MoCam Middleware Software Development Kit for NINTENDO 3DS Programming Manual

Version 1.0.2

The content of this document is highly confidential and should be handled accordingly.

© 2013 Nintendo
CONFIDENTIAL

Released: January 16, 2013

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

© 2013 Nintendo

Table of Contents

Code

© 2013 Nintendo

CONFIDENTIAL

1 About this document

This document is a guide explaining basic programming using the MoCam SDK for CTR API.

The content of this document is aimed at people who are familiar with C/C++ programming.

For a detailed specification of the MoCam SDK API, please see the HTML Function Reference Manual.

If not done yet, please have a look to the CTR-MoCam_SDK_Quickstart_Guide-en_US.pdf document; it will give you a quick overview of the MoCam SDK for CTR package and its installation steps.

© 2013 Nintendo

CONFIDENTIAL

2 Overview

2.1 Introduction

The MoCam SDK API is composed of both a C API that conforms to the C99 standard and a C++ API.
There is a perfect match between the C and the C++ API functionalities.
You'll see below the syntactic equivalences between these two APIs.

2.2 Syntactic philosophy

The MCcam SDK API is using the same naming convention as the Nintendo CTR SDK API.

The C++ API is using namespaces to categorize objects, functions and types.

The `mw::mo` namespace is the root namespace of Mobiclip middleware SDKs.

The C API uses prefixes obtained by concatenating namespace strings together.

Code 2-1 Example of MoCam SDK C++ API usage

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

Code 2-2 Example of MoCam SDK C API usage

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

C functions that use a MoCam object handle as a first parameter are replaced in the C++ API by methods on the corresponding MoCam object.

Also note that output parameters provided or retrieved by address in the C API are provided or retrieved by reference in the C++ API.

© 2013 Nintendo

CONFIDENTIAL

2.3 Modules description

Below you will find a list of modules available in the MoCam SDK API, sorted by their corresponding namespace. For most of them there is a corresponding library file that must be included in your `OMakefile` project to correctly build your application.

2.3.1 namespace mw::mo::mocam

This namespace only specifies error codes returned by MoCam API and typedef used for the memory interface.

2.3.2 namespace mw::mo::mocam::autofocus

2.3.2.1 mw::mo::mocam::autofocus::Handle class

This class is used to automatically adjust the 3D focus of two stereo images (typically left-eye and right-eye CTR camera captured images).
As explained in chapter 2.2, all of its methods have a C equivalent with the `mwmomocamautofocusHandle` prefix and a handle as their first parameter.
The library name to use in your `OMakefile` project is `libmw_mo_MoCam.(option).a`.

2.3.3 namespace mw::mo::helper

This namespace contains various C++ classes that implement useful features for the developer like camera or Y2R management.

These classes are not part of the MoCam SDK API, these are application driven classes that are used by the sample codes provided with the SDK and their source code is available.

Developers can choose to use them as is for their own usage, modify them or write their own.

© 2013 Nintendo

CONFIDENTIAL

3 MoCam SDK usage guide

3.1 Autofocus

To use autofocus functionality of MoCam SDK, you must include the `mocam_Autofocus.h` file and have its path added to the `INCLUDE` section of your `OMakefile` project.

Code 3-1 Inclusion of the mocam_Autofocus.h file in your source code

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

Please have a look at the MoCam SDK sample codes for additional information on the MoCam SDK API usage.

3.1.1 Autofocus initialization and finalization

Before calling any method on a `mw::mo::mocam::autofocus::Handle` object, you have to call its `Initialize` method, providing the allocation and de-allocation functions to be used to allocate and de-allocate this object's memory.

Code 3-2 Example of MoCam autofocus initialization

```
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);
```

You can call the `Finalize` method on a calibration object you no longer use to free its internal memory. This otherwise would be called further on by the C++ destructor as needed.

Code 3-3 MoCam autofocus finalization

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

© 2013 Nintendo

CONFIDENTIAL

3.1.2 Calculate focus on two images

To calculate the focus on two images, we can use `mw::mo::mocam::autofocus::Handle::Calculate` function.
Once the focus is calculated, it will be used by cropping the first image left columns, and by cropping the second image right columns. It means that the maximum possible range will usually depend on screen resolution and camera capture resolution.
In the example below, we are calling focus calculation on two CTR camera captured images, asking the autofocus to calculate a focus value ranging from 0 to `CAMERA_WIDTH-SCREEN_WIDTH`, meaning we will have to crop both images by `CAMERA_WIDTH-SCREEN_WIDTH` pixels maximum, letting us `SCREEN_WIDTH` pixels to display the result.

Code 3-4 Example of MoCam focus calculation

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

3.1.3 Retrieving autofocus result

Once we have calculated the focus, we can retrieve the result by calling `mw::mo::mocam::autofocus::Handle::Get`.
To avoid focus “erratic” and “jumping” behavior, this function allows the user to use a damping factor to damp down fast focus variations over time.

Code 3-5 Example of MoCam focus retrieving

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

Once the focus has been obtained, we can use it directly into OpenGL matrices, by adding the value to the translation of the first image, and subtracting the value to the translation of the second image.

© 2013 Nintendo

CONFIDENTIAL

3.1.4 Advanced usage

Depending on the input images resolution and the calculation range, the focus calculation can use too much CPU time.
For some usages it could be fine, but for other scenarios, it can become problematic.
To overcome this issue, you can call the focus calculation in a background thread, and use `mw::mo::mocam::autofocus::Handle::SetCallback` to control the CPU time used by the calculation.

Code 3-6 MoCam autofocus 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);
}
```

Now that we have a thread in place, we can make our thread sleep for a given duration in our callback. This will allow focus calculation to “spread” on multiple video frames.
For reading simplification purpose, thread synchronization has been removed in the next example.

© 2013 Nintendo

CONFIDENTIAL

Code 3-7 MoCam autofocus thread usage

```
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);
  }
}
```


© 2013 Nintendo

CONFIDENTIAL

3.1.5 Forcing focus to a certain value

As the focus returned by `mw::mo::mocam::autofocus::Handle::Get` is an interpolation of the internal value, it's sometimes important to be able to force internal focus value to a certain number.

For example, when a user wants to switch off autofocus calculation, then switch on again after a while (with completely different images), it will be convenient to reset internal focus value to zero. In order to do that, use `mw::mo::mocam::autofocus::Handle::Set` function.

Code 3-8 Example of usage of autofocus Set function

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

3.1.6 Precaution of use

The camera image displayed on LCD screen is usually compensated using StereoCameraCalibrationData.

Please note that using MoCam on a raw uncompensated camera image from OUT camera and simply using calculated offset values for display could result in a few differences.

© 2013 Nintendo

Revision History

Version	Revision Date	Category	Description
1.0.2	2011/05/23		• Harmonized capitalization.
1.0.1	2011/05/13	-	• Removed typo errors.
1.0.0	2011/03/15	-	• Initial version.

© 2013 Nintendo

CONFIDENTIAL

All company and product names in this document are the trademarks or registered trademarks of their respective companies.

©2013 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed, or loaned in whole or in part without the prior approval of Nintendo.