ゲームプログラミング

▶　基礎編－第１１回　フレームレートの制御　◀

ゲームは、一定の間隔で画面を更新する必要があります。毎回異なるタイミングで画面を更新すると、ぎこちない動きになってしまいます。たいていのゲームでは、１秒間に６０回画面を更新しており、１フレームを約16ミリ秒の間隔で描画しています。

フレームレートの制御

ゲームプログラムでは、メインループ処理(フリップの間隔)を一定のタイミングで行う必要があります。そうしないと、ゲームのスピードが処理時間や環境により異なってしまい、速い環境ほど高速に動作してしまいます。

１秒間に何回画面を更新するのかをフレームレート(FPS:Frame Per Second)と呼びます。フレームレートを設定し、これにあわせてゲームを動かします。たいていのゲームでは６０FPSで、１フレームを約16.67ミリ秒間隔で処理しています。

フレームレートは、DirectDrawやDirectX Graphicsで設定できるようになっていますが、設定するとエラーになってしまうことが多いので、プログラムで実現する必要があります。

時間を取得するAPI

フレームレートを制御するには、処理にかかった時間を取得し、次のフレーム処理を開始するタイミングを取る必要があります。時間を取得するAPIはいくつかあり、それぞれ精度が異なります。これらの関数のほかに、指定した時間だけプログラムを休止し、CPU使用権をほかのアプリケーションに渡すSleep関数があります。これらを組み合わせてフレームレートを制御します。

○GetTickCount関数

システムを起動した後の経過時間を、ミリ秒単位で取得します。精度は、システムタイマの分解能による制限を受けます。システムタイマの分解能を取得するには、GetSystemTimeAdjustment関数を使います。経過時間はDWORD型で保存されており、システムを約49.71日間連続して動作させると、経過時間は0に戻ります。

○timeGetTime関数

システムを起動した後の経過時間を、ミリ秒単位で取得します。精度は、OSによって異なります。精度を取得するには、timeGetDevCaps関数を使います。経過時間はDWORD型で保存されており、システムを約49.71日間連続して動作させると、経過時間は0に戻ります。

WindowsNT/2000/XPでは、timeGetTime関数の精度は、環境によっては５ミリ秒以上になる場合があります。timeGetTime関数の精度は、timeBeginPeriod関数とtimeEndPeriod関数を使って設定することができます。

Windows95/98/Meでは、timeGetTime関数の精度は１ミリ秒です。timeBeginPeriod関数とtimeEndPeriod関数でどのような精度に設定しても、１ミリ秒の精度を変更することはできません。

なお、timeGetTime関数は、ライブラリ"winmm.lib"が必要になります。

○QueryPerformanceCounter関数

高分解能パフォーマンスカウンタが存在する場合、そのカウンタの現在の値を取得します。パフォーマンスカウンタは0.838マイクロ秒の分解能を持ち、カウンタが１周するまで約700,000年かかります。

高分解能パフォーマンスカウンタの周波数(１秒あたりのカウント数)を取得するには、QueryPerformanceFrequency関数を使います。

○ 課題 ○

方法１から方法７の中から適切なものを選んでフレームレートの制御を行い、これにあわせてゲームが動くようにプログラムを変更しましょう。

○方法１
　ゲーム処理のメインループの最後で、Sleep関数を使い一定時間休止し、速度を調整する方法です。

```
void TestProc()
{
    // ゲームのメインループ処理
    (省略)

    DDFlip();

    Sleep(15);    // 休止
}
```

○方法２
　１フレームを処理するのにかかった時間を

　　フレーム終了時間 － フレーム開始時間 ＝ １フレームを処理するのにかかった時間

という式で求めます。休止時間は以下の式で求め、Sleep関数で休止します。

　　１フレームにかけられる時間 － １フレームを処理するのにかかった時間 ＝ 休止時間
　　(あまった時間が負の場合は、処理落ちになります)

　時間の取得には、GetTickCount関数を使います。

－追加１－

```
/*****************************************************************************************************/
/*                                          定　　数                                                 */
/*****************************************************************************************************/
#define FPS          60                     // １秒間に画面を更新する回数。フレーム／秒
#define FRAME_DIRAY ((DWORD)(1000 / FPS))   // フレーム周期。１フレームあたりの最大処理時間。
                                            // １フレームの処理時間が、この値を越えると処理落ちします
```

－追加２－

```
/*****************************************************************************************************/
/*                                        グローバル変数                                             */
/*****************************************************************************************************/
static DWORD   g_dwStartTime = 0;   // フレーム開始時間
```

－追加３－

```
/*****************************************************************************************************/
/*                                       フレーム構築開始                                            */
/*****************************************************************************************************/
void BeginFrame()
{
    g_dwStartTime = GetTickCount();
}
```

－追加４－

```
/*****************************************************************************************************/
/*                                       フレーム構築終了                                            */
/*****************************************************************************************************/
void EndFrame()
{
    // 休止時間を計算
    const long   lWaitTime = FRAME_DIRAY - (GetTickCount() - g_dwStartTime);

    if(0 < lWaitTime)
        Sleep((DWORD)lWaitTime);    // あまった時間は休止
}
```

－使用例－

```
void TestProc()
{
    BeginFrame();      // フレーム開始

    // ゲームのメインループ処理
    (省略)

    DDFlip();

    EndFrame();        // フレーム終了
}
```

○方法３
　方法２のGetTickCount関数をtimeGetTime関数に変更したものです。

```
void BeginFrame()
{
    g_dwStartTime = timeGetTime();
}


void EndFrame()
{
    // 休止時間を計算
    const long   lWaitTime = FRAME_DIRAY - (timeGetTime() - g_dwStartTime);

    if(0 < lWaitTime)
        Sleep((DWORD)lWaitTime);    // あまった時間は休止
}
```

○方法４
　１フレームを処理するのにかかった時間を

　　フレーム終了時間 － 前回のフレーム終了時間

という式で求めます。休止は方法２と同じ方法で行い、時間の取得にはGetTickCount関数を使います。

－追加１－

```
/*****************************************************************************************************/
/*                                          定　　数                                                 */
/*****************************************************************************************************/
#define FPS         60
#define FRAME_DIRAY ((DWORD)(1000 / FPS))
```

－追加２－

```
/*****************************************************************************************************/
/*                                      フレームレート制御                                           */
/*****************************************************************************************************/
void WaitFrame()
{
    static DWORD   dwLastTime = 0;  // 終了時間

    // ウェイト
    const long   lWaitTime = FRAME_DIRAY - (GetTickCount() - dwLastTime);
    if(0 < lWaitTime)
        Sleep((DWORD)lWaitTime);

    dwLastTime = GetTickCount();    // 終了時間を保存
}
```

－使用例－

```
void TestProc()
{
    // ゲームのメインループ処理
    (省略)

    DDFlip();

    WaitFrame();       // フレームレート制御
}
```

○方法５
　方法４のGetTickCount関数をtimeGetTime関数に変更したものです。

```
void WaitFrame()
{
    static DWORD   dwLastTime = 0;  // 終了時間

    // ウェイト
    const long   lWaitTime = FRAME_DIRAY - (timeGetTime() - dwLastTime);
    if(0 < lWaitTime)
        Sleep((DWORD)lWaitTime);
```

```
    dwLastTime = timeGetTime();      // 終了時間を保存
}
```

○方法６

QueryPerformanceCounter関数で時間を取得し、１ミリ秒未満の休止時間がある場合は、ビジーループで時間を消費する方法です。

ビジーループとは、以下のような「何もしない」ループのことです。

```
while(true)
    ;    // なにもしない
```

ビジーループを長時間行うと、CPUを猛烈に消費し、Windowsが不安定になってしまいます。しかし、１ミリ秒に満たない時間を休止する場合は、このループでCPUを浪費させて休止します。

－追加１－

```
/*****************************************************************************************************/
/*                                          定　　数                                                 */
/*****************************************************************************************************/
static const DWORD    FPS         = 60;              // フレーム数
static const double   FRAME_DIRAY = 1000.0 / FPS;    // １フレーム周期
```

－追加２－

```
/*****************************************************************************************************/
/*                                        グローバル変数                                             */
/*****************************************************************************************************/
static double    g_dFrequency  = 0.0;    // パフォーマンス周波数
```

－追加３－

ゲームを初期化したときに、パフォーマンス周波数を初期化します。

```
    // パフォーマンス周波数設定
    __int64   n64Frequency;
    if(0 == QueryPerformanceFrequency((LARGE_INTEGER*)&n64Frequency))
        return false;
    g_dFrequency = n64Frequency / 1000.0;    // ミリ秒に変換
```

－追加４－

```
/*****************************************************************************************************/
/*                                       フレームレート制御                                          */
/*****************************************************************************************************/
void WaitFrame()
{
    static __int64   n64LastTime = 0;    // 終了時間

    // 休止時間取得
    __int64   n64CurrentTime;
    QueryPerformanceCounter((LARGE_INTEGER*)&n64CurrentTime);
    double   dWaitTime = FRAME_DIRAY - (n64CurrentTime - n64LastTime) / g_dFrequency;
    if(0.0 <= dWaitTime) {
        // 整数部休止
        Sleep((DWORD)dWaitTime);

        // 小数部休止
        __int64   n64CurrentTime2;
        QueryPerformanceCounter((LARGE_INTEGER*)&n64CurrentTime2);
        dWaitTime -= (n64CurrentTime2 - n64CurrentTime) / g_dFrequency;
        do {
            QueryPerformanceCounter((LARGE_INTEGER*)&n64LastTime);
        } while((n64LastTime - n64CurrentTime2) / g_dFrequency < dWaitTime);
    } else {
        // 現時間取得
        QueryPerformanceCounter((LARGE_INTEGER*)&n64LastTime);
    }   // if(dWaitTime)
}
```

○方法７
　方法６を変更し、すべての休止時間をSleep(0)のループで消費して休止する方法です。Sleep関数に０を指定すると、ほかのアプリケーションにCPU使用権を譲ります。

```
void WaitFrame()
{
    static __int64   n64LastTime = 0;        // 終了時間

    // 待機時間取得
    __int64   n64CurrentTime;
    QueryPerformanceCounter((LARGE_INTEGER*)&n64CurrentTime);
    const double   WAIT_TIME = FRAME_DIRAY - (n64CurrentTime - n64LastTime) / g_dFrequency;

    // ループで時間を消費
    do {
        Sleep(0);
        QueryPerformanceCounter((LARGE_INTEGER*)&n64LastTime);
    } while((n64LastTime - n64CurrentTime) / g_dFrequency < WAIT_TIME);
}
```

・付録
　以下のDrawFPS関数は、フレームレートを表示する関数です。DDFlip関数の直前に実行すると、画面左上にフレームレートを描画します(DDUtilsのDDGetDC関数とDDReleaseDC関数が必要です)。

```
/*****************************************************************************************************/
/*                                          FPS描画                                                  */
/*****************************************************************************************************/
void DrawFPS()
{
    static int     nFPS       = 0;  // FPS
    static int     nFPSCnt    = 0;  // フレームカウンタ
    static DWORD   dwLastTime = 0;  // 終了時間

    nFPSCnt++;
    const DWORD   dwTime = timeGetTime();
    if(dwTime - dwLastTime >= 1000) {
        nFPS       = nFPSCnt;
        nFPSCnt    = 0;
        dwLastTime = dwTime;
    }

    // FPS描画
    HDC   hDC = DDGetDC(DDS_BACKBUF);
    if(NULL != hDC) {
        TCHAR   szMes[8];
        wsprintf(szMes, "FPS:%3d", nFPS);
        TextOut(hDC, 0, 0, szMes, lstrlen(szMes));
        DDReleaseDC(DDS_BACKBUF);
    }
}
```