MoCam Middleware Software Development Kit for NINTENDO 3DS Programming Manual

2011-05-25

Version 1.0.2

任天堂株式会社発行

本ドキュメントの内容は、機密情報であるため、厳重な取り扱い、管理を行ってください。

目次

コード

1 本書について

本書は MoCam SDK for CTR API を使用した基本プログラミングについて説明するプログラミングガイドです。

本書の内容は C/C++ プログラミングに精通した読者を想定しています。

MoCam SDK API の詳細な仕様については、「HTML 関数リファレンスマニュアル」を参照してください。

“CTR-MoCam_SDK_Quickstart_Guide-ja_JP.pdf” をまだ参照していない場合は、これに目を通しておくことを推奨します。MoCam SDK for CTR パッケージの概要とそのインストール手順について説明しています。

2 概要

2.1 はじめに

MoCam SDK API は、C99 規格に準拠した C API と C++ API から構成されています。

C と C++ API の機能は完全に整合しています。

この 2 つの API 構文の同等性についても後述しています。

2.2 構文の考え方

MoCam SDK API は NINTENDO CTR SDK API と同じ命名規則を採用しています。

C++ API は名前空間を使用して、オブジェクト、関数、型を分類しています。

mw::mo 名前空間は Mobiclip ミドルウェア SDK のルートとなる名前空間です。

C API では、名前空間の文字列をすべて連結して接頭辞変換を行います。

コード 2-1 MoCam SDK C++ API 使用例

```
mw::mo::mocam::autofocus::Handle aHandle;
aHandle.Initialize(myMalloc,myFree);
aHandle.Calculate(src0,src1,640,480,640*2,128);
```

コード 2-2 MoCam SDK C API 使用例

```
mwmomocamautofocusHandle* aHandle;
mwmomocamautofocusHandleInitialize(&cHandle,myMalloc,myFree);
mwmomocamautofocusHandleCalculate(&aHandle,src0,src1,640,480,640*2,128,false);
```

最初のパラメータとして MoCam オブジェクトハンドルを使用する C 関数は、C++ API の対応する MoCam オブジェクトメソッドに置き換えられます。
また、C API ではアドレスで提供または取得される出力パラメータが、C++ API ではリファレンスで提供または取得されます。

2.3 モジュールの説明

以下は MoCam SDK API で使用可能な名前空間別のモジュール一覧です。これらのモジュールの多くは、アプリケーションを正しくビルドするためにそれぞれの `OMakefile` プロジェクトに含める必要のあるライブラリファイルです。

2.3.1 名前空間 mw::mo::mocam

この名前空間は MoCam API から返されるエラーコードとメモリインターフェイスで使用する typedef のみを定義します。

2.3.2 名前空間 mw::mo::mocam::autofocus

2.3.2.1 mw::mo::mocam::autofocus::Handle クラス

このクラスは 2 つのステレオ画像(通常、左右の CTR カメラでキャプチャされた画像)の 3D フォーカスをオートマチックに調整するために使用します。

2.2 節で説明したように、このクラスのすべてのメソッドは `mwmomocamautofocusHandle` 接頭辞をもち、第一パラメータがハンドルである C と等しい関数が存在します。

`OMakefile` プロジェクトで使用するライブラリ名は `libmw_mo_MoCam.(option).a` です。

2.3.3 名前空間 mw::mo::helper

この名前空間には、カメラや Y2R 管理など開発者にとって役立つ機能を実装した C++ の各種クラスが含まれます。

これらのクラスは、MoCam SDK API には含まれていません。これらは SDK を備えたサンプルコードで使用されるアプリケーション主導型クラスであり、そのソースコードを使用可能です。

これらのクラスは、開発者が各自の目的に合わせて使用したり独自に書き直したりすることもできます。

3 MoCam SDK の使用ガイド

3.1 オートフォーカス

MoCam SDK のオートフォーカス機能を使用するには、mocam_Autofocus.h ファイルをインクルードし、そのパスを OMakefile プロジェクトの INCLUDE セクションに追加する必要があります。

コード 3-1 mocam_Autofocus.h ファイルをソースコードにインクルード

```
#include "mocam_Autofocus.h"
```

MoCam SDK API 使用法の追加情報については、MoCam SDK サンプルコードを参照してください。

3.1.1 オートフォーカスの初期化と終了処理

mw::mo::mocam::autofocus::Handle オブジェクトのメソッドを呼び出す際、事前に Initialize メソッドを呼び出す必要があります。このメソッドは、オブジェクトのメモリ割り当て/割り当て解除のための関数を備えています。

コード 3-2 MoCam オートフォーカス初期化の例

```
extern void* myMalloc (size_t size); //The memory allocation function
extern void  myFree   (void* ptr);   //The memory de-allocation function

mw::mo::mocam::autofocus::Handle aHandle;
aHandle.Initialize(myMalloc,myFree);
```

内部メモリを割り当て解除して使用を終了するときには、オートフォーカスオブジェクトの Finalize メソッドを呼び出します。それ以外では、このメソッドは必要に応じて C++ デストラクタに呼び出されます。

コード 3-3 MoCam オートフォーカス終了処理

```
aHandle.Finalize();
```

3.1.2 2 つの画像におけるフォーカスの計算

2 つの画像のフォーカスを計算するには、mw::mo::mocam::autofocus::Handle::Calculate 関数を使用します。

一旦フォーカスが計算されると、1 番目の画像の左カラムと 2 番目の画像の右カラムがクロッピングに使用されます。つまり、最大の可能範囲は通常、画面の解像度とカメラキャプチャの解像度に依存します。

下の例では、2 つの CTR カメラでキャプチャした画像におけるフォーカス計算を呼び出して、0 から CAMERA_WIDTH-SCREEN_WIDTH までの範囲をもつフォーカス値を計算するためのオートフォーカスを要求します。つまり、両方の画像を最大の CAMERA_WIDTH-SCREEN_WIDTH ピクセルまでをクロップし、SCREEN_WIDTH ピクセルを残して結果を表示できるようにしています。

コード 3-4 MoCam オートフォーカス計算の例

```
aHandle.Calculate(
        camera.GetImageChannel(0),
        camera.GetImageChannel(1),
        CAMERA_WIDTH,CAMERA_HEIGHT,CAMERA_WIDTH*2,
        CAMERA_WIDTH-SCREEN_WIDTH);
```

3.1.3 オートフォーカス結果の取得

一旦フォーカスが計算されると、mw::mo::mocam::autofocus::Handle::Get を呼び出して結果を取得することができます。

フォーカスが“不安定”および“画像が跳ぶ”状態を避けるために、この関数はユーザにダンピングファクタを使用して時間に対する早いフォーカス変化を抑えるようにします。

コード 3-5 MoCam フォーカス取得の例

```
f32 focus;
cHandle.Get(1.5f,focus);
```

一旦フォーカスが取得されると、1 番目の画像の並進移動にその値を足し、2 番目の画像の並進移動からその値を引くことによって直接 OpenGL 行列に使用します。

3.1.4 上級向けの使用法

入力画像の解像度と計算範囲によって、フォーカス計算は CPU 時間を使用しすぎる可能性もあります。

一部の用途では問題ありませんが、他の場合では問題が生じる場合があります。

この欠点を克服するために、バックグラウンドのスレッドでフォーカス計算を呼び出し、mw::mo::mocam::autofocus::Handle::SetCallback を使って計算に使用される CPU 時間を制御できます。

コード 3-6 MoCam オートフォーカス SetCallback

```
static void AutofocusThread(mw::mo::mocam::autofocus::Handle* aHandle)
{
  for(;;)
          aHandle->Calculate(
                  camera.GetImageChannel(0),
                  camera.GetImageChannel(1),
                  CAMERA_WIDTH,CAMERA_HEIGHT,CAMERA_WIDTH*2,
                  CAMERA_WIDTH-SCREEN_WIDTH);
}

static void AutofocusCallback(void* param)
{
}

static void AutofocusExample(mw::mo::mocam::autofocus::Handle* aHandle)
{
  aHandle->SetCallback(AutofocusCallback,0);

  nn::os::Thread autofocusThread;
  autofocusThread.StartUsingAutoStack(AutofocusThread,aHandle,4096,16);
}
```

スレッドができたので、呼び戻しの中で与えられた期間だけスレッドをスリープさせることが可能です。これは、フォーカス計算を複数のビデオフレームに“拡散”させます。

下の例は、読みやすくするために、スレッド間の同期制御の部分を除いています。

コード 3-7 MoCam オートフォーカススレッドの使用法

```
static void AutofocusThread(mw::mo::mocam::autofocus::Handle* aHandle)
{
  for(;;)
  {
  // Thread synchronization
  […]
          aHandle->Calculate(
                  camera.GetImageChannel(0),
                  camera.GetImageChannel(1),
                  CAMERA_WIDTH,CAMERA_HEIGHT,CAMERA_WIDTH*2,
                  CAMERA_WIDTH-SCREEN_WIDTH);
  }
}

static void AutofocusCallback(void* param)
{
  u32 sleepTime=*(u32*)param;
  nn::os::Thread::Sleep(nn::fnd::TimeSpan::FromMilliSeconds(sleepTime));
}

static void AutofocusExample(mw::mo::mocam::autofocus::Handle* aHandle)
{
  // Make the thread sleep during 30 ms
  aHandle->SetCallback(AutofocusCallback,30);

  nn::os::Thread autofocusThread;
  autofocusThread.StartUsingAutoStack(AutofocusThread,aHandle,4096,16);

  for(;;)
  {
  // Camera capture
  […]
          f32 focus;
  aHandle->Get(1.5f,focus);
  // Display the image centered
          tx0=CAMERA_WIDTH-SCREEN_WIDTH+focus;
  tx1=CAMERA_WIDTH-SCREEN_WIDTH-focus;
  Display(camera.GetImageChannel(0),tx0,
          camera.GetImageChannel(1),tx1);
  }
}
```

3.1.5 ある値にフォーカスを強制

mw::mo::mocam::autofocus::Handle::Get から返されるフォーカスは内部の値を変換したものなので、場合によっては、内部のフォーカス値を強制的にある値に置き換えられることが重要です。

例えば、ユーザがオートフォーカス計算をオフにして、しばらく経過した後に（まったく異なる画像で）またオンにしたい場合、内部のフォーカス値をゼロにリセットできると便利です。そうするには mw::mo::mocam::autofocus::Handle::Set 関数を使用します。

コード 3-8 オートフォーカス Set 関数の使用例

```
f32 focus;
if (autofocusEnabled)
aHandle->Get(1.5f,focus);
else
{
focus=0.f;
aHandle->Set(0.f);
}
```

3.1.6 使用上の注意事項

LCD 画面上に表示する画像は StereoCameraCalibrationData を使用して補正を行ったカメラ画像です。OUT カメラから得られた生のカメラ画像を MoCam に渡して算出したオフセット値をそのまま使用しますと表示に際して若干の誤差が生じますのでご注意ください。

改訂履歴

版	改訂日	分類	改訂内容
1.0.2	2011-05-25	-	・Mocam を MoCam に統一。
1.0.1	2011-05-13	-	・いくつかの誤字を修正。
1.0.0	2011-03-15	-	・初版

記載されている会社名、製品名等は、各社の登録商標または商標です。