della stringa battuta

Figura e - I parametri in gioco nella definizione dei caratteri e delle linee di scrittura.

(escluso il CR) addizionando la larghezza dei caratteri presenti nella stringa stessa (larghezza caratteristica presente nella funzione CharWidth). Ad esempio, i wp, quando si utilizza una scrittura centrata, adottano una routine del tipo: ▼

```
_larghezza_finestra = WINDOW(2) ' calcola la larghezza della
finestra corrente
_y = 150
_stringa$ = "MCMicrocomputer"
_lunghezza_stringa = WIDTH(_stringa$)
CALL MOVETO ((_larghezza_finestra - _lunghezza_stringa)/2, _y)
PRINT _stringa$
```

Caratteri, dall'antico al moderno

Fino all'avvento del System 6.07 tutti i font sullo schermo erano di tipo bitmap, vale a dire erano rappresentati da punti legati tra di loro a formare un carattere. Il disegnatore di font era chiamato sempre a disegnare una mappa, appunto, di punti per ogni glifo nella misura che gli era necessaria. In questo modo un Courier 12 punti piano e uno corsivo (come anche grassetto, ombreggiato e così via) dovevano essere disegnati separatamente. Alla fine ogni glifo era rappresentato da un disegno, punto per punto, e la cosa era ben evidente usando lo zoom di pacchetti come MacPaint.

Questo approccio presentava vantaggi e svantaggi; i primi erano essenzialmente rappresentati da una facile operatività del disegnatore (non a caso i font bitmap erano estremamente poco costosi e talora freeware), uniti a una certa maggiore velocità di rappresentazione sullo schermo. Gli svantaggi erano invece numerosi, specie quando occorreva visualizzare una grandezza non posseduta dal carattere. QuickDraw era chiamato a un lavorio, in questo caso, enorme: cercava, per primo, se esisteva un carattere di grandezza doppia di quello richiesto e lo riduceva (era la tecnica utilizzata anche dalle stampanti laser sotto QD); in caso negativo cercava un carattere pari a metà, e lo ingrandiva. Se nemmeno questo avveniva sceglieva il carattere della massima grandezza disponibile e lo riduceva. Se neppure questa strada era disponibile (ad esempio un documento redatto con un font poi non disponibile su un'altra macchina), andava a cercare un eventuale font presente solo nell'applicazione e, infine, quale ultima ratio, utilizzava il font di Sistema. Generalmente l'installazione delle grandezze 9, 10 e 12 permetteva di lavorare con una certa tranquillità (i caratteri 5, 20, 6 e 24 si presentavano con caratteristiche decenti), ma appena si passava a misure superiori l'odiato fenomeno della seghettatura dei bordi aveva la meglio.

Dopo l'avvento del System 6.07 è esploso il fenomeno dell'outline anche per i font da schermo. In base a un principio abbastanza simile a quello già adottato con successo nella gestione dei font PostScript per stampante laser, il simbolo non fu più descritto in termini di punti, ma come una funzione matematica basata sulle curve parametriche di Bezier (chissà che una delle prossime volte non si parli un poco più a fondo di questo affascinate mondo). Apple, proprietaria della tecnologia, introdusse immediatamente questo tipo di rappresentazione e lo adattò anche ai font che forniva di corredo al sistema operativo (anche se, per ovvi motivi di compatibilità continuò a fornire ancora i bitmap). I font si chiamarono TrueType, in ossequio al fatto che obbedivano a una vera descrizione del contorno del glifo per tutte le misure desiderate. Questo approccio superò di fatto tutte le barriere legate alla grandezza dei caratteri, abbattendo il fatidico muro dei 127 punti massimi (TrueType può maneggiare grandezze fino a 32767, con tutte, dico tutte le grandezze intermedie).

TrueType superava, inoltre, un subdolo problema legato ai caratteri in bitmap. Esisteva infatti una limitazione nell'algoritmo di QuickDraw destinato a maneggiare la scalatura, per consentire la visualizzazione su schermi in b/n; quando un'immagine era ridotta di quattro volte QuickDraw, se anche uno solo dei quattro pixel era nero, lasciava il pixel di risultato nero (altrimenti il blocchetto sarebbe scomparso dallo schermo). Anche nel caso del miglior risultato, il carattere appariva frastagliato, troppo scuro o macchiato, e, in ogni caso, di cattiva qualità.

Con le font di tipo outline il problema d'incanto spariva, in quanto i punti erano legati solo alla funzione matematica che li creava, e non a una rappresentazione precedente. Per dirla in altri termini, quando si scalava, in basso o in alto, un carattere, venivano mutati i parametri in gioco nella formula rappresentativa, veniva tracciato un contorno immaginario del risultato, e solo allora il bitmap veniva chiamato in causa, per «riempire» l'outline.

Allora scriviamo due righe di programma del tipo: ▼

```
_font = XXXXX ' sostituire XXXXX con l'ID number del font che si
desidera analizzare
FOR _j = 1 to 200
  _grandezza = _j
  _vero = 0
  CALL RealFont _font, _grandezza, _vero
  IF _vero THEN PRINT "La misura "; _grandezza; " e' disponibile"
NEXT _j
```

e facciamo girare il programma sostituendo a XXXXX un numero di identificazione di un font bitmapped e TrueType. Sarà possibile scoprire quali grandezze sono effettivamente presenti, per il primo tipo, ma è meraviglioso constatare come facendo la stessa operazione per TT la risposta, addirittura elevando il loop fino a 32767 sia sempre affermativa. E voglio darvi un'occasione per giocherellare; provate a lavorare con numeri negativi!

Concludendo

Penso che per questa puntata possa bastare. A parte la digressione sui font, che speriamo almeno sia stata piacevole, abbiamo solo voluto far notare che un'approfondita conoscenza delle infinite chiamate al Toolbox permetterebbe a qualunque implementatore di linguaggi di realizzare tool semplici e complessi a proprio piacimento. Creare una funzione del «C», una procedura in Pascal o una funzione locale degli ultimi potentissimi Basic, tagliata ad hoc per le nostre esigenze, diviene, con un poco di allenamento (ma neanche tanto) un giochetto da ragazzi. Da qui a passare alle librerie personali (a patto che siano ben documentate) alle quali attingere in caso di bisogno il passo è breve. E se è vero, come è vero, che oggi una buona biblioteca di routine ben collaudate vale almeno quanto un blocco di keyword implementate nel linguaggio stesso, studiarsi a fondo il Toolbox del Mac è sicuramente un investimento che ripaga nel tempo. A risentirci. MC