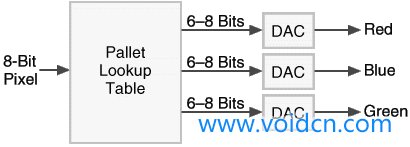
16.1 调色板的使用

传统意义上，调色板是画家混合颜色所使用的画板。这个词也用于表示艺术家用来绘画的全部颜色范围。在计算机图形学中，调色板时指图形输出设备上(例如视频显示器)可用的颜色范围。这个词也可指支持 256 种颜色模式的显示适配卡上的查找表。

16.1.1 视频硬件

在显卡上的调色板查找表的工作原理如下：



在 8 位视频模式中，每一个像素用 8 位二进制数据表示。像素值被送到一个包含 256 个 RGB 值的查找表。这些 RGB 数值可以是完整的 24 位宽，也可以窄一点，典型是 18 位宽(即红、绿、蓝分别由 6 位表示)。每一种颜色的数值被输入到数模转换器，而它产生的红、绿、蓝模拟信号被输出到显示器。

调色板的查找表一般可由计算机软件设置为任意的数值，但是实现与设备无关的接口(例如微软 Windows)还存在着一些困难。首先，Windows 必须提供一个软件接口，使应用程序可以访问调色板管理器而不必直接和硬件打交道。第二个问题显得更严重些：由于所有应用程序共享同一个显卡并且同时运行，当一个应用程序使用调色板查找表时，会影响到另一个程序。

在 Windows 3.0 中引入的 Windows 调色板管理器就是用来解决这些问题的。Windows 保留了 256 种颜色中的 20 种为自己所用，让应用程序使用其余 236 种颜色。(在某些情况下，应用程序可使用 256 种颜色种除了黑河白以外的 254 种颜色，不过需要费些功夫。)Windows 保留的 20 种颜色列在表 16-1 中，这些颜色又叫“静态颜色”。

表 16-1 256 色模式中保留的 20 种颜色

像 素 位 RGB 值 颜色名称 像 素 位 RGB 值 颜色名称

00000000 00 00 00 黑 11111111 FF FF FF 白

00000001 80 00 00 深红 11111110 00 FF FF 青色

00000010 00 80 00 深绿 11111101 FF 00 FF 紫红

00000011 80 80 00 暗黄 11111100 00 00 FF 蓝

00000100 00 00 80 深蓝 11111011 FF FF 00 黄

00000101 80 00 80 深紫红 11111010 00 FF 00 绿

00000110 00 80 80 深青色 11111001 FF 00 00 红

00000111 C0 C0 C0 浅灰 11110000 80 80 80 深灰

00001000 C0 DC C0 美元绿(MoneyGreen) 11110111 A0 A0 A4 中灰

00001001 A6 CA F0 天蓝 11110110 FF FB F0 乳白 当运行在 256 色视频模式上时，Windows 会维护一个“系统调色板”，它和显卡上的硬件调色板查找表示相同的。默认的系统调色板展示在表 16-1 中。 应用程序可使用逻辑调色板改变其他的 236 种颜色。如果多个应用程序使用逻辑调色板，Windows 会给活动窗口最高优先级。(正如你所知道的，活动窗口具有高亮的标题栏并出现在其他窗口之上。)让我们通过一个简单程序来看他是如何工作的。

在运行本章其余部分的程序时，建议把显卡转换到 256 色模式。在桌面单击鼠标右键，从快捷菜单中选择【属性】，再选择【设置】标签。

16.1.2 显示灰色图像

如下所示的 GRAYS1 程序没有用到 Windows 调色板管理器，而是试图采用正常显示。它用到了 65 个不同的灰色色调作为从黑色到白色的颜色过渡。

/\*----------------------------------------------------------

GRAYS1.C -- Gray Shades

(c) Charles Petzold, 1998

-----------------------------------------------------------\*/

#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevinstance,

PSTR szCmdLine, int iCmdShow)

{

static TCHAR szAppName[] = TEXT("Grays1");

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, TEXT("Shades of Gray #1"),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static int cxClient, cyClient;

HBRUSH hBrush;

HDC hdc;

int i;

PAINTSTRUCT ps;

RECT rect;

switch (message)

{

case WM\_SIZE:

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

// Draw the fountain of grays

for (i = 0; i < 65; i++)

{

rect.left = i \* cxClient / 65;

rect.top = 0;

rect.right = (i + 1) \* cxClient / 65;

rect.bottom = cyClient;

hBrush = CreateSolidBrush(RGB(min(255, 4 \* i),

min(255, 4 \* i),

min(255, 4 \* i)));

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

}

EndPaint(hwnd, &ps);

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

在 WM\_PAINT 消息期间，程序调用 FillRect 函数 65 次，每一次都使用不同灰色色调的画刷。灰色色调对应于不同的 RGB 值(0, 0, 0)、(4, 4, 4)、(8, 8, 8)，以此类推知道最后一组数值(255, 255, 255)。这最后一组数值也是我们为什么要在 CreateSolidBrush 函数中使用 min 宏的原因。

在 256 色视频模式上运行程序时，会看到从黑色到白色的 65 种灰色色调，几乎所有的颜色都是用抖动技术(dithering)画出来的。唯一的纯色是黑色、深灰色(128、128、128)、浅灰色(192, 192, 192)和白色。其他颜色是由各种位模式把这些纯色组合而成的。如果我们用 65 种灰色色调显示线或文字而不是填充区域，Windows 就不会使用抖动技术，而只会使用这四种纯色。如果显示位图，Windows 会使用 20 种 Windows 标准颜色来近似显示图像。要证明这一点，你可以运行最后一章的任何程序，再载入一个彩色或灰度 DIB。Windows 在正常情况下不会使用抖动技术显示位图。

如下所示的 GRAYS2 程序展示了非常重要的调色板管理器的功能和消息，而几乎没用什么外部代码。

/\*----------------------------------------------------------

GRAYS2.C -- Gray Shades Using Palette Manager

(c) Charles Petzold, 1998

-----------------------------------------------------------\*/

#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevinstance,

PSTR szCmdLine, int iCmdShow)

{

static TCHAR szAppName[] = TEXT("Grays2");

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, TEXT("Shades of Gray #2"),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HPALETTE hPalette;

static int cxClient, cyClient;

HBRUSH hBrush;

HDC hdc;

int i;

LOGPALETTE \* plp;

PAINTSTRUCT ps;

RECT rect;

switch (message)

{

case WM\_CREATE:

// Set up a LOGPALETTE structure and create a palette

plp = (LOGPALETTE \*)malloc(sizeof(LOGPALETTE) + 64 \* sizeof(PALETTEENTRY));

plp->palVersion = 0x0300;

plp->palNumEntries = 65;

for (i = 0; i < 65; i++)

{

plp->palPalEntry[i].peRed = (BYTE)min(255, 4 \* i);

plp->palPalEntry[i].peGreen = (BYTE)min(255, 4 \* i);

plp->palPalEntry[i].peBlue = (BYTE)min(255, 4 \* i);

plp->palPalEntry[i].peFlags = 0;

}

hPalette = CreatePalette(plp);

free(plp);

return 0;

case WM\_SIZE:

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

// Select and realize the palette in the device context

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

// Draw the fountain of grays

for (i = 0; i < 65; i++)

{

rect.left = i \* cxClient / 65;

rect.top = 0;

rect.right = (i + 1) \* cxClient / 65;

rect.bottom = cyClient;

hBrush = CreateSolidBrush(PALETTERGB(min(255, 4 \* i),

min(255, 4 \* i),

min(255, 4 \* i)));

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

}

EndPaint(hwnd, &ps);

return 0;

case WM\_QUERYNEWPALETTE:

if (!hPalette)

return FALSE;

hdc = GetDC(hwnd);

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

InvalidateRect(hwnd, NULL, TRUE);

ReleaseDC(hwnd, hdc);

return TRUE;

case WM\_PALETTECHANGED:

if (!hPalette || (HWND)wParam == hwnd)

break;

hdc = GetDC(hwnd);

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

UpdateColors(hdc);

ReleaseDC(hwnd, hdc);

break;

case WM\_DESTROY:

DeleteObject(hPalette);

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

通常，使用调色板管理器的第一步是调用 CreatePalette 函数生成一个逻辑调色板。逻辑调色板包括程序需要的所有颜色，更确切地说是 236 种颜色。GRAYS2 程序在处理 WM\_CREATE 消息时产生逻辑调色板。它初始化 LOGPALETTE(“逻辑调色板”)结构的字段，然后把这个结构的指针传给 CreatePalette 函数。CreatePalette 函数返回一个逻辑调色板的句柄，将它存在静态变量 hPalette 中。

LOGPALETTE 结构是这样定义的：

typedef struct

{

WORD palVersion

WORD palNumEntries;

PALETTEENTRY palPalEntry[1];

}

LOGPALETTE, \* PLOGPALETTE;

LOGPALETTE 结构第一个字段总是设置为 0x0300，表示与 Windows 3.0 兼容。第二个字段设置为调色板查找表中的条目数。第三个字段是一个 PALETTEENTRY 结构的数组，数组中的每一个单元对应于调色板的一个条目。PALETTEENTRY 结构定义如下：

typedef struct

{

BYTE peRed;

BYTE peGreen;

BYTE peBlue;

BYTE ppeFlags;

}

PALETTEENTRY, \* PPALETTEENTRY;

每一个 PALETTEENTRY 结构定义了我们在调色板中所需要的一个 RGB 颜色值。

请注意，LOGPALETTE 被定义为一个数组，而这个数组只包含一个 PALETTEENTRY 结构变量。你需要预留足够大的存储空间以保持一个 LOGPALETTE 结构以及额外的 PALETTEENTRY 结构。程序 GRAYS2 需要 65 种灰色色调，所以它分配了足够的内存，以存下一个 LOGPALETTE 结构和额外的 64 个 PALETTEENTRY 结构。它把 palNumEntries 字段设置为 65。GRAYS2 接着通过一个从 0 到 64 的循环，计算出灰度级(它为循环索引值的 4 倍，但不能超过 255)，并把该结构的 peRed、peGreen、peBlue 字段设置为这个灰度级，同时把 peFlags 字段设为 0。程序将该内存块的指针传给 CreatePalette 函数，把调色板的句柄保存在静态变量中，然后释放内存。

逻辑调色板是一个 GDI 对象。当应用程序完成了任务后，它应该销毁自己所产生的任何逻辑调色板。WndProc 在 WM\_DESTROY 消息期间调用 DeleteObject 函数，销毁逻辑调色板。

注意，逻辑调色板与设备环境是相互独立的。实际使用逻辑调色板之前，必须把调色板与设备环境联系在一起。在 WM\_PAINT 消息期间，SelectPalette 函数把逻辑调色板选入设备环境。除了还需要第三个参数之外，SelectPalette 与 SelectObject 函数很类似。通常三个参数设置为 FALSE。如果 SelectPalette 的第三个参数设置为 TRUE，则该调色板总是“背景调色板”(background palette)。也就是说，当其他程序实现了它们的调色板之后，系统调色板中剩下来的条目可被这个调色板使用。

在任何时候，只有一个逻辑调色板可被选入设备环境。函数返回的是上一次被选入设备环境中的逻辑调色板句柄。你可以把这个句柄存起来，以便下次选入到设备环境中。

RealizePalette 函数使 Windows “实现” 设备环境中的逻辑调色板。它把逻辑颜色映射到系统调色板上，而系统调色板对应的是显示适配卡的硬件调色板。调用这个函数时，真正的工作才开始。Windows 必须确认调用这个函数的窗口是活动的还是不活动的，也许还需要通知其他窗口：系统调色板正在改变。(下面我们将讨论这个通知过程。)

你也许还记得 GRAYS1 使用了 RGB 宏来指定画刷的颜色。RGB 宏构建了一个 32 位长整数(被称为 COLORREF 值)，最高字节是 0，三个低字节代表红、绿、蓝三种颜色的强度。

使用 Windows 调色板管理器的程序可以继续使用 RGB 颜色值来指定颜色。但是这些 RGB 颜色值不能使用逻辑调色板所提供的额外颜色。使用 RGB 颜色就相当于没有利用调色板管理器。要使用逻辑调色板的其他颜色，应使用 PALETTERGB 宏。“调色板 RGB”(Palette RGB)颜色与 RGB 颜色非常相似，不同之处是 COLORREF 的高字节被设成 2 而不是 0。

下面是一些重要的规则。

如果要使用逻辑调色板的颜色，请使用调色板 RGB 值或者调色板索引值，而不要使用通常的 RGB 值。(我将很快讨论调色板索引。)如果使用了通常的 RGB 值，得到的颜色将是标准颜色而不是来自逻辑调色板的颜色。

当还没有把调色板选入设备环境时，不要使用调色板 RGB 值或调色板索引值。

虽然可以使用调色板 RGB 值指定一种颜色，但是如果这个颜色并不存在于逻辑调色板中，那么最终得到的颜色仍会从逻辑调色板中选出。

例如，在 GRAYS2 的 WM\_PAINT 消息处理中，当选中和实现了逻辑调色板后，如果你试图显示红色，那么显示出来的颜色会是灰色。你需要使用 RGB 颜色值去选择不在逻辑调色板中的颜色。

注意，GRAYS2 程序从不检查显卡驱动程序是否支持调色板管理。当 GRAYS2 运行的视频模式不支持调色板管理时(也就是说，除 256 色之外的所有视频模式)，GRAYS2 在功能上相当于 GRAYS1。

16.1.3 调色板的消息

如果多个 Windows 程序使用调色板管理器，活动窗口会得到使用调色板的优先权。最近活跃的窗口获得第二个优先级，以此类推。每当一个新程序变成活动的时，Windows 调色板管理器一般必须重新调整系统调色板查找表。

如果程序在它的逻辑调色板上指定了一种颜色，而且是 20 种保留颜色中的一种，那么 Windows 将把该逻辑调色板条目影射到那种颜色。还有，如果两个以上程序在它们的逻辑调色板中指定了同样的颜色，那么这些程序将共享系统调色板条目。程序可以通过把 PALETTEENTRY 结构的 peFlags 字段设为一个常数 PC\_NOCOLLAPSE，来阻止这种默认的共享行为。(其他两个可能的标志使 PC\_EXPLICIT 和 PC\_RESERVED。PC\_EXPLICIT 用来显示系统调色板，PC\_RESERVED 则用于调色板动画。我将在本章后面进一步介绍这两种标志。)

为了帮助组织系统调色板，Windows 调色板管理器具有两个可发送到主窗口的消息。

第一个消息是 WM\_QUERYNEWPALETTE。当窗口即将变成活动窗口时，这个消息会发生到主窗口。如果程序使用调色板管理器在窗口上绘图，则必须处理这个消息。GRAYS2 展示了处理过程。程序获得设备环境句柄，选入调色板，调用 RealizePalette 函数，然后使窗口无效，进而产生 WM\_PAINT 消息。如果窗口过程实现了逻辑调色板，则消息处理返回 TRUE，否则返回 FALSE。

只要系统调色板因为 WM\_QUERYNEWPALETTE 消息而发生了变化，Windows 就会发送 WM\_PALETTECHANGED 消息到所有的主窗口，它先通知最活跃的窗口，然后顺序通知窗口链上的其他窗口。这就允许前端窗口具有优先权。发送到窗口的过程的 wParam 值是活动窗口的句柄。只有当 wParam 值不等于程序的窗口句柄，使用调色板管理器的程序才会处理这个消息。

通常，任何使用自定义调色板的程序，在处理 WM\_PALETTECHANGED 消息时，都要调用 SelectPalette 和 RealizePalette。当后续的窗口调用 RealizePalette 函数时，Windows 首先会检查逻辑调色板的 RGB 值是否与系统调色板加载的 RGB 值相同。如果两个程序需要同种颜色，它们可共用同一个系统调色板条目。下一步，Windows 会寻找还没有被用到的系统调色板条目。如果全用到了，那么逻辑调色板的颜色将被映射到最接近的系统颜色，即 20 种保留颜色之一。

如果你不关心在程序不活动时客户区会如何显示，就不需要处理 WM\_PALETTECHANGED 消息，否则你有两种选择。GRAYS2 展示了其中一种选择：在处理 WM\_QUERYNEWPALETTE 消息时，它获得了设备环境，选入了调色板，然后调用了 RealizePalette 函数。此时，它可以调用 InvalidateRect 函数来处理 WM\_QUERYNEWPALETTE 消息。但是 GRAYS2 缺调用了 UpdateColors 函数。这个函数一般比重绘窗口更有效，它改变了窗口里的像素的值，这样就可以保留之前使用的颜色。

使用调色板管理器的大多数程序，它们处理 WM\_QUERYNEWPALETTE 和 WM\_PALETTECHANGED 消息的方式会跟 GRAYS2 中展示的基本一样。

16.1.4 调色板索引方式

如下所示的 GRAYS3 程序与 GRAYS2 非常相似，但它在处理 WM\_PAINT 消息时用了一个叫 PALETTENINDEX 的宏，而没有使用 PALETTERGB 宏。

/\*----------------------------------------------------------

GRAYS3.C -- Gray Shades Using Palette Manager

(c) Charles Petzold, 1998

-----------------------------------------------------------\*/

#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevinstance,

PSTR szCmdLine, int iCmdShow)

{

static TCHAR szAppName[] = TEXT("Grays3");

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, TEXT("Shades of Gray #3"),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HPALETTE hPalette;

static int cxClient, cyClient;

HBRUSH hBrush;

HDC hdc;

int i;

LOGPALETTE \* plp;

PAINTSTRUCT ps;

RECT rect;

switch (message)

{

case WM\_CREATE:

// Set up a LOGPALETTE structure and create a palette

plp = (LOGPALETTE \*)malloc(sizeof(LOGPALETTE) + 64 \* sizeof(PALETTEENTRY));

plp->palVersion = 0x0300;

plp->palNumEntries = 65;

for (i = 0; i < 65; i++)

{

plp->palPalEntry[i].peRed = (BYTE)min(255, 4 \* i);

plp->palPalEntry[i].peGreen = (BYTE)min(255, 4 \* i);

plp->palPalEntry[i].peBlue = (BYTE)min(255, 4 \* i);

plp->palPalEntry[i].peFlags = 0;

}

hPalette = CreatePalette(plp);

free(plp);

return 0;

case WM\_SIZE:

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

// Select and realize the palette in the device context

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

// Draw the fountain of grays

for (i = 0; i < 65; i++)

{

rect.left = i \* cxClient / 65;

rect.top = 0;

rect.right = (i + 1) \* cxClient / 65;

rect.bottom = cyClient;

hBrush = CreateSolidBrush(PALETTEINDEX(i));

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

}

EndPaint(hwnd, &ps);

return 0;

case WM\_QUERYNEWPALETTE:

if (!hPalette)

return FALSE;

hdc = GetDC(hwnd);

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

InvalidateRect(hwnd, NULL, TRUE);

ReleaseDC(hwnd, hdc);

return TRUE;

case WM\_PALETTECHANGED:

if (!hPalette || (HWND)wParam == hwnd)

break;

hdc = GetDC(hwnd);

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

UpdateColors(hdc);

ReleaseDC(hwnd, hdc);

break;

case WM\_DESTROY:

DeleteObject(hPalette);

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

“调色板索引”颜色与调色板 RGB 颜色由很大不同。调色板索引颜色的高字节的值是 1，而低字节的值是设备环境中当前被选中的逻辑调色板的索引。在 GRAYS3 中，逻辑调色板有 65 个条目。对应这些条目的索引值是 0 到 64。例如

PALETTEINDEX (0) 指黑色

PALETTEINDEX(32) 指中灰色

PALETTEINDEX(64) 指白色

使用调色板索引比使用 RGB 值更有效，因为 Windows 不再需要搜索最接近的颜色。

16.1.5 查询对调色板的支持

可以很容易地验证，当 Windows 在 16 位或 24 位视频模式运行时，GRAYS2 和 GRAYS3 程序会正常运行。但是在有些情况下，希望使用调色板管理器的 Windows 应用程序可能需要首先确认设备驱动程序是否支持调色板管理器。为此可以调用 GetDeviceCaps 函数，并提供视频显示的设备环境句柄和 RASTERCAPS 参数。该函数返回一个由一系列标志位组合而成的整数。可以通过对返回值和常数 RC\_PALETTE 按位与运算来检测调色板的支持情况：

RC\_PALETTE & GetDeviceCaps (hdc, RASTERCAPS)

如果这个值不是零，则说明视频显示的设备驱动程序支持调色板操作。此时，GetDeviceCaps 函数还可以提供其他三个重要的数据。以下函数调用将返回显卡的调色板查找表的总大小：

GetDeviceCaps (hdc, SIZEPALETTE)

这个值是能同时显示的颜色的总数。由于调色板只用于 8 位像素的视频显示模式，因此这个值将是 256。

函数调用将返回调色板查找表中设备驱动程序为系统所保留的颜色总数：

GetDeviceCaps (hdc, NUMRESERVED)

这个值将是 20。 如果不是用调色板管理器，这个数字将是 Windows 应用程序在 256 色视频模式下所能使用的纯色的数目。一个程序要使用其他的 236 种颜色，就必须使用调色板管理器。另外还有一项，函数调用将返回加载到硬件调色板查找表中的 RGB 颜色值的分辨率(用二进制位表示)：

GetDeviceCaps (hdc, COLORRES)

这些二进制位会输入到一个数模转换器。一些显示适配器只使用 6 位 ADC，所以这个返回值将是 18。其他视频卡使用 8 位 ADC，这个返回值将是 24。

我们建议 Windows 应用程序应检测颜色分辨率以便正确地显示颜色。例如，如果颜色分辨率是 18，那么程序要求显示 128 个灰度级就没有任何意义，因为此时最多只存在 64 个灰度级。要求 128 个灰度级会在硬件调色板查找表中填入多余的重复条目。

16.1.6 系统调色板

我曾提到过，Windows 的系统调色板直接对应于显示适配卡的硬件调色板查找表。(但是，硬件调色板查找表的严格上分辨率也许比系统调色板低。)如果应用程序要得到系统调色板的部分或全部 RGB 条目，可调用下面的函数：

GetSystemPaletteEntries (hdc, uStart, uNum, &pe);

只有当视频适配器模式支持调色板操作时，这个函数才能工作。第二个和第三个参数是无符号整数，它们分别对应于所需要的第一个调色板条目索引和总共所需的条目个数。最后的参数是指向 PALETTENENTRY 结构的指针。

有几种方法可使用这个函数。应用程序可以如下定义一个 PALETTEENTRY 结构：

PALETTEENTRY pe;

然后像下面这样多次调用 GetSystemPaletteEntries：

GetSystemPaletteEntries (hdc, i, 1, &pe);

这里，i 可以是从 0 到任何小于 255 的数。(这个 255 是通过制定 SIZEPALETTE 为参数而调用 GetDeviceCaps 所得到的返回值。)要想得到系统调色板的所有条目，还可通过以下的方法：定义一个指向 PALETTEENTRY 结构的指针，然后分配一块足以容纳多个 PALETTEENTRY 结构的内存，结构的数量由调色板的大小指定。

GetSystemPaletteEntries 函数确实允许你检查硬件调色板查找表。系统调色板的条目以像素位增加的顺序排列，这些像素位被用来表示视频显示缓冲区的颜色。我将在下面展示如何做到这一点。

16.1.7 其他的调色板函数

前面说过，Windows 程序只能间接地改变系统调色板。第一步是创建程序要用的逻辑调色板，它基本上时一个 RGB 颜色值的数组。CreatePalette 函数不会造成系统调色板或显卡的调色板查找表的任何变化。逻辑调色板必须选入设备环境中，并在它被使用之前实现。

应用程序可调用如下函数来查询逻辑调色板的 RGB 颜色值：

GetPaletteEntries (hPalette, uStart, uNum, &pe);

这个函数的使用方法和 GetSystemPaletteEntries 相同。但要注意，它的第一个参数是指向逻辑调色板的句柄，而不是指向设备环境的句柄。

如果逻辑调色板已经创建，则相应有一个函数允许你改变逻辑调色板中的值：

SetPaletteEntries (hPalette, uStart, uNum, &pe);

请再一次记住，调用这个函数不会引起系统调色板的变化，即便该调色板当前已被选入设备环境。这个函数同样不同改变逻辑调色板的大小。要想改变它的大小，请使用 ResizePalette 函数。

下面的函数把一个 RGB 颜色参考值作为最后的参数，返回逻辑调色板的索引值，它对应于最为接近该参考色的 RGB 颜色：

iIndex = GetNearestPaletteIndex (hPalette, cr);

第二个参数是 COLORREF 值。如果你愿意，可以在以后调用 GetPaletteEntries 得到逻辑调色板中的实际 RGB 颜色值。

在 8 位视频模式下，如果程序需要多于 236 种自定义颜色，则可调用 GetSystemPaletteUse。这允许程序设置 254 种自定义颜色，系统仅保留黑色和白色。但是，这种情况只应发生在程序被放大到占据了全屏幕的时候，而且，程序应该设置一些系统颜色为黑色和白色以便保证标题栏、菜单等诸如此类的东西还能看得见。

16.1.8 光栅操作的问题

在第 5 章我们了解到，GDI 允许使用多种“绘图模式”或“光栅操作”来绘制直线或填充区域。你可以调用 SetROP2 设置绘图模式。这里的“2”标志着两个对象间进行二元光栅操作。三元(tertiary)光栅操作由 BItBlt 及类似函数完成。这些光栅操作决定了所绘制对象与表面像素如何结合在一起。例如，你可以画一条线使得线上的像素与显示的像素进行位异或操作。

光栅操作的工作原理是在像素上进行二进制位操作。改变调色板可能影响到这些光栅操作的工作方式。光栅操作是针对像素位的，这些像素位也许与实际的颜色不相关。

你可以运行程序 GRAYS2 和 GRAYS3 来验证这一点。拖动顶部或底部边框，改变窗口尺寸。拖动过程中，Windows 会用光栅操作技术翻转背景像素位，以显示被拖动的边框。这样做的目的是保证边框在拖动过程中总是可以被看见。但是对于 GRAYS2 和 GRAYS3 程序，你也许会看到各种各样的随机颜色。这些颜色正好对应于调色板中没被使用的条目。造成这种现象的原因是由于翻转显示像素位所造成的。可见颜色并没有被翻转，只是像素位变化了。

正如表 16-1 所示，20 种标准的颜色被放在系统调色板的顶部或底部，这样关山操作的结果仍然正常。然而，一旦开始改变调色板，特别是如果改变了保留的颜色，那么对彩色对象的光栅操作可能会变得没有任何意义。

你唯一能保证的是光栅操作对黑色和白色继续有效。黑色时系统调色器的第一个条目(所有像素位设置为0)，白色是最后一个条目(所有像素位设置为 1)。这些条目不能被改变。如果需要预测光栅操作对彩色对象的结果，你可以读取系统调色板查找表，看一下各像素位值对应的 RGB 颜色值。

16.1.9 查看系统调色板

在 Windows 下运行的程序与逻辑调色板打交道。Windows 会设置系统调色板的颜色，以便最好地服务于那些使用逻辑调色板的程序。系统调色板反映了显示适配卡的硬件查找表的情况。因此，了解系统调色板可以帮助调色调色板应用程序。

由于存在着三种完全不同的解决问题的方法，所以我将用三个不同的程序分别展示系统调色板的内容。

/\*------------------------------------------

SYSPAL1.C -- Displays system palette

(c) Charles Petzold, 1998

------------------------------------------\*/

#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

TCHAR szAppName[] = TEXT("SysPal1");

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevinstance,

PSTR szCmdLine, int iCmdShow)

{

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, TEXT("System Palette #1"),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

if (!hwnd)

return 0;

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

BOOL CheckDisplay(HWND hwnd)

{

HDC hdc;

int iPalSize;

hdc = GetDC(hwnd);

iPalSize = GetDeviceCaps(hdc, SIZEPALETTE);

ReleaseDC(hwnd, hdc);

if (iPalSize != 256)

{

MessageBox(hwnd, TEXT("This program requires that the video")

TEXT("display mdoe have a 256-color palette."),

szAppName, MB\_ICONERROR);

return FALSE;

}

return TRUE;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static int cxClient, cyClient;

static SIZE sizeChar;

HDC hdc;

HPALETTE hPalette;

int i, x, y;

PAINTSTRUCT ps;

PALETTEENTRY pe[256];

TCHAR szBuffer[16];

switch (message)

{

case WM\_CREATE:

if (!CheckDisplay(hwnd))

return -1;

hdc = GetDC(hwnd);

SelectObject(hdc, GetStockObject(SYSTEM\_FIXED\_FONT));

GetTextExtentPoint32(hdc, TEXT("FF-FF-FF"), 10, &sizeChar);

ReleaseDC(hwnd, hdc);

return 0;

case WM\_DISPLAYCHANGE:

if (!CheckDisplay(hwnd))

DestroyWindow(hwnd);

return 0;

case WM\_SIZE:

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

SelectObject(hdc, GetStockObject(SYSTEM\_FIXED\_FONT));

GetSystemPaletteEntries(hdc, 0, 256, pe);

for (i = 0, x = 0, y = 0; i < 256; i++)

{

wsprintf(szBuffer, TEXT("%02X-%02X-%02X"),

pe[i].peRed, pe[i].peGreen, pe[i].peBlue);

TextOut(hdc, x, y, szBuffer, lstrlen(szBuffer));

if ((x += sizeChar.cx) + sizeChar.cx > cxClient)

{

x = 0;

if ((y += sizeChar.cy) > cyClient)

break;

}

}

EndPaint(hwnd, &ps);

return 0;

case WM\_PALETTECHANGED:

InvalidateRect(hwnd, NULL, FALSE);

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

与 SYSPAL 系列的其他程序一样，只有当使用 SIZEPALETTE 参数调用 GetDeviceCaps 函数的返回值是 256 时，SYSPAL1 程序才能运行。

注意，一旦 SYSPAL1 收到 WM\_PALETTECHANGED 消息，它的客户区将变为无效。在处理由此产生的 WM\_PAINT 消息期间，SYSPAL1 调用 GetSystemPaletteEntries 函数，调用时用到了一个 256 个 PALETTEENTRY 结构的数组。RGB 值被作为文本字符串显示在客户区中。当执行这个程序时，请注意 20 种保留的颜色是 RGB 列表中的前 10 个颜色和最后 10 个颜色，如表 16-1 所示。

尽管 SYSPAL1 的确展示了有用的信息，但与实际看到 256 种颜色还不完全一样。如下所示的 SYSPAL2 解决了这个问题。

/\*------------------------------------------

SYSPAL2.C -- Displays system palette

(c) Charles Petzold, 1998

------------------------------------------\*/

#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

TCHAR szAppName[] = TEXT("SysPal2");

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevinstance,

PSTR szCmdLine, int iCmdShow)

{

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, TEXT("System Palette #1"),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

if (!hwnd)

return 0;

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

BOOL CheckDisplay(HWND hwnd)

{

HDC hdc;

int iPalSize;

hdc = GetDC(hwnd);

iPalSize = GetDeviceCaps(hdc, SIZEPALETTE);

ReleaseDC(hwnd, hdc);

if (iPalSize != 256)

{

MessageBox(hwnd, TEXT("This program requires that the video")

TEXT("display mdoe have a 256-color palette."),

szAppName, MB\_ICONERROR);

return FALSE;

}

return TRUE;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HPALETTE hPalette;

static int cxClient, cyClient;

HBRUSH hBrush;

HDC hdc;

int i, x, y;

LOGPALETTE \* plp;

PAINTSTRUCT ps;

RECT rect;

switch (message)

{

case WM\_CREATE:

if (!CheckDisplay(hwnd))

return -1;

plp = (LOGPALETTE \*)malloc(sizeof(LOGPALETTE) + 255 \* sizeof(PALETTEENTRY));

plp->palVersion = 0x0300;

plp->palNumEntries = 256;

for (i = 0; i < 256; i++)

{

plp->palPalEntry[i].peRed = i;

plp->palPalEntry[i].peGreen = 0;

plp->palPalEntry[i].peBlue = 0;

plp->palPalEntry[i].peFlags = PC\_EXPLICIT;

}

hPalette = CreatePalette(plp);

free(plp);

return 0;

case WM\_DISPLAYCHANGE:

if (!CheckDisplay(hwnd))

DestroyWindow(hwnd);

return 0;

case WM\_SIZE:

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

SelectPalette(hdc, hPalette, FALSE);

RealizePalette(hdc);

for (y = 0; y < 16; y++)

for (x = 0; x < 16; x++)

{

hBrush = CreateSolidBrush(PALETTEINDEX(16 \* y + x));

SetRect(&rect, x \* cxClient / 16, y \* cyClient / 16,

(x + 1) \* cxClient / 16, (y + 1)\*cyClient / 16);

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

}

EndPaint(hwnd, &ps);

return 0;

case WM\_PALETTECHANGED:

if ((HWND) wParam != hwnd)

InvalidateRect(hwnd, NULL, FALSE);

return 0;

case WM\_DESTROY:

DeleteObject(hPalette);

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

SYSPAL2 在 WM\_CREATE 消息期间生成了一个逻辑调色板。但是注意，该逻辑调色板的所有 256 个数值都是调色板的索引，范围从 0 到 255。peFlags 字段被改为 PC\_EXPLICIT，这个标志意味着：“逻辑调色板条目的低阶字(low-order word)被指定为硬件调色板的索引。这个标志允许应用程序列出显示设备调色板的内容”。因此，这个标志特别适合我们想要做的事。

在 WM\_PAINT 消息期间，SYSPAL2 把调色板选入设备环境并实现它。这样做不会造成系统调色板的重新调整，而是允许程序使用 PALETTEINDEX 宏指定系统调色板的颜色。SYSPAL2 利用此方法显示了 256 个矩形。同样，如果运行程序，请注意顶端的 10 种颜色和底部的 10 种颜色，它们是图 16-1 中的 20 种保留颜色。当运行的程序使用了自定义的逻辑调色板时，显示的结果会变化。

如果你想在看 SYSPAL2 的颜色的同时也看到对应的 RGB 值，那么可以同时运行第 8 章的 WHATCLR 程序。

SYSPAL 系列的第三个版本使用一种我最近才发现的技术——尽管我研究 Windows 调色板管理器已经有七年了。

几乎所有的 GDI 函数都会直接地或者间接地用 RGB 值来指定颜色。在 GDI 内部，这个值被转换为对应于那种颜色的某些像素位。在有些视频模式下(例如在 16 位或 24 位模式中)，这种转换时相当简单的。在其他视频模式下(4 位或 8 位颜色)，这种转换会涉及最相近颜色的搜索。

但是，有两个 GDI 函数，它们允许直接用像素位指定颜色。当然这两个函数是与设备息息相关的。而且它们与设备太有关了，以至于它们可以直接显示视频显示适配器上的实际调色板查找表的内容。这两个函数是 BitBlt 和 StretchBlt。

如下所示的 SYSPAL3 展示了如何用 StretchBlt 来显示系统调色板的颜色。

/\*------------------------------------------

SYSPAL3.C -- Displays system palette

(c) Charles Petzold, 1998

------------------------------------------\*/

#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

TCHAR szAppName[] = TEXT("SysPal3");

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevinstance,

PSTR szCmdLine, int iCmdShow)

{

HWND hwnd;

MSG msg;

WNDCLASS wndclass;

wndclass.style = CS\_HREDRAW | CS\_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

if (!RegisterClass(&wndclass))

{

MessageBox(NULL, TEXT("This program requires Windows NT!"),

szAppName, MB\_ICONERROR);

return 0;

}

hwnd = CreateWindow(szAppName, TEXT("System Palette #3"),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

if (!hwnd)

return 0;

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

BOOL CheckDisplay(HWND hwnd)

{

HDC hdc;

int iPalSize;

hdc = GetDC(hwnd);

iPalSize = GetDeviceCaps(hdc, SIZEPALETTE);

ReleaseDC(hwnd, hdc);

if (iPalSize != 256)

{

MessageBox(hwnd, TEXT("This program requires that the video")

TEXT("display mdoe have a 256-color palette."),

szAppName, MB\_ICONERROR);

return FALSE;

}

return TRUE;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HBITMAP hBitmap;

static int cxClient, cyClient;

BYTE bits [256];

HDC hdc, hdcMem;

int i;

PAINTSTRUCT ps;

switch (message)

{

case WM\_CREATE:

if (!CheckDisplay(hwnd))

return -1;

for (i = 0; i < 256; i++)

bits[i] = i;

hBitmap = CreateBitmap(16, 16, 1, 8, &bits);

return 0;

case WM\_DISPLAYCHANGE:

if (!CheckDisplay(hwnd))

DestroyWindow(hwnd);

return 0;

case WM\_SIZE:

cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

hdcMem = CreateCompatibleDC(hdc);

SelectObject(hdcMem, hBitmap);

StretchBlt(hdc, 0, 0, cxClient, cyClient,

hdcMem, 0, 0, 16, 16, SRCCOPY);

DeleteDC(hdcMem);

EndPaint(hwnd, &ps);

return 0;

case WM\_PALETTECHANGED:

if ((HWND)wParam != hwnd)

InvalidateRect(hwnd, NULL, FALSE);

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}

在 WM\_CREATE 消息期间，SYSPAL3 调用 CreateBitmap 函数，产生了一个 16 \* 16 像素的位图，每像素 8 位。该函数的最后一个参数是一个 256 字节的数组，分别存有 0 到 255 的值。这些是可能的 256 种像素位的值。在 WM\_PAINT 消息期间，程序把该位图选入内存设备环境中，然后使用 StretchBlt 将它显示在整个客户区内。Windows 简单地把位图的像素位输入到视频显示硬件，以便允许这些像素位去访问调色板查找表的 256 个条目。在收到 WM\_PALETTECHANGED 消息时，程序的客户区甚至不需要被设置成无效的，因为任何查找表的变化都会立即反映到 SYSPAL3 的显示上。