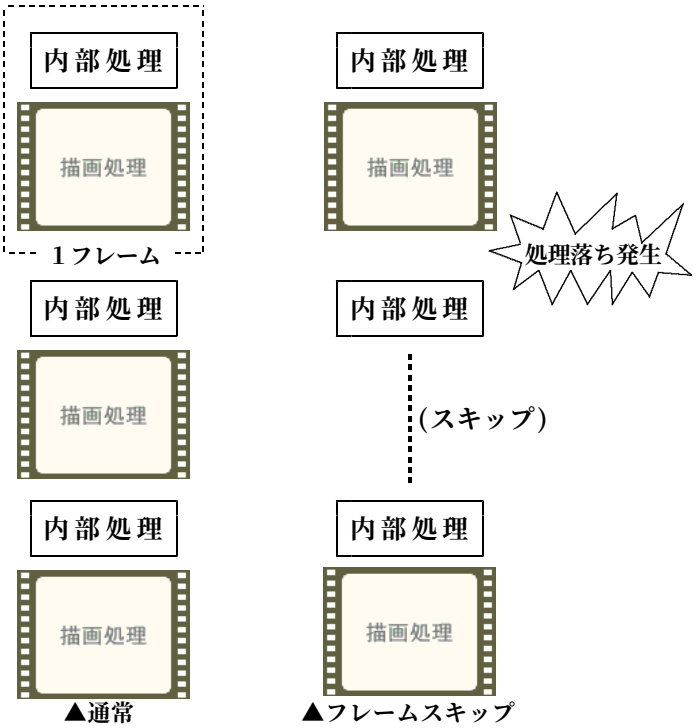
オブジェクト指向と
ゲームプログラミング

▷　フレームワーク編 － 第１５回　フレームスキップ　◁

フレームスキップ

PCでは、速いものから遅いものまでさまざまな環境が考えられます。遅い環境の場合、プログラムによってはあるフレームの処理が規定時間内に終了せず、「処理落ち」になる場合があります。処理落ちすると、プログラムの進行が遅くなってしまいます。遅い環境でも、速い環境と同じような速度でプログラムを進行させたい場合があります。このような場合、フレームスキップという方法を導入します。

フレームスキップとは、描画処理の負荷により、あるフレームの処理が規定時間内に終了できなかった場合、遅れた時間を取り戻すまで以降の描画処理を省略するという方法です。処理時間のほとんどが描画処理に掛かる時間です。特に、3Dではその傾向が強くなります。描画処理を省略すれば、かなりの高速化が期待できます。フレームスキップを導入すれば、遅い環境でも(いくつかのフレームは描画されなくなりますが)速い環境と同じ速度でプログラムを動作させることができます。



たとえば、60FPSのタイミングでプログラムを進行させる場合、１フレームあたりの処理時間は1/60秒(16.67ミリ秒)以内に終了しなければなりません。しかし、あるフレームを処理した際、1/60秒を超えてしまった場合には、「処理落ち」となってしまいます。フレームスキップでは、以降のフレームの描画処理をすべて省略します。省略するのは描画処理だけで、それ以外の内部処理(キャラクターの移動や衝突検出など)は通常どおり行います。本来の時間に追いつくまで描画処理を省略し続けます。こうすれば、どのような環境でも同じ速度でプログラムを動作させられるというわけです。

課　　題

フレームスキップを導入する場合、以下のCFPSTimerクラスを作成しましょう。

(１)FPSの制御を行うクラスCFPSTimerを作成します。CFPSTimerクラスのヘッダファイル(FPSTimer.hpp)を以下のように作成しましょう。

－FPSTimer.hpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
     Microsoft Windows2000/XP

  【コンパイラ】
     Microsoft Visual C++ 2005

  【プログラム】
     FPSTimer.hpp
                FPS制御タイマクラスヘッダ

  【履歴】
     * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

#pragma once

/*********************************************************************************************/
/*                                        インクルードファイル                                 */
/*********************************************************************************************/
#include <windows.h>

/*********************************************************************************************/
/*                                      FPS制御タイマクラス定義                                */
/*********************************************************************************************/
class CFPSTimer {
public:
    virtual ~CFPSTimer();

    void Wait();
    void Reset() { ※(４) }

    bool IsSkip() const { return (m_SurplusTime < 0) ? true : false; }

    void SetFPS(const UINT inFPS) { ※(５) }
    UINT GetFPS() const { return m_FPS; }

    void DrawFPS(const HDC hDC, const int inX, const int inY);

    // シングルトンインスタンスの取得
    static CFPSTimer& GetInstance()
    {
        ?????? CFPSTimer   theFPSTimer;
        return theFPSTimer;
    }

private:
    CFPSTimer();
    bool Initialize();

    UINT      m_FPS;            // １秒あたりのフレーム数
    __int64   m_Frequency;      // パフォーマンスカウンタ周波数
    __int64   m_MaxWait;        // 最大休止時間

    __int64   m_LastTime;       // 前フレーム終了時間
    __int64   m_SurplusTime;    // 余剰時間
```

```
    UINT        m_PeriodMin;        // マルチメディアタイマ最小分解能

    // フレームレート計測用変数
    int         m_DrawCount;        // 描画数
    int         m_SkipCount;        // スキップ数
    int         m_DrawFPS;          // フレームレート描画用
    int         m_DrawSkip;         // スキップ数描画用
    DWORD       m_LastDraw;         // 前描画時間
};

/*****************************************************************************************************/
/*                                          マクロ定義                                               */
/*****************************************************************************************************/
inline CFPSTimer& FPSTimer() { return CFPSTimer::GetInstance(); }
```

(２)CFPSTimerクラスのメンバは、以下のとおりです。

CFPSTimer プライベートなメンバ：コンストラクタ

CFPSTimerオブジェクトを構築します。

~CFPSTimer デストラクタ

CFPSTimerオブジェクトを解放します。

GetInstance クラス関数：アクセス関数

アプリケーションで唯一のCFPSTimerオブジェクトを取得します。

書式 `static CFPSTimer& GetInstance();`

Return CFPSTimerオブジェクトの参照

Wait 待機

FPSを安定させるための待機関数です。

Reset 設定

タイマをリセットします。

IsSkip 判定

描画処理を省略すべきどうかを判定します。

書式 `bool IsSkip() const;`

Return 描画省略：true　それ以外：false

SetFPS 設定

１秒あたりのフレーム数を設定します。

書式 `void SetFPS(const UINT inFPS);`

inFPS FPS

GetFPS アクセス関数

CFPSTimerオブジェクトに設定されているFPSを取得します。

書式 `UINT GetFPS() const;`

Return FPS

DrawFPS 描画

１秒あたりの描画数、スキップ数を指定されたデバイスコンテキストに描画します。

書式 `void DrawFPS(const HDC hDC, const int inX, const int inY);`

hDC デバイスコンテキストのハンドル
inX 描画先のｘ座標
inY 描画先のｙ座標

Initialize プライベートなメンバ：初期化

CFPSTimerオブジェクトが使用するタイマの初期設定を行います。

書式 `bool Initialize();`

Return 成功：true　それ以外：false

m_FPS メンバ変数

１秒あたりのフレーム数です。

書式 `UINT m_FPS;`

m_Frequency　メンバ変数

高分解能パフォーマンスカウンタの周波数です。

```
__int64 m_Frequency;
```

m_MaxWait　メンバ変数

１フレームあたりの最大休止時間です。

```
__int64 m_MaxWait;
```

m_LastTime　メンバ変数

前フレームの終了時間です。

```
__int64 m_LastTime;
```

m_SurplusTime　メンバ変数

「最大休止時間 － フレームを処理するのに掛かった時間」を累積した値を保持します。

```
__int64 m_SurplusTime;
```

m_PeriodMin　メンバ変数

マルチメディアタイマの最小分解能を保持します。

```
UINT m_PeriodMin;
```

m_DrawCount　メンバ変数

フレームレートの描画に使用する、描画数を保持する変数です。

```
int m_DrawCount;
```

m_SkipCount　メンバ変数

フレームレートの描画に使用する、描画省略数を保持する変数です。

```
int m_SkipCount;
```

m_DrawFPS　メンバ変数

フレームレートの描画に使用する、実際に描画される描画数を保持する変数です。

```
int m_DrawFPS;
```

m_DrawSkip　メンバ変数

フレームレートの描画に使用する、実際に描画される描画省略数を保持する変数です。

```
int m_DrawSkip;
```

m_LastDraw　メンバ変数

フレームレートの描画に使用する、前回の描画時点の時間を保持する変数です。

```
DWORD m_LastDraw;
```

(３)CFPSTimerクラスのソースファイル(FPSTimer.cpp)を以下のように作成しましょう。

－FPSTimer.cpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
    Microsoft Windows2000/XP

  【コンパイラ】
    Microsoft Visual C++ 2005

  【プログラム】
    FPSTimer.cpp
                FPS制御タイマクラス

  【履歴】
    * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

/*********************************************************************************************/
```

```
/*                                    インクルードファイル                                    */
/*****************************************************************************************/
#include ※ここは各自考えましょう

/*****************************************************************************************/
/*                                    静的リンクライブラリ                                    */
/*****************************************************************************************/
#pragma comment(lib, "winmm.lib")

/*****************************************************************************************/
/*                                    デフォルトコンストラクタ                                */
/*****************************************************************************************/
CFPSTimer::CFPSTimer()
    : m_FPS(0), m_Frequency(0), m_MaxWait(0), m_LastTime(0), m_SurplusTime(0), m_PeriodMin(0),
      m_DrawCount(0), m_SkipCount(0), m_DrawFPS(0), m_DrawSkip(0), m_LastDraw(0)
{
    Initialize();
    SetFPS(60);
    Reset();
}

/*****************************************************************************************/
/*                                    デストラクタ                                          */
/*****************************************************************************************/
CFPSTimer::~CFPSTimer()
{
    if(m_PeriodMin != 0)
        ::timeEndPeriod(m_PeriodMin);
}

/*****************************************************************************************/
/*                                    タイマ初期化                                          */
/*****************************************************************************************/
bool CFPSTimer::Initialize()
{
    // マルチメディアタイマ能力取得
    TIMECAPS   tc;
    ::timeGetDevCaps(&tc, sizeof(tc));
    m_PeriodMin = tc.wPeriodMin;

    // マルチメディアタイマ分解能設定
    ::timeBeginPeriod(m_PeriodMin);

    // パフォーマンスカウンタ周波数取得
    ::QueryPerformanceFrequency((LARGE_INTEGER*)&m_Frequency);

    return true;
}

/*****************************************************************************************/
/*                                    待　機                                                */
/*****************************************************************************************/
void CFPSTimer::Wait()
{
    // 現時間取得
    ※(6)

    // 待機、超過時間設定
    __int64   wait_time = m_MaxWait - (current_time - m_LastTime);
    m_SurplusTime += wait_time;          // 余剰時間の累積
    if(m_SurplusTime < 0) {
        wait_time = 0;                   // 余剰時間が負の場合は追いつくまでウェイトしない
        m_SkipCount++;
    } else {
        if(wait_time != m_SurplusTime)
            wait_time = m_SurplusTime;   // 追いついた場合のウェイト調整
        m_SurplusTime = 0;
        m_DrawCount++;
    }

    // 待機
    ※(7)
}
```

```
/*****************************************************************************************************/
/*                                          フレームレート描画                                       */
/*****************************************************************************************************/
void CFPSTimer::DrawFPS(const HDC hDC, const int inX, const int inY)
{
    const DWORD   CURRENT_TIME = ※(８);                          // 現時間
    const DWORD   ELAPSED_TIME = CURRENT_TIME - m_LastDraw;      // 経過時間
    if(ELAPSED_TIME >= 1000) {
        m_DrawFPS   = m_DrawCount * 1000 / ELAPSED_TIME;
        m_DrawSkip  = m_SkipCount * 1000 / ELAPSED_TIME;
        m_DrawCount = 0;
        m_SkipCount = 0;
        m_LastDraw  = CURRENT_TIME;
    }

    // FPS描画
    TCHAR   info[32];
    ::wsprintf(info, "FPS:%3d Skip:%3d", m_DrawFPS, m_DrawSkip);
    ::TextOut(hDC, inX, inY, info, ::lstrlen(info));
}
```

(４)Reset関数を実装します。以下のうち適切なものを選びましょう。

```
void Reset() { m_LastTime = ::GetTickCount(); m_SurplusTime = 0; }
void Reset() { m_LastTime = ::timeGetTime();  m_SurplusTime = 0; }
void Reset() { ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime); m_SurplusTime = 0; }
```

(５)SetFPS関数を実装します。以下のうち適切なものを選びましょう。

```
void SetFPS(const UINT inFPS) {m_FPS = inFPS; m_MaxWait = 1000 / inFPS; }
void SetFPS(const UINT inFPS) {m_FPS = inFPS; m_MaxWait = m_Frequency / inFPS; }
```

(６)以下のうち適切なものを選んで「※(６)」に入れましょう。

・６－１

```
__int64   current_time = ::GetTickCount();   // 現時間取得
```

・６－２

```
__int64   current_time = ::timeGetTime();    // 現時間取得
```

・６－３

```
// 現時間取得
__int64   current_time;
::QueryPerformanceCounter((LARGE_INTEGER*)&current_time);
```

(７)以下のうち適切なものを選んで「※(７)」に入れましょう。

・７－１

```
// 待機
if(wait_time > 0)
    ::Sleep((DWORD)wait_time);

m_LastTime = ::GetTickCount();      // 終了時間の設定
```

・７－２

```
// 待機
if(wait_time > 0)
    ::Sleep((DWORD)wait_time);

m_LastTime = ::timeGetTime();       // 終了時間の設定
```

・７－３

```
// 待機処理
if(wait_time > 0) {
    // Sleepによる休止(wait_timeのパフォーマンス秒をミリ秒に変換して休止)
    ::Sleep((DWORD)(wait_time * 1000 / m_Frequency));

    // ミリ秒未満の休止
```

```
        __int64   current_time2;
        ::QueryPerformanceCounter((LARGE_INTEGER*)&current_time2);
        wait_time -= current_time2 - current_time;
        do {
            ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime);
        } while(wait_time > m_LastTime - current_time2);
    } else {
        // 現時間取得
        ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime);
    }
```

・７－４

```
    // ループで待機時間を消費
    do {
        ::Sleep(0);
        ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime);
    } while(wait_time > m_LastTime - current_time);
```

(８)「※(８)」に「::GetTickCount()」または「::timeGetTime()」のうち適切なものを入れましょう。

(９)アプリケーションがアクティブ状態に復帰した場合に、CFPSTimerオブジェクトをリセットする必要があります。リセットしないとアプリケーションが休止されていた時間に比例してかなりのフレームがスキップされる場合があります。
CGameApp::OnActivateApp関数に、CFPSTimerオブジェクトをリセットする処理を追加しましょう。

(10)メインループのフレームレート制御処理をCFPSTimerクラスに変更します。CGameApp::Main関数を以下のように変更しましょう

```
int CGameApp::Main()
{
    // メインループ
    while(true) {
        // メッセージ処理
        if(MessageLoop() == false)
            break;
        // ゲーム処理
        if(m_GameProc.???(m_Active) == false)
            Release();          // アプリケーション終了
        // ウェイト
        FPSTimer().Wait();
    }

    return 0;                   // 正常終了
}
```

(11)CGameApp::Initialize関数に以下のプログラムを追加しましょう。

```
FPSTimer().SetFPS(60);      // FPS設定
```

(12)CTestScene::CTestScene関数の最後に以下のプログラムを追加しましょう。

```
FPSTimer().Reset();         // タイマリセット
```

(13)CTestScene::ActiveProc関数を以下のように変更しましょう。

```
int CTestScene::ActiveProc()
{
    // 内部処理

    // 描画処理
    if(FPSTimer().IsSkip() == true)
        return 0;

    // ここに、機能テストの描画処理を記述します

    return 0;
}
```

応用問題　CFixTimerクラスとCFPSTimerクラスの基底クラスCTimerを作成しましょう。