DUO Dense3D Samples

Overview

This article is a review of the `C++` Dense3D samples that ship with the DUO SDK. This will be updated as new features are added or specifications updated to the DUO API. In the `Developers/Samples` folder within the SDK download you can find the latest version of these samples with a `cmake` project to generate specific IDE projects (Visual Studio/Qt Creator).

Prerequisites

Before reading this guide it is recommended to review our API and SDK Docs to get an understanding of the design, common practices and usage. With DUO devices you will always receive a "frame" which contains all relevant sensor data. The following examples showcase the usage of the DUOFrame structure while using different device functions. Also make sure you have the latest SDK download.

Understanding of the DUO Dense3D API which provides high level processor for generation of a depth map from the image frames.

- DUOLib.lib & DUOLib dynamic library that match your compiler and architecture (x86/x64)
- DUO Dense3D Libraries
- OpenCV v2.4.7+
- CMake 2.8+

Methods/Structures

Here are some of the Device API methods used through-out the examples:

- `EnumerateResolutions` - Lists available resolutions.
    - `DUOResolutionInfo` - Structure used for resolution information.
- `DUOFrameCallback` - Callback on frame update.
- `OpenDUO` - Opens a new connection to the device.
- `StartDUO` - Initializes capture on the device.
- `StopDUO` - Un-Initializes capture on the device.
- `CloseDUO` - Closes the connection to the device.

And Dense3D API specifics methods:

- `Dense3DOpen` - Initialize the depth processor instance.
- `Dense3DClose` - Close the depth processor instance.
- `Dense3DGetDepth` - Returns the depth processor data.
- `SetDense3DCalibration` - Sets the calibration data.

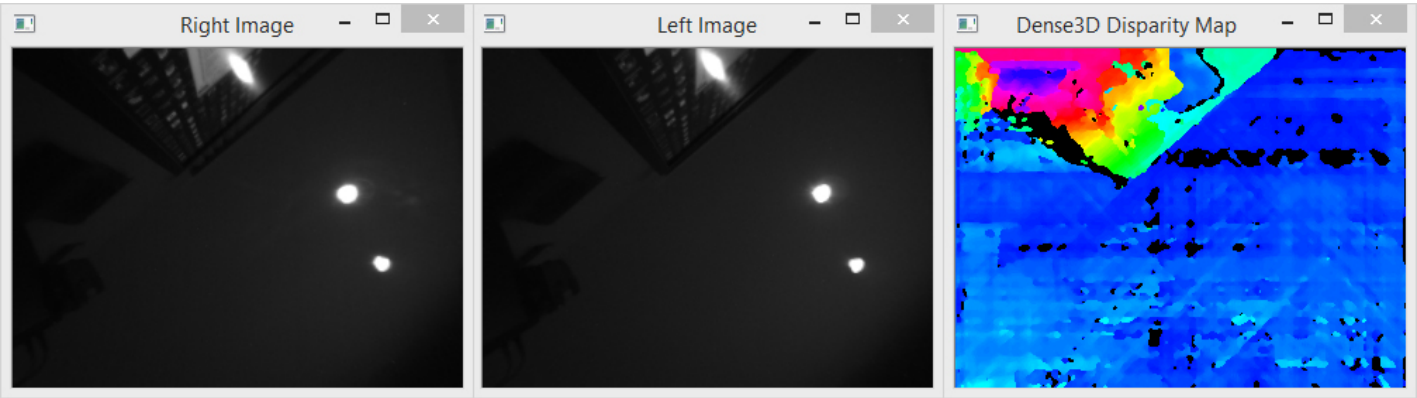
Include/Linkage

The including/linkage of the DUOLib and Dense3D libraries (which may vary dependant on compiler).

```
#include "DUOLib.h"
#include "Dense3D.h"
#pragma comment(lib, "DUOLib.lib")
#pragma comment(lib, "Dense3DLib.lib")
```

Sample 01

Generating a depth map with Dense3D Middleware



In this sample we showcase how to extract depth via our Dense3D API and libraries.

Here are some of the key Dense3D functions/structures:

- `Dense3DOpen` - Initializes the Dense3D processor.
- `Dense3DGetDepth` - Returns a depth map based on processor results.
- `GetDense3DNumDisparities` - Returns the current number of disparities.

Sample.h

```
#ifndef SAMPLE_H
#define SAMPLE_H
```

```
// Include some generic header files
...

// Include DUO API header file
#include

#include
using namespace cv;

// Some global variables
static DUOInstance _duo = NULL;
static PDUOFrame _pFrameData = NULL;

...

// One and only duo callback function
// It sets the current frame data and signals that the new frame data is ready
static void CALLBACK DUOCallback(const PDUOFrame pFrameData, void *pUserData)
{
    _pFrameData = pFrameData;
    SetEvent(_evFrame);
}

// Opens, sets current image format and fps and starts capturing
static bool OpenDUOCamera(int width, int height, float fps)
{
    if(_duo != NULL)
    {
        // Stop capture
        StopDUO(_duo);
        // Close DUO
        CloseDUO(_duo);
        _duo = NULL;
    }

    // Find optimal binning parameters for given (width, height)
    // This maximizes sensor imaging area for given resolution
    int binning = DUO_BIN_NONE;
    if(width <= 752/2)
        binning += DUO_BIN_HORIZONTAL2;
    else if(width <= 752/4)
        binning += DUO_BIN_HORIZONTAL4;
    if(height <= 480/4)
        binning += DUO_BIN_VERTICAL4;
    else if(height <= 480/2)
        binning += DUO_BIN_VERTICAL2;

    // Check if we support given resolution (width, height, binning, fps)
    DUOResolutionInfo ri;
    if(!EnumerateResolutions(&ri, 1, width, height, binning, fps))
        return false;

    if(!OpenDUO(&_duo))
        return false;
```

```
    char tmp[260];
    // Get and print some DUO parameter values
    GetDUODeviceName(_duo,tmp);
    printf("DUO Device Name:      '%s'\n", tmp);
    GetDUOSerialNumber(_duo, tmp);
    printf("DUO Serial Number:    %s\n", tmp);
    GetDUOFirmwareVersion(_duo, tmp);
    printf("DUO Firmware Version: v%s\n", tmp);
    GetDUOFirmwareBuild(_duo, tmp);
    printf("DUO Firmware Build:   %s\n", tmp);

    // Set selected resolution
    SetDUOResolutionInfo(_duo, ri);

    // Start capture
    if(!StartDUO(_duo, DUOCallback, NULL))
        return false;
    return true;
}

// Waits until the new DUO frame is ready and returns it
static PDUOFrame GetDUOFrame()
{
    if(_duo == NULL)
        return NULL;
    if(WaitForSingleObject(_evFrame, 1000) == WAIT_OBJECT_0)
        return _pFrameData;
    else
        return NULL;
}

// Stops capture and closes the camera
static void CloseDUOCamera()
{
    if(_duo == NULL)
        return;
    // Stop capture
    StopDUO(_duo);
    // Close DUO
    CloseDUO(_duo);
    _duo = NULL;
}

static void SetExposure(float value)
{
    if(_duo == NULL)
        return;
    SetDUOExposure(_duo, value);
}

static void SetGain(float value)
{
    if(_duo == NULL)
```

```
        return;
    SetDUOGain(_duo, value);
}

static void SetLed(float value)
{
    if(_duo == NULL)
        return;
    SetDUOLedPWM(_duo, value);
}

static void SetVFlip(bool value)
{
    if(_duo == NULL)
        return;
    SetDUOVFlip(_duo, value);
}

static void SetCameraSwap(bool value)
{
    if(_duo == NULL)
        return;
    SetDUOCameraSwap(_duo, value);
}

static void SetUndistort(bool value)
{
    if(_duo == NULL)
        return;
    SetDUOUndistort(_duo, value);
}

typedef struct
{
    unsigned short w, h;
    double left[12];
    double right[12];
} INTRINSICS;

typedef struct
{
    double rotation[9];
    double translation[3];
} EXTRINSICS;

static bool GetCameraParameters(INTRINSICS *intr, EXTRINSICS *extr)
{
    if(_duo == NULL)
        return false;
    return (GetDUOIntrinsics(_duo, intr) && GetDUOExtrinsics(_duo, extr));
}

#endif // SAMPLE_H
```

Sample.cpp

Considering the header file is mainly helper functions and includes in the actual implementation we go through the following steps:

- **Step 1** - Create DUO and Dense3D Instances
- **Step 2** - Configure DUO and Dense3D Parameters
- **Step 3** - Get DUO Frame (Left and Right Image Data) and pass it to Dense3D
- **Step 4** - Render the depthmap using the color lookup table

```
#include "Sample.h"
// Include Dense3D API header file
#include

#define WIDTH    320
#define HEIGHT    240
#define FPS        30

Vec3b HSV2RGB(float hue, float sat, float val)
{
    float x, y, z;

    if(hue == 1) hue = 0;
    else         hue *= 6;

    int i = static_cast(floorf(hue));
    float f = hue - i;
    float p = val * (1 - sat);
    float q = val * (1 - (sat * f));
    float t = val * (1 - (sat * (1 - f)));

    switch(i)
    {
        case 0: x = val; y = t; z = p; break;
        case 1: x = q; y = val; z = p; break;
        case 2: x = p; y = val; z = t; break;
        case 3: x = p; y = q; z = val; break;
        case 4: x = t; y = p; z = val; break;
        case 5: x = val; y = p; z = q; break;
    }
    return Vec3b((uchar)(x * 255), (uchar)(y * 255), (uchar)(z * 255));
}

int main(int argc, char* argv[])
{
    // build color lookup table for depth display
    Mat colorLut = Mat(cv::Size(256, 1), CV_8UC3);
    for(int i = 0; i < 256; i++)
```

```
        colorLut.at(i) = (i==0) ? Vec3b(0, 0, 0) : HSV2RGB(i/256.0f, 1, 1);

    printf("DUOLib Version:       v%s\n", GetLibVersion());
    printf("Dense3D Version:      v%s\n", Dense3DGetLibVersion());

    // Open DUO camera and start capturing
    if(!OpenDUOCamera(WIDTH, HEIGHT, FPS))
    {
        printf("Could not open DUO camera\n");
        return 0;
    }

    Dense3DInstance dense3d;
    if(!Dense3DOpen(&dense3d))
    {
        printf("Could not open Dense3D library\n");
        // Close DUO camera
        CloseDUOCamera();
        return 0;
    }

    if(!SetDense3DLicense(dense3d, "XXXXX-XXXXX-XXXXX-XXXXX-XXXXX")) // Get your license from duo3d.
com/account
    {
        printf("Invalid Dense3D license\n");
        // Close DUO camera
        CloseDUOCamera();
        // Close Dense3D library
        Dense3DClose(dense3d);
        return 0;
    }
    if(!SetDense3DImageSize(dense3d, WIDTH, HEIGHT))
    {
        printf("Invalid image size\n");
        // Close DUO camera
        CloseDUOCamera();
        // Close Dense3D library
        Dense3DClose(dense3d);
        return 0;
    }

    INTRINSICS intr;
    EXTRINSICS extr;
    if(!GetCameraParameters(&intr, &extr))
    {
        printf("Could not get DUO camera calibration data\n");
        // Close DUO camera
        CloseDUOCamera();
        // Close Dense3D library
        Dense3DClose(dense3d);
        return 0;
    }

    // Set Dense3D parameters
```

```
SetDense3DCalibration(dense3d, &intr, &extr);
SetDense3DNumDisparities(dense3d, 3);
SetDense3DSADWindowSize(dense3d, 2);
SetDense3DP1(dense3d, 800);
SetDense3DP2(dense3d, 1600);
SetDense3DPreFilterCap(dense3d, 0);
SetDense3DUniquenessRatio(dense3d, 0);
SetDense3DSpeckleWindowSize(dense3d, 0);
SetDense3DSpeckleRange(dense3d, 0);

// Set exposure, LED brightness and camera orientation
SetExposure(75);
SetLed(25);
SetVFlip(true);
// Enable retrieval of undistorted (rectified) frames
SetUndistort(true);

// Create Mat for left & right frames
Mat left = Mat(Size(WIDTH, HEIGHT), CV_8UC1, NULL);
Mat right = Mat(Size(WIDTH, HEIGHT), CV_8UC1, NULL);

// Create Mat for disparity and depth map
Mat1f disparity = Mat(Size(WIDTH, HEIGHT), CV_32FC1);
Mat3f depth3d = Mat(Size(WIDTH, HEIGHT), CV_32FC3);

// Run capture loop until  key is pressed
while((cvWaitKey(1) & 0xff) != 27)
{
    // Capture DUO frame
    PDUOFrame pFrameData = GetDUOFrame();
    if(pFrameData == NULL) continue;

    // Set the image data
    left.data = (uchar*)pFrameData->leftData;
    right.data = (uchar*)pFrameData->rightData;

    // Process Dense3D depth map here
    if(Dense3DGetDepth(dense3d, pFrameData->leftData, pFrameData->rightData,
                       (float*)disparity.data, (PDense3DDepth)depth3d.data))
    {
        uint32_t disparities;
        GetDense3DNumDisparities(dense3d, &disparities);
        Mat disp8;
        disparity.convertTo(disp8, CV_8UC1, 255.0/disparities);
        Mat mRGBDepth;
        cvtColor(disp8, mRGBDepth, COLOR_GRAY2BGR);
        LUT(mRGBDepth, colorLut, mRGBDepth);
        imshow("Dense3D Disparity Map", mRGBDepth);
    }
    // Display images
    imshow("Left Image", left);
    imshow("Right Image", right);
}
// Close DUO camera
```

```
    CloseDUOCamera();
    // Close Dense3D library
    Dense3DClose(dense3d);
    return 0;
}
```

Resources

Related

- DUO API
- DUO SDK
- DUO Developers
- DUO Devices
- DUO Downloads