

Creating a Basic Application That Uses Kinect Fusion

Nick Zuzow

April 1, 2015

Executive Summary:

In this application note I will go over the Kinect Fusion Basics project that comes with the Windows SDK. This project is used to show how to use the Windows SDK to create an application that utilizes Kinect Fusion. However some of it can be a bit hard to follow, so I will be explaining some of the key functions that are used. The project that is referenced in this tutorial can be found in the C++ samples section of the Kinect Toolkit. This tutorial will use visual studio 2013 and the code examples will be written in C++.

Key words:

Kinect Fusion, Visual Studio 2013, C++, Windows SDK

Requirements:

- Kinect for Windows v2
- Windows 8 or 8.1
- Windows SDK 1.8
- Kinect Toolkit 1.8
- Visual Studio 2013
- A Graphics Card with 1GB or greater of RAM

Introduction:

Kinect Fusion is an API that is released with newer versions of the Windows SDK. Kinect Fusion is used to build Windows applications that take input from a Kinect for Windows sensor, and output 3D models of the scene. Kinect Fusion works by integrating the depth data from Kinect over time from multiple viewpoints. Kinect Fusion can handle multiple viewpoints by keeping track of the cameras position as it relates to the objects in the scene. This means that while the Kinect is physically being moved, the Kinect Fusion API will compare the elements found within an individual frame with the other frames to estimate the direction that the Kinect is moving.

The process pipeline of Kinect Fusion can be found in figure 1 below. As we can see, the data comes in as Raw Depth data from the Kinect camera. It then is converted from Raw Depth to floating point depth in meters. It then goes through the camera tracking phase where elements within a frame are detected. This is also where the cameras position or pose is calculated. From here, the depth data goes through the integration phase. This is where the data is taken from a known sensor pose and integrated into a single volumetric representation of the space around the camera. The integration phase is processed continuously for every frame. This allows for dynamic change in the scene, as well as a continuous average, which can lead to smoother cleaner output. The last step in the process is for the reconstruction volume to be raycast from a sensor pose and shaded to make the 3D reconstruction more visible.

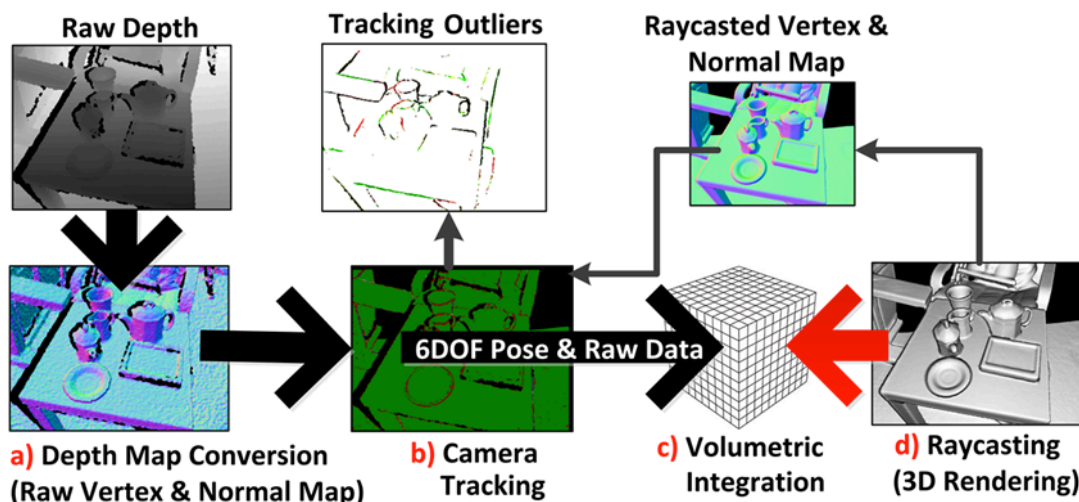


Figure 1 - Process Pipeline

Microsoft releases some sample code with the Windows SDK to give people examples of using Kinect Fusion. This code works instantly after downloading as long as you have the proper dependencies installed. The problem however is that the code examples sometimes can be hard to understand. Through this application note we are going to step through some of the functions found in the Kinect Fusion Basics sample and give more of an understanding about how they work.

Objective:

After reading through this application note, a user should have a better understanding of the downloaded sample code used to create a Kinect Fusion enabled Windows application. This application note is aimed at people who are interested in learning how to write Windows applications with Kinect Fusion functionality. This application note assumes the reader has a background in C++ programming, and visual studio, however no background in programming for the Kinect is necessary.

The following tutorial will focus on the functions that need to be written and called in order to create an application that will take in depth data from a Kinect v2 sensor, and display a 3D model as output. The specific code that is displayed in the tutorial can be found in the sample projects directory that comes with the Windows SDK.

Tutorial:**Step 1:**

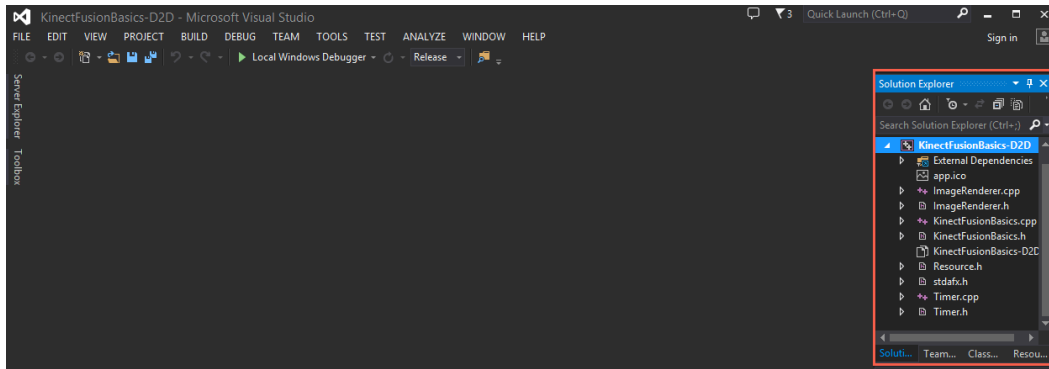
First we need to download the C++ source code for the Kinect Fusion Basics project. Assuming that you have all of the required downloads installed, this can be found by clicking Start > All Programs > Kinect for Windows SDK 1.8 > Developer Toolkit Browser. This will open up the Toolkit Browser. Click on the C++ tab at the top, and scroll down until you find the Kinect Fusion Basics project. Then click the install button, which will download the project to your computer.

Step 2:

Now go to the location of the downloaded project, and open up the visual studio solution file.

Step 3:

Once the solution opens in visual studio 2013 we can see the included files on the right hand side in the Solution Explorer pane.



We can see that there are multiple files, however we will be primarily talking about the KinectFusionBasics.cpp, and ImageRender.cpp files.

Step 4:

Now open the KinectFusionBasics.cpp file. When we open it we see that it contains a class named CKinectFusionBasics. Within this class we have multiple functions, however there are a few main functions I would like to focus on.

wWinMain:

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE, _In_ LPWSTR, _In_ int nCmdShow)
{
    CKinectFusionBasics application;
    application.Run(hInstance, nCmdShow);
}
```

The wWinMain function is the first function in this class. It is a very small function however it is the main function of the application, and it is needed for the application to work.

CKinectFusionBasics::Run:

This is the function responsible for creating the application window and starting the processing. The following code can also be found within this method:

```

while (WM_QUIT != msg.message)
{
    // Explicitly check the Kinect frame event since MsgWaitForMultipleObjects
    // can return for other reasons even though it is signaled.
    Update();

    while (PeekMessageW(&msg, nullptr, 0, 0, PM_REMOVE))
    {
        // If a dialog message will be taken care of by the dialog proc
        if ((hWndApp != nullptr) && IsDialogMessageW(hWndApp, &msg))
        {
            continue;
        }

        TranslateMessage(&msg);
        DispatchMessageW(&msg);
    }
}

```

This is the while loop that the program lives in while it is running. This is essentially a high level message handler. The program will only break out of this loop when it receives the message that the user has closed the program. When it receives a message (or an event) from the console and it isn't the quit message, it calls the appropriate actions based on the type of the method.

CKinectFusionBasics::Update():

The Update method is very important. This method is responsible for the main processing of the application. This includes checking to see if the Kinect sensor has moved, acquiring the next frame, and buffering the depth data before calling the processing function.

```

if (m_coordinateMappingChangedEvent != NULL &&
    WAIT_OBJECT_0 == WaitForSingleObject((HANDLE)m_coordinateMappingChangedEvent, 0))
{
    OnCoordinateMappingChanged();
    ResetEvent((HANDLE)m_coordinateMappingChangedEvent);
}

```

In the code above we see the if statement that is checking to see if the physical Kinect camera has moved. This is extremely important, because special functionality is required to be able to appropriately map scenes that are captured with a moving camera. If the software didn't detect when the camera is moving then the resulting models would not be accurate because it would consistently update the same position in the model with new items.

```

hr = pDepthFrame->get_RelativeTime(&currentTimestamp);
if (SUCCEEDED(hr) && currentTimestamp - m_lastFrameTimeStamp > cResetOnTimeStampSkippedMilliseconds * 10000
    && 0 != m_lastFrameTimeStamp)
{
    m_bResetReconstruction = true;
}
m_lastFrameTimeStamp = currentTimestamp;

```

Kinect Fusion also has a feature where it will reset the model window if there has been no activity for a specific amount of time. This feature however can cause some systems that don't have a powerful enough graphics card to consistently start redrawing the scene. This is something to be aware of.

CKinectFusionBasics::ProcessDepth():

The ProcessDepth method is the last main method we are going to discuss. The name of this method gives a good indication into the functionality that it produces. ProcessDepth is called from the Update method and it is used to handle many important pieces of Kinect Fusion. The flow of this method is as follows. It receives a frame, performs error checking on the frame data, updates the new pose of the camera if necessary, calls the function to create the point cloud, calls the function to shade the point cloud, and calls the draw function.

This method makes calls to DepthToDepthFloatFrame, ProcessFrame, GetCurrentWorldToCameraTransform, CalculatePointCloud, and NuiFusionShadePointCloud. These are all functions that are part of the NuiKinectFusionApi, which is the main API behind Kinect Fusion. These functions are used to do the following:

DepthToDepthFloatFrame: Used to convert the specified array of Kinect depth pixels to a [NUI_FUSION_IMAGE_FRAME](#) structure.

ProcessFrame: Used to process a depth frame through the KinectFusion pipeline.

GetCurrentWorldToCameraTransform: Used to get the current internal world-to-camera transform (camera view pose).

CalculatePointCloud: Used to calculate a point cloud by raycasting into the reconstruction volume.

NuiFusionShadePointCloud: Produces two shaded images from the point cloud frame, based on point position and surface normal.

The draw function that is called from the ProcessDepth method is a part of the ImageRenderer class. This class is defined in the ImageRenderer.cpp file, and is used to draw the image to the screen using other functions found within the NuiKinectFusionApi.

More information about the NuiKinectFusionApi can be found at <https://msdn.microsoft.com/en-us/library/dn188696.aspx>

Conclusion:

Throughout this application note we have discussed some of the more complex methods found within the Kinect Fusion Basics sample code that comes with the Windows SDK. Hopefully this walk through has made this sample easier

to understand and will allow you to have a better foundation to integrate Kinect Fusion into your new Windows applications in the future.

References:

"Kinect Fusion Basics D2D C++ Sample." Kinect Fusion Basics D2D C++ Sample. N.p., n.d. Web. 02 Apr. 2015.

"Kinect Fusion." Kinect Fusion. N.p., n.d. Web. 01 Apr. 2015.

"NuiFusionShadePointCloud." NuiFusionShadePointCloud. N.p., n.d. Web. 03 Apr. 2015.