Intel® Perceptual Computing SDK

My First C++ Application

LEGAL DISCLAIMER

THIS DOCUMENT CONTAINS INFORMATION ON PRODUCTS IN THE DESIGN PHASE OF DEVELOPMENT.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

INTEL MAY MAKE CHANGES TO SPECIFICATIONS AND PRODUCT DESCRIPTIONS AT ANY TIME, WITHOUT NOTICE. DESIGNERS MUST NOT RELY ON THE ABSENCE OR CHARACTERISTICS OF ANY FEATURES OR INSTRUCTIONS MARKED "RESERVED" OR "UNDEFINED." INTEL RESERVES THESE FOR FUTURE DEFINITION AND SHALL HAVE NO RESPONSIBILITY WHATSOEVER FOR CONFLICTS OR INCOMPATIBILITIES ARISING FROM FUTURE CHANGES TO THEM. THE INFORMATION HERE IS SUBJECT TO CHANGE WITHOUT NOTICE. DO NOT FINALIZE A DESIGN WITH THIS INFORMATION.

THE PRODUCTS DESCRIBED IN THIS DOCUMENT MAY CONTAIN DESIGN DEFECTS OR ERRORS KNOWN AS ERRATA WHICH MAY CAUSE THE PRODUCT TO DEVIATE FROM PUBLISHED SPECIFICATIONS. CURRENT CHARACTERIZED ERRATA ARE AVAILABLE ON REQUEST.

CONTACT YOUR LOCAL INTEL SALES OFFICE OR YOUR DISTRIBUTOR TO OBTAIN THE LATEST SPECIFICATIONS AND BEFORE PLACING YOUR PRODUCT ORDER.

COPIES OF DOCUMENTS WHICH HAVE AN ORDER NUMBER AND ARE REFERENCED IN THIS DOCUMENT, OR OTHER INTEL LITERATURE, MAY BE OBTAINED BY CALLING 1-800-548-4725, OR BY VISITING INTEL'S WEB SITE HTTP://WWW.INTEL.COM.

ANY SOFTWARE SOURCE CODE REPRINTED IN THIS DOCUMENT IS FURNISHED UNDER A SOFTWARE LICENSE AND MAY ONLY BE USED OR COPIED IN ACCORDANCE WITH THE TERMS OF THAT LICENSE ANY SOFTWARE SOURCE CODE REPRINTED IN THIS DOCUMENT IS FURNISHED UNDER A SOFTWARE LICENSE AND MAY ONLY BE USED OR COPIED IN ACCORDANCE WITH THE TERMS OF THAT LICENSE

INTEL, THE INTEL LOGO, INTEL CORE, INTEL MEDIA SOFTWARE DEVELOPMENT KIT (INTEL MEDIA SDK) ARE TRADEMARKS OR REGISTERED TRADEMARKS OF INTEL CORPORATION OR ITS SUBSIDIARIES IN THE UNITED STATES AND OTHER COUNTRIES.

MPEG IS AN INTERNATIONAL STANDARD FOR VIDEO COMPRESSION/DECOMPRESSION PROMOTED BY ISO. IMPLEMENTATIONS OF MPEG CODECS, OR MPEG ENABLED PLATFORMS MAY REQUIRE LICENSES FROM VARIOUS ENTITIES, INCLUDING INTEL CORPORATION.

*OTHER NAMES AND BRANDS MAY BE CLAIMED AS THE PROPERTY OF OTHERS.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Introduction

The Intel® Perceptual Computing SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The library's purpose is to lower barriers to using these algorithms and shift the application developers' focus from coding the algorithm details to innovating on the usage of these algorithms for next generation human computer experience.

Assuming that you have followed the **`Getting Started`** guide to install the SDK, we will go through the steps to create and run a simple C++ application using the SDK.

First, it's always a good habit to check if the camera and the SDK are installed correctly. Here are the steps:

- Launch the **`capture_viewer`** application from the startup menu: **`Startup→Intel® Perceptual Computing SDK→Tools→Capture Viewer.`**
- Select a color stream, a depth stream and an audio stream from the **`DepthSense 325 Audio/Video Capture`** module.
- Select **`Control→Display`** to render the streams.

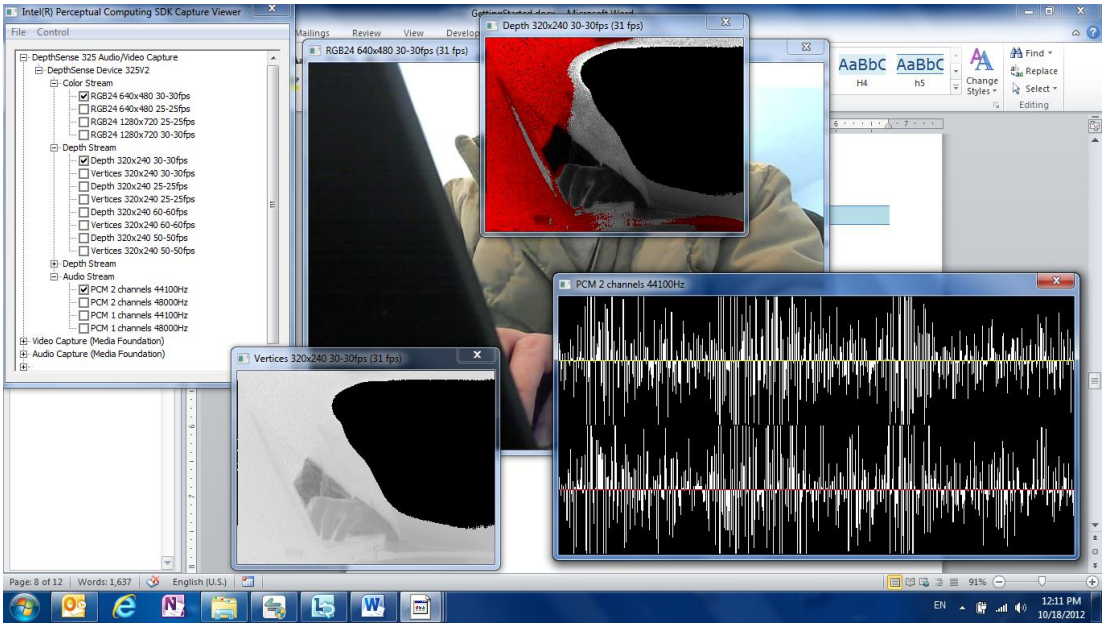

Figure 1: Camera Testing for Color, Depth and Audio

If your screen looks similar to the one shown in Figure 1, the camera is working as expected. It's time to create a simple application. To do something similar to the **`capture_viewer`** application, we will develop an application that shows color and depth pictures.

My First C++ Application

Launch Microsoft* Visual Studio* 2010. Create a new console application called **`ShowImages`** (see Figure 2).

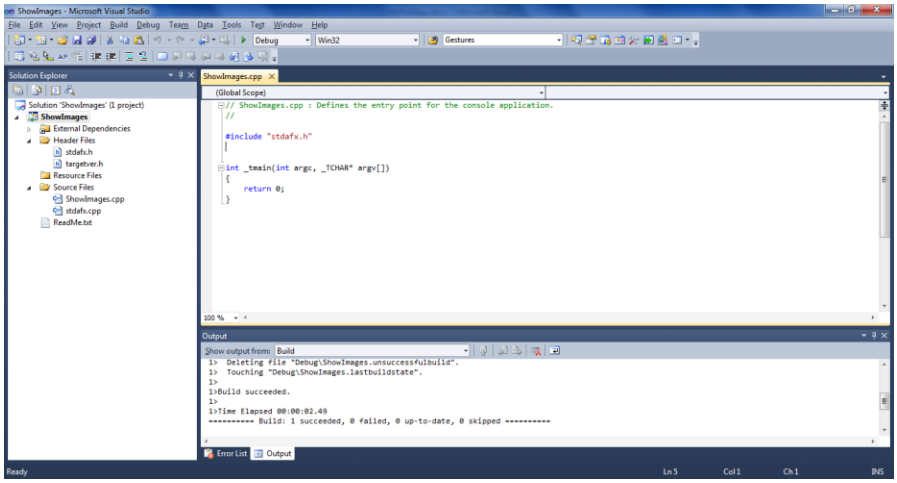

Figure 2: New Console Application

Add the SDK include files and library paths, as illustrated in Figure 3, Figure 4, and Figure 5, to the project so that the compiler and linker can find the SDK files as follows:

- Select **`Project→Properties`**, and then the **`VC++Directories`** tab. Add **`$(PCSDK_DIR)/include`** to the include directories, as illustrated in Figure 3. We will need to use some sample utilities. Add **`$(PCSDK_DIR)/sample/common/include`** to the include directories as well.

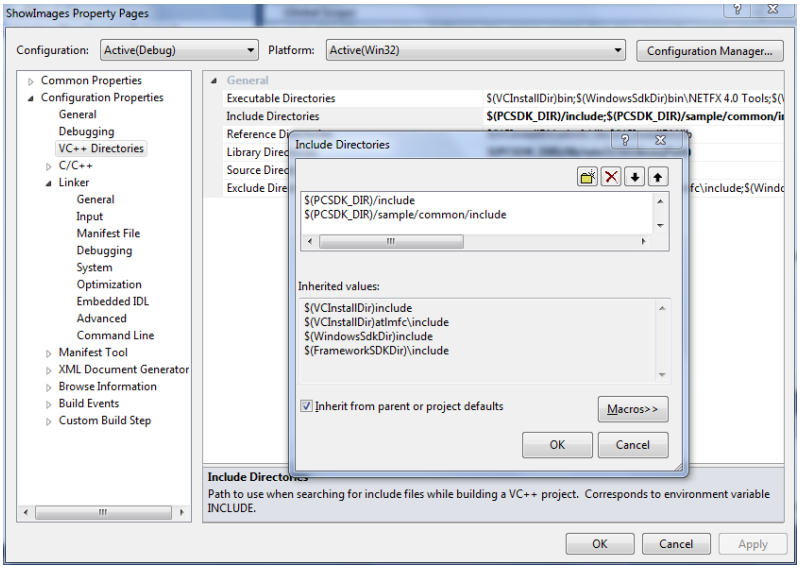

Figure 3: SDK Application Project Include Settings

- Add **`$(PCSDK_DIR)/lib/$(PlatformName)`** and **`$(PCSDK_DIR)/sample/common/lib/$(PlatformName)/$(PlatformToolset)`** to the library directories, as in Figure 4.

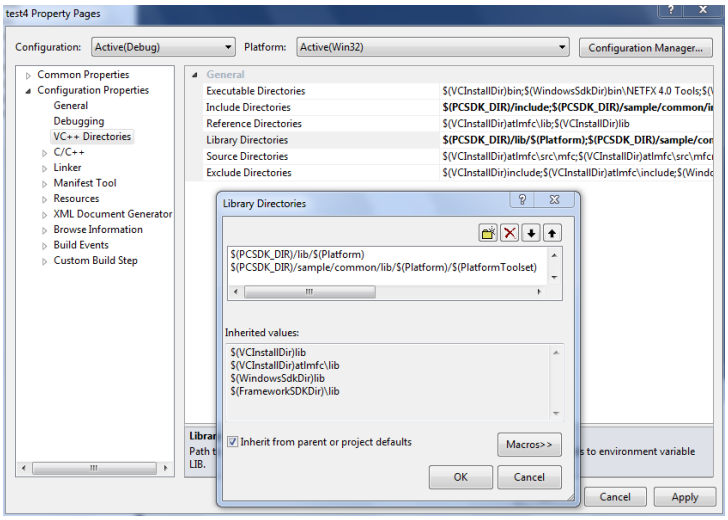

Figure 4: SDK Application Project Library Path Settings

- Go to the **`Linker→Input`** tab and add **`libpxc_d.lib`** and **`libpxcutils_d.lib`** (for the **`Debug`** build), or **`libpxc.lib`** and **`libpxcutils.lib`** (for the **`Release`** build) to the **`Additional Dependencies`** tab, as illustrated in Figure 5. Here **`libpxc`** is the SDK library code, and the **`libpxcutils`** is the sample utility library.

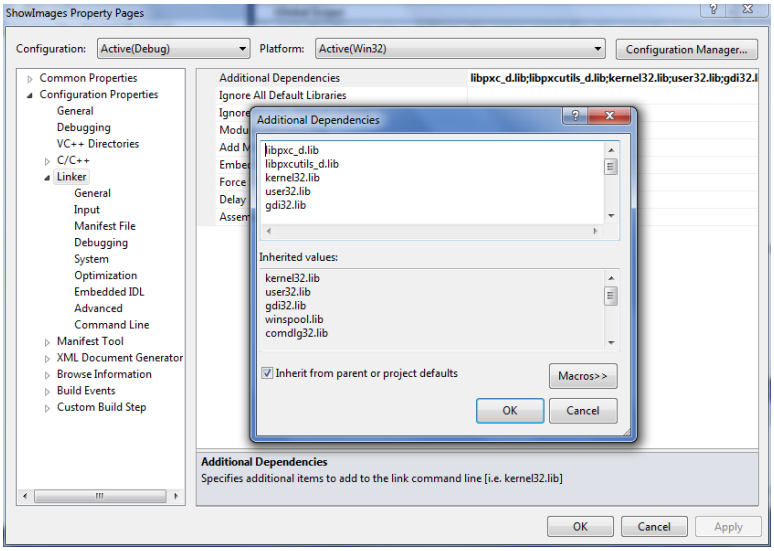

Figure 5: SDK Application Project Library Settings

- Go to the **`C/C++→Code Optimization`** tab and change the **`Runtime Library`** setting to be **`Multi-Threaded Debug (/MTd)`**, or **`Multi-threaded (/MT)`**, according to your project **`Debug`** or **`Release`** mode, as illustrated in Figure 6.

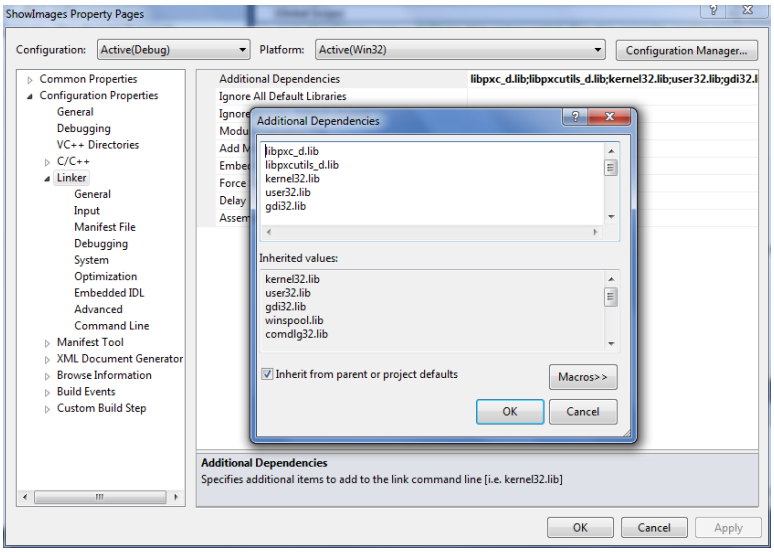

Figure 6: SDK Application Code Optimization Settings

For our first C++ SDK application (shown in Example 1), we are reading color pictures from the camera and rendering them.

```
#include "stdafx.h"
#include "util_render.h"
#include "util_pipeline.h"

int _tmain(int argc, _TCHAR* argv[]) {
	UtilPipeline pipeline;
	pipeline.EnableImage(PXCImage::COLOR_FORMAT_RGB32);
	pipeline.Init();
	UtilRender color_render(L"Color Stream");
	for (;;) {
		if (!pipeline.AcquireFrame(true)) break;
		PXCImage *color_image=pipeline.QueryImage(PXCImage::IMAGE_TYPE_COLOR);
		if (!color_render.RenderFrame(color_image)) break;
		pipeline.ReleaseFrame();
	}
	pipeline.Close();
	return 0;
}
```

Example 1: Simple Image Capture Application

The code uses the **`UtilPipeline`** utility classes. We first enable color image capturing by the **`pipeline.EnableImage(PXCImage::COLOR_FORMAT_RGB32)`** function, and then initialize the pipeline. In the for loop, the **`AcquireFrame`** function waits for a color sample to be available, then we render it using the **`UtilRender::RenderFrame`** function. We use the **`ReleaseFrame`** function to indicate that there are no more operations for the current frame. It's time to go read the next frame.

Run the code; you should see the color rendering window similar the one in in Figure 7. Click the close button on the top to close the application.

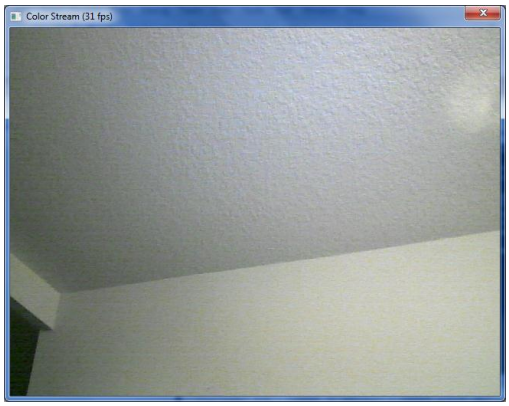

Figure 7: The Sample Capture Application Render Window

You may notice that the color window is of the default resolution. This is usually 640x480. To change to a different resolution, simply enter the resolution as part of the **`EnableImage`** function as illustrated in Figure 8. Compile and run it. The application will show a 720p resolution color window.

```
#include "stdafx.h"
#include "util_render.h"
#include "util_pipeline.h"

int _tmain(int argc, _TCHAR* argv[]) {
	UtilPipeline pipeline;
	pipeline.EnableImage(PXCImage::COLOR_FORMAT_RGB32, 1280, 720);
	pipeline.Init();
	UtilRender color_render(L"Color Stream");
	for (;;) {
		if (!pipeline.AcquireFrame(true)) break;
		PXCImage *color_image=pipeline.QueryImage(PXCImage::IMAGE_TYPE_COLOR);
		if (!color_render.RenderFrame(color_image)) break;
		pipeline.ReleaseFrame();
	}
	pipeline.Close();
	return 0;
}
```

Figure 8: Simple Capture with An Explicit Resolution.

Now, let's go one more step and add a depth stream in a different rendering window. It's easy to do. The code is shown in Figure 9.

```
#include "stdafx.h"
#include "util_render.h"
#include "util_pipeline.h"

int _tmain(int argc, _TCHAR* argv[])
{
	UtilPipeline pipeline;
	pipeline.EnableImage(PXCImage::COLOR_FORMAT_RGB32);
	pipeline.EnableImage(PXCImage::COLOR_FORMAT_DEPTH);
	pipeline.Init();
	UtilRender color_render(L"Color Stream");
	UtilRender depth_render(L"Depth Stream");
	for (;;) {
		if (!pipeline.AcquireFrame(true)) break;
		PXCImage *color_image=pipeline.QueryImage(PXCImage::IMAGE_TYPE_COLOR);
		PXCImage *depth_image=pipeline.QueryImage(PXCImage::IMAGE_TYPE_DEPTH);
		if (!color_render.RenderFrame(color_image)) break;
		if (!depth_render.RenderFrame(depth_image)) break;
		pipeline.ReleaseFrame();
	}
	pipeline.Close();
	return 0;
}
```

Figure 9: Color and Depth Image Capture

Compile and run it. You should see two windows as shown in Figure 10: one for the color stream and the other for the depth stream. Closing any of the color and depth windows will close the application.

Figure 10: Color and Depth Image Capture Render Windows