# DirectDraw. Introduzione per i neofiti

## *1 Introduzione*

DirectDraw è una sottolibreria delle librerie multi-purpose della Microsoft. Per poter accedere alle funzioni in essa contenuta si può agire in due modi: linkare una libreria che viene fornita insieme al SDK più recente o attraverso le API di windows con i metodi per accedere a oggetti COM.
Preferisco scegliere la seconda opzione perché, in primo luogo, permette di caricare la libreria in fase di Run-Time e non in Load-Time, fornendo all'utente un migliore controllo per gli errori, e in seconda perché è System-Indipendent tra Windows9x e WindowsNT.
Gli oggetti COM sono una nuova forma delle API di windows per accedere a funzioni di sistema senza dover ogni volta chiamare una funzione di libreria, ma accedendo a queste funzioni come classi di C++ (sono dei puntatori a classi di funzioni solo **virtual**, dunque dal C si può accedere ai membri virtual di una classe attraverso un puntatore contenuto nella struttura stessa).
Così la classe `IDirectDraw` sarà sempre usata in forma di un suo puntatore: `LPDIRECTDRAW`, e `IDirectDrawSurface` in `LPDIRECTDRAWSURFACE`.
Ogni funzione COM restituisce un `HRESULT`, un valore non segnato, che se vale zero vuol dire che la funzione è andata a buon termine, altrimenti rappresenta un codice di errore. Ho segnalato con una variabile `HRESULT hr` la lettura di questo valore.
Ogni interfaccia COM (`IDirectDraw`, `IDirectDrawSurface`, `IDirectDrawClipper`… etc) contiene 3 metodi sempre (ereditarietà dalla classe `IUnknown`). Il più importante è la funzione **`Release`** che libera l'oggetto e la memoria da esso occupata. Al momento della chiusura del programma bisognerà rilasciare tutte le superfici, i clipper e le istanze degli oggetti IDirectDraw in questa maniera.
La seconda funzione è **`AddRef`** che aumenta il contatore interno alla classe. **`Release`** decrementa questo contatore e solo quando giunge a zero viene rilasciata effettivamente la classe.
La terza funzione è **`QueryInterface`** che permette di ottenere da un oggetto un altro oggetto suo parente (tipo da IDirectDraw si può richiedere l'oggetto IDirectDraw3, una versione più recente dell'interfaccia).

Per poter ottenere il meglio da questo documento è necessaria una conoscenza base del sistema Windows e della creazione di finestre e la gestione dei messaggi, in quanto data per scontata.

## *2 Accedere all'Interfaccia*

Per poter usare gli oggetti DirectDraw prima di tutto è necessario includere la libreria *ddraw.h*.
Per poter accedere agli oggetti COM da un programma invece l'header è compreso in *windows.h* (se cioè non fosse includere *objbase.h*).
Per prima cosa chiamare la funzione per inizializzare le funzioni COM:

```
hr = CoInitialize(NULL);
```

Alla fine del programma poi sarà necessario liberare la memoria allocata con **`CoUninitialize`**.
Creata una variabile per ospitare la classe (una variabile del tipo `LPDIRECTDRAW`) è ora possibile inizializzarla con la funzione **`CoCreateInstance`**:

```
hr = CoCreateInstance(CLSID_DirectDraw, NULL, CLSCTX_ALL, IID_IDirectDraw, (void
**) &lpDD1);
```

dove *lpDD1* è l'oggetto `LPDIRECTDRAW`. Le GUID per specificare l'oggetto COM (CLSID_DirectDraw, IID_IDirectDraw) e la definizione propria delle classi (IDirectDraw, LPDIRECTDRAW) sono contenute nell'header *ddraw.h* da includere. E' necessario o linkare all'eseguibile la libreria *dxguid.lib*, o altrimenti definire INITGUID prima di includere l'header *ddraw.h*.
Inizializzare l'oggetto:

```
hr = IDirectDraw_Initialize(lpDD1, NULL);
```

IDirectDraw_Initialize è una macro. Si può anche usare:

```
hr = lpDD1->Initialize(NULL);
```

Tutte le funzioni delle DirectDraw hanno una macro corrispondente, per permettere di eseguire la medesima istruzione in un file C (che non supporta gli oggetti) o in un file C++. Nel seguito userò indistintamente le due forme di accesso alle istruzioni membro.
A questo punto è necessario avere una finestra di Windows a cui appoggiarsi per i messaggi. Avendo la finestra si può selezionarla nell'oggetto `IDirectDraw`:

```
hr = lpDD1->SetCooperativeLevel(hWnd, dwFlags );
```

*dwFlags* può essere una o più di delle macro, comunque consiglio per applicazioni a schermo intero la combinazione *DDSCL_EXCLUSIVE* | *DDSCL_FULLSCREEN*| *DDSCL_ALLOWREBOOT*, altrimenti per applicazioni in finestra *DDSCL_NORMAL*.
Una applicazione DirectDraw a schermo intero permette di selezionare il modo video e i bit di colore, se non di permettere il Page Flipping. Il modo a finestra non permette ovviamente tutto ciò.
Se si è scelta un'applicazione a schermo intero è possibile ora scegliere il modo video:

```
hr = lpDD1->SetDisplayMode( width, height, bitperpixel);
```

Mettere i parametri giusti. Ovviamente non tutte le combinazioni saranno possibili, e dipende molto anche dal drive video. Per avere un elenco dei modi video si può usare il metodo **`EnumDisplayModes`**, vedere la documentazione per informazioni.

## *3 Le superfici.*

Le superfici sono una forma di astrazione della memoria video. Sono oggetti anche essi, e servono in primo luogo a sfruttare l'hardware video per operazioni di blitting o altri effetti speciali che l'hardware specifico consente. Alcuni effetti possono essere eseguiti solo se la memoria video associata all'oggetto risieda nella memoria video della scheda o nella memoria di sistema.
Nonostante superfici come la primaria (quella proiettata sullo schermo) siano già create fisicamente, è necessaria lo stesso una procedura per ottenere un puntatore ad un oggetto che permetta la manipolazione della suddetta. Le superfici create od ottenute avranno lo stesso bit depth della superficie primaria, e la loro dimensione massima è limitata dalle stesse dimensioni della superficie primaria (dall'Interfaccia IDirectDraw2 in poi questo limite non sussiste più).
Per ottenere una superficie basta riempire in modo opportuno una struttura `DDSURFACEDESC` e darla in pasto alla funzione **`CreateSurface`**.
Per esempio, per ottenere la superficie primaria (indispensabile per lavorare in ogni caso) bisogna fare:

```
DDSURFACEDESC ddsd;
ZeroMemory( &ddsd, sizeof(DDSURFACEDESC) );
ddsd.dwSize = sizeof(DDSURFACEDESC);

ddsd.dwFlags = DDSD_CAPS;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;
```

Se si è a schermo intero e  si vuole avere una superficie con delle superfici secondarie su cui fare il PageFlipping bisogna fare invece queste righe aggiuntive:

```
ddsd.dwFlags = DDSD_BACKBUFFERCOUNT | DDSD_CAPS;
ddsd.ddsCaps.dwCaps = DDSCAPS_COMPLEX | DDSCAPS_FLIP | DDSCAPS_PRIMARYSURFACE;
ddsd.dwBackBufferCount = Quante superfici servono, solitamente 1, al massimo 2.
```

E infine:

```
hr = lpDD1->CreateSurface( &ddsd, &lpddsPrimary, NULL );
```

Dove *lpddsPrimary* è un oggetto della classe LPDIRECTDRAWSURFACE. Ogni superficie possiede anch'essa dei propri metodi. Uno è quello per ottenere le superfici attaccate, come degli Z-Buffer o le BackBuffer Surface (quelle su cui disegnare per fare il page-flipping per intenderci):

```
DDSCAPS ddscaps;
ZeroMemory(&ddscaps, sizeof(ddscaps));
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
hr = lpddsPrimary->GetAttachedSurface(&ddscaps, &lpddsBack);
```

Dove *lpddsBack* è la prima delle superfici attaccate (sempre del tipo LPDIRECTDRAWSURFACE).

Per lavorare è necessario spesso creare superfici partendo da zero. Le superifici secondarie vengono create nel modo seguente:

```
LPDIRECTDRAWSURFACE lpDDS;
DDSURFACEDESC ddsd;
HRESULT hr;

ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd);
ddsd.dwWidth = width;
ddsd.dwHeight = height;

ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH;

ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
```

Di default le superfici vengono create nella memoria video se questa è disponibile, altrimenti nella memoria di sistema. Per motivi di performance (operazioni massicce sull'immagine) è possibile richiedere memoria di sistema con la flag DDSCAPS_SYSTEMMEMORY:

```
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN | DDSCAPS_SYSTEMMEMORY;

hr = IDirectDraw_CreateSurface(lpDD1, &ddsd, &lpDDS, NULL);
```

A questo punto lpDDS è un nuovo oggetto IDirectDrawSurface correttamente inizializzato.
Le dimensioni della superficie solo limitate dalla superficie primaria e non è possibile creare una superficie con bitdepth diverso dalla superficie primaria.

## *4 Operazioni base sulle Superfici*

Tra superfici è possibile copiare (in modo hardware) i dati attraverso due funzioni:
**Blt** e **BltFast**, la seconda è più veloce solo in caso di copia tra superfici senza alcun supporto hardware. La prima invece permette di avere molteplici effetti di copia, tra cui rotazioni, scala, e altri…

```
lpDDSDestinazione->BltFast( x, y, lpDDSSorgente, NULL, DDBLTFAST_WAIT);
```

Al posto del NULL è possibile inserire un puntatore a una struttura RECT rappresentante la porzione della superfice sorgente da copiare.
La funzione non viene eseguita se la superfice sorgente esce da uno qualsiasi dei bordi della superficie destinazione.

```
lpDDSDestinazione->Blt(lpDestRect, lpDDSrcSurface,  lpSrcRect, dwFlags,
lpDDBltFx);
```

**Blt** è la forma più generale per eseguire la copia tra superfici. E' possibile specificare un rettangolo di Destinazione (o NULL se si vuole considerare tutta la superfice) e un rettangolo di Sorgente (o NULL se si vuole come sempre considerare la superfice). Se le due dimensioni non corrispondono verrà eseguito un ridimensionamento dell'immagine.
*dwFlags* normalmente deve essere posto a DDBLT_WAIT e *lpDDBltFx* a NULL.

*lpDDBltFx* è un puntatore a una struttura DDBLTFX che contiene effetti sulla copia. Queste opzioni vengono poi abilitate con una particolare flag in *dwFlags*. Per esempio la flag DDBLT_COLORFILL che abilita il membro dwFillColor in DDBLTFX permette di riempire la superficie di destinazione con il colore specificato in dwFillColor (in questo caso per esempio *lpDDSrcSurface* e *lpSrcRect* verranno ignorati).
Se l'hardware lo permette è possibile anche eseguire una copia asincrona attraverso la flag DDBLT_ASYNC. Guardare la documentazione per la spiegazione di tutte le flag e gli effetti. Attenzione che molti degli effetti segnalati, anche se inseriti nella documentazione possono non venire supportati dal proprio hardware, e molti non sono supportanti su nessun hardware esistente (e probabilmente non lo saranno mai).

## 5 Disegno diretto sulle Superfici

Per disegnare su superfici si può operare in due modi distinti: accedere alla memoria direttamente o usare le GDI.
• Nel primo modo si ottiene un puntatore alla memoria e le informazioni necessarie al disegno:

```
DDSURFACEDESC ddsd;
ZeroMemory(lpddsd, sizeof(DDSURFACEDESC));
ddsd.dwSize = sizeof(DDSURFACEDESC);

hr = IDirectDrawSurface_Lock(lpDDS, NULL, &ddsd, DDLOCK_SURFACEMEMORYPTR |
DDLOCK_WAIT, NULL);
```

*lpDDS* è sempre un puntatore a una IDirectDrawSurface. A questo punto *ddsd* contiene le dimensioni della superfice, il pitch (ddsd.*lPitch*) ovvero l'occupazione fisica di una linea, e un puntatore a void (ddsd.*lpSurface*) all'inizio della memoria.
Per avanzare alla riga *y* basta aggiungere a questo puntatore il valore *y* * *ddsd.lPitch*. Non dare mai per scontato che il Pitch sia ben diverso da quello ottenuta da [BytePerPixel * Numero di Pixel per riga]: l'hardware per motivi di ottimizzazione potrebbe modificare questo valore.
Alla fine del disegno è necessario rilasciare il lock sulla surface:

```
IDirectDrawSurface_Unlock(lpDDS, NULL);
```

• Usando le GDI si usa un metodo nettamente più lento, ma si possono usare le funzioni di disegno interne di Windows (tra cui infatti i Fonts non supportati in altro modo dal DirectDraw). In questo modo si ottiene un *Device Contest* dove è possibile lavorare:

```
HDC hdc;
hr = IDirectDrawSurface_GetDC( lpddsSurface, &hdc);
```

Vedere la documentazione di Windows per conoscere le funzioni di disegno. Internamente GetDC chiama Lock, dunque alla fine del disegno, per rilasciare il *DC* e liberare il lock:

```
IDirectDrawSurface_ReleaseDC( lpddSurface, hdc);
```

## 6 Interfacce diverse

Esistono al momento attuale (fine 2001) 4 versioni dell'oggetto IDirectDraw e 5 delle IDirectDrawSurface. Per accedere a queste interfacce è necessario solo possedere l'header del *ddraw.h* più recente. Ogni versione recente delle interfacce grosso modo mantiene la compatibilità verso il basso, permettendo (se il programma è costruito in un certo modo) poche modifiche al passaggio di un'interfaccia nuova.
E' possibile creare un interfaccia partendo da zero (nel caso delle IDirectDraw) usando **CoCreateInstance**, e da queste ottenere le corrispondenti IDirectDrawSurface usando man mano usando **CreateSurface**. Altrimenti è possibile partendo da un oggetto, attraverso la funzione membro **QueryInterface** un'altra versione del medesimo oggetto. Per esempio ottenere la IDirectDraw7 partendo da IDirectDraw o normalmente per ottenere la IDirectDrawSurface3 o IDirectDrawSurface4 partendo da una versione precedente (visto che la creazione di superfici dal

DirectX3 al DirectX7 era limitata alla IDirectDraw2) o infine creare una IDirectDrawSurface (qualsiasi versione) partendo da una qualsiasi versione del IDirectDraw.
La sintassi è semplice:
*OggettoBase* -> QueryInterface ( *GUID del nuovo oggetto*, *puntatore al puntatore dell'oggetto*).

Esempio1
```
LPDIRECTDRAW2 lpDD2;
lpDD1->QueryInterface(IID_IDirectDraw2, (LPVOID *)&lpDD2);
```

Esempio2
```
LPDIRECTDRAWSURFACE2 lpdds2;
DDSURFACEDESC ddsd;
...
hr = lpDD1->QueryInterface(IID_IDirectDrawSurface2, (LPVOID *)&lpdds2);
hr = lpdds2->Initialize(lpDD1, &ddsd);
```

## *Appendice A. Effetti*

• E' possibile specificare un colore chiave (reso trasparente) sulle superfici. In questo modo il blitting copierà sulla destinazione solo i byte diversi da questo colore[1].

```
DDCOLORKEY ddck;
ddck.dwColorSpaceHighValue = ddck.dwColorSpaceLowValue = /colore/;
lpdds->SetColorKey(DDCKEY_SRCBLT , &ddck);
```

*lpdds* è ovviamente un puntatore a una IDirectDrawSurface, mentre al posto di *colore* va messo il codice del colore da filtrare. Non lasciatevi ingannare dai due valori per il colore a modo di range. Nessun hardware finora li supporta e devono essere posti uguali. Il valore di colore deve essere nel formato colore della superficie. Al momento del Blitting è necessario in ogni caso specificare che la superficie sorgente abbia un colore chiave:

```
lpDDSDestinazione->BltFast(x, y, lpDDSSorgente, NULL, DDBLTFAST_WAIT |
DDBLTFAST_SRCCOLORKEY);
```

• E' possibile fare il page-flipping tra superfici attraverso la funzione **Flip**:

```
hr = IDirectDrawSurface_Flip( lpddsPrimary, NULL, DDFLIP_WAIT);
```

Il Flipping è possibile solo sulle superfici (anche non primarie) a cui in fase di creazione però siano stati assegnati dei BackBuffer.

## *Appendice B. I Clipper*

Nel modo in finestra è necessario avere accorgimenti particolari per evitare di cancellare zone di schermo coperte da finestre non nostre. Prima di tutto bisogna sempre evitare di scrivere direttamente sulla superficie primaria, ma sempre in una secondaria, delle dimensioni della finestra e fare un blitting (attenzione che con i clipper, BltFast non funziona).
La creazione del Clipper è la seguente:

```
LPDIRECTDRAWCLIPPER lpClipper;
hr = lpDD1->CreateClipper(0, &lpClipper, NULL);
hr = lpClipper->SetHWnd(0, hWnd);
hr = lpddsPrimary->SetClipper(lpClipper);
```

A questo punto ogni Blitting fatto con **Blt** sulla superficie primaria verrà compiuto in modo tale da non sovrascrivere finestre altrui.

[1] E' possibile fare altri tipi di copia con trasparenza, ma questa è quella più comunemente usata.

## *Appendice C. Le Palette*

Se il modo video lo consente (8 bit per pixel) è possibile assegnare alla superficie primaria (e solo a quella) una palette di colori. Il procedimento è il seguente:

```
LPDIRECTDRAWPALETTE ddpal;
if(lpDDS->GetPalette(&ddpal)!=0)
      {
      lpDD1->CreatePalette(DDPCAPS_8BIT | DDPCAPS_ALLOW256 | DDPCAPS_INITIALIZE,
lpPalette, &ddpal, NULL);
      lpDDS->SetPalette(ddpal);
      }
      else
      {
      ddpal->SetEntries(0, 0, 256, lpPalette);
      }

ddpal->Release();
```

E' necessario liberare subito l'oggetto IDirectDrawPalette visto che quando si esegue il GetPalette o il SetPalette il contatore interno viene aumentato automaticamente.
*lpPalette* è un array di **PALETTEENTRY**, dove ogni indice contiene al suo interno le componenti rosso, verde e blu:

```
typedef struct tagPALETTEENTRY { // pe
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

Il dato in *peFlags* viene ignorato.

## *Appendice D. DirectX8*

La nuova versione del DirectX, l'ottava, ha abbandonato il supporto della grafica 2d pura, ma che con la versione 7 è sufficientemente implementato. Non sono state prodotte le interfacce IDirectDraw8 e IDirectDrawSurface8, a favore di altre interfacce orientate esclusivamente alla grafica 3D.

## *Appendice E. Riferimento*

| Oggetto | Puntatore | Commento | Funzioni membro Importanti |
|---|---|---|---|
| IDirectDraw | LPDIRECTDRAW | Interfaccia base (presente da Windows95 in poi) del DirectDraw. | Release<br>CreateClipper<br>CreateSurface<br>CreatePalette<br>EnumDisplayModes<br>GetAvaiableTotalMem<br>GetCaps<br>Initialize<br>SetCooperativeLevel<br>SetDisplayMode |
| IDirectDraw2 | LPDIRECTDRAW2 | Interfaccia disponibile dal DirectX3 in poi. | GetAvailableVidMem |
| IDirectDraw4 | LPDIRECTDRAW4 | Interfaccia disponibile dal DirectX6 in poi | RestoreAllSurfaces |
| IDirectDraw7 | LPDIRECTDRAW7 | Interfaccia disponibile dal DirectX7 in poi. | |
| IDirectDrawSurface | LPDIRECTDRAWSURFACE | Interfaccia base per manipolare superfici. | Release<br>Blt<br>BltFast<br>Flip<br>GetAttachedSurface<br>GetCaps<br>GetDC<br>IsLost<br>Lock<br>ReleaseDC<br>Restore<br>Unlock |
| IDirectDrawSurface2 | LPDIRECTDRAWSURFACE2 | | PageLock<br>PageUnlock |
| IDirectDrawSurface3 | LPDIRECTDRAWSURFACE3 | Disponibile dal DirectX5 in poi | |
| IDirectDrawSurface4 | LPDIRECTDRAWSURFACE4 | | |
| IDirectDrawSurface7 | LPDIRECTDRAWSURFACE7 | | |
| IDirectDrawPalette | LPDIRECTDRAWPALETTE | | Release<br>GetEntries<br>SetEntries |
| IDirectDrawClipper | LPDIRECTDRAWCLIPPER | | SetHWnd |

Si possono ottenere molte informazioni dall'header *ddraw.h* (ovviamente se si possiede l'SDK con l'help sulle DirectX tutto ciò non è strettamente necessario).