ゲームプログラミング

▶　DirectDraw－第５回　ピクセルフォーマット　◀

DirectDrawでもっとも理解しにくいといわれるのがピクセルフォーマットです。ピクセルフォーマットが理解できれば、サーフェイスのピクセルを直接加工し、高度な画像効果を再現することができます。

RGBカラー

Windowsのピクセルは、RGBと呼ばれる光の三原色、赤(Red)、緑(Green)、青(Blue)を調合して作られます。これらを調合して色を作成するためにRGBマクロが提供されており、赤、緑、青の輝度をそれぞれ0～255の256段階で指定することができます。たとえば、ピンク色(赤：255、緑：128、青：192)を表現するには、

```
COLORREF    rgbColor = RGB(255, 128, 192);
```

とします。RGBマクロは、調合された色をCOLORREF型の32ビット値で返します。この値は、デバイスコンテキストを用いて描画するときに使用します。しかし、DirectDrawサーフェイスに描画するときには、COLORREF型の値を使用することはできません。

ピクセルフォーマット

ピクセルフォーマットは、サーフェイス上で１ピクセルを構成するビット数と、そのビットの並びかたのことです。基本的にディスプレイモードとサーフェイスのピクセルフォーマットは同一です。同じ色でもディスプレイモードにより表現の方法が変わります。

○８ビット(256色)モード
サーフェイスの１ピクセルは、８ビットで表現されます。ピクセルは、パレットエントリの番号(インデックス値)を示しています。たとえば赤は、赤のパレットインデックスの値になります。

○16ビット(High Color)モード
High Colorモードでは、サーフェイスの１ピクセルを16ビットで表現します。しかし、16は３で割り切れないため、余分な１ビットをどうするかによっていくつかのピクセルフォーマットが存在します。

５：５：５
各成分に５ビットずつ割りあてたピクセルフォーマットです。

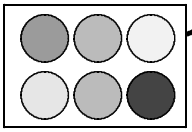
５：６：５
16ビットを赤５ビット、緑６ビット、青５ビットと割りあてたピクセルフォーマットです。

○24ビット(True Color)モード
サーフェイスの１ピクセルは、赤８ビット、緑８ビット、青８ビットの計24ビットで表現されます。このモードをサポートしないビデオカードもあるので注意が必要です。

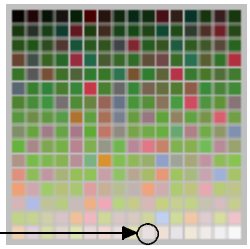
○32ビット(True Color)モード
サーフェイスの１ピクセルは、ダミー８ビット、赤８ビット、緑８ビット、青８ビットの計32ビットで表現されます。32ビットですが、色の構成は24ビットモードの８：８：８と同じであり、色数が増えるわけではありません。
32ビットモードは、24ビットモードよりメモリを多く消費しますが、32ビットというデータ型はシステムにとって相性がよいため、演算処理や転送効率に高いパフォーマンスが望めます。

ディスプレモードとピクセルフォーマットの関係
(R:255,G:0,B:0の場合)

①８ビット(256色)モード

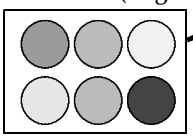


F 9　８ビット使用
パレットインデックスを示す



パレットテーブル

②16ビット(High Color)モード　５：５：５

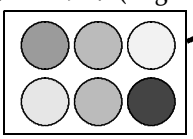


F C 0 0　16ビット使用

R R R R R G G G G G B B B B B

赤５ビット 緑５ビット 青５ビット

③16ビット(High Color)モード　５：６：５

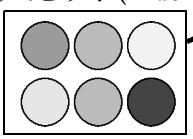


8 F 0 0　16ビット使用

R R R R R G G G G G G B B B B B

赤５ビット　緑６ビット　青５ビット

④24ビット(True Color)モード

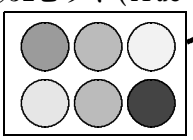


F F 0 0 0 0　24ビット使用

R R R R R R R R G G G G G G G G B B B B B B B B

赤８ビット　緑８ビット　青８ビット

⑤32ビット(True Color)モード



0 0 F F 0 0 0 0　32ビット使用

R R R R R R R R G G G G G G G G B B B B B B B B

ダミー８ビット　赤８ビット　緑８ビット　青８ビット

⑥COLORREF型
RGB(255, 0, 0)→

0 0 0 0 0 0 F F　32ビット型変数

0 0 0 0 0 0 0 0 B B B B B B B B G G G G G G G G R R R R R R R R

つねに０　青８ビット　緑８ビット　赤８ビット

※ハイカラー以上では、ピクセルの並びが図のとおりにならない場合があります。

○ 課題 ○

以下のプログラムは、RGBマクロが返すCOLORREF型の色値を、サーフェイス上で表現される色値に変換するDDColorMatch関数です。この関数は、カラーキーや塗りつぶしの値を決定するときに役に立ちます。関数の仕様をよく読んで解析し、DDUtilsに追加しましょう。

DDColorMatch関数
－説明－
DDColorMatch関数は、RGBマクロで調合されたCOLORREF型の色値を、指定されたサーフェイスのピクセルフォーマットのビット列に変換します。

－パラメータ－
const DDSRFC dds…色値を取得するサーフェイス。DDSRFC列挙体型を指定
const COLORREF rgbColor…変換する色

－戻り値－
関数が成功した場合は、変換された色情報のビット列，それ以外はCLR_INVALIDを返します。

－処理の流れ－
①左上(作業用)ピクセルの保存
サーフェイスの左上ピクセルを作業用として使用するため、現在の左上ピクセルをGetPixel関数で取り出し、保存しておきます。

②左上ピクセルへ描画
DDColorMatch関数の２つ目の引数で与えられた色を、SetPixel関数でサーフェイスに書き込みます。SetPixel関数はデバイスコンテキストを用いた描画なので、サーフェイスのピクセルフォーマットに適合するように変換して描画されます。

③サーフェイスのロック
②で変換されて書き込まれた色情報のビット列をサーフェイスから取り出します。サーフェイスのデータへ直接的にアクセスするには、Lockメソッドでサーフェイスをロックします。Lockメソッドは、サーフェイスをロックし、ロックされた領域の左上のピクセルのアドレス(ポインタ)を取得することができます。このポインタを使って、サーフェイスのピクセルへ直接的にアクセスすることができます。

④サーフェイスのピクセルデータの取り出し
サーフェイスをロックすると、Lockメソッドに渡したDDSURFACEDESC2構造体変数にサーフェイスの情報が格納されます。その中のddpfPixelFormatメンバは構造体で、この構造体のdwRGBitCountメンバには、１ピクセルあたりのビット数が格納されています。これを利用し、CopyMemory関数でサーフェイスから左上の１ピクセルのビット列を取り出します。

⑤サーフェイスのロックを解除
サーフェイスのロックを、Unlockメソッドで解除します。

⑥左上ピクセルの復元
①で保存した左上ピクセルを復元します。

```
/*****************************************************************************************************/
/*                                        サーフェイスカラー照合                                       */
/*****************************************************************************************************/
DWORD DDColorMatch(const DDSRFC dds, const COLORREF rgbColor)
{
#ifdef _DEBUG
    if(NULL == g_lpDDSurface7[dds]) {
        OutputDebugString("*** Error - 対象サーフェイス未初期化(DDColorMatch)\n");
        return CLR_INVALID;
    }
#endif

    // 指定された色をサーフェイスに書き込む
    HDC   hDC;
    if(DD_OK != g_lpDDSurface7[dds]->GetDC(&hDC)) {
        OutputDebugString("*** Error - サーフェイスDC取得失敗(DDColorMatch)\n");
        return CLR_INVALID;
    }
```

```
    COLORREF    rgb = GetPixel(hDC, 0, 0);    // 元のピクセルを保存しておく
    SetPixel(hDC, 0, 0, rgbColor);
    g_lpDDSurface7[dds]->ReleaseDC(hDC);

    // サーフェイスをロックしてポインタを取得
    DDSURFACEDESC2    ddsd2;
    ddsd2.dwSize = sizeof(ddsd2);
    if(DD_OK != g_lpDDSurface7[dds]->Lock(NULL, &ddsd2, DDLOCK_WAIT, NULL)) {
        OutputDebugString("*** Error - ロック失敗(DDColorMatch)\n");
        return CLR_INVALID;
    }

    // 書き込まれた色情報のビット列を取得
    DWORD    dwColor = 0;
    if(0 == ddsd2.ddpfPixelFormat.dwRGBBitCount % 8)
        CopyMemory(&dwColor, ddsd2.lpSurface, ddsd2.ddpfPixelFormat.dwRGBBitCount / 8);
    else {
        CopyMemory(&dwColor, ddsd2.lpSurface, ddsd2.ddpfPixelFormat.dwRGBBitCount / 8 + 1);
        dwColor &= (1 << ddsd2.ddpfPixelFormat.dwRGBBitCount) - 1;  // マスク
    }

    g_lpDDSurface7[dds]->Unlock(NULL);

    // ピクセル復元
    if(CLR_INVALID != rgb) {
        if(DD_OK == g_lpDDSurface7[dds]->GetDC(&hDC)) {
            SetPixel(hDC, 0, 0, rgb);
            g_lpDDSurface7[dds]->ReleaseDC(hDC);
        }
    }

    return dwColor;
}
```

※この関数は、アルファ情報付きサーフェイス(テクスチャなど)の場合、デバイスコンテキストがアルファ値をサポートしないため正しく動作しません。この関数で取得した値に、アルファ情報を加えることにより正しい色値にすることができます。