

Kinect 개발 입문서

C/C++에서의 Kinect 개발

강 찬 석

2012-12-12



한국과학기술연구원

실감교류로보틱스연구센터

1. 대상

이 글은 Kinect 개발을 처음 접하는 개발자를 대상으로 작성하는 가이드이며, 전체적인 맥락은 기관 프로젝트 진행시 구현하려고 했던 hand Gesture Estimation 을 구현하는데 있어서 배웠던 내용을 바탕으로 했다.

이 글은 크게

- Kinect 가 제공하는 기능
- 설치방법
- OpenCV 를 사용한 기본적인 기능 활용
- 기타적으로 유용하게 쓰일 함수들

나뉘지며 기타로 부연설명이 필요한 부분에 대해서는 주석이나 참고문헌에 명시한다.

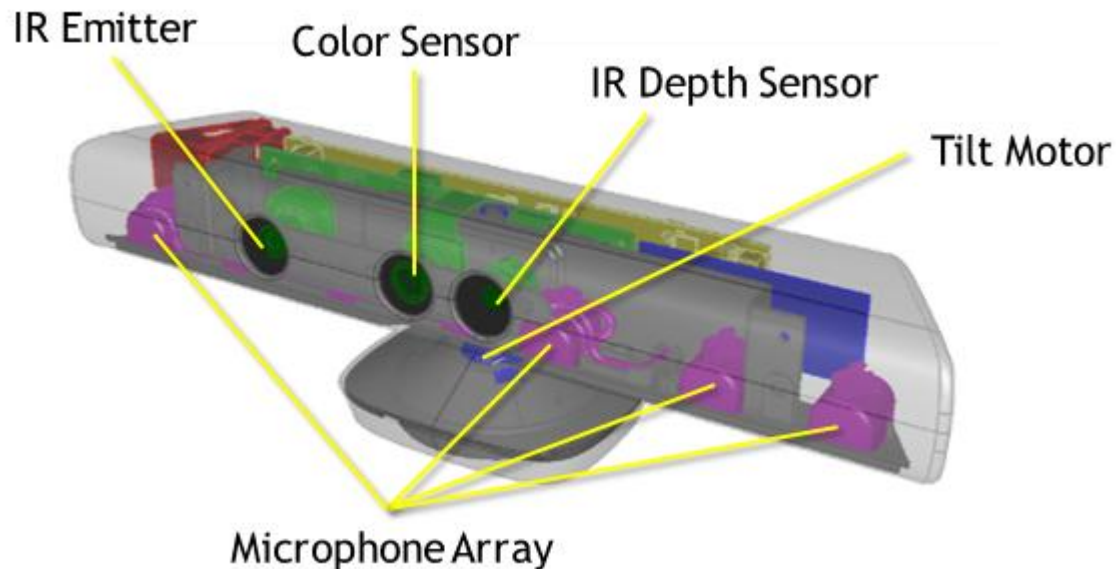
(이미지가 작아서 안 보일 때는 문서에서 확대하면 잘 보인다.)

2. 차례

1. C/C++로 개발하는 키넥트의 대상	-----	1p
2. 차례	-----	2p
3. Kinect 가 제공하는 기능	-----	3p
4. 설치 방법		
4-1. 개발 환경 조성 및 유의 사항	-----	4p
4-2. Kinect SDK 와 OpenNI 동시 설치	-----	5p
4-3. OpenCV 설치	-----	6p
4-4. (option) 3Gear system 설치	-----	7p
5. Visual Studio 상에서 환경 설정	-----	11p
6. Kinect 기본 기능 활용		
6-1. colorStream 출력	-----	15p
6-2. DepthStream 출력	-----	19p
6-3. SkeletonStream 활용	-----	27p
6-4. IRStream 출력	-----	35p
7. 기타 활용할 수 있는 함수		
7-1. 실좌표계로의 좌표변환	-----	40p
7-2. Distance Transform	-----	41p
7-3. Background Subtraction	-----	44p
7-4. 손의 ConvexityDefects	-----	46p
7-5. Blob Detection	-----	49p
8. 출처 및 참고 문헌	-----	50p

3. Kinect 의 기능

키넥트는 Microsoft 에서 출시한 DepthCAM 이다. 하지만 기기 자체는 이스라엘의 PrimeSense¹사에서 만든 기기의 라이선스를 따서 만들어졌다. 유사 기기로는 ASUS 사의 Xtion 제품군이 있다.



2

키넥트는 RGB 카메라, IR 센서, 4-array microphone, 3 축 센서로 구성되어 있으며, 기타로 제공되는 스펙은 다음과 같다.

Kinect	Array Specifications
Viewing angle	43° vertical by 57° horizontal field of view
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression
Accelerometer characteristics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

¹ PrimeSense : <http://www.primesense.com>

² Specification for the Kinect : <http://msdn.microsoft.com/en-us/library/jj131033.aspx>

기존에는 Xbox 360 의 주변 기기로 활용되었지만 현재는 데스크탑 어플리케이션 개발에 특화된 Kinect for Windows 가 나와있는 상태이며, 기존의 기기에 비해서 성능이 향상되었다. 특히 Near Mode 의 제공으로 키넥트를 조금 더 가까운 거리에서 활용할 수 있게 되었다. 현재 연구실에서 보유중인 기기는 Kinect for windows ³이다.

4. 설치 방법

4-1. 개발 환경 조성 및 유의 사항

Kinect SDK 는 MS 에서 제공하는 공식 개발 툴이기 때문에 모든 제품의 연동이 MS 사의 제품과 연계되어 있다. 공식적으로 언급하는 개발환경은 다음과 같다.

- 인텔/AMD 계열의 x86/x64 프로세서
- Windows 7 / 8
- Visual Studio 2010 / 2012
- .net framework (C# , VB) & C++

OpenNI⁴는 PrimeSense 사에서 만든 키넥트를 동작시키기 위해 개발된 오픈소스 드라이버로, 이를 개발할 수 있는 SDK 가 같이 제공되고 있다. MS 사의 키넥트 역시 이 기기와 하드웨어가 동일하기 때문에 이 드라이버를 적용해서 개발할 수 있다. OpenNI 를 개발할 수 있는 환경은 다음과 같다.

- Windows XP 이상의 운영체제 / Linux Ubuntu 10.10 이상

이 밖에도 CL NUI⁵나 Libfreenect 같은 드라이버들도 존재한다.

키넥트는 기본적으로 USB 의 전원으로 동작하는 기기이다. 또한 USB 의 대역폭을 많이 사용한다. 키넥트를 연결했을 때 발생하는 오류 두 가지 중

³ Kinect for windows : <http://www.microsoft.com/en-us/kinectforwindows/>

⁴ OpenNI : <http://openni.org/>

⁵ CL NUI / Libfreenect: http://openkinect.org/wiki/Main_Page

하나는 usb 로 공급되는 전원이 부족해서 나타나며, 또 다른 하나는 대역폭의 부족 현상 때문에 발생한다. 보통 이 문제들은 키넥트를 여러 대 연결했을 때 나타난다. 이 때 키넥트는 반드시 usb 허브(전원장치가 별도로 제공되는 것은 예외)가 아닌 pc 본체와 직접적으로 연결되어야 하며, 메인보드 상에 제공되는 usb 컨트롤러당 키넥트를 1 개만 연결해야 한다. 쉽게 말하자면 키넥트 외에는 사용하는 usb 장치가 되도록이면 없게끔 환경을 구축해야 한다는 것이다.

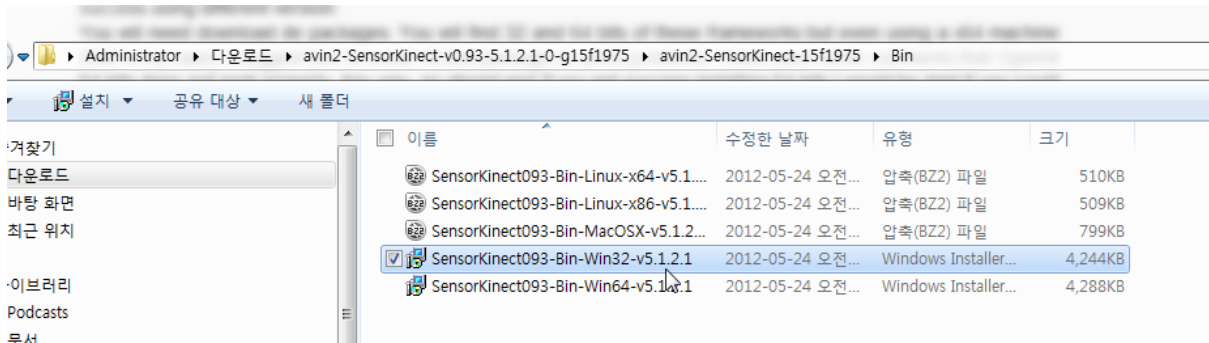
4-2. Kinect SDK 와 OpenNI 설치

Kinect SDK 기반의 개발은 installer 파일로 제공되기 때문에 설치가 무척 쉽다. 하지만 OpenNI 로도 같이 개발할 수 있도록 만든 것이 있기 때문에 그 부분을 설명하고자 한다.

우선 다음과 같은 준비물이 필요하다.

- Kinect for Windows 나 Kinect for Xbox
- Kinect for windows SDK & Developer Toolkit
(<http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx>)
- OpenNI v1.5.4.0 (32bit)
(<http://www.openni.org/Downloads/OpenNIModules.aspx> 에서 OpenNI Binaries 에서 32bit 선택)
- NITE v1.5.2.21 (32bit)
(<http://www.openni.org/Downloads/OpenNIModules.aspx> 에서 Middleware Binaries 에서 32bit 선택)
- KinectSensor v0.93
(<https://github.com/avin2/SensorKinect>)
- Bridge Driver
(<http://code.google.com/p/kinect-mssdk-openni-bridge/>에서 Kinect SDK 의 버전에 맞게 설치)

우선 키넥트가 연결된 상태라면 키넥트 연결을 해제하고 프로그램 설치 / 관리에서 키넥트와 관련된 모든 프로그램을 **삭제**한다. 우선 앞에서 받은 OpenNI와 NITE를 차례대로 설치한다. 중간에 드라이버를 설치한다는 말이 나오면 반드시 설치해준다. 그 후에 다음으로 받은 KinectSensor를 받아서 설치한다. 이때 유의할 내용은 이 설치형 파일에는 윈도우뿐만 아니라 리눅스에 대한 설치형 파일이 포함되어 있다는 것이고 여기서 선택해야 될 플랫폼은 win32이다.

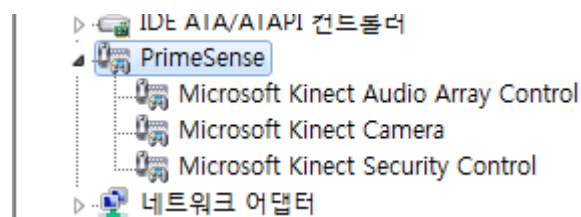


위 파일을 설치한 후에 ProgramFiles\OpenNI\Sample\bin에 있는 파일이 실행되는지를 확인한다.



OpenNI에서는 기본적으로 GestureGenerator라는 제스처 세트를 제공하며 여기에는 손 흔들기와 같은 간단한 제스처에 대해서 정의되어 있다. 또한 Kinect SDK보다 DepthStream의 Noise 간섭이 덜 하기 때문에 조금 더 깔끔한 이미지를 얻을 수 있다.

여기까지 온 상태에서 키넥트를 연결해제 한 후 Kinect SDK와 Developer Toolkit을 차례대로 설치해준다. 그 후에 키넥트를 연결하면 장치관리자가 다음과 같이 나열된다.



이 상태에서 앞에서 받은 bridge driver 파일에 들어있는 install 을 관리자권한으로 실행시키면 된다. 이때 중간에 OK 라는 문구가 떠 있으면 정상적으로 설치가 된 것이다. **Kinect SDK 를 설치한 후에 OpenNI 샘플이 실행되지 않는 경우가 발생하면 이 브릿지 드라이버 설치 과정을 다시 수행하면 해결된다.**

이렇게 설치하면 C 프로젝트상에서 OpenNI 에서 제공하는 API 와 Kinect SDK 에서 제공하는 API 를 동시에 사용할 수 있게 된다. 참고로 OpenNI 는 오픈소스형 드라이버이기 때문에 Processing⁶이나 OpenCV⁷ / Point Cloud Library⁸의 기본 모듈로 사용할 수 있으며 이를 활용한 프로젝트들이 온라인상에 많이 공개되어 있다.

유의할 점은 이렇게 설치한 후에는 Kinect 를 두 대 이상 사용할 수 없기 때문에 뒤에서 소개할 3 Gear System 과 같은 MultiKinect 를 활용한 프로젝트라면 위와 같은 멀티 플랫폼이 아닌 단일 개발환경을 구축해야 한다. 만약 테스트를 한 후 원래대로 복원해야 한다면 앞에서 언급한 설정과정을 반복해서 수행해주면 된다.

4-3. OpenCV 설치

키넥트로부터 받은 데이터를 시각적으로 표현하는 방법이 다양하게 있지만 여기서는 비전 라이브러리인 OpenCV 를 사용해서 표현하고자 한다. 이 글을 쓸때의 OpenCV 버전은 v2.4.3 이고 버전이 달라짐에 따라서 설치방법이 달라질 수 있음을 알려둔다. (<http://sourceforge.net/projects/opencvlibrary/>)

기본적으로 OpenCV 는 설치형 바이너리를 제공하며 이를 설치하면 프로젝트 구축에 필요한 라이브러리 파일과 dll 파일, 헤더파일들이 설치된다. 만약 OpenCV 내에서 OpenNI 만을 가지고 프로젝트를 구현하고자 했을 때는 기본적으로 만들어져 있는 코드(cap_openni.cpp)를 활용하면 된다.

⁶ Processing : <http://www.processing.org/>

⁷ OpenCV : <http://opencv.org/>

⁸ Point Cloud Library : <http://pointclouds.org/>

우선 opencvWbuildWx86Wv10Wbin 폴더로 들어가면 dll 파일들이 존재하는데 이 파일들을 전부 Windows\System32 폴더에 옮겨준다. 물론 프로젝트 생성시에 이 파일 경로를 지정해줄 수도 있으나 Visual Studio 2010에서는 프로젝트 생성시마다 이 설정을 매번 해줘야 하기 때문에 기본적으로 불러오는 폴더에 dll 파일을 넣는 과정을 거친다. 물론 exe 파일 실행 시에도 이 dll 파일들이 System32 폴더 안에 들어있어야 정상적으로 실행된다.

그 후에 C 드라이브 상에 임의의 폴더를 생성한다. 그 중에서 원본 폴더내의 build 폴더에 들어있던 include 폴더와 opencvWbuildWx86Wv10 에 있는 lib 폴더를 옮겨준다. 이는 라이브러리와 헤더파일 경로 설정을 용이하게 하기 위해서다.

마지막으로 설치해줘야 하는 것은 Intel Threading Building Block(TBB)⁹인데 병렬연산에 대한 지원을 해주는 것이다. 역시 이 파일도 압축을 푼 뒤에 생성되는 tbb.dll 파일과 tbb_debug.dll 파일을 아까 dll 파일을 넣었던 System32 폴더에 넣어주면 된다.

4-4. 3 Gear systems 설치

3 Gear systems¹⁰은 키넥트 두 대를 사용해서 손을 모델링해주는 tool 이다. 두 대를 사용했기 때문에 한 대를 사용했을 때보다 Occlusion 으로 인한 문제가 완화되었다. 이 tool 에는 finger 를 활용한 제스처를 구현하고, 관련 어플리케이션을 개발하는데 필요한 샘플과 API 들이 포함되어 있다. 홈페이지에는 정식 프레임이 판매하고 있지만 연구실에서는 다음 환경에서 구현했다.



⁹ Inter Threading Building Block : <http://threadingbuildingblocks.org/ver.php?fid=174>

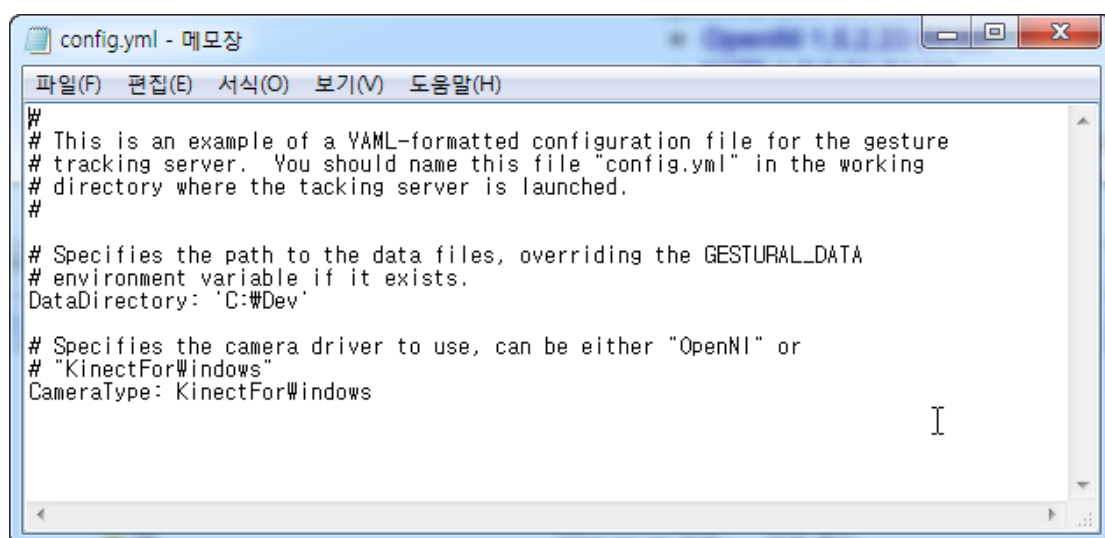
¹⁰ 3 Gear Systems : <http://www.threegear.com/>

앞에서 언급한 유의 사항과 같이 키넥트 두 대를 연결하는데 있어서는 반드시 서로 다른 usb 컨트롤러에 연결되어 있어야 한다. 또한 앞에서 소개한 설치법은 단일 키넥트에서만 개발할 수 있기 때문에 이 tool 을 사용하기 위해서는 Kinect SDK 나 OpenNI 단일 플랫폼으로만 개발하여야 한다.

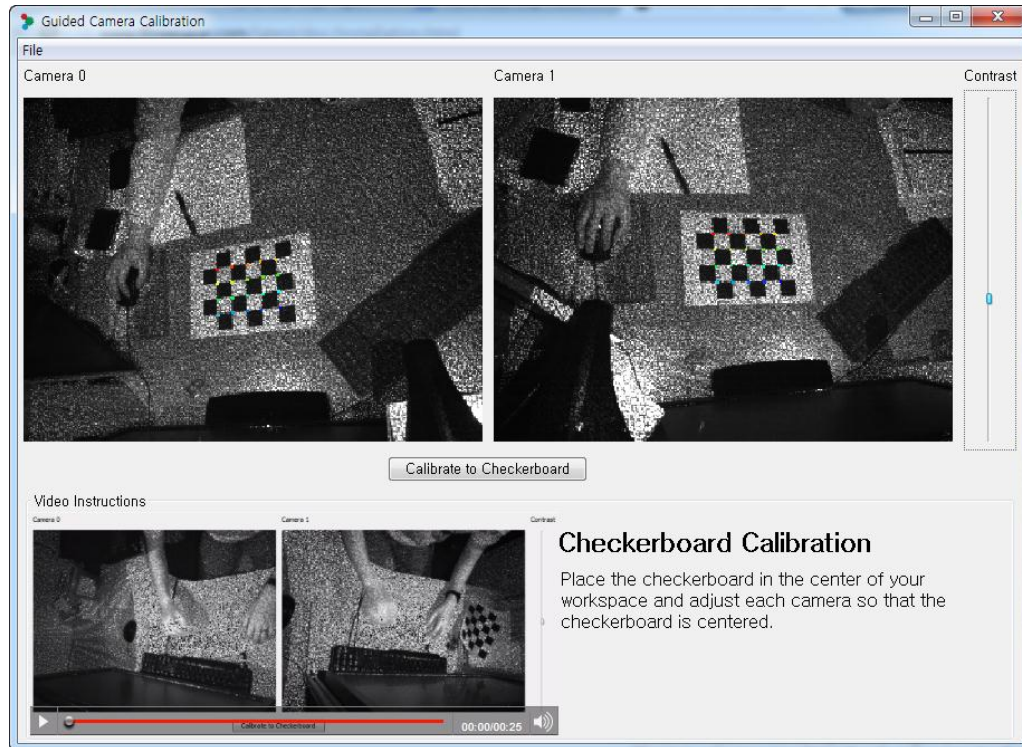
이 Tool 은 64bit 기반이기 때문에 64bit 용 Kinect SDK 를 사용하는 사람이라면 그대로 활용해도 된다. Kinect SDK 는 설치 운영체제에 따라서 자동으로 결정되기 때문에 이를 참고하면 된다. 참고로 Java Runtime Environment(JRE) x64 도 설치되어 있어야 한다.

- 진행 과정은 다음과 같다.

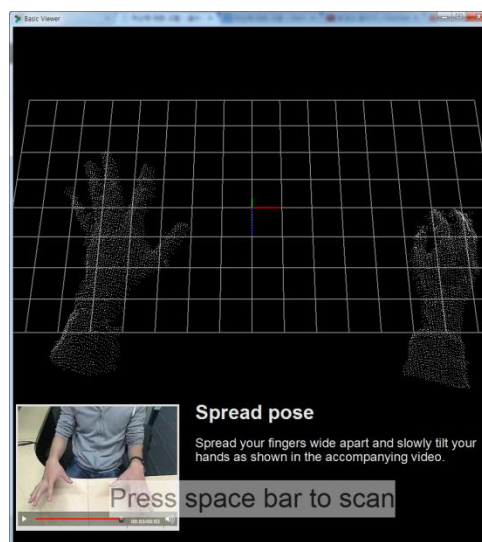
- SDK 64bit Binaries 를 받아서 해당 폴더에 집어넣는다.
C:\Dev\GesturalUserInterface
C:\Dev\GesturalUserInterface\lib
- SDK data 를 받아서 해당 폴더에 집어넣는다.
C:\Dev\data\CameraCalibration
C:\Dev\data\Icons
C:\Dev\data\LeftHandSkinningExample
- GesturalUserInterface 폴더에 있는 config.yml 파일을 다음과 같이 수정한다.



- Command 창에서 CameraCalibrationtest.bat 을 실행한다.

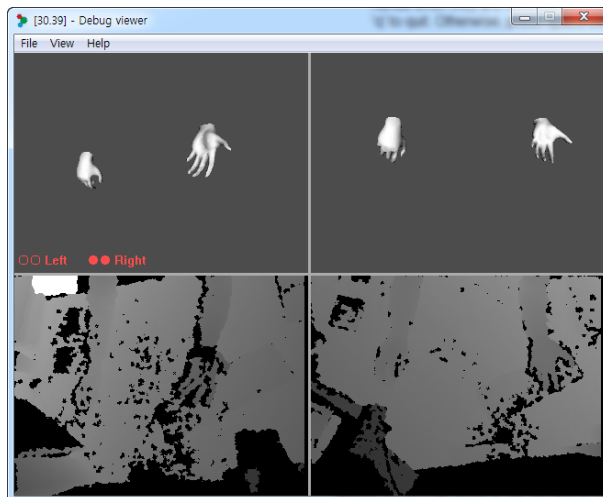


- Command 창에서 handCalibration.bat shape 후 사용자 이름을 입력한다.
ex) handCalibration.bat shape kcsgoodboy



- Pose Calibration 을 실시한다.
ex) handCalibration.bat pose kcsgoodboy

- 만들어진 정보를 가지고 DB 를 생성한다.
ex) handCalibration.bat sample kcsgoodboy
- DB 가 만들어진 상태에서 tracking server 를 연다.
ex) handdriver.bat kcsgoodboy



- 이 상태에서 Sample 을 실행시킬 수 있다.
ex) assembly_trainer.bat kcsgoodboy

현재 3 gear systems 는 30 일 평가판으로 제공되고 있으면 추가적으로
연구목적에 한해서 라이선스를 제공하고 있다. 필요한 사람은 다음 링크¹¹를
참고하면 된다.

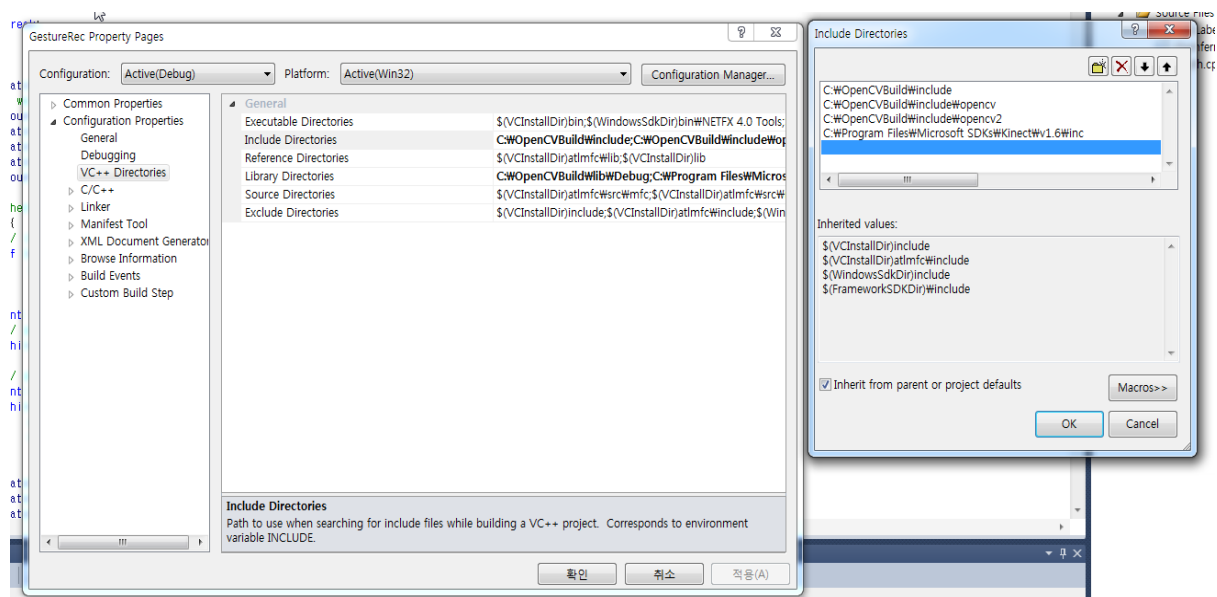
5. Visual studio 에서의 환경 설정

이 과정에서 수행되는 것은 OpenCV 와 Kinect SDK 의 헤더파일 및 라이브러리의
파일 경로 설정이다.

Visual Studio 상에서 빈 Visual C++ 프로젝트를 하나 생성하고, 우선 Debug
모드로 개발할지 Release 모드로 결정할 것인지 여부를 정한다. 프로젝트의 기본
설정가 Debug 모드인 상태에서 모드 변경시 기존에 설정했던 내용들이 유지되지

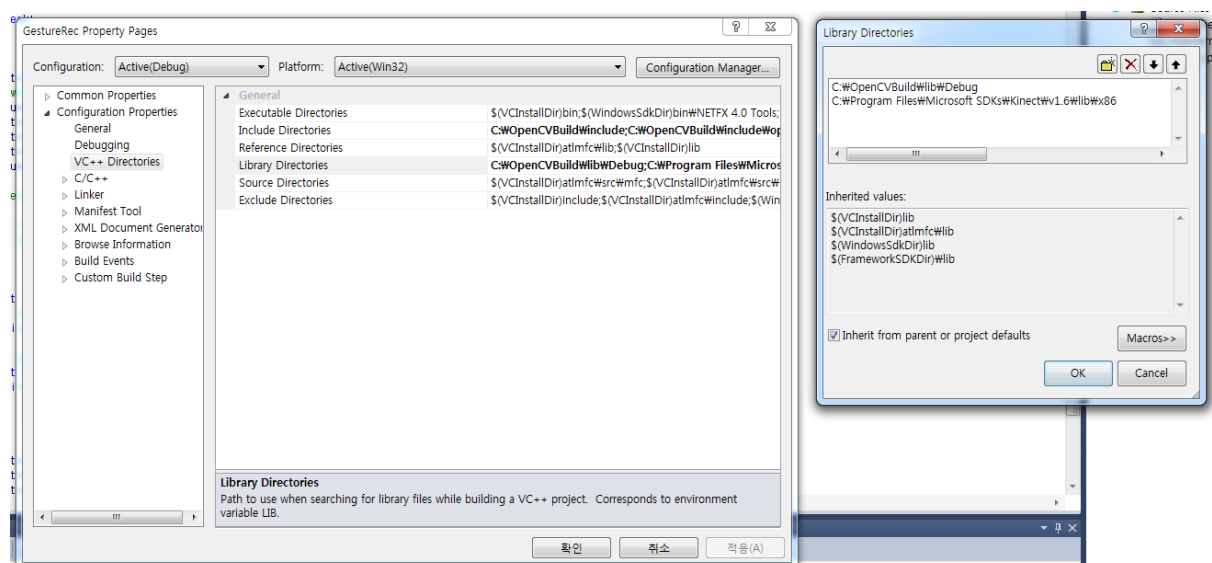
¹¹ 3 Gear Systems License : <http://www.threegear.com/licensing.html>

않기 때문에 번거로움을 피하기 위해서는 사전에 이 부분에 대해서 확인해야 한다. 여기서는 Debug 모드에서의 환경설정을 소개한다.



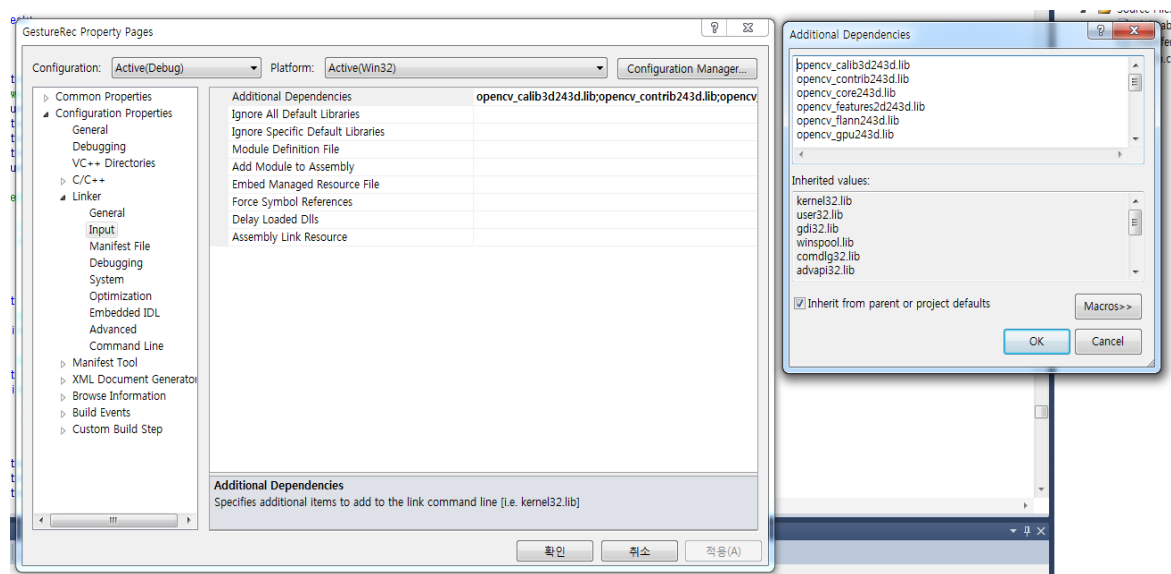
우선 Solution explorer 상에서 프로젝트를 선택하고 Properties 메뉴로 들어간다. 맨 처음에 선언해줘야 할 것은 프로젝트에 사용될 헤더파일의 경로이다. 이 부분은 VC++ directories 의 Include Directories 에서 이뤄진다.

앞에서 언급한 바와 같이 include 폴더를 opencvBuild 라는 폴더에 모아둔 상태이며 그 안에 들어있는 모든 폴더를 연결해준다. 추가적으로 Kinect SDK 헤더파일을 사용하기 위해서 Kinect SDK 의 헤더 파일의 경로도 연결시켜준다.



다음은 라이브러리 폴더 지정인데 이 부분도 VC++의 Library Directories 에서 설정할 수 있다.

역시 라이브러리도 아까의 include 폴더와 같이 하나의 폴더 안에 넣어놔으며, Kinect SDK 의 라이브러리를 사용하기 위해서 파일 경로를 지정했다.



마지막으로 라이브러리를 어떤 것을 쓸 것인지를 정의하는 부분을 Linker 에서 설정해줘야 한다.

여기서 Additional Dependencies 를 채워줘야 하는데 debug 모드에서 사용되는 라이브러리에는 파일명에 'd'가 붙어있다. 이를 참고하고 기본적으로 쓰이는 라이브러리를 아래와 같이 삽입하면 된다.

opencv_calib3d243d.lib

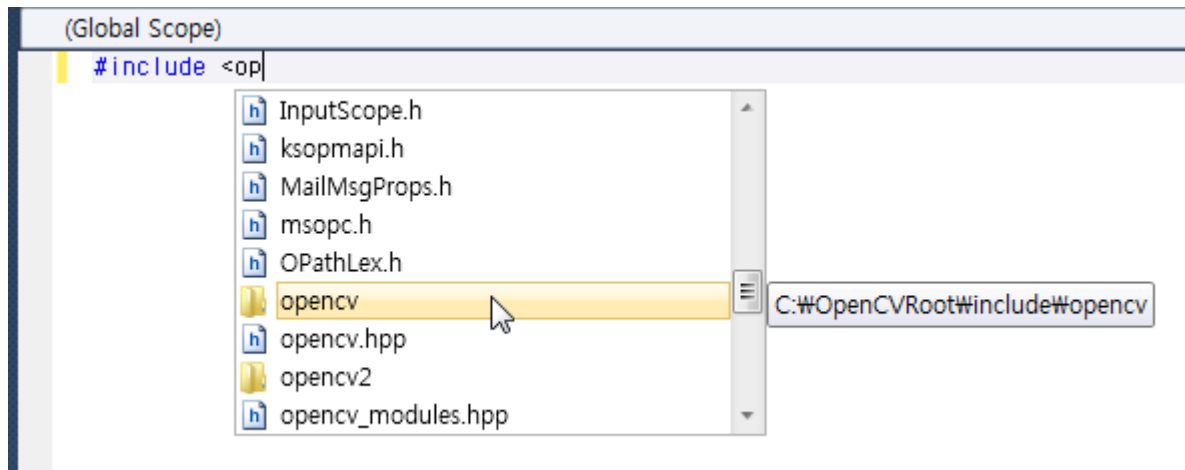
opencv_core243d.lib

opencv_highgui243d.lib

opencv_imgproc243d.lib

opencv_video243d.lib

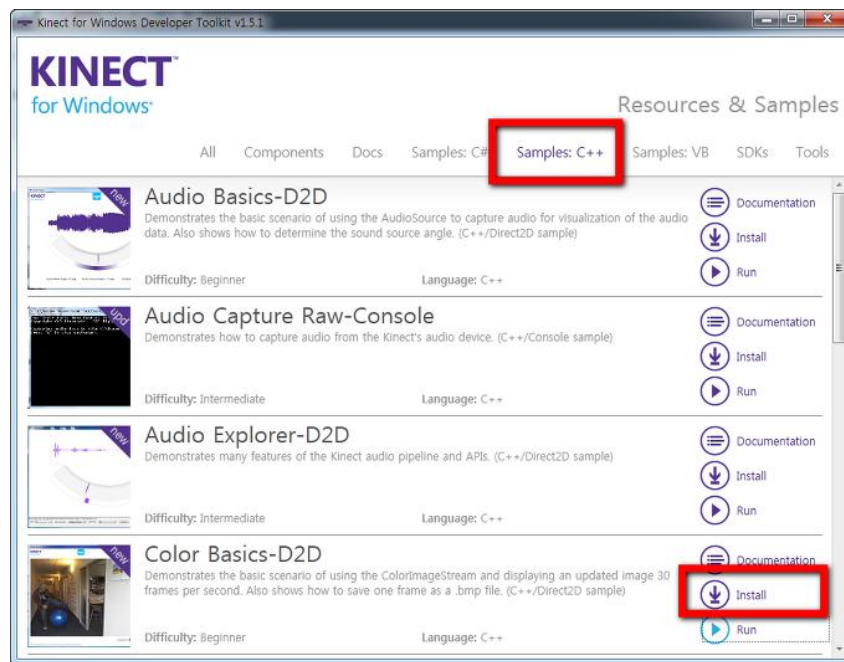
Kinect10.lib



여기까지 삽입하면 기본적인 환경설정은 끝난다. 경로가 제대로 설정되었는지 확인해보려면 cpp 파일 상에 opencv 헤더파일과 Kinect SDK의 헤더파일(NuiApi.h)가 호출되는지를 확인하면 된다. 앞으로 나오는 프로젝트의 예제는 이 과정이 사전에 이뤄졌음을 염두에 두고 언급한다.

6. Kinect 기본 Data 활용

Kinect Developer Toolkit에서는 C++ 환경에서 만들어진 샘플들을 제공하고 있다. 이 샘플들은 Direct2D를 기반으로 작성되어 있고, 이 문서에서는 OpenCV 상에서 출력하는 것을 목적으로 하고 있기 때문에 약간 다른 내용으로 진행된다. 그러나 기반은 거의 똑같다. 궁금한 내용이 있으면 참고를 하면 좋을 것 같다.



6-1. ColorStream 출력

- 헤더 파일 및 네임스페이스 추가

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv/cxcore.h>

#include <tchar.h>
#include <windows.h>
#include <NuiApi.h> // Microsoft Kinect SDK

// STL
#include <stdio.h>
#include <iostream>

using namespace std;
using namespace cv;

#define COLOR_WIDTH 640
#define COLOR_HEIGHT 480
```


- 프로그램 구성은 다음과 같이 이뤄진다,

- 키넥트를 초기화 (자신이 원하는 기능을 활성화하는 과정)
- 이미지 출력을 위한 자료형 선언
- 매초마다 그 자료형을 뿌려주는 과정
- 종료시 자료형에 할당된 메모리 해제

- 키넥트 초기화를 위한 함수 생성

```
int _tmain(int argc, _TCHAR* argv[])
{
    HANDLE colorStreamHandle;
    HANDLE nextColorFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    ...
    HRESULT hr;
    ...
    InitializeKinect();
    ...
}
```

- InitializeKinect()의 정의

```
void InitializeKinect()
{
    bool FailToConnect;
    ...
    do
    {
        HRESULT hr = NuInititalize(NUI_INITIALIZE_FLAG_USES_COLOR);
        ...
        if(FAILED(hr))
        {
            system("cls");
            cout<<"\nFailed to Connect!\n\n";
            FailToConnect = true;
            system("PAUSE");
        }
        else
        {
            cout<<"\nConnection Established!\n\n";
            FailToConnect = false;
        }
    }
    while(FailToConnect);
}
```

여기서 키넥트의 기능을 활성화시켜주는 부분이 바로 NuiInitialize()이다. 다음 표를 보고 원하는 기능을 괄호안에 플래그 형식으로 삽입해주면 된다.

NUI_INITIALIZE Flags

(http://msdn.microsoft.com/en-us/library/hh855368.aspx#NUI_INITIALIZE)

Constant	Description
NUI_INITIALIZE_DEFAULT_HARDWARE_THREAD	This flag was deprecated in version 1.5; it is no longer used.
NUI_INITIALIZE_FLAG_USES_AUDIO	Initialize the sensor to provide audio data.
NUI_INITIALIZE_FLAG_USES_COLOR	Initialize the sensor to provide color data.
NUI_INITIALIZE_FLAG_USES_DEPTH	Initialize the sensor to provide depth data.
NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX	Initialize the sensor to provide depth data with a player index.
NUI_INITIALIZE_FLAG_USES_SKELETON	Initialize the sensor to provide skeleton data.

예시 : Depth 와 Color 를 병행해서 사용할 때

NuiInitialize(NUI_INITIALIZE_FLAG_USES_COLOR | NUI_INITIALIZE_FLAG_USES_DEPTH)

- IplImage 와 창 생성

```
int _tmain( int argc, _TCHAR* argv[])
{
    HANDLE colorStreamHandle;
    HANDLE nextColorFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    .....
    HRESULT hr;
    .....
    InitializeKinect();
    .....
    IplImage *Color = cvCreateImage(cvSize(COLOR_WIDTH,COLOR_HEIGHT), IPL_DEPTH_8U,4);
    .....
    cvNamedWindow( "Color Image",CV_WINDOW_AUTOSIZE);
    .....
}
```

- NuiImageStreamOpen 에서 활성화된 기능에 대한 해상도 및 이벤트 지정

```

int _tmain(int argc, _TCHAR* argv[])
{
    HANDLE colorStreamHandle;
    HANDLE nextColorFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    HRESULT hr;

    InitializeKinect();

    IplImage *Color = cvCreateImage(cvSize(COLOR_WIDTH, COLOR_HEIGHT), IPL_DEPTH_8U, 4);

    cvNamedWindow("Color Image", CV_WINDOW_AUTOSIZE);

    hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_COLOR, NUI_IMAGE_RESOLUTION_640x480, 0, 2, nextColorFrameEvent, &colorStreamHandle);

    if(FAILED(hr))
    {
        cout<<"Could not open ImageStream"<<endl;
        return hr;
    }
}

```

- 루프 내에서 프레임을 계속 받아오는 과정

```

while(1)
{
    WaitForSingleObject(nextColorFrameEvent, 1000);
    createRGBImage(colorStreamHandle, Color);

    if(cvWaitKey(10) == 0x001b)
    {
        break;
    }
}

```

- 일반 웹캠에서 수행하는 OpenCV 예제를 돌리려면 IplImage 로 선언된 color 를 건드리면 된다. (ex : Edge , Contour, color-based Tracking)

- 종료된 후에 키넥트를 꺼주는 과정

```

while(1)
{
    WaitForSingleObject(nextColorFrameEvent, 1000);
    createRGBImage(colorStreamHandle, Color);

    if(cvWaitKey(10) == 0x001b)
    {
        break;
    }
}

NuiShutdown();
return 0;
}

```

- NuiShutdown() 후에 cvReleaseImageHeader(&color)가 빠짐

- createRGBImage () 정의

```
int createRGBImage(HANDLE h, IpImage* Color)
{
    const NUI_IMAGE_FRAME *pImageFrame = NULL;

    HRESULT hr = NuImageStreamGetNextFrame(h, 1000, &pImageFrame);

    if(FAILED(hr))
    {
        cout<<"Create RGB Image Failed\n";
        return -1;
    }

    INuiFrameTexture *pTexture = pImageFrame->pFrameTexture;
    NUI_LOCKED_RECT LockedRect;
    pTexture->LockRect(0, &LockedRect, NULL, 0);

    if(LockedRect.Pitch != 0)
    {
        BYTE * pBuffer = (BYTE*)LockedRect.pBits;
        cvSetData(Color, pBuffer, LockedRect.Pitch);
        cvShowImage("Color Image", Color);
    }

    NuImageStreamReleaseFrame(h, pImageFrame);

    return 0;
}
```

- 여기까지가 기본적인 ColorStream 의 출력 과정이다.

6-2. DepthStream 출력

- 헤더파일 추가 및 전역변수 선언

```
Depth.cpp
(Global Scope)
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv/cxcore.h> // 이미지 출력을 위한 OpenCV용 헤더파일

#include <Windows.h>
#include <tchar.h> // win32 어플리케이션을 위한 헤더
#include <NuiApi.h> // Kinect API 사용을 위한 헤더파일

#include <stdio.h> // standard IO

#define DEPTH_WIDTH 640
#define DEPTH_HEIGHT 480 //640x480에 대해서 창을 출력함
```

- DepthStream 을 처리할 이벤트 생성

```
Depth.cpp x dist.h
(Global Scope)
#include <opencv2/cv.h>
#include <opencv2/highgui.h>
#include <opencv2/core.h> // 이미지 출력을 위한 OpenCV용 헤더파일

#include <Windows.h>
#include <tchar.h> // win32 어플리케이션을 위한 헤더
#include <NuiApi.h> // Kinect API 사용을 위한 헤더파일

#include <stdio.h> // standard IO

#define DEPTH_WIDTH 640
#define DEPTH_HEIGHT 480 //640x480에 대해서 창을 출력함

int _tmain(int argc, _TCHAR argv[])
{
    HANDLE DepthStreamHandle; // 지금 이벤트를 담을 핸들러
    HANDLE NextDepthFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL); // 다음 이벤트에 대한 핸들러
    HRESULT hr; // 핸들러 결과를

}
```

- IplImage 와 Window 선언

```
Depth.cpp x _tmain.cpp
#include <opencv2/cv.h>
#include <opencv2/highgui.h>
#include <opencv2/core.h> // 이미지 출력을 위한 OpenCV용 헤더파일

#include <Windows.h>
#include <tchar.h> // win32 어플리케이션을 위한 헤더
#include <NuiApi.h> // Kinect API 사용을 위한 헤더파일

#include <stdio.h> // standard IO

#define DEPTH_WIDTH 640
#define DEPTH_HEIGHT 480 //640x480에 대해서 창을 출력함

int _tmain(int argc, _TCHAR argv[])
{
    HANDLE DepthStreamHandle; // 지금 이벤트를 담을 핸들러
    HANDLE NextDepthFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL); // 다음 이벤트에 대한 핸들러
    HRESULT hr; // 핸들러 결과를

    IplImage *Depth = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4); //Depth 이미지를 담을 자료형 선언
    ...
    cvNamedWindow("DepthImage", CV_WINDOW_AUTOSIZE); // 이미지 출력 창 설정

}
```

- Kinect 초기화 및 기능 활성화

```

void InitializeKinect()
{
    bool NotConnectKinect;

    do
    {
        HRESULT hr = NuiInitialize(NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX);
        if(FAILED(hr))
        {
            system("cls");
            printf("\nFailed to Connect!\n");
            NotConnectKinect = true;
            system("PAUSE");
        }
        else
        {
            printf("Connect Established!\n");
            NotConnectKinect = false;
        }
    }
    while(NotConnectKinect);
} // Kinect 연결과정 초기화

```

- 활성화 기능에 대한 세부설정 및 이벤트 지정

```

int _tmain(int argc, _TCHAR argv[])
{
    HANDLE DepthStreamHandle; // 지금 이벤트를 받을 핸들러
    HANDLE NextDepthFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL); // 다음 이벤트에 대한 핸들러
    HRESULT hr; // 핸들러 결과물

    IplImage *Depth = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4); //Depth 이미지를 받을 자료형 선언
    cvNamedWindow("DepthImage", CV_WINDOW_AUTOSIZE); // 이미지 출력 창 설정

    InitializeKinect(); // Kinect 초기화

    hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_DEPTH_AND_PLAYER_INDEX, NUI_IMAGE_RESOLUTION_640x480, 0, 2,
        NextDepthFrameEvent, &DepthStreamHandle); // 키넥트 이미지를 출력하기 위한 스트림 개방

    if(FAILED(hr))
    {
        printf("Could not open DepthStream\n");
        return hr;
    }
}

```

* 참고사항 : NuiImageStreamOpen 시 NuiInitialize 로 활성화하지 않은 기능을 사용할 경우 오류 발생 -> 동일한 플래그로 지정해줘야 함

- 루프내에서 프레임 받아옴. 종료시 메모리 해제 및 키넥트 종료

```

Depth.cpp • dist.h
(Global Scope) _tmain(i
int _tmain(int argc, _TCHAR argv[])
{
    HANDLE DepthStreamHandle; // 지금 이벤트를 받을 핸들러
    HANDLE NextDepthFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL); // 다음 이벤트에 대한 핸들러
    HRESULT hr; // 핸들러 결과를

    IplImage *Depth = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4); //Depth 이미지를 받을 자료형 선언

    cvNamedWindow("DepthImage", CV_WINDOW_AUTOSIZE); // 이미지 출력 창 설정

    InitializeKinect(); // Kinect 초기화

    hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_DEPTH_AND_PLAYER_INDEX, NUI_IMAGE_RESOLUTION_640x480, 0, 2,
        NextDepthFrameEvent, &DepthStreamHandle); // 키넥트 이미지를 출력하기 위한 스트림 개방

    if(FAILED(hr))
    {
        printf("Could not open DepthStream\n");
        return hr;
    }

    while(1)
    {
        WaitForSingleObject(NextDepthFrameEvent, INFINITE);
        createDepthImage(DepthStreamHandle, Depth);

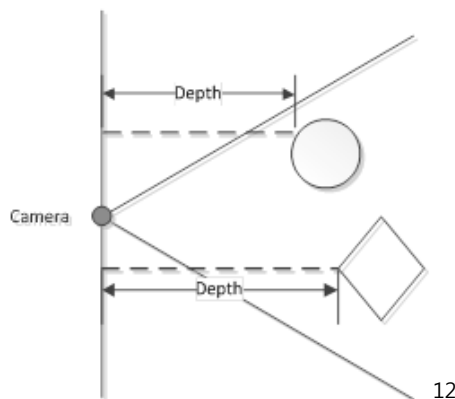
        if(cvWaitKey(10) == 0x001b)
        {
            break;
        }

        cvReleaseImage(&Depth);
        cvDestroyWindow("DepthImage");
        NuiShutdown();

        return 0;
    }
}

```

- 역시 여기서도 depth 를 이용하면 다양한 작업을 할 수 있다. 기존 웹캠과는 다르게 depth 에는 해당 픽셀 속에 거리값이 들어 있기 때문에 그 값을 읽어오면 해당 픽셀과 키넥트간의 거리를 알 수 있다. 여기서 알아야 할 점은 키넥트와 사물간의 거리를 인지하는 방법이다.



¹² Depth Stream Values : <http://msdn.microsoft.com/en-us/library/hh973078.aspx>

키넥트가 물체를 인지하는 Depth 는 키넥트와의 직접적인 거리가 아닌 수직거리이다. 즉, 키넥트를 기준(0,0,0)으로 하는 상대적인 좌표계에서의 z 값이 Depth 거리이다. 이 걸 알기 위해서는 해당 픽셀값을 NuiDepthPixelToDepth()의 인자로 넣어줘야 한다.

- createDepthImage() 정의

```
int createDepthImage(HANDLE h, IplImage *Depth)
{
    const NUI_IMAGE_FRAME *pImageFrame = NULL;

    HRESULT hr = NuiImageStreamGetNextFrame(h, 1000, &pImageFrame);

    if(FAILED(hr))
    {
        printf("Creating DepthImage is Failed\n");
        return -1;
    }

    INuiFrameTexture *pTexture = pImageFrame->pFrameTexture;
    NUI_LOCKED_RECT LockedRect;

    pTexture->LockRect(0, &LockedRect, NULL, 0);

    if(LockedRect.Pitch != 0)
    {
        BYTE *pBuffer = (BYTE*)LockedRect.pBits;
        RGBQUAD *rgbrun = rgb;
        USHORT *pBufferRun = (USHORT*) pBuffer;
    }
}
```

- 픽셀 단위로 이동하면서 받아온 값을 표현해줌

```
if(LockedRect.Pitch != 0)
{
    BYTE *pBuffer = (BYTE*)LockedRect.pBits;
    RGBQUAD *rgbrun = rgb;
    USHORT *pBufferRun = (USHORT*) pBuffer;

    for(int y = 0 ; y < DEPTH_HEIGHT ; y++)
    {
        for(int x = 0 ; x < DEPTH_WIDTH ; x++)
        {
            RGBQUAD quad = Nui_ShortToQuad_Depth(*pBufferRun);
            pBufferRun++;
            *rgbrun = quad;
            rgbrun++;
        }
    }
}
```


- depth 값을 시각적으로 표현하기 위해서 RGBQUAD 자료형(r,g,b 에 대한 속성을 지정할 수 있음)으로 표현했고 이를 변환해주는 Nui_ShortToQuad_Depth 를 만들었다.

- Nui_ShortToQuad_Depth 정의

```
(Global Scope)
#include <opencv\cv.h>
#include <opencv\highgui.h>
#include <opencv\cxcore.h> // 이미지 출력을 위한 OpenCV용 헤더파일

#include <Windows.h>
#include <tchar.h> // win32 어플리케이션을 위한 헤더
#include <NuiApi.h> // Kinect API 사용을 위한 헤더파일

#include <stdio.h> // standard IO

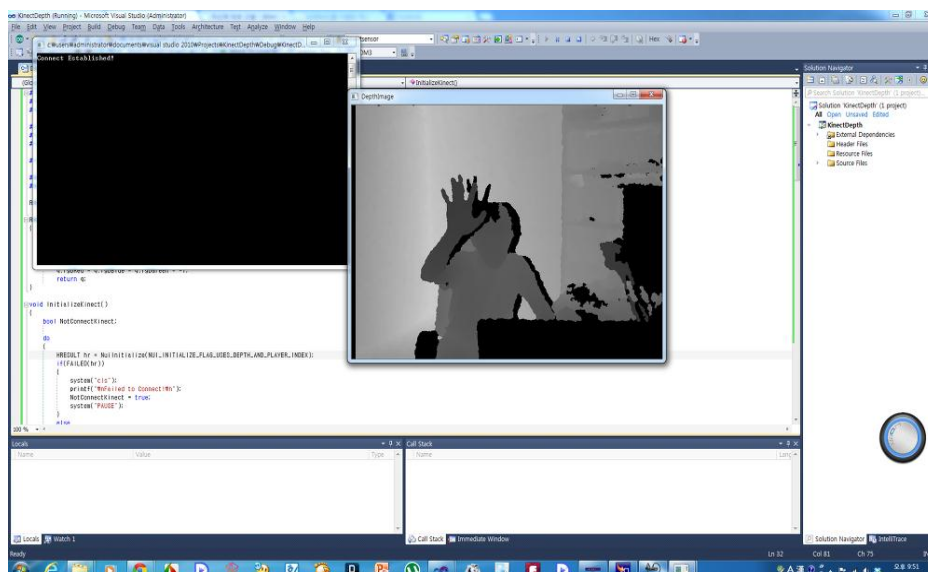
#define DEPTH_WIDTH      640
#define DEPTH_HEIGHT     480 //640x480에 대해서 창을 출력함

RGBQUAD rgb[640*480];

RGBQUAD Nui_ShortToQuad_Depth(USHORT s)
{
    USHORT realDepth = (s&0xffff) >> 3;
    BYTE l = 255-(BYTE)(256+realDepth/(0x0fff));

    RGBQUAD q;
    q.rgbRed = q.rgbBlue = q.rgbGreen = ~l;
    return q;
}
```

- 결과



Depth 값은 GrayScale 이기 때문에 rgb 값이 모두 동일한 값을 가진다. 이를 변환하는 공식이 위와 같고, 위의 내용을 잘 활용하면 특정 영역내의 Depth Value 만 가져올 수 있다. 참고로 위에서는 realDepth 를 구하기 위해서 받은 인자를 3bit shift 했는데 이에 대한 원리는 다음과 같다.

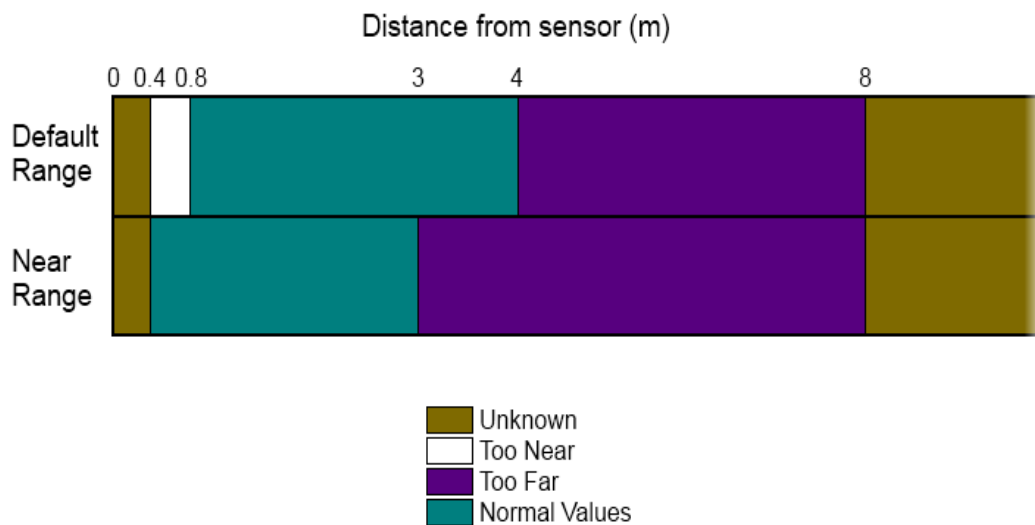
Extended Depth Data

Some applications need depth data beyond the "too far" limit or closer than the "too near" limit even if the resolution or accuracy goes down. Starting with 1.6.0, the depth and segmentation data can be retrieved in either of two formats:

- Full depth information - Each pixel is represented by a structure with two fields: a 16-bit depth and a 16-bit player index. All detected depth values, including those outside the reliable range, are reported. Pixels whose depth is unknown (could not be detected) are reported with a depth value of "0". *Introduced in 1.6.*
- Packed depth information - Each pixel is represented by one 16-bit value. The 13 high-order bits contain the depth value; the 3 low-order bits contain the player index. Any depth value outside the reliable range is replaced with a special value to indicate that it was too near, too far, or unknown.

위에 따르면 한 픽셀을 표현하는데 필요한 depth 는 최소 16bit 이고 이 중 상위 13bit 은 해당 픽셀의 depthValue, 하위 3bit 은 playerId 값이다. 그렇기 때문에 하위 3bit 을 shift 함으로써 depthValue 를 구할 수 있다. 물론 13bit 으로 표현할 수 있는 range 의 한계가 있기 때문에 이를 확장해서 16bit 으로 확장해서 사용할 수 있다. 이 부분은 조금 있다가 더 설명하고자 한다. 아무튼 여기까지 마치면 기본적인 Depth 를 뽑아낼 수 있게 된다.

그런데 Kinect for Windows 에서는 기존의 Xbox 용 Kinect 와는 다르게 Near Mode 를 제공한다. 위의 이미지에서는 표현이 안되었지만, 4m 이상의 Depth 에 대해서는 Too Far Value 를 내보내고 정확히 표현하지 못한다. Kinect SDK v1.6 에서는 4m 이상의 Data 를 표현하기 위해 새로운 플래그가 도입되었다.



13

위의 표는 키넥트가 인식할 수 있는 DepthRange 를 나타내며 Near Mode 를 활성화시키면 위와 같이 40cm 의 거리에서도 인지할 수 있게 된다. 이를 포함해서 4m 이상의 value 를 보여주는 플래그도 있는데 msdn 에 이 플래그들이 정리되어 있다.

NUI_IMAGE Flags

Constant	Value
NUI_IMAGE_STREAM_FLAG_DISTINCT_OVERFLOW_DEPTH_VALUES	0x00040000
NUI_IMAGE_STREAM_FLAG_ENABLE_NEAR_MODE	0x00020000
NUI_IMAGE_STREAM_FLAG_SUPPRESS_NO_FRAME_DATA	0x00010000
NUI_IMAGE_STREAM_FLAG_TOO_FAR_IS_NONZERO	0x00040000

그리고 이 플래그를 사용하기 위해서는 다음과 같이 코드를 삽입하면 된다.

¹³ Depth Space Range : <http://msdn.microsoft.com/en-us/library/hh973078.aspx>

```

int _tmain(int argc, _TCHAR argv[])
{
    HANDLE DepthStreamHandle; // 지금 이벤트를 받을 핸들러
    HANDLE NextDepthFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL); // 다음 이벤트에 대한 핸들러
    HRESULT hr; // 핸들러 결과를

    IplImage *Depth = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4); //Depth 이미지를 받을 자료형 선언
    cvNamedWindow("DepthImage", CV_WINDOW_AUTOSIZE); // 이미지 출력 창 설정
    InitializeKinect(); // Kinect 초기화

    hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_DEPTH_AND_PLAYER_INDEX, NUI_IMAGE_RESOLUTION_640x480, 0, 2,
        NextDepthFrameEvent, &DepthStreamHandle); // 키넥트 이미지를 출력하기 위한 스트림 개방

    NuiImageStreamSetImageFrameFlags(DepthStreamHandle, NUI_IMAGE_STREAM_FLAG_ENABLE_NEAR_MODE);
}

```

NuiImageStreamSetImageFrameFlags 에 현재 받고 있는 Depth handle 과 자신이 원하는 플래그를 정하면 해당 기능을 사용할 수 있다. 마찬가지로 or 연산을 지원하기 때문에 같이 사용하고자 할 때는 OR 연산자를 붙여 추가시키면 된다.

6-3. SkeletonStream 출력

기본적인 Initialize 과정은 ColorStream 이나 DepthStream 을 뽑는 것과 비슷한데 Skeleton 을 사용할 것이기 때문에 이에 대한 플래그도 선언해줘야 한다.

- Skeleton 을 사용하기 위한 플래그를 NuiInitialize 에 넣어줌

```

void InitializeKinect()
{
    bool FailToConnect;

    do
    {
        HRESULT hr = NuiInitialize(NUI_INITIALIZE_FLAG_USES_COLOR | NUI_INITIALIZE_FLAG_USES_SKELETON);
        if(FAILED(hr))
        {
            system("cls");
            cout<<"\nFailed to Connect!\n\n";
            FailToConnect = true;
            system("PAUSE");
        }
        else
        {
            cout<<"\nConnection Established!\n\n";
            FailToConnect = false;
        }
    }
    while(FailToConnect);
}

```

- SkeletonStream 을 뽑기 위한 이벤트 핸들러 생성

```
int _tmain(int argc, _TCHAR* argv[])
{
    HANDLE colorStreamHandle;
    HANDLE nextColorFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    HANDLE skeletonStreamHandle;
    HANDLE nextSkeletonFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    HRESULT hr;

    InitializeKinect();

    IplImage *Color = cvCreateImage(cvSize(COLOR_WIDTH,COLOR_HEIGHT), IPL_DEPTH_8U,4);
    IplImage *Gray = cvCreateImage(cvSize(COLOR_WIDTH,COLOR_HEIGHT), IPL_DEPTH_8U,1);
    IplImage *Skeleton = cvCreateImage(cvSize(COLOR_WIDTH,COLOR_HEIGHT), IPL_DEPTH_8U,4);

    cvNamedWindow("Color Image",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Skeleton Image",CV_WINDOW_AUTOSIZE);
}
```

- Tracking 기능 활성화

```
hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_COLOR, NUI_IMAGE_RESOLUTION_640x480,0.2,nextColorFrameEvent,&colorStreamHandle);
if(FAILED(hr))
{
    cout<<"Could not open ImageStream"<<endl;
    return hr;
}

hr = NuiSkeletonTrackingEnable(nextSkeletonFrameEvent,0);
if(FAILED(hr))
{
    cout<<"Could not open SkeletonStream"<<endl;
    return hr;
}
```

- For 루프 내에서 createSkeleton 사용

```
while(1)
{
    WaitForSingleObject(nextColorFrameEvent,1000);
    createRGBImage(colorStreamHandle,Color);
    WaitForSingleObject(nextSkeletonFrameEvent,0);
    createSkeleton(skeletonStreamHandle,Skeleton);

    cvCvtColor(Color,Gray,CV_RGBA2GRAY);

    cvShowImage("Gray Image",Gray);

    if(cvWaitKey(10) == 0x001b)
    {
        break;
    }
}

NuiShutdown();

cvReleaseImageHeader(&Color);
cvReleaseImage(&Gray);

cvDestroyAllWindows();
return 0;
}
```

- CreateSkeleton() 정의

```
void createSkeleton(HANDLE h, IplImage* Skeleton)
{
    NUI_SKELETON_FRAME skeletonFrame = {0};
    /*IplImage *Skeleton_clear = cvCreateImage(cvSize(COLOR_WIDTH,COLOR_HEIGHT), IPL_DEPTH_8U,4);
    cvCopy(Skeleton_clear,Skeleton);*/

    HRESULT hr = NuiSkeletonGetNextFrame(0,&skeletonFrame);
    if(FAILED(hr))
    {
        return;
    }

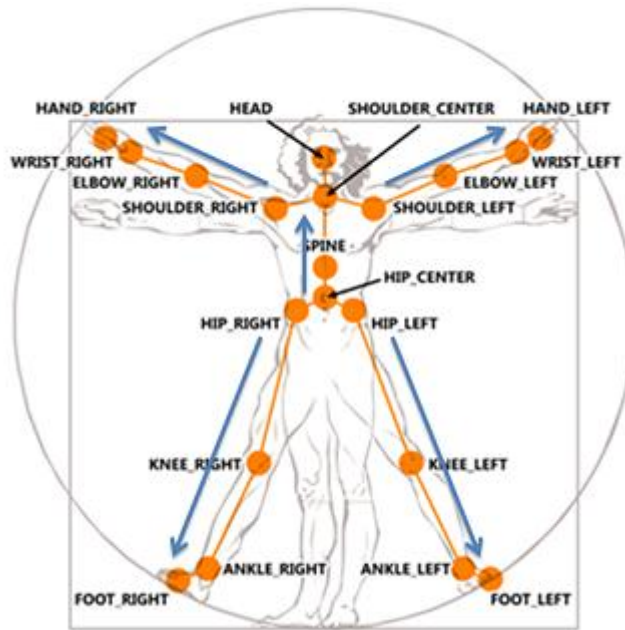
    NuiTransformSmooth(&skeletonFrame,NULL);

    for(int i = 0 ; i<NUI_SKELETON_COUNT;i++)
    {
        NUI_SKELETON_TRACKING_STATE state = skeletonFrame.SkeletonData[i].eTrackingState;

        if(NUI_SKELETON_TRACKED == state)
        {
            drawSkeleton(skeletonFrame.SkeletonData[i],Skeleton);
        }
        cvShowImage("Skeleton Image",Skeleton);
    }

    //cvReleaseImage(&Skeleton_clear);
}
```

- 여기서 지금 주석처리된 부분이 해제 되어야 정상적인 Skeleton Stream 이 프레임단위로 나오게 된다. 위의 과정을 거치지 않으면 그려진 Line 들이 그대로 Frame 상에 남게 된다. 그래서 빈 IplImage 를 생성해서 매초마다 갱신하게 하는 구조를 가진다.
- 키넥트는 최대 6 명의 사람을 인지할 수 있지만 Skeleton Tracking 을 통해서 잡을 수 있는 사람은 최대 2 명이다. 이 둘의 차이는 Joint 의 Position 을 뽑을 수 있냐 없냐의 차이이다.



14

- 키넥트는 사람의 관절을 20 개 영역으로 나눠서 처리한다. 물론 각각의 관절에는 TrackingID 가 할당되어 있기 때문에 여러 사람이 들어가도 한 사람의 관절을 계속 따라갈 수 있다. 즉 Player Index 를 통한 인식은 위와 같은 20 개의 관절 값을 뽑을 수 없고, skeleton Tracking 을 통해 잡은 사람은 각 관절값을 뽑을 수 있다.
- 참고로 키넥트의 관절 인식에 관한 논문은 다음 링크를 보면 도움이 된다.

(<http://research.microsoft.com/pubs/145347/BodyPartRecognition.pdf>)

- Skeleton Tracking 의 State 는 총 3 가지로 나뉜다.
 - Tracked : 관절의 연결과 위치가 정확히 인지된 상태 (바른 값)
 - PositionOnly : 관절의 위치값은 뽑을 수 있지만 정확히 어느 관절인지 인지못함 (틀린 값)
 - NotTracked : 인지가 안되는 상태

즉 이상태를 활용하면 관절이 정확히 추적이 됐을때만 고려해서 실행시킬 수 있다.

¹⁴ Skeleton Position and tracking state : <http://msdn.microsoft.com/en-us/library/jj131025.aspx>

- Tracking Position 을 잡을 CvPoint 변수 선언

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv/cxcore.h>

#include <tchar.h>
#include <windows.h>
#include <NuiApi.h> // Microsoft Kinect SDK

// STL
#include <stdio.h>
#include <iostream>

using namespace std;
using namespace cv;

#define COLOR_WIDTH 640
#define COLOR_HEIGHT 480

CvPoint points[NUI_SKELETON_POSITION_COUNT];
```

- NUI_SKELETON_POSITION_COUNT = 20 이다. 이 부분은 앞에서 설명한 부분으로 말그대로 관절 20 개의 position 을 담을 자료형을 선언한 것이다.

- drawSkeleton 정의

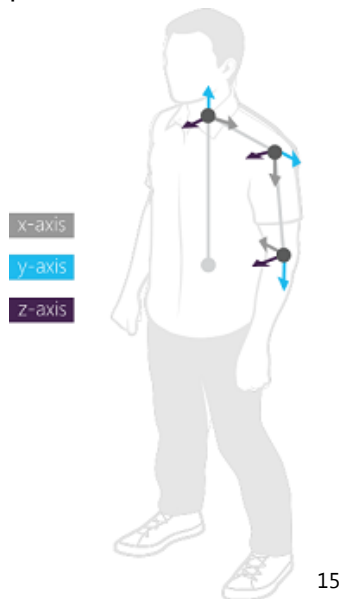
```
void drawSkeleton(const NUI_SKELETON_DATA &position, IplImage *Skeleton)
{
    for(int i = 0; i < NUI_SKELETON_POSITION_COUNT; ++i)
    {
        points[i] = SkeletonToScreen(position.SkeletonPositions[i]);
    }

    drawBone(position, NUI_SKELETON_POSITION_SHOULDER_RIGHT, NUI_SKELETON_POSITION_ELBOW_RIGHT, Skeleton);
    drawBone(position, NUI_SKELETON_POSITION_ELBOW_RIGHT, NUI_SKELETON_POSITION_WRIST_RIGHT, Skeleton);
    drawBone(position, NUI_SKELETON_POSITION_WRIST_RIGHT, NUI_SKELETON_POSITION_HAND_RIGHT, Skeleton);

    drawBone(position, NUI_SKELETON_POSITION_SHOULDER_LEFT, NUI_SKELETON_POSITION_ELBOW_LEFT, Skeleton);
    drawBone(position, NUI_SKELETON_POSITION_ELBOW_LEFT, NUI_SKELETON_POSITION_WRIST_LEFT, Skeleton);
    drawBone(position, NUI_SKELETON_POSITION_WRIST_LEFT, NUI_SKELETON_POSITION_HAND_LEFT, Skeleton);
}
```

- Tracked 가 된 상태에서 20 개의 관절값을 앞에서 선언한 Point 에 집어넣는다 그래서 이렇게 받은 값을 cvCircle 로 표현하게 되면 그 점이 관절이 되게 된다.

- 밑의 drawBone 은 뼈를 그려주는 함수인데 여기서 받는 인자인 position 에 대해서 따져봐야 한다.



- 기본적으로 키넥트로부터 받아오는 Skeleton 의 position 은 x,y,z,w 값을 가진 Vector 4 형 꼴이다. 따라서 IplImage 상에 바로 표현을 해줄 수 없고, 이를 화면의 픽셀 좌표로 변환시켜주는 함수가 필요하다. 이 역할을 SkeletonToScreen()이 해준다. 그래서 여기를 통해서 나온 결과 값이 drawBone 에 들어가게 되는 것이다.

- SkeletonToScreen() 정의

```

CvPoint SkeletonToScreen(Vector4 skeletonPoint)
{
    LONG x, y;
    USHORT depth;

    NuiTransformSkeletonToDepthImage(skeletonPoint, &x, &y, &depth, NUI_IMAGE_RESOLUTION_640x480);

    float screenPointX = static_cast<float>(x);
    float screenPointY = static_cast<float>(y);

    return cvPoint(screenPointX, screenPointY);
}

```

- 위에 쓰인 NuiTransformSkeletonToDepthImage 는 해당 픽셀의 skeleton joint 값을 받아서 ColorStream 과 맞춰주는 일종의 Mapper 이다.

¹⁵ Hierarchical Rotation : <http://msdn.microsoft.com/en-us/library/hh973073.aspx>

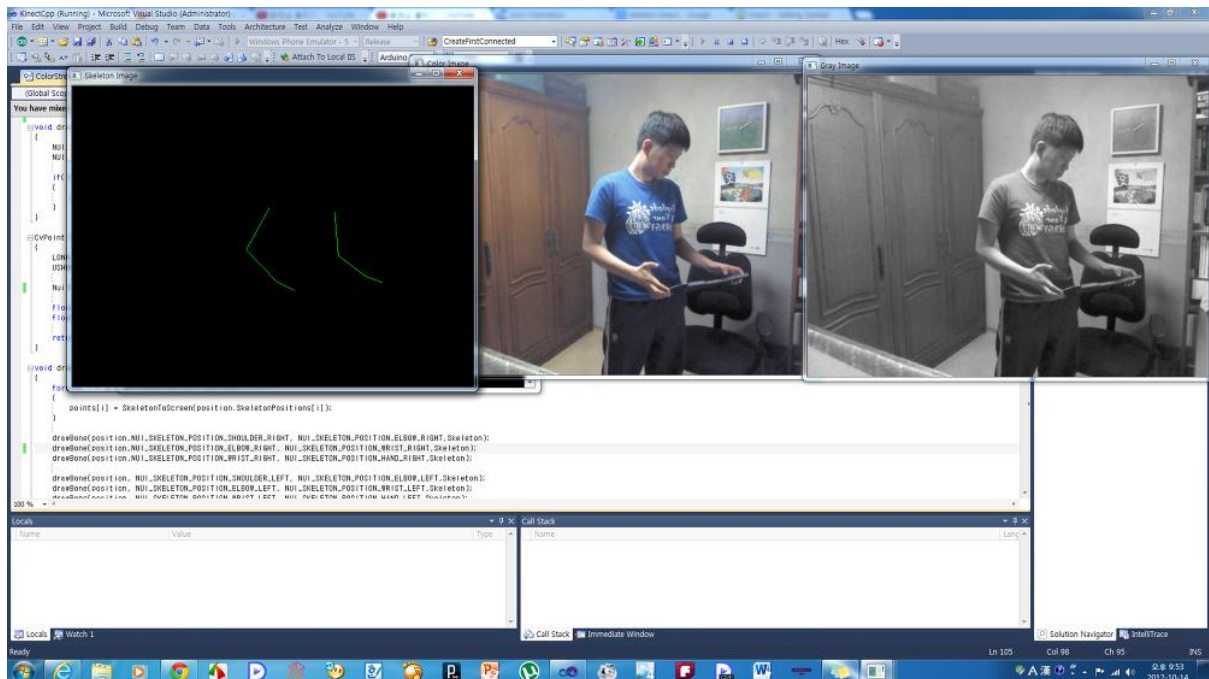
- 지금은 화면상에 그리는 것이 목적이기 때문에 ColorStream 의 Resolution 을 고려해줘야 한다.

- DrawBone() 정의

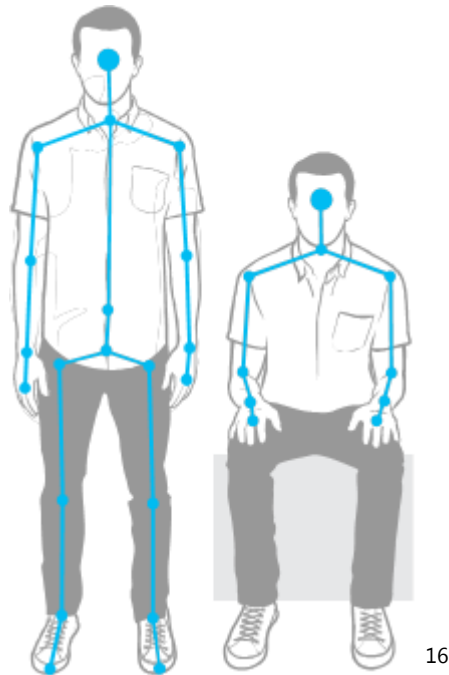
```
void drawBone(const NUI_SKELETON_DATA &position, NUI_SKELETON_POSITION_INDEX j1, NUI_SKELETON_POSITION_INDEX j2, IplImage *Skeleton)
{
    NUI_SKELETON_POSITION_TRACKING_STATE j1state = position.eSkeletonPositionTrackingState[j1];
    NUI_SKELETON_POSITION_TRACKING_STATE j2state = position.eSkeletonPositionTrackingState[j2];

    if(j1state == NUI_SKELETON_POSITION_TRACKED && j2state == NUI_SKELETON_POSITION_TRACKED)
    {
        cvLine(Skeleton, points[j1], points[j2], CV_RGB(0,255,0), 1, 8, 0);
    }
}
```

- 여기까지 마치면 기본적인 SkeletonStream 을 표현할 수 있다.



기본적으로 Kinect for Windows에서는 Seated Mode Skeleton Tracking 을 지원한다. 이때는 기존의 Default Mode 가 20 개의 관절을 인식했던 것과 달리 상반신(10 개의 관절) 만 인식할 수 있게끔 할 수 있다.



역시 이 기능을 사용하기 위해서는 다음과 같이 활성화 구문에 플래그를 달아주면 된다.

NUI_SKELETON Flags

Constant	Value
NUI_SKELETON_TRACKING_FLAG_SUPPRESS_NO_FRAME_DATA	0x00000001
NUI_SKELETON_TRACKING_FLAG_TITLE_SETS_TRACKED_SKELETONS	0x00000002
NUI_SKELETON_TRACKING_FLAG_ENABLE_SEATED_SUPPORT	0x00000004
NUI_SKELETON_TRACKING_FLAG_ENABLE_IN_NEAR_RANGE	0x00000008

예시) NuiSkeletonTrackingEnable(

m_hNextSkeletonEvent, NUI_SKELETON_TRACKING_FLAG_ENABLE_SEATED_SUPPORT);

¹⁶ Tracking Mode : <http://msdn.microsoft.com/en-us/library/hh973077.aspx>

6-4. IRStream 출력

Kinect SDK 가 v1.6 으로 업데이트 되면서 새롭게 추가된 기능이 바로 적외선 이미지를 뽑아올 수 있는 기능이다. MS 의 말로는 low light intensity 의 환경에서도 사람을 뽑을 수 있다고 한다. 유의할 점은 이 이미지 자체는 ColorStream 을 기반으로 하기 때문에 colorImage 와 동시에 호출할 수 없다.

- 필요한 header 파일과 자료형 선언

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv/cxcore.h>

#include <Windows.h>
#include <tchar.h>
#include <NuiApi.h>

#include <stdio.h>

#define DEPTH_WIDTH      640
#define DEPTH_HEIGHT     480

#define COLOR_WIDTH      640
#define COLOR_HEIGHT     480

void InitializeKinect();
void createIRImage(HANDLE h, IplImage *InfraRed);

RGBQUAD Nui_ShortToQuad_IR(USHORT s);
RGBQUAD m_irWk[COLOR_WIDTH * COLOR_HEIGHT];
```

- IR 이미지도 GrayScale 로 표현이 되기 때문에 위와 같이 RGBQUAD 형으로 처리한다.

- IRStream 을 다룰 이벤트 생성

```
int _tmain(int argc, _TCHAR argv[])
{
    HANDLE IRStreamHandle;
    HANDLE nextIRFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    HRESULT hr;
```

- IR 을 뿌려줄 IplImage 와 window 생성

```
int _tmain(int argc, _TCHAR argv[])
{
    HANDLE IRStreamHandle;
    HANDLE nextIRFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    HRESULT hr;

    InitializeKinect();

    IplImage *InfraRed = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4);
    cvNamedWindow("IR Image", CV_WINDOW_AUTOSIZE);
}
```

- NuiImageStreamOpen 을 통해서 IRStream 활성화

```
int _tmain(int argc, _TCHAR argv[])
{
    HANDLE IRStreamHandle;
    HANDLE nextIRFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    HRESULT hr;

    InitializeKinect();

    IplImage *InfraRed = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4);
    cvNamedWindow("IR Image", CV_WINDOW_AUTOSIZE);

    hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_COLOR_INFRARED, NUI_IMAGE_RESOLUTION_640x480, 0, 2, nextIRFrameEvent, &IRStreamHandle);

    if(FAILED(hr))
    {
        printf("Could not open IRStream\n");
        return hr;
    }
}
```

- 유의할 점은 NuiInitialize 에서 활성화되는 기능 color 이지만 ImageStream 을 활성화시킬때는 INFRARED 가 붙는다. 앞에서 말했지만 ColorStream 과 IRStream 은 같이 뽑을 수 없다.

- While 구문 내에서 IRStream 을 다룰 함수 생성

```

int _tmain(int argc, _TCHAR argv[])
{
    HANDLE IRStreamHandle;
    HANDLE nextIRFrameEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    HRESULT hr;

    InitializeKinect();

    IplImage *InfraRed = cvCreateImage(cvSize(DEPTH_WIDTH, DEPTH_HEIGHT), IPL_DEPTH_8U, 4);

    cvNamedWindow("IR Image", CV_WINDOW_AUTOSIZE);

    hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_COLOR_INFRARED, NUI_IMAGE_RESOLUTION_640x480, 0, 2, nextIRFrameEvent, &IRStreamHandle);

    if(FAILED(hr))
    {
        printf("Could not open IRStream\n");
        return hr;
    }

    while(1)
    {
        WaitForSingleObject(nextIRFrameEvent, 1000);
        createIRImage(IRStreamHandle, InfraRed);

        if(cvWaitKey(10) == 0x001b)
        {
            break;
        }
    }
}

```

- 종료부분 설정

```

    while(1)
    {
        WaitForSingleObject(nextIRFrameEvent, 1000);
        createIRImage(IRStreamHandle, InfraRed);

        if(cvWaitKey(10) == 0x001b)
        {
            break;
        }
    }

    NuiShutdown();

    cvReleaseImageHeader(&InfraRed);

    cvDestroyAllWindows();
    return 0;
}

```

- NuiInitialize() 정의

```
void InitializeKinect()
{
    bool FailToConnect;
    do
    {
        HRESULT hr = NuiInitialize(NUI_INITIALIZE_FLAG_USES_COLOR | NUI_INITIALIZE_FLAG_USES_DEPTH | NUI_INITIALIZE_FLAG_USES_SKELETON);
        if(FAILED(hr))
        {
            system("cls");
            printf("Failed to connect!\n\n");
            FailToConnect = true;
            system("PAUSE");
        }
        else
        {
            printf("Connection Established!\n\n");
            FailToConnect = false;
        }
    } while (FailToConnect);
}
```

- createIRImage() 정의

```
void createIRImage(HANDLE h, IplImage *InfraRed)
{
    const NUI_IMAGE_FRAME *pImageFrame = NULL;
    HRESULT hr = NuiImageStreamGetNextFrame(h, 0, &pImageFrame);

    if(FAILED(hr))
    {
        return;
    }

    INuiFrameTexture *pTexture = pImageFrame->pFrameTexture;
    NUI_LOCKED_RECT LockedRect;

    pTexture->LockRect(0, &LockedRect, NULL, 0);

    if(LockedRect.Pitch != 0)
    {
        BYTE *pBuffer = (BYTE*)LockedRect.pBits;
        RGBQUAD *rgbrun = m_irWk;
        USHORT *pBufferRun = (USHORT*) pBuffer;

        for(int y = 0; y < COLOR_HEIGHT; y++)
        {
            for(int x = 0; x < COLOR_WIDTH; x++)
            {
                RGBQUAD quad = Nui_ShortToQuad_IR(*pBufferRun);
                pBufferRun++;
                *rgbrun = quad;
                rgbrun++;
            }
        }

        cvSetData(InfraRed, (BYTE*)m_irWk, InfraRed->widthStep);
        cvShowImage("IR Image", InfraRed);
    }

    NuiImageStreamReleaseFrame(h, pImageFrame);
}
```

- 전체적인 코드 구성은 DepthImage 를 뽑는 것과 동일하다. 여기서는 Nui_shortToQuad_IR()이라는 함수를 통해서 데이터를 픽셀로 변환하는 과정을 거친다.

- Nui_shortToQuad_IR() 정의

```
RGBQUAD Nui_ShortToQuad_IR(USHORT s)
{
    USHORT pixel = s>>8;
    BYTE intensity = pixel;
    RGBQUAD q;
    q.rgbBlue = intensity;
    q.rgbGreen = intensity;
    q.rgbRed = intensity;
    return q;
}
```

형태 자체는 depth 와 비슷하지만 IR 이미지를 뽑을 때는 8bit 을 shift 한다. 이 값을 RGBQUAD 의 rgbBlue, rgbGreen, rgbRed 에 넣어준다.



여기까지 하면 IR 이미지를 뽑을 수 있게 된다.

7. 기타 활용할 수 있는 함수

7-1. 실좌표계값으로 변환시켜주는 함수 : NuiTransformDepthImageToSkeleton()

기관과제로 진행했던 프로젝트에선 손의 DepthData 를 뽑아내서 키넥트의 상대적인 위치로 바꿔주는 것이 필요했다. 하지만 DepthFrame 상의 Pixel 을 읽어오면 해당 Pixel 의 z 값(키넥트와 대상간의 상대적인 거리)만 구할 수 있다. 이때 위에서 언급한 NuiTransformDepthImageToSkeleton 을 넣게 되면 Vector4 형 자료를 구할 수 있고, 여기서 나오는 x,y,z 값이 실좌표계값이다. 샘플 코드는 다음과 같다.

```
void SetCurHandPos(CvRect handRect, CvPoint COM, IplImage *handDepth, Vector4 vec)
{
    if(handRect.width > 20)
    {
        if(position == LEFT)
        {
            COM = cvPoint(0,0);
        }
        else
        {
            cvSetImageROI(handDepth, handRect);
            GetContour(handDepth);
            COM = distTF(handDepth);
            cvResetImageROI(handDepth);
        }
    }

    cvDrawRect(handDepth, cvPoint(handRect.x, handRect.y), cvPoint(handRect.x+handRect.width, handRect.y+handRect.height), CV_RGB(255,255,255), 1, 8, 0);
    cvCircle(handDepth, cvPoint(COM.x, COM.y), 3, CV_RGB(255,255,255), -1, 8, 0);

    vec.x = 0; vec.y = 0;

    Handpos.x = COM.x;
    Handpos.y = COM.y;

    vec = NuiTransformDepthImageToSkeleton((LONG)COM.x, (LONG)COM.y, realDepth, NUI_IMAGE_RESOLUTION_640x480);
    printf("x: %.3f y: %.3f z: %.3f\n", vec.x, vec.y, vec.z);
}
```

자신이 원하는 지점의 PixelPosition을 구함

앞에서 구한 PixelPosition을 대입

PixelPosition에서의 depthValue를 구함

* HandPos 는 전역적으로 선언되어 있는 CvPoint.

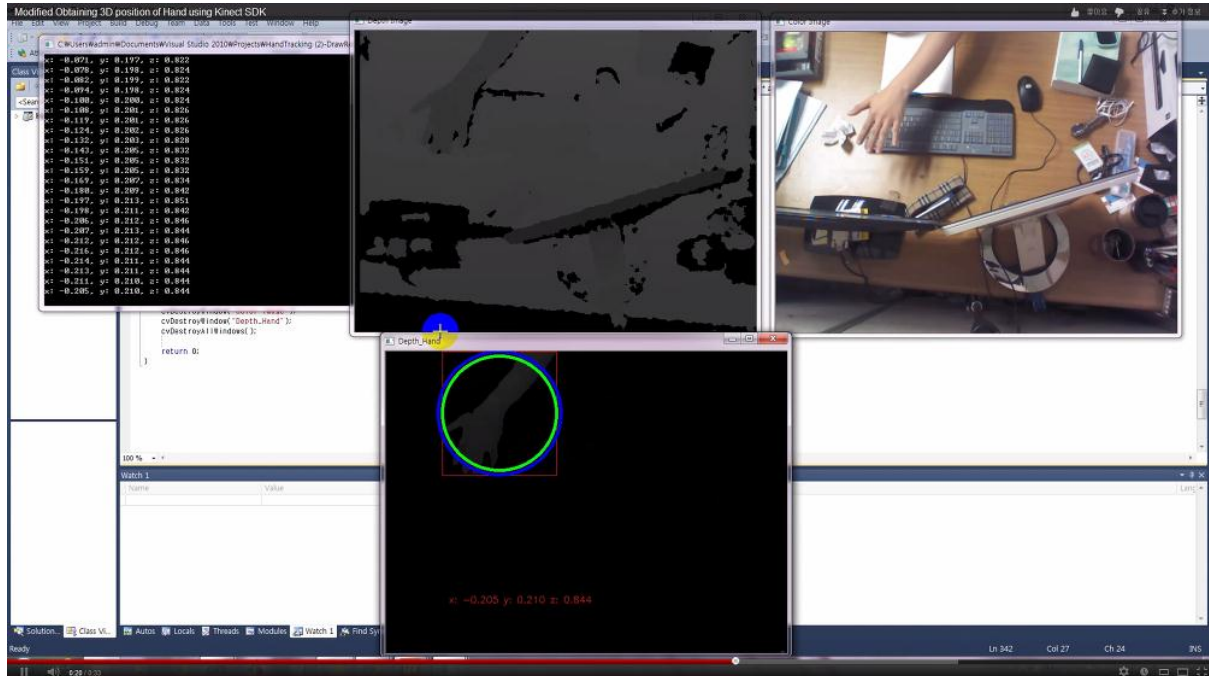
특정 픽셀 위치에서의 depthValue 는 다음과 같이 구할 수 있다.

```
if(LockedRect.Pitch != 0)
{
    BYTE * pBuffer = (BYTE*)LockedRect.pBits;
    RGBQUAD *rgbrun = m_rgbWk;
    USHORT *pBufferRun = (USHORT*) pBuffer;

    for( int y = 0 ; y < DEPTH_HEIGHT ; y++ )
    {
        for( int x = 0 ; x < DEPTH_WIDTH ; x++ )
        {
            RGBQUAD quad = Nui_ShortToQuad_Depth( *pBufferRun );
            if(x == Handpos.x && y == Handpos.y)
            {
                realDepth = *pBufferRun;
                pBufferRun++;
                *rgbrun = quad;
                rgbrun++;
            }
        }
    }

    cvSetData(Depth, (BYTE*) m_rgbWk, Depth->widthStep);
    cvShowImage("Depth Image", Depth);
}
```

앞에서 언급한 createDepthImage()에서 handPos 일때, 그점에서의 depthValue 를 잡도록 했다. 결과는 다음과 같다.



여기서 x는 키넥트를 기준으로 오른쪽일때 음수,왼쪽일때 양수를 가진다. Y는 수직방향, z는 키넥트와 대상간의 떨어진 거리를 나타낸다.

7-2. Distance Transform 을 통한 손 중심점 유추

출처 : <http://cafe.naver.com/opencv/3264>

관련 논문 : Parametric Correspondence and Chamfer Matching:
Two new Techniques for Image Matching – Barrows et al

Distance Transform이란 프레임상의 한 픽셀에서 특정 외곽선까지 이르는 거리를 변환시켜서 표현하는 방법이다. 이 거리를 grayscale로 표현하게 되면 하나의 Distance Map이 나오게 된다. 이 특성을 활용해서 손 중심점을 유추했다. 기본적인 Distance Map의 결과는 다음과 같다.


```

CvPoint distTF(IplImage* img) //손의 무게중심을 찾기위한..
{
    float mask[3];
    IplImage* dist = 0;
    IplImage* dist8u = 0;
    IplImage* dist32s = 0;
    int max;

    dist = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
    dist8u = cvCloneImage( img );
    dist32s = cvCreateImage( cvGetSize(img), IPL_DEPTH_32S, 1 );

    mask[0] = 1.f;
    mask[1] = 1.5f;

    cvDistTransform( img, dist, CV_DIST_USER, 3, mask, NULL );

    cvConvertScale( dist, dist, 1000 , 0 );
    cvPow( dist, dist, 0.5 );

    cvConvertScale( dist, dist32s, 1.0, 0.5 );
    cvAndS( dist32s, cvScalarAll(255), dist32s, 0 );
    cvConvertScale( dist32s, dist8u, 1, 0 );

    for(int i= max=0;i<dist8u->height;i++)
    {
        int index=i * dist8u->widthStep;
        for(int j=0;j<dist8u->width;j++)
        {
            if((unsigned char)dist8u->imageData[index + j]>max)
            {
                max=(unsigned char)dist8u->imageData[index + j];
                p.x =j; p.y =i;
            }
        }
    }

    cvReleaseImage(&dist);
    cvReleaseImage(&dist8u);
    cvReleaseImage(&dist32s);

    return p;
}


```

7-3. Background subtraction

배경을 없애는 이유는 키넥트가 화면을 바라봤을 때 기기의 특성으로 인한 잡음이 발생하고 이로 인해서 결과에 영향을 주기 때문이다. 즉 잡음이 발생하는 지점을 여러 프레임에 나눠서 누적인 후에 결과 프레임에서 빼주면 결국 배경이 제거된 형태를 취할 수 있다. 물론 이 방법이 아닌 Learning OpenCV ¹⁷책에 나온 배경 제거법을 사용해도 가능할 거라 생각한다. 샘플 코드는 다음과 같다

- Main 구문 상에서 FrameCount 를 선언한다.

```
int FrameCount = 0;
int handCount = 0;
Vector4 vec;
vec.x = 0, vec.y = 0, vec.z = 0, vec.w = 0;
```



- 그 후 while 루프 내에서 초기 frameCount 가 20 이 될 때까지의 배경을 계속 검사한다.

```
while(1)
{
    GenerateImage(nextColorFrameEvent,nextDepthFrameEvent,colorStreamHandle,depthStreamHandle,Color,Depth);
    cvSmooth(Depth,Depth,CV_GAUSSIAN,3,0,0,0);
    cvCvtColor(Color,gray,CV_RGBA2GRAY);
    cvCvtColor(Depth,dummy,CV_BGRA2GRAY);
    BackgroundModeling(FrameCount,dummy,mask_Back,hand);
    if(FrameCount >=20)
    {
        cvCopy(hand,handDepth);
        Labeling(hand,handRect);
        SetCurHandPos(handRect,COM,handDepth,vec);
        cvShowImage("Depth_Hand",handDepth);
    }
    FrameCount++;
    if(cvWaitKey(10) == 0x001b)
    {
        break;
    }
}
```

¹⁷ Learning OpenCV – Gary Bradski : <http://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/0596516134>

- BackgroundModeling 부분은 다음과 같이 구성된다.

여기서 20Frame 전까지는 그 값을 누적하고 20Frame 이 넘으면 그 누적된 값을 빼주는 형태이다.

```
void BackgroundModeling(int FrameCount, IplImage *dummy, IplImage *mask_Back, IplImage *hand)
{
    if(FrameCount == 0)
    {
        cvCopy(dummy, mask_Back);
    }
    else if(FrameCount < 20)
    {
        accumulateBackground(dummy, mask_Back);
    }
    else if(FrameCount >= 20)
    {
        backgroundSub(dummy, mask_Back, hand);
    }
}
```

- accumulateBackground 는 다음과 같이 구성된다.

픽셀값을 일었을 때 기존의 픽셀값보다 새로 들어온 픽셀값이 더 크면 그 값으로 누적되는 형태이다

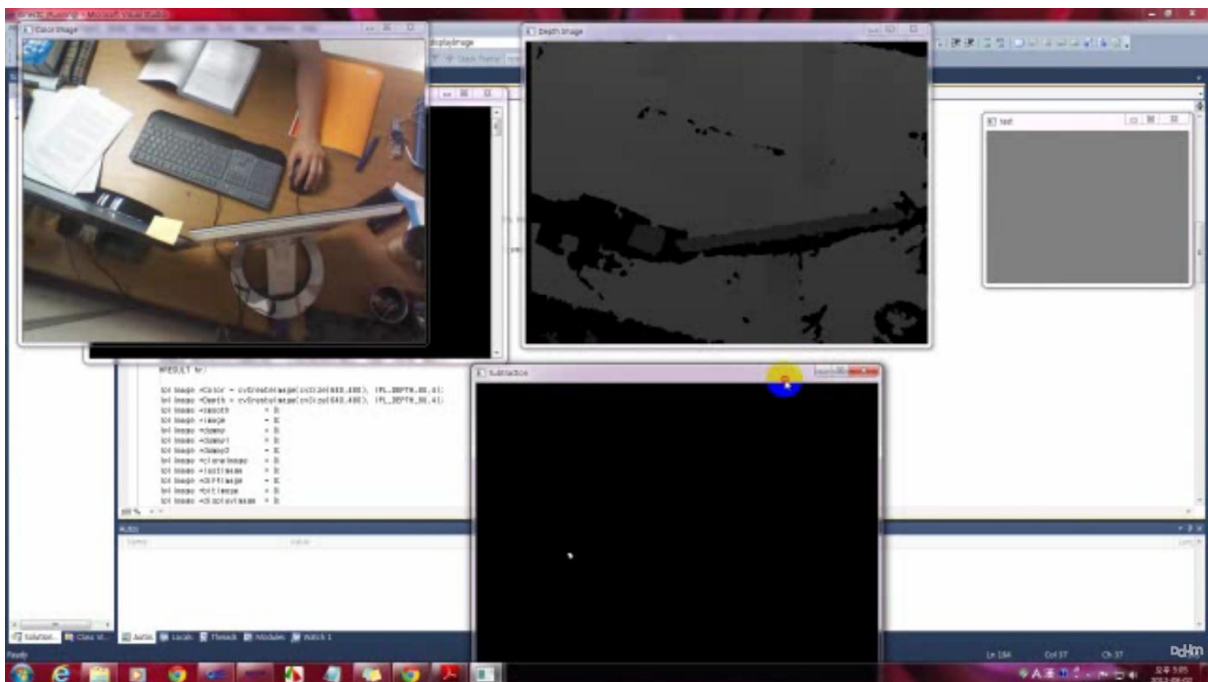
```
void accumulateBackground( IplImage *I, IplImage* mask1 ){
    for(int i = 0; i < DEPTH_HEIGHT; i++)
        for(int j = 0; j < DEPTH_WIDTH; j++)
        {
            if(mask1->imageData[i*DEPTH_WIDTH + j] < I->imageData[i*DEPTH_WIDTH + j])
                mask1->imageData[i*DEPTH_WIDTH + j] = I->imageData[i*DEPTH_WIDTH + j];
        }
}
```

- 20Frame 이 넘으면 앞에서 구한 결과값을 토대로 배경에서 제거해줘야 한다.

```
void backgroundSub(IplImage *I, IplImage *mask1, IplImage* mask2)
{
    for(int i = 0; i < DEPTH_HEIGHT; i++)
    {
        for(int j = 0; j < DEPTH_WIDTH; j++) // Width와 Height이 순차적으로 넘어갈 때
        {
            mask2->imageData[i*DEPTH_WIDTH + j] = 0; // 결과 이미지의 픽셀값 초기화
            if(I->imageData[i*DEPTH_WIDTH + j] > 20)
            {
                if(mask1->imageData[i*DEPTH_WIDTH + j] == 0)
                    mask2->imageData[i*DEPTH_WIDTH + j] = (char)255;
                else
                {
                    int nDiff = mask1->imageData[i*DEPTH_WIDTH + j] - I->imageData[i*DEPTH_WIDTH + j];
                    if(nDiff > 5){
                        if(nDiff+3 > 255)
                            mask2->imageData[i*DEPTH_WIDTH + j] = (char)255;
                        else
                            mask2->imageData[i*DEPTH_WIDTH + j] = nDiff+3;
                    }
                }
            }
        }
    }

    cvErode(mask2, mask2, NULL, 1);
    cvDilate(mask2, mask2, NULL, 1);
    cvSmooth(mask2, mask2, CV_MEDIAN, 3, 0, 0);
}
```

다만 문제가 있다면 배경이 제거된 상태에서 다른 물체가 그 제거된 위치로 이동하면 그때의 픽셀값이 본연의 값이 아닌 다른 값이 나오게 되는 것이다. 이 부분은 따로 코드로 구성하던지, 환경이 정해진 상태에서만 사용해야 된다. 그리고 마지막에는 위의 배경을 제거한 상태에서도 잡음이 발생하여 그걸 해결하기 위해서 Erode / Dilate 연산을 실시하였다. 결과는 다음과 같이 나온다.



7-4. 손의 Contour 및 ConvexityDefects

앞에서 나온 방법을 사용하면 배경이 제거된 상태에서 손의 중심점 및 그 손의 실좌표점을 구할 수 있다. 원래 구상하려던 방법은 손의 FingerTip 을 구한 후에 그 지점의 realDepth 를 읽어옴으로써 손끝의 실좌표점을 구하려고 했었다.

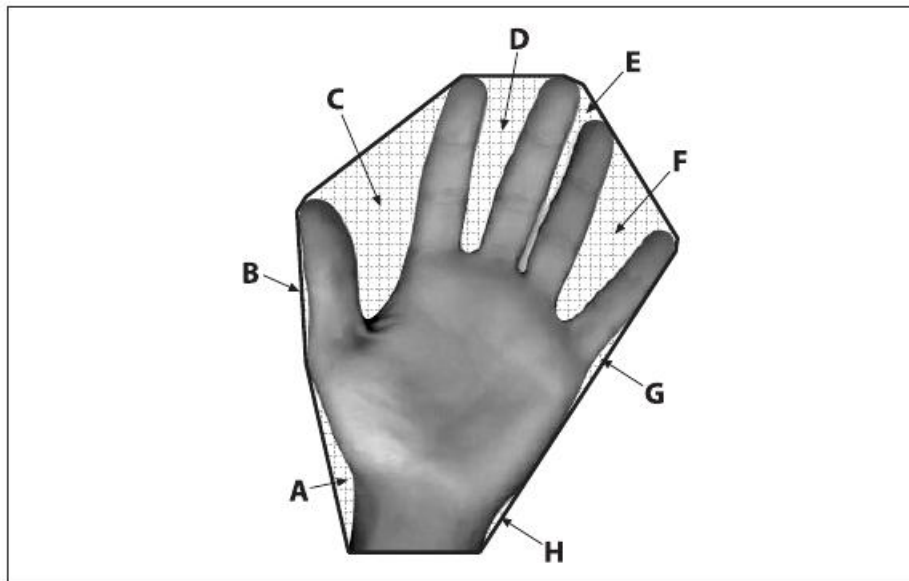


Figure 8-12. Convexity defects: the dark contour line is a convex hull around the hand; the gridded regions (A-H) are convexity defects in the hand contour relative to the convex hull

18

위와 같이 손에 대한 ConvexHull 을 구한 후 hull 속의 Defects 를 구하면 FingerTip 을 얻을 수 있다.

- 초기 변수 선언

```

CvRect GetContour(IplImage *src)
{
    CvMemStorage *storage1 = cvCreateMemStorage(0);
    CvMemStorage *storage2 = cvCreateMemStorage(0);
    CvMemStorage *storage3 = cvCreateMemStorage(0);
    CvSeq* contours = 0;
    CvSeq* ptseq = cvCreateSeq(CV_SEQ_KIND_GENERIC | CV_32SC2, sizeof(CvContour), sizeof(CvPoint), storage1);
    CvSeq* defects = cvCreateSeq( CV_SEQ_KIND_GENERIC | CV_32SC2, sizeof(CvSeq), sizeof(CvPoint), storage2);
    CvSeq* hull;
    CvSeq* defectPoint = cvCreateSeq( CV_SEQ_ELTYPE_POINT, sizeof(CvSeq), sizeof(CvPoint), storage3);
    CvPoint pt0;

    IplImage *tmp = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U, 1);
    IplImage *test = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U, 1);
}

```

- GrayScale / 1 channel image 로 만든 후에 이미지의 Contour 를 찾는다.

```

if(src->nChannels == 3)
    cvCvtColor(src, tmp, CV_BGR2GRAY);
else
    cvCopy(src, tmp);

cvThreshold(tmp, test, 1, 255, CV_THRESH_BINARY);
cvThreshold(tmp, tmp, 1, 255, CV_THRESH_BINARY);
cvFindContours(tmp, storage1, &contours, sizeof(CvContour), CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE); //contour에 해당하는 점들을 List화 시킴
cvZero(tmp);
//cvSmooth(tmp, tmp, CV_MEDIAN, 7, 7, 0, 0);

```


- 이때 만든 Contour 에는 잡음으로 인해서 깔끔하게 나오지 않기 때문에 간략하게 축약하는 방법을 쓰는게 좋고 여기서는 그중에 Douglas- Peucker Approximation 을 사용했다. 그 후에 Contour 와 hull, defect 를 구하였다.

```
contours = cvApproxPoly(contours, sizeof(CvContour), storage1, CV_POLY_APPROX_DP, 3, 1); // douglas Peucker Approx 적용
if(contours)
    cvDrawContours(tmp, contours, CV_RGB(255, 255, 255), CV_RGB(255, 255, 255), 100, 1);
hull = cvConvexHull2(contours, 0, CV_CLOCKWISE, 0); //찾은 Contour 사이에서 Hull을 찾음
defects = cvConvexityDefects(contours, hull, NULL); // Contour hull 사이 Defect를 찾아냄
```

- 그 후에 cvSeq 에 들어 있는 defects 를 탐색하는 과정을 거친다.

여기까지 하면 손끝점에 대한 정보가 defectArray[i]에 들어있게 된다.

```
for(; defects; defects = defects->h_next) //defect도 하나의 sequence에 저장됨
{
    int total = defects->total;
    if(!total)
        continue;

    CvConvexityDefect* defectArray = (CvConvexityDefect*)malloc(sizeof(CvConvexityDefect)*total);
    cvCvtSeqToArray(defects, defectArray, CV_WHOLE_SEQ); // sequence에 있는 내용들을 배열로 옮김
    for(int i = 0; i < total; i++)
    {
        //cvCircle(src, *(defectArray[i].depth_point), 5, CV_RGB(255, 255, 255), -1, 8, 0);
        cvCircle(src, *(defectArray[i].start), 5, CV_RGB(255, 255, 255), -1, 8, 0);
        //cvCircle(src, *(defectArray[i].end), 5, CV_RGB(255, 255, 255), -1, 8, 0); // 특정 조건에 따른 Point 표현
    }
}
```

- Hull 에 대해서 그리고 싶다면 아래에 코드를 더 작성하면 된다.

```
pt0.x = 0, pt0.y = 0;
CvPoint start_pt;
CvPoint end_pt;

for(int x = 0; x < hull->total; x++)
{
    CvPoint hull_pt = **CV_GET_SEQ_ELEM(CvPoint*, hull, x);

    if(pt0.x == 0 && pt0.y == 0)
    {
        pt0 = hull_pt;
        end_pt = pt0;
        start_pt = hull_pt;
    }

    double dx = hull_pt.x - pt0.x;
    double dy = hull_pt.y - pt0.y;
    double dx2 = pow(dx, 2);
    double dy2 = pow(dy, 2);
    double fDist = sqrt(dx2 + dy2);

    if(fDist > 15)
    {
        //cvCircle(src, hull_pt, 3, CV_RGB(255, 255, 255), 3, 8, 0);
        //cvLine(src, pt0, hull_pt, CV_RGB(255, 255, 255), 2, 8);
        pt0 = hull_pt;
    }

    /*if(x == hull->total - 1)
        cvLine(src, hull_pt, end_pt, CV_RGB(255, 255, 255), 2, 8); */
}
```

7-5. Blob Detection

Blob Detection 자체는 손에 해당하는 픽셀을 잡고 관련 정보를 관심영역으로 선언하기 위해서 필요한 방법이며, 잡음도 하나의 blob 으로 인식될 수 있기 때문에 생성에 있어서 제한 조건을 두어야 한다(blob 의 최소 크기나 최대 크기에 대한 제한)

많은 blob detection 방법이 있지만 여기서 사용한 방법은 마틴님이 만든 blob library 이다. 자세한 방법은 <http://martinblog.tistory.com/826> 을 참고하면 된다. 유의할 점은 depth 같은 경우에는 손이 겹쳐있는 경우에도 다른 PixelData 를 가진다. 그렇기 때문에 해당 프레임 상에서는 3 개의 blob 이 생겨야 한다. 그리고 같은 depthValue 를 가진 blob 끼리는 같은 blob 이라고 인정되어야 한다. 하지만 위 라이브러리에서는 그 부분에 대한 정의가 되어 있지 않고 간단히 구현할 수 있는 내용들로 표현되어 있다. 따라서 손동작 인식시 필요한 내용은 개별적으로 구현해야 된다.

출처 및 도움이 될만한 자료

- Kinect for Windows SDK Programming Guide
키넥트 기능 사용법을 가르쳐주는 MS 공식 라이브러리
-> <http://msdn.microsoft.com/en-us/library/hh855348.aspx>
- Kinect for Windows SDK Interaction Guide
MS 에서 제공하는 공식적인 인터랙션 가이드라인
-> <http://msdn.microsoft.com/en-us/library/jj663791.aspx>
- Kinect for windows SDK Reference
Kinect 개발에 사용되는 C++/C# Reference
-> <http://msdn.microsoft.com/en-us/library/jj572480.aspx>
- OpenNI Programming Guide Book
OpenNI 로 개발시 참고할 수 있는 문서
-> <http://openni.org/Documentation/ProgrammerGuide.html>
- Vassilis Athitsos 교수 홈페이지
Chamfer Matching 을 통한 Hand Pose Estimation 연구
-> <http://vlm1.uta.edu/~athitsos/>
 - Chamfer Matching 을 활용한 hand Pose Estimation Sample
-> http://vlm1.uta.edu/~athitsos/handpose/hand_pose.zip
- Antonis Argyros 교수 홈페이지
single Kinect 를 활용한 3D Hand Tracking 에 대해서 연구
-> <http://www.ics.forth.gr/~argyros/>
 - 3D hand Tracking Sample
: <http://cvrlcode.ics.forth.gr/handtracking/>
- RT Hand-Tracking with a color Glove
 - 3 Gear System 의 기반 논문
-> <http://people.csail.mit.edu/rywang/handtracking/>
- Mobile Robot Programming Toolkit (MRPT)
 - Kinect 를 사용한 Camera Calibration 및 localization / SLAM 샘플 제공 (CL NUI 기반)
-> http://www.mrpt.org/Kinect_and_MRPT#win.freenect