

SỬ DỤNG KỸ THUẬT DEVICE CONTEXT ẢO

1 Mục đích

- Sử dụng kỹ thuật MemDC.
- Thao tác trên ảnh bitmap.

2 Kỹ thuật MemDC

2.1 Ý tưởng chính:

Nếu thực hiện nhiều thao tác vẽ trực tiếp trên device context của màn hình thì sẽ bị tình trạng màn hình bị “giật”.

Khi sử dụng kỹ thuật MemDC có thể hạn chế được hiện tượng “giật” màn hình:

- Trước tiên cần tạo ra 1 device context ảo trong bộ nhớ tương thích với device context thật sự cần xử lý.
- Sau đó, tất cả mọi thao tác vẽ sẽ được thực hiện trên dc ảo này thay vì thực hiện trực tiếp trên dc thật.
- Cuối cùng chép nội dung của dc ảo vào dc thật. Xem như chỉ thao tác trực tiếp trên device context thật.

2.2 Khai báo biến

Khai báo bổ sung biến thành phần trong lớp `CxxxView` (trong file `xxxView.h`):

```
CDC          m_MemDC;  
CBitmap     m_MemBitmap, *m_pOldBitmap;
```

2.3 Các hàm xử lý

2.3.1 Tạo MemDC

Cần phải tạo ra memdc trước khi sử dụng.

Tạo hàm `OnCreate` ứng với sự kiện `WM_CREATE` của lớp `CxxxView`

```
int CxxxView::OnCreate(LPCREATESTRUCT lpCreateStruct)  
{  
    if (CView::OnCreate(lpCreateStruct) == -1)  
        return -1;  
  
    // TODO: Add your specialized creation code here  
    // Lấy dc của màn hình  
    CClientDC dc(this);  
  
    // Lấy kích thước màn hình.  
    int MaxX = ::GetSystemMetrics(SM_CXSCREEN);  
    int MaxY = ::GetSystemMetrics(SM_CYSCREEN);  
  
    // Tạo ra m_MemDC tương thích với dc của màn hình  
    m_MemDC.CreateCompatibleDC(&dc);
```

```
// Tạo ra 1 đối tượng CBitmap (kích thước MaxX × MaxY)
// tương thích với dc màn hình
m_MemBitmap.CreateCompatibleBitmap(&dc, MaxX, MaxY);

// Đưa đối tượng bitmap m_MemBitmap vào m_MemDC
m_pOldBitmap = m_MemDC.SelectObject(&m_MemBitmap);
return 0;
}
```

2.3.2 Hủy MemDC

Cần hủy MemDC sau khi sử dụng xong.

Trong hàm destructor của lớp `CxxxView`, bổ sung các hàm hủy đối tượng MemDC và MemBitmap

```
CxxxView::~CxxxView()
{
    m_MemDC.DeleteDC();
    m_MemBitmap.DeleteObject();
    m_Objects.RemoveAll();
}
```

2.3.3 Sử dụng MemDC

Có thể viết code trong hàm `OnDraw` để sử dụng MemDC như sau :

```
void CxxxView::OnDraw(CDC* pDC)
{
    CVD4Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rect;
    GetClientRect(rect);

    m_MemDC.FillSolidRect(rect, RGB(59, 112, 168));
    int i, n=m_Objects.GetSize();
    for (i=0; i<n; i++)
        ((CEllipse*)m_Objects[i])->Draw(&m_MemDC); //Vẽ lên memDC

    pDC->BitBlt(0, 0, rect.Width(), rect.Height(),
               &m_MemDC, 0, 0, SRCCOPY);
}
```

Build và thực hiện thử chương trình

- Vẫn còn hiện tượng “giật” màn hình
- Lý do: mỗi khi cần vẽ lại cửa sổ của View, sự kiện `WM_ERASEBKGD` sẽ được tự động phát sinh để yêu cầu xóa toàn bộ nền của View bằng cách tô màu trắng vào vùng client của View. Khi đã sử dụng MemDC thì việc tô màu nền có thể thực hiện bằng lệnh `FillSolidRect` trong đoạn code trên đây. Như vậy, thao tác tô màu nền vùng client của View trong sự kiện xóa nền `WM_ERASEBKGD` là hoàn toàn không cần thiết và sẽ gây ra hiện tượng “giật” màn hình.

- Giải pháp: override hàm xử lý sự kiện `WM_ERASEBKGD` của `CxxxView`: Vào Class Wizard để tạo ra hàm `OnEraseBkgnd` tương ứng với sự kiện `WM_ERASEBKGD` của `CxxxView`.

```

BOOL CxxxView::OnEraseBkgnd(CDC* pDC)
{
    // TODO: Add your message handler code here and/or call default
    return TRUE;
    // return CView::OnEraseBkgnd(pDC);
}
    
```

Build và thực hiện chương trình để xem thử hiệu quả của việc sử dụng `memdc`.

2.3.4 Hàm `BitBlt`

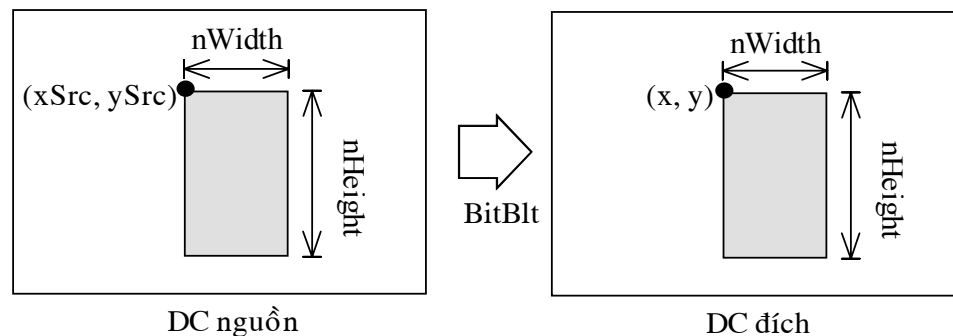
Hàm `BitBlt` của lớp `CDC` có tác dụng chép nội dung bitmap từ device context nguồn sang device context đích.

```

BOOL BitBlt(int x, int y, int nWidth, int nHeight,
            CDC* pSrcDC, int xSrc, int ySrc,
            DWORD dwRop );
    
```

với

<code>x, y</code>	Góc trái trên của vùng device context đích
<code>nWidth, nHeight</code>	Kích thước vùng device context đích
<code>pSrcDC</code>	Con trỏ đến device context nguồn
<code>xSrc, ySrc</code>	Góc trái trên của vùng ảnh được chép trong device context nguồn
<code>dwRop</code>	Chế độ chép ảnh



3 Thao tác trên ảnh bitmap

3.1 Tạo ảnh bitmap trong Resource

Trong cửa sổ Resource View, chọn chức năng Insert (hay dùng chức năng Resource trong menu Insert), sau đó chọn loại resource là bitmap. Trong phần Bitmap của Resource View sẽ xuất hiện thêm 1 bitmap (ví dụ có ID là `IDB_BITMAP1`)

Lưu ý: ảnh bitmap tạo trong resource theo cách này có thể là ảnh nhị phân (2 màu), ảnh 16 màu hay 256 màu. Không thể tạo ra ảnh true color (ví dụ ảnh 24bit) trong resource bằng cách này.

3.2 Import ảnh bitmap vào Resource

Có thể import ảnh bitmap có sẵn vào resource theo cách sau:

- Vào Resource View, chọn chức năng Import.
- Trong hộp thoại Import Resource, nhập vào ô File name *.bmp
- Chọn file bitmap đã lưu sẵn trên đĩa để import vào resource.

Lưu ý: nếu ảnh bitmap nhiều hơn 256 màu (tức là ảnh true color, ví dụ như ảnh 24 bit) thì không thể xem hay sửa đổi trong Resource View của Visual C++. Tuy nhiên, tất cả ảnh bitmap này đều có thể sử dụng bình thường như các ảnh bitmap có ít hơn hay bằng 256 màu.

3.3 Load ảnh bitmap từ Resource

Mọi thao tác trên ảnh bitmap trong resource được thực hiện thông qua lớp `CBitmap`.

Sử dụng hàm `LoadBitmap` của lớp `CBitmap` để load ảnh bitmap trong resource. Sau khi sử dụng xong thì cần hủy đối tượng bitmap bằng hàm `DeleteObject`.

Ví dụ:

```
CBitmap Pic;  
Pic.LoadBitmap(IDB_BITMAP1);  
... ..  
Pic.DeleteObject();
```

3.4 Load ảnh bitmap từ tập tin

Sử dụng hàm `LoadImage` để load bitmap từ tập tin :

```
HANDLE LoadImage(  
    HINSTANCE hinst,  
    LPCTSTR lpszName,  
    UINT uType,  
    int cxDesired,  
    int cyDesired,  
    UINT fuLoad  
);
```

Một ví dụ mở tập tin (mở tập tin Bitmap với tên là `m_strFileName`)

```
bmRead = (HBITMAP) LoadImage(NULL, m_strFileName,  
    IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
```

3.5 Hiển thị CBitmap trên device context (DC)

Xây dựng hàm `PutBitmap` để hiển thị `pBitmap` lên `pDC` tại vị trí góc trái trên là `TopLeft` và chế độ chép ảnh là `dwRop` (mặc định là `SRCCOPY`)

```
void PutBitmap(CDC* pDC, CBitmap* pBitmap,
              CPoint TopLeft, DWORD dwRop = SRCCOPY )
{
    BITMAP bm;
    CDC TempDC;
    TempDC.CreateCompatibleDC(pDC);
    TempDC.SelectObject(pBitmap);
    pBitmap->GetBitmap(&bm);
    pDC->BitBlt(TopLeft.x, TopLeft.y, bm.bmWidth, bm.bmHeight,
               &TempDC, 0, 0, dwRop);
    TempDC.DeleteDC();
}
```

Giải thích: cấu trúc `BITMAP` được sử dụng để lấy các thông số của 1 đối tượng `CBitmap`, chẳng hạn như kích thước, số bit màu/pixel...

3.6 Ví dụ:

Tạo ra 1 bitmap trong resource view (ví dụ: `IDB_BITMAP1`).

Xây dựng hàm `PutBitmap` như đã trình bày ở trên

Xây dựng tiếp hàm `OnDraw` như sau:

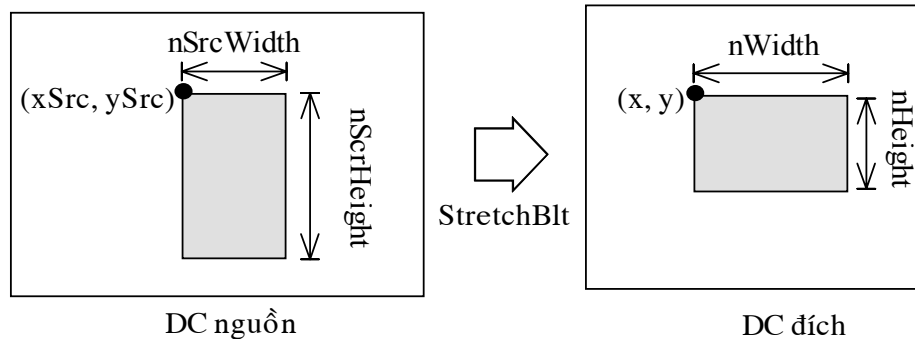
```
void CxxxView::OnDraw(CDC* pDC)
{
    CVD4Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rect;
    GetClientRect(rect); //Lấy cửa sổ vùng client
    m_MemDC.FillSolidRect(rect, RGB(59, 112, 168));
    CBitmap Pic;
    Pic.LoadBitmap(IDB_BITMAP1);
    PutBitmap(&m_MemDC, &Pic, CPoint(0, 0), SRCCOPY);
    Pic.DeleteObject();

    int i, n=m_Objects.GetSize();
    for (i=0; i<n; i++)
        ((CEllipse*)m_Objects[i])->Draw(&m_MemDC);

    pDC->BitBlt(0, 0, rect.Width(), rect.Height(),
               &m_MemDC, 0, 0, SRCCOPY);
}
```

4 Mở rộng

4.1 Hàm StretchBlt



Cũng giống như hàm `BitBlt`, hàm `StretchBlt` thực hiện thao tác chép nội dung của 1 device context sang 1 device context khác. Tuy nhiên, hàm `StretchBlt` cho phép phóng to hay thu nhỏ ảnh.

```

BOOL StretchBlt( int x, int y, int nWidth, int nHeight,
                CDC* pSrcDC,
                int xSrc, int ySrc, int nSrcWidth, int nSrcHeight,
                DWORD dwRop);
    
```

với

x, y	Góc trái trên của vùng device context đích
$nWidth, nHeight$	Kích thước vùng device context đích
$pSrcDC$	Con trỏ đến device context nguồn
$xSrc, ySrc$	Góc trái trên của vùng ảnh được chép trong device context nguồn
$nSrcWidth, nSrcHeight$	Kích thước vùng device context nguồn
$dwRop$	Chế độ chép ảnh

4.2 Xử lý file BMP

Những ảnh bitmap được lưu trữ sẵn trong Resource của project (được tạo ra trong Visual C hay import từ file bmp vào project) được xem là các ảnh bitmap phụ thuộc thiết bị (DDB – Device Dependent Bitmap).

Đối với các file ảnh BMP chưa được import vào trong Resource của project, mọi thao tác đọc/ghi/hiển thị ảnh được sử dụng thông qua

5 Phụ lục

5.1 Các chế độ chép ảnh trong hàm BitBlt và StretchBlt

- BLACKNESS** Turns all output black.
- DSTINVERT** Inverts the destination bitmap.

- **MERGECOPY** Combines the pattern and the source bitmap using the Boolean AND operator.
- **MERGEPAINT** Combines the inverted source bitmap with the destination bitmap using the Boolean OR operator.
- **NOTSRCCOPY** Copies the inverted source bitmap to the destination.
- **NOTSRCERASE** Inverts the result of combining the destination and source bitmaps using the Boolean OR operator.
- **PATCOPY** Copies the pattern to the destination bitmap.
- **PATINVERT** Combines the destination bitmap with the pattern using the Boolean XOR operator.
- **PATPAINT** Combines the inverted source bitmap with the pattern using the Boolean OR operator. Combines the result of this operation with the destination bitmap using the Boolean OR operator.
- **SRCAND** Combines pixels of the destination and source bitmaps using the Boolean AND operator.
- **SRCCOPY** Copies the source bitmap to the destination bitmap.
- **SRCERASE** Inverts the destination bitmap and combines the result with the source bitmap using the Boolean AND operator.
- **SRCINVERT** Combines pixels of the destination and source bitmaps using the Boolean XOR operator.
- **SRCPAINT** Combines pixels of the destination and source bitmaps using the Boolean OR operator.
- **WHITENESS** Turns all output white.

5.2 Các hàm xử lý Bitmap DIB (Device Independent Bitmap)

Function	Description
AlphaBlend	Displays a bitmap with transparent or semitransparent pixels.
BitBlt	Performs a bit-block transfer.
CreateBitmap	Creates a bitmap.
CreateBitmapIndirect	Creates a bitmap.
CreateCompatibleBitmap	Creates a bitmap compatible with a device.
CreateDIBitmap	Creates a device-dependent bitmap (DDB) from a DIB.

CreateDIBSection	Creates a DIB that applications can write to directly.
ExtFloodFill	Fills an area of the display surface with the current brush.
GetBitmapDimensionEx	Gets the dimensions of a bitmap.
GetDIBColorTable	Retrieves RGB color values from a DIB section bitmap.
GetDIBits	Copies a bitmap into a buffer.
GetPixel	Gets the RGB color value of the pixel at a given coordinate.
GetStretchBltMode	Gets the current stretching mode.
GradientFill	Fills rectangle and triangle structures.
LoadBitmap	Loads a bitmap from a module's executable file.
MaskBlt	Combines the color data in the source and destination bitmaps.
PlgBlt	Performs a bit-block transfer.
SetBitmapDimensionEx	Sets the preferred dimensions to a bitmap.
SetDIBColorTable	Sets RGB values in a DIB.
SetDIBits	Sets the pixels in a bitmap using color data from a DIB.
SetDIBitsToDevice	Sets the pixels in a rectangle using color data from a DIB.
SetPixel	Sets the color for a pixel.
SetPixelV	Sets a pixel to the best approximation of a color.
SetStretchBltMode	Sets the bitmap stretching mode.
StretchBlt	Copies a bitmap and stretches or compresses it.
StretchDIBits	Copies the color data in a DIB.
TransparentBlt	Performs a bit-block transfer of color data.