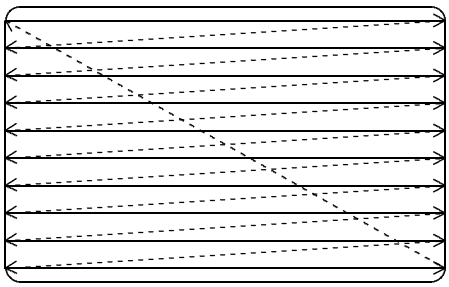
オブジェクト指向と ゲームプログラミング

▶ DirectX Graphics編 － 第６回　描画のしくみ ◀

ディスプレイ描画の仕組み

DirectX Graphicsでは、3Dオブジェクトをレンダリングし、その内容をビデオカードがDAC(Digital to Analog Converter：デジタル情報をモニタ表示用のアナログ信号に変換する装置)を介してモニタに送ることで描画を行います。この動作はある一定の周期に合わせて行われます。

ブラウン管(CRT:Cathode Ray Tube)を用いたディスプレイは、ブラウン管の奥からレーザーが発射されるようになっており、レーザーが左から右に移動しながら下に流れていき、目の残像現象を用いて映像を表示しています。この動作を走査といい、いちばん下まで走査が終わったら最上段に戻ります。この最上段に戻る動作を垂直帰線といい、この間は描画が行われません。一般的なディスプレイは、この動作を１秒間に60～120回行います。１秒間に行う垂直帰線動作回数のことをリフレッシュレートと呼び、多ければ多いほど、画面を書き換える回数が増え、ちらつきを感じにくくなります。

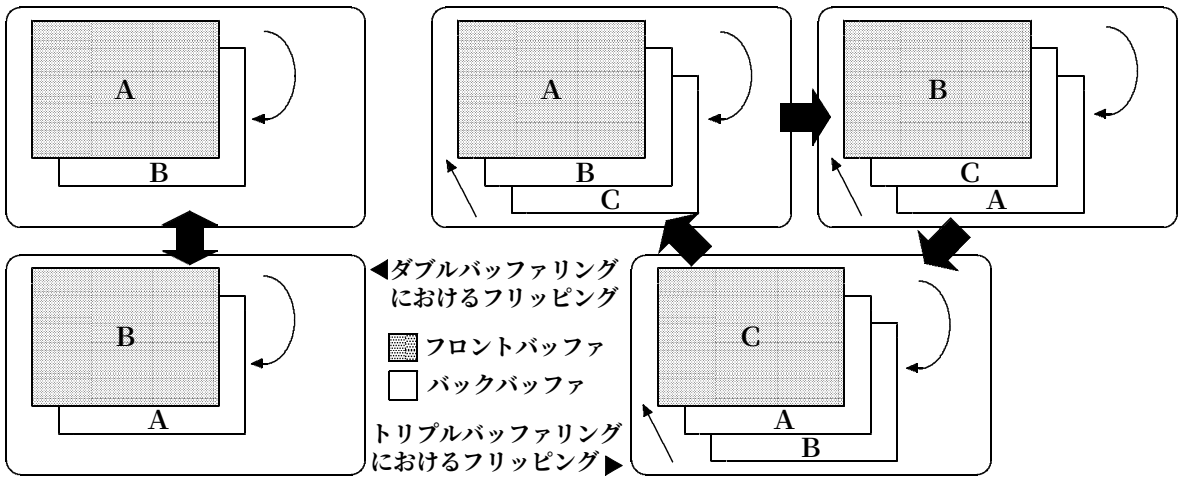


▲ディスプレイに映像が表示される様子。
ブラウン管の奥から発射されたレーザーの
走査によって画面を構成しています。

液晶ディスプレイ(LCD:Liquid Crystal Display)には、垂直帰線はありません。液晶ディスプレイでは、画面全体がいっぺんに描画されています。各画素は次の画面描き替えが発生するまでその内容を保持するため、ブラウン管のような走査線や垂直帰線は必要ない、というわけです。

スワップチェイン

画面に出力する１シーンのことをフレームと呼びます。このフレームを保持しておくメモリ領域のことをフレームバッファと呼びます。フレームバッファを２枚用意し、その時点でのフレームを描画している間に次のフレームをもう１枚のバッファに書き込むことで、現フレームの描画時に、次のフレーム作成処理が行き詰まるのを防ぐ手法のことをダブルバッファリングと呼び、フレームバッファを３枚用意する手法をトリプルバッファリングと呼びます。



このようにいくつかのフレームバッファを関連付けて、ローテーションさせていくシステムのことをスワップチェイン(フリッピングチェイン)と呼びます。このスワップチェインのなかで選択されているバッファをフロントバッファ、非選択の状態にあるバッファをバックバッファと呼びます。

フロントバッファとバックバッファを瞬時に入れ換える動作をフリップと呼びます。フリップは、実際にはフロントバッファとバックバッファのポインタ(フレームバッファのアドレス)を交換しているだけで、バックバッファの内容を入れ換えたりすることはありません。よって、フリップに掛かる処理負荷は微々たるものでしかありません。

DirectX Graphicsでは、フリップを行うと、フロントバッファとバックバッファのポインタを内部的に交換するので、ディスプレイの描画内容がバックバッファのものに瞬時に入れ換えられます。フリップするタイミングは、ディスプレイの垂直帰線と同期させることが可能で、ちらつきはまったくおこりません。バックバッファが複数枚ある場合は、接続されている順にフリップされます。垂直帰線を無視してフリップさせることもできますが、走査中にフリップを行うと、ディスプレイの途中までが前画面の画像、途中からが次画面の画像になってしまいます。これをティアリングと呼びます。

ティアリング

ティアリングとは、走査線が画面上を走っているときに描画が次のフレームに進んでしまったため、画面上に前後のフレームが混じってしまう現象のことです。以下の例では、走査線が画面の中央付近を走っているときに次のフレームに進んだため、画面上半分は前のフレーム、画面下半分は次のフレームが描画され、表示が乱れてしまった状態です。

前のフレーム

次のフレーム

走査線の位置で前後のフレームが混じって
表示されてしまう現象が「ティアリング」

ディスプレイモード

解像度と色数の組み合わせを、ディスプレイモードと呼びます。ディスプレイモードを表現するときは、ｘ解像度、ｙ解像度、色数の３つの値で記述します。たとえば、640×480×256(または640×480×８)は、画面の幅が640ピクセル、画面の高さが480ピクセル、色数が256色であることを示します。

サポートされるディスプレイモードは、グラフィックカードによってあらかじめ選択肢が決まっています。特殊なディスプレイモードを使用すると、特定の製品でしか動作しなくなってしまいます。

ディスプレイモードに関する一般的な用語を以下に示します。

ピクセル

画面に表示される画像は、ピクセルの集合によって構成されます。画面上で個別に色を指定できる単位をピクセルと呼びます。ピクセルに設定された色が組み合わさることで、画面全体のイメージが形成されます。

ピクセルが使用しているメモリには、画面に表示する色を示す値が格納されています。画面に表示する画像は、グラフィックカード上のメモリ(VRAM：ビデオラム)に配置されます。

解像度

解像度は通常、画面の幅(ｘ方向の解像度)と高さ(ｙ方向の解像度)を使って表現します。解像度は、ｘ×ｙという形式で表現します。たとえば、640×480という解像度は、画面の幅が640ピクセル、高さが480ピクセルで、全部で307,200ピクセルがあることを示します。
使用できる解像度はいくつかありますが､解像度が大きくなれば転送量が多くなり速度が低下します。できるだけ多くのピクセルを使って鮮明な画像を表示するのが理想的です。しかし実際には、見栄えをある程度保ちつつ速度も遅すぎないように、速度と解像度のバランスを考える必要があります。

色数

色数とは、画面に同時に表示できる色の数です。画面の解像度の場合と同じように、選択できる色数は限られています。一般的な色数は256色、65,536色、16,777,216色です。
色数を指定するときは、ピクセルあたりのビット数(bpp:bits per pixel)を使用します。たとえば、ピクセルあたりのビット数が８ならば、使用できる色の数は256色になります($2^8=256$)。
使用する色の数が多いほど、画質は向上します。しかし、色数が増えるほど転送量も多くなるので、速度と色数のバランスを考える必要があります。

パレット

パレットには、使用する色の情報が格納されています。１つ１つの色の情報は数値として、パレットエントリに格納されます。パレットエントリに含まれている色の情報は、赤、緑、青の３つの値で構成され、それぞれに８ビット割り当てられ、16,777,216色表現できます。画面に表示されるときにこれらの値を組み合わせた色が表示されます。
パレットは、通常は256色以下の色数のときに使用され、16,777,216色中最大256色をパレットエントリに選択します。ピクセルは、色値そのものではなく、パレットエントリのインデックス値で表現されます。パレットを操作することで、さまざまな画面効果を作り出すことができます。たとえば、画像を変更することなく、パレットエントリの内容だけを変更して色を変化させることができます。

ハイカラー

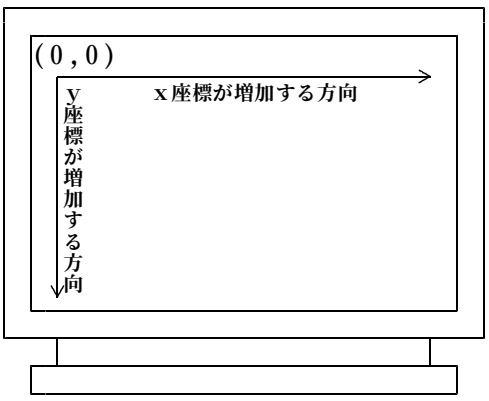
ピクセルあたりのビット数が16の色数をハイカラー(High Color)と呼びます。ハイカラーには２種類あります。ピクセルを赤５ビット、緑５ビット、青５ビットの32,768色で表現する場合と、赤５ビット、緑６ビット、青５ビットの65,536色で表現する場合があります。どちらが使われるかはグラフィックカードにより異なります。

フルカラー

色数が16,777,216色をフルカラー(Full Colorまたはトゥルーカラー:True Color)と呼びます。ピクセルは赤８ビット、緑８ビット、青８ビットで表現されます。１ピクセルは24ビットで表現することができますが、24ビットちょうどを扱う変数の型がないため、ダミーの８ビットを追加して32ビットになる場合があります。どちらになるかはグラフィックカードにより異なります。古い製品では24ビットが多く、最新のものはほとんどが32ビットです。32ビットの方が転送効率がよいため速度が向上する場合がありますが、メモリを多く消費してしまいます。

スクリーン座標

画面上の位置を指定するときは、スクリーン座標で指定します。スクリーン座標は、２つの値(ｘ,ｙ)を順番に記述して表現します。ｘ座標は水平方向の位置を、ｙ座標は垂直方向の位置を指定します。画面の左上角はｘ座標とｙ座標がともに０です。座標の値は右または下へ向かって増加します。
ｘ座標は画面の左から右へ向かって増加し、ｙ座標は画面の上から下へ向かって増加します。



フレームの更新

IDirect3DDevice9::Presentメソッドは、バックバッファをフロントバッファにフリップまたはコピーによって転送し、フレームを更新します。フリップが行われるのは、D3DPRESENT_PARAMETERS構造体のSwapEffectメンバがD3DSWAPEFFECT_FLIPである場合、またはフルスクリーンモードでD3DSWAPEFFECT_DISCARDが指定されている場合です。

```
// フレーム更新(pD3DDeviceは、Direct3DDevice9オブジェクト)
pD3DDevice->Present(NULL, NULL, NULL, NULL);
```

ビューポートの消去

ビューポートは、コンピュータ内部に表現された3D空間をレンダリングターゲット(特に指定しない場合はバックバッファ)のどこに描画するのかを設定するものです。デフォルトでは、レンダリングターゲット全体になっています。
ビューポートには、z値など前回レンダリングに使用した情報が残っている場合があるので、レンダリング前にすべての情報を消去しておきます。
ビューポートの消去は、IDirect3DDevice9::Clearメソッドで行います。このメソッドを呼び出すと、レンダリングターゲットを指定した色で塗りつぶし、zバッファをクリアすることができます。

```
// ビューポート消去
pD3DDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                  D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
```

シーンの開始と終了

DirectX Graphicsでは、レンダリング前に、IDirect3DDevice9::BeginSceneメソッドを呼び出してシーンの開始を宣言し、システムにレンダリングの開始を通知する必要があります。

```
// シーン開始
pD3DDevice->BeginScene();
```

レンダリングがすべて終了した場合は、IDirect3DDevice9::EndSceneメソッドを呼び出してシーンの終了を宣言し、システムにレンダリングの終了を通知します。

```
// シーン終了
pD3DDevice->EndScene();
```

ダイアログボックスの表示

フルスクリーンモードでフリッピングを行う場合、メニューやメッセージボックスを表示させようとすると、表示される場合とされない場合があります。
これは、スワップチェインのフレームのひとつだけが、ウィンドウなどのGDIが書き込める能力を持つためです。GDIが書き込めるフレームがフロントバッファになったときだけ表示されます。
IDirect3DDevice9::SetDialogBoxModeメソッドを呼び出すと、Presentメソッドの動作が変わり、必ずメニューやダイアログボックスが表示されるようになります。

```
// メッセージボックスの表示(hWndはウィンドウのハンドル)
pD3DDevice->SetDialogBoxMode(TRUE);      // ダイアログ表示ON
MessageBox(hWnd, "エラーが発生しました", "情報", MB_ICONSTOP | MB_OK);
pD3DDevice->SetDialogBoxMode(FALSE);     // ダイアログ表示OFF
```

課　題

3Dオブジェクトを描画するために必要な関数を作成しましょう。

(１)CDXGraphics9クラスに、3Dオブジェクトを描画するのに必要な関数を追加します。以下のプロトタイプを適切な場所に追加しましょう。

```
void Clear(const DWORD inFlags = D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, const D3DCOLOR inColor = 0);
                    // ビューポートクリア
```

```
    void UpdateFrame();    // フレーム更新

    bool BeginScene();     // シーン開始
    void EndScene();       // シーン終了
```

Clear関数の「inFlags = D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER」および「inColor = 0」により、引数inFlagsとinColorの指定が省略可能になります。省略した場合、代入されている値が指定されます。このように、デフォルト値が設定されている引数をデフォルト引数と呼びます。

(２)ビューポートの情報を消去するClear関数を実装します。以下のプログラムの足りない部分を補い、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          ビューポートクリア                                         */
/*****************************************************************************************************/
void CDXGraphics9::Clear(const DWORD inFlags, const D3DCOLOR inColor)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_Clear)\n");
        return;
    }
#endif
    m_pD3DDevice->?????(0, NULL, inFlags, inColor, 1.0f, 0);
}
```

(３)バックバッファをプライマリサーフェスに転送し、画面を更新するUpdateFrame関数を実装します。以下のプログラムの足りない部分を補い、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          フレーム更新                                             */
/*****************************************************************************************************/
void CDXGraphics9::UpdateFrame()
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_UpdateFrame)\n");
        return;
    }
#endif

    if(m_pD3DDevice->???????(NULL, NULL, NULL, NULL) != D3D_OK) {
        if(m_pD3DDevice->TestCooperativeLevel() == D3DERR_DEVICENOTRESET)
            Reset();    // フレームの更新に失敗した場合、デバイスをリセットして復帰を試みる
    }
}
```

(４)シーンを開始するBeginScene関数を実装します。以下のプログラムの足りない部分を補い、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          シーン開始                                               */
/*****************************************************************************************************/
bool CDXGraphics9::BeginScene()
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_BeginScene)\n");
        return false;
    }
#endif

    if(m_pD3DDevice->??????????() != D3D_OK) {
        ::OutputDebugString("*** Error - シーン開始失敗(CDXGraphics9_BeginScene)\n");
        return false;
    }

    return true;
}
```

(５)シーンを終了するEndScene関数を実装します。以下のプログラムの足りない部分を補い、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          シーン終了                                               */
/*****************************************************************************************************/
void CDXGraphics9::EndScene()
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_EndScene)\n");
        return;
    }
#endif
    m_pD3DDevice->????????();
}
```

(６)フリップを行ってもメッセージボックスが表示されるように、新しいメッセージボックス関数を作成します。以下のプログラムの足りない部分補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                        メッセージボックス                                         */
/*****************************************************************************************************/
int CDXGraphics9::MessageBox(LPCTSTR lpText, LPCTSTR lpCaption, UINT uType)
{
    int   ret;

    if(m_pD3DDevice != NULL) {
        m_pD3DDevice->SetDialogBoxMode(TRUE);
        ret = ::MessageBox(m_PresentParams.hDeviceWindow, lpText, lpCaption, uType);
        m_pD3DDevice->SetDialogBoxMode(FALSE);
    } else {
        ret = ::MessageBox(::GetDesktopWindow(), lpText, lpCaption, uType);
    }

    return ret;
}
```

※プロトタイプも忘れずに追加しましょう。

(７)CGameApp::OnClose関数を以下のように変更しましょう。

```
LRESULT CGameApp::OnClose(const HWND hWnd, const WPARAM wParam, const LPARAM lParam)
{
    if(DXGraphics().MessageBox("終了しますか？", CGameApp::APP_NAME,
                               MB_ICONQUESTION | MB_YESNO | MB_DEFBUTTON2) == IDYES)
        Release();

    return 0;
}
```