AMD Media SDK

User Guide

January 2015

rev 1.1

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA 94088-3453
www.amd.com

For AMD Media SDK:

URL:
http://developer.amd.com/tools-and-sdks/heterogeneous-computing/media-sdk/

Forum:
http://devgurus.amd.com/community/media-sdk

Contents

FiguresAMD Media SDK User Guide

Tables

Chapter 1
Overview

1.1 Introduction

The AMD Media SDK allows developers to access CPU, GPU compute shaders, and hardware accelerators for media processing. Developers can access this functionality by easy to use APIs and software components for popular multimedia frameworks.

Figure 1.1 shows a typical developer workflow using the Media SDK.

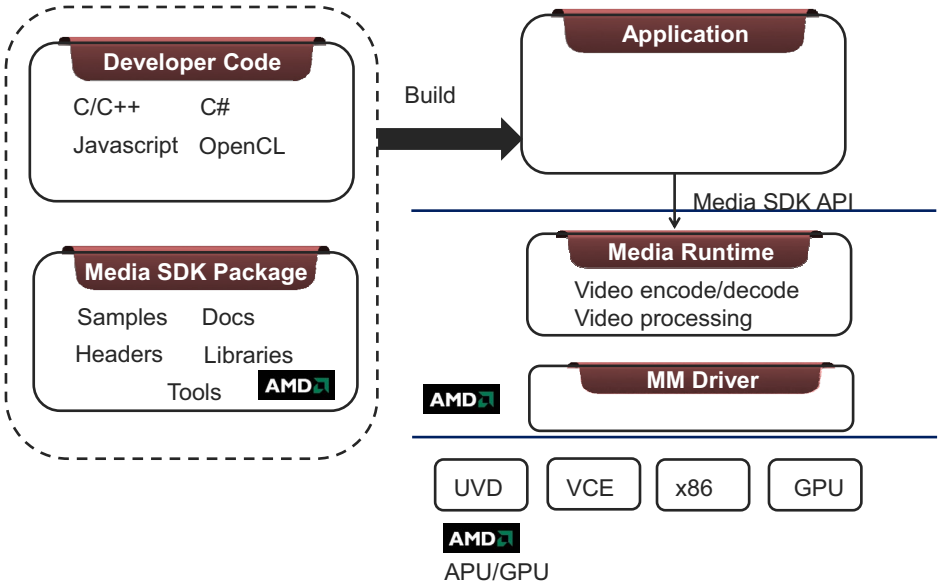


Figure 1.1 Developer Workflow with Media SDK

The AMD APUs and discrete GPUs provide the following components for use in multimedia processing:

- Unified Video Decoder (UVD) – Fixed-function hardware accelerator that supports MPEG2, MPEG4 Part 2, DivX, H.264, WMV9, and VC-1 decoding.
- Video Coding Engine (VCE) – Fixed-function hardware accelerator that supports H.264 AVC, and SVC encoding.
- Graphics Processing Unit (GPU) – Typically used for data parallel processing.
- CPU cores – CPU cores are typically used for serial processing.

1.2 Media Foundation Transform (MFT) Solution

For developers using Microsoft's Media Foundation framework, the Media SDK provides Media Foundation Transforms (MFTs); this allows developers to use UVD and VCE for video decoding and encoding, respectively. The Media SDK also provides a Video Quality (VQ) MFT for video pre/post processing.

Figure 1.2 shows a typical multimedia processing pipeline in a multimedia application. Note that developers can use MFTs from a variety of sources, including their own, to build powerful multimedia applications easily and efficiently.

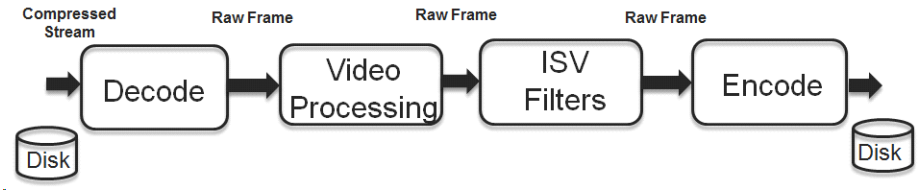


Figure 1.2 Pipeline in a Video Edit Application

1.3 AMF Library

AMD provides the AMD Multimedia Framework (AMF) to allow developers to harness the capabilities of the UVD and VCE blocks. The AMF includes a set of APIs and libraries that developers can use to realize multimedia processing use-cases. The AMF library in the Media SDK provides a C++ API for using the AMF features on AMD APUs and discrete GPUs.

1.4 AMF-DEM Library

On AMD APUs and discrete GPUs, there is a direct physical connection between the display controller and the Video Coding Engine (VCE); this feature is known as Display Encode Mode (DEM). Developers can use DEM for screen capture and video encode in one process. This is ideal for low-latency applications like wireless display and remote desktop. The AMF-DEM library in the Media SDK provides a C++ API for using the DEM feature on AMD APUs and discrete GPUs.

1.5 Samples

The Media SDK samples provide optimized reference code for the following targeted applications use cases.

- Video editing, transcode, and playback.
- Remote desktop and wireless display.

The sample code addresses application-specific optimizations such as inter-op with OpenCL and C++ AMP-based components and buffer sharing on GPUs.

The samples are targeted for both Windows and Windows Store applications.

1.6 Windows 8/8.1 Store Application Development

The Media SDK lets Windows 8/8.1 Store application developers use:

- UVD and VCE for video decoding and encoding through standard Windows 8/8.1 Store media processing APIs.
- Video Quality (VQ) MFT for video pre/post processing. Note that the VQ MFT must be bundled with the Windows 8/8.1 Store application.

1.7 Additional Information

For more information on the Microsoft Media Foundation framework:
http://msdn.microsoft.com/en-us/library/windows/desktop/ms697062(v=vs.85).aspx

For a copy of the OpenCL Programming Guide:
http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf

VQ MFT API Reference

AMF Reference Manual

AMF-DEM API Reference

Chapter 2
Getting Started

This chapter describes how to build and run a sample.

2.1 Prerequisites

1. Ensure that following are installed.
   - AMD Media SDK.
   - Microsoft Visual Studio 2010 or later.
   - Windows Software Development Kit (SDK) – This Contains headers, libraries, and a selection of tools used to create applications on Windows OS. It also includes the Windows Application Certification Kit (ACK) to test applications for the Windows 8/8.1 Certification Program, as well as the Windows 7 Logo program. Microsoft Windows SDK for Windows 8/8.1 is compatible with the current Media SDK release.
   - AMD Catalyst Driver – See the *MediaSDK_ReleaseNotes* for the driver version.
2. Navigate to the samples `build\Windows` folder.
   This is in:

`$<installDirectory>\samples\<sample_category>\<sample_name>\build\windows`

2.2 Sample Build

1. Open one of the following solution files. (The example used in the following steps is the transcodeVq sample.)

```
$<installDirectory>\samples\mediaFoundation\transcodeVq\build\windows\transcodeVqVs12.sln
$<installDirectory>\samples\mediaFoundation\transcodeVq\build\windows\transcodeVqVs10.sln
```

2. Open the `transcodeVqVs10.sln` solution file with Microsoft Visual Studio 2010 Professional Edition, or open the `transcodeVqVs12.sln` solution file with Microsoft Visual Studio 2012 Professional Edition.
   - To build all the solutions, select `Build > Build Solution`.
   - Select the project file in the Solutions Explorer.
   - The executable `transcodeVq.exe` is created in the following folders for 32-bit builds and 64-bit builds:

```
$<installDirectory>\samples\mediaFoundation\transcodeVq\bin\x86\
$<installDirectory>\samples\mediaFoundation\transcodeVq\bin\x86_64\
```

– Depending on the build (i.e. 32-bit or 64-bit), the post build step copies the appropriate Video Quality (VQ) MFT `.dll` file from the `$<installDirectory>\dll\videoQualityMFT\windowsClassicDesktop\` folder into the relevant `\bin\` directory.

2.3 Sample Execution

On the command prompt, change to the directory that contains the executable; then, execute the sample. See the sample document for the usage of each sample. For samples with a GUI, the GUI is launched when running the executable.

For example, the transcodeVq sample can be executed by using the following command:

```
transcodeVq.exe -i <input.avi> -o <output.asf> -c <config file> -l <0, 1, or 2 >
```

where: `-l` enables the logging; `0` means no logging; `1` generates the log at the API level; `2` generates logs at the transcoding session level.

The log file `<sample_name>/build/windows/<sample_name>Errorlog.txt` contains log dumps in case of an error; it is empty if the sample executed successfully.

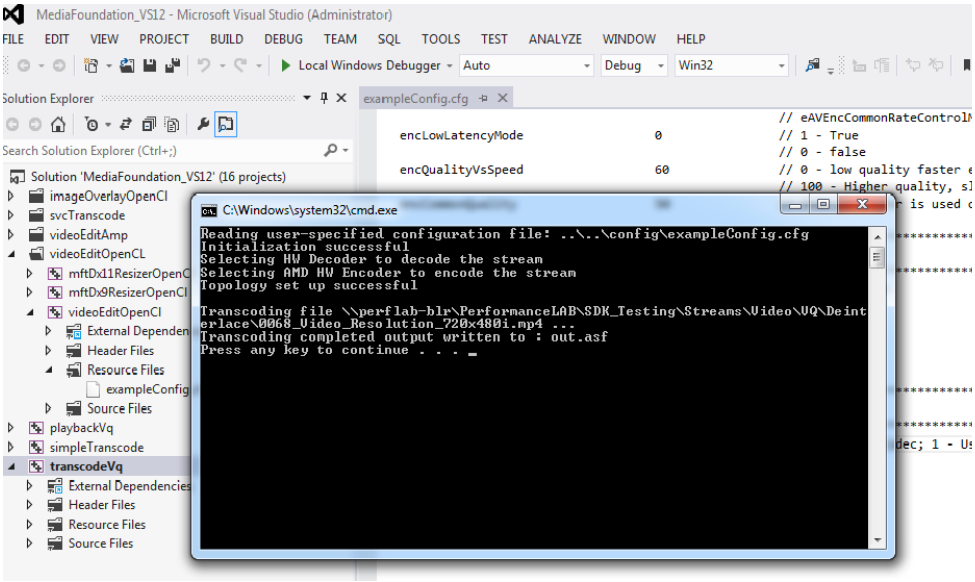
The configuration file associated with this sample is located in:

```
$<installDirectory>\samples\mediaFoundation\transcodeVq\config\
```

For details on each of the configuration parameters, see the sample document.

`Output.asf` contains the H.264 encoded stream with video quality improvements.

The following is a screenshot of the console.



See the individual sample documentation for more information on how to build and run the respective sample. Individual sample documentation can be found in `<sample_name>/docs` folder.

Chapter 3
Using Media Foundation

Media Foundation enables the development of applications and components for using digital media on Windows platforms. The basic Media Foundation components are the Media Source, Media Sink, and the Media Foundation Transform (MFT).

3.1 Overview

A media source is an object that acts as the source of multimedia data, either compressed or uncompressed. It can encapsulate various data sources, like a file, a network server, or a camcorder, with source-specific functionality abstracted by a common interface.

A source object can use a source resolver object to create a media source from a uniform resource indicator (URI), file, or byte stream. Support for non-standard protocols can be added by creating a source resolver for them. A source object also can use a sequencer object to create a sequence of sources (a playlist), or to merge multiple sources into single logical source.

For more information about Media Source, see:

http://msdn.microsoft.com/en-us/library/windows/desktop/ms697527(v=vs.85).aspx

Figure 3.1 shows some of the components of an AVI Media Source object. In this case, the Media Source parses an AVI file and demuxes the video and audio streams for further processing.

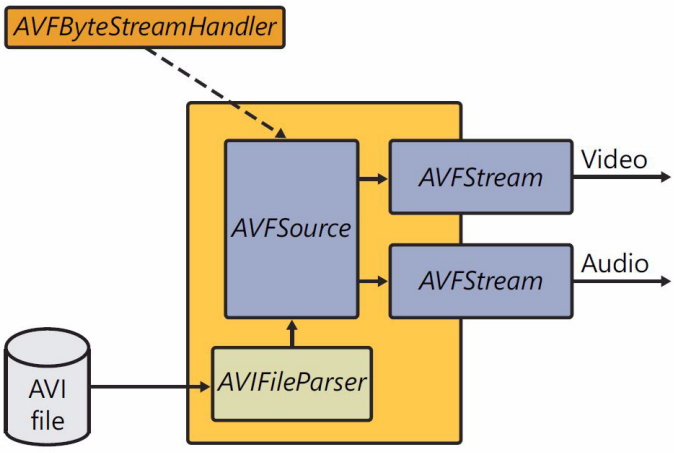


Figure 3.1 MF AVI Source Object

A media sink is the recipient of processed multimedia data. A media sink can either be a renderer sink, which renders the content on an output device, or an archive sink, which saves the content onto a persistent storage system such as a file.

A renderer sink takes uncompressed data as input, whereas an archive sink can take either compressed or uncompressed data.

For more information on Media Sink, see:
http://msdn.microsoft.com/en-us/library/windows/desktop/ms701626(v=vs.85).aspx

Figure 3.2 shows the components of an AVI Media Sink object. Here, the Media Sink muxes the audio and video channels and generates and AVI output file.

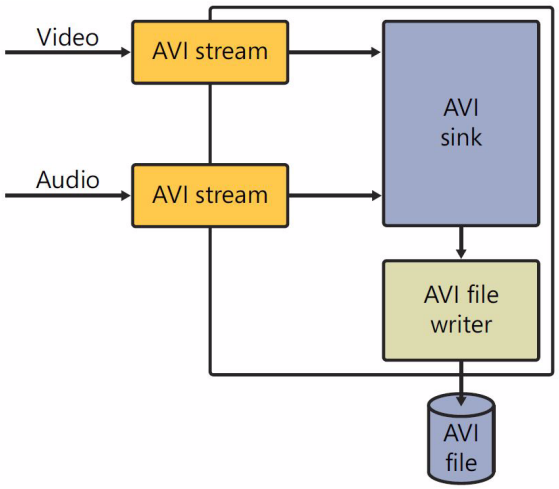


Figure 3.2 Media Foundation AVI Sink Object

The data from media sources to sinks are acted upon by MFTs; MFTs are certain functions which transform the data into another form. Some examples of MFTs are multiplexers and demultiplexers, codecs or DSP effects like reverb. Figure 3.3 is a diagram of this MF topology.

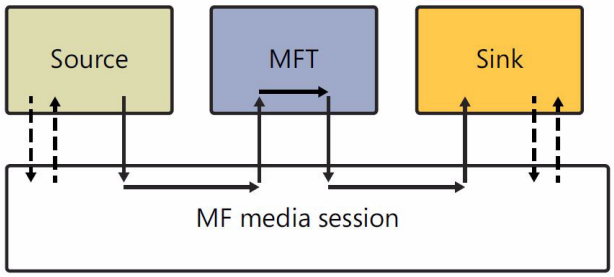


Figure 3.3 Media Foundation Topology

See this MSDN link for further information on Media Foundation Transforms:
http://msdn.microsoft.com/en-us/library/windows/desktop/ms703138(v=vs.85).aspx

3.2 Direct3D

A video MFT is considered Direct3D-aware if it can process samples that contain Direct3D surfaces. Direct3D is supported in a video MFT to enable hardware-accelerated decoding, using DirectX Video Acceleration (DXVA). For example, to decode using Direct3D 11, the decoder must have a pointer to a Direct3D 11 device. The Direct3D 11 device is created externally to the decoder. In a Media Session scenario, the video renderer creates the Direct3D 11 device. The DXGI Device Manager is then used to share the Direct3D 11 between components. The DXGI Device Manager exposes the IMFDXGIDeviceManager interface. The pipeline sets the IMFDXGIDeviceManager pointer on the decoder by sending the `MFT_MESSAGE_SET_D3D_MANAGER` message.

Direct3D is a Microsoft DirectX API for abstracting the communication between a graphics application and the graphics hardware drivers. Direct3D uses hardware acceleration, if available on the graphics card; this permits hardware acceleration of the entire 3D rendering pipeline (or only a partial acceleration). It is presented as an abstract layer and provides a low-level interface to 3D functions such as transformations, clipping, lighting, materials, textures, depth buffering, etc.

3.3 Media Pipeline

The media pipeline components are presented as discrete components. An application must choose which source types, transforms, and sinks are needed for a particular video processing task. It also must set up the connections between the components (a topology) to complete the data-flow pipeline (Figure 3.4).

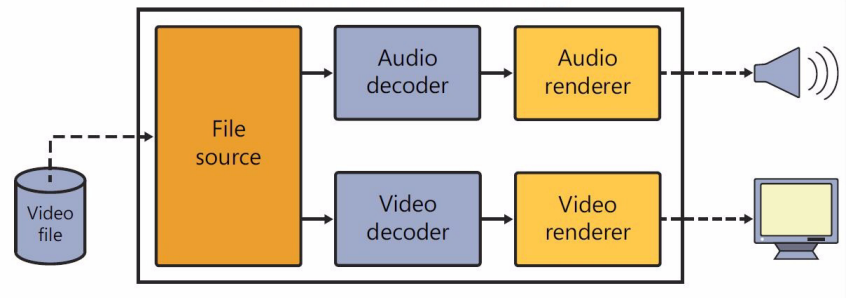


Figure 3.4 Media Foundation Topology for Playback

For example, in Figure 3.4, to play back a compressed audio/video file, the pipeline consists of a file source object, a demultiplexer for the specific file container format to split the audio and video streams, codecs to decompress the audio and video streams, and DSP processors for audio and video effects, and, finally, the EVR renderer.

Similarly, for a video capture application, the camcorder is the source for both video and audio. The codec MFTs process these to compress the data and feed

it to a multiplexer that coalesces the streams into a container. Finally, a file sink or a network sink write the results to a file, or it streams them over a network.

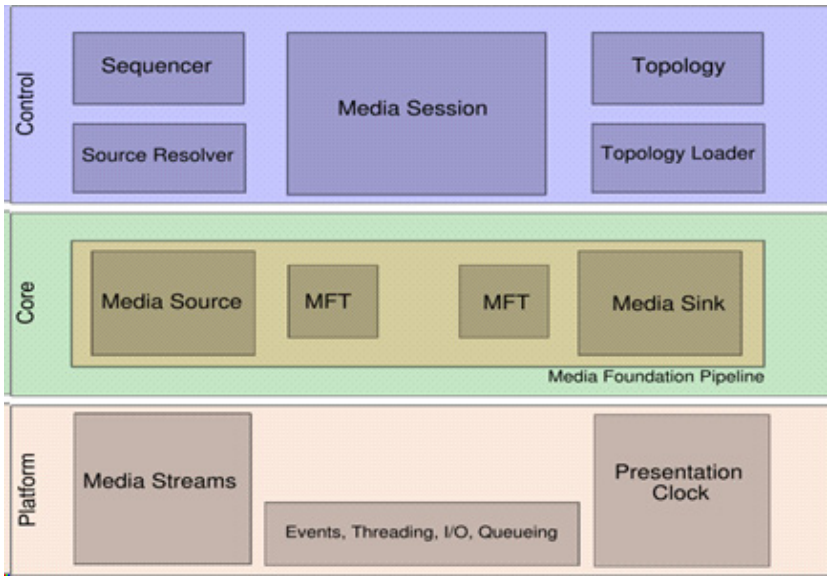


Figure 3.5 Media Foundation Architecture

The application also must co-ordinate the flow of data between the pipeline components. Applications running in the control layer choose the source type, transform, and sink needed for the video processing task; they also set up the connections between the components to complete the data flow pipeline. The control layer (see Figure 3.4) must “pull” (request) samples from one pipeline component and pass them onto the next component for data-flow within the pipeline. This is in contrast to older DirectShow “push” model, where a pipeline component pushes data to the next component.

The control layer components must propagate the data through the pipeline at such a rate that the rendering synchronizes with the presentation clock. The rendering rate is embedded in the multimedia stream as metadata. The source objects extract the metadata and pass it on to the subsequent component in the pipeline. Each MF object passes the metadata on until it reaches the renderer. There are two types of metadata: coded metadata (information about bit rate and presentation timings), and descriptive metadata (such as title and author names). Coded metadata is passed to the object that controls the pipeline session; descriptive metadata is accessible by the application.

For information about how to write the Media Foundation Pipeline, see: http://msdn.microsoft.com/en-us/library/windows/desktop/ms703912(v=vs.85).aspx.

3.4 Media Session Objects

Media Foundation provides a Media Session object that can be used to set up the topologies and facilitate a data flow, relieving the application from this task. It exists in the control layer and exposes a topology loader object. The application

specifies the required pipeline topology to the loader, which then creates the necessary connections between the components. The media session object synchronizes with the presentation clock. It creates the presentation clock object, and passes a reference to it to the sink. Then, it uses the timer events from the clock to propagate data along the pipeline. It also changes the state of the clock to handle pause, stop, or resume requests from the application.

The following code snippet explains how to build a TranscodeVQ topology.

```
// Initialize the COM
hr = CoInitializeEx(nullptr, COINIT_MULTITHREADED);
RETURNIFFAILED(hr);

// Start up Media Foundation platform.
hr = MFStartup(MF_VERSION);
RETURNIFFAILED(hr);

// Create Partial Topology
hr = MFCreateTopology(&mPartialTopology);
RETURNIFFAILED(hr);

////////////////////////////////////////////////////////////////////
// Create source filter
hr = MFCreateSourceResolver(&sourceResolver);
RETURNIFFAILED(hr);
hr = sourceResolver->CreateObjectFromURL(sourceFileName,
MF_RESOLUTION_MEDIASOURCE |
MF_RESOLUTION_CONTENT_DOES_NOT_HAVE_TO_MATCH_EXTENSION_OR_MIME_TYPE,
nullptr, &objectType, &sourceObject);
RETURNIFFAILED(hr);
hr = sourceObject->QueryInterface(IID_PPV_ARGS(mediaSource));
RETURNIFFAILED(hr);

// Create the source node
hr = MFCreateTopologyNode(MF_TOPOLOGY_SOURCESTREAM_NODE, &mSourceNode);
RETURNIFFAILED(hr);
hr = mSourceNode->SetObject(mediaSource);

////////////////////////////////////////////////////////////////////
// Create sink filter
hr = MFCreateASFProfile(&sinkAsfProfile);
RETURNIFFAILED(hr);
hr = MFCreateASFContentInfo(&asfContentInfo);
RETURNIFFAILED(hr);
hr = MFCreateASFMediaSinkActivate(sinkFileName, asfContentInfo,
mediaSinkActivate);
RETURNIFFAILED(hr);

hr = MFCreateTopologyNode(MF_TOPOLOGY_SOURCESTREAM_NODE, &mSinkNode);
RETURNIFFAILED(hr);
hr = mSinkNode->SetObject(mediaSinkActivate);

////////////////////////////////////////////////////////////////////
// Create the Video Decoder, Encoder & Video Processing MFTs
hr = MFTEnumEx( MFT_CATEGORY_VIDEO_DECODER,
flags,
&inputRegisterTypeInfo,
&outputRegisterTypeInfo,
&ppActivate,
&registeredDecodersNumber);
RETURNIFFAILED(hr);
hr = ppActivate[0]->ActivateObject(IID_PPV_ARGS(decoderTransform));
```

```
RETURNIFFAILED(hr);
hr = MFCreateTopologyNode(MF_TOPOLOGY_TRANSFORM_NODE, &mDecoderNode);
hr = mDecoderNode->SetObject(decoderTransform);

//////////////////////////////////////////////////////////////////////
hr = mMftBuilderObjPtr->createVqTransform(deviceManagerPtr,&vqTransform);
RETURNIFFAILED(hr);
hr = MFCreateTopologyNode(MF_TOPOLOGY_TRANSFORM_NODE, &mVqNode);
hr = mVqNode->SetObject(vqTransform);

//////////////////////////////////////////////////////////////////////
hr = MFTEnumEx( MFT_CATEGORY_VIDEO_ENCODER,
flags,
&inputRegisterTypeInfo,
&outputRegisterTypeInfo,
&ppActivate,
&registeredDecodersNumber);
RETURNIFFAILED(hr);
hr = ppActivate[i]->ActivateObject(IID_PPV_ARGS(&encoderTransform));
RETURNIFFAILED(hr);
hr = MFCreateTopologyNode(MF_TOPOLOGY_TRANSFORM_NODE, &encoderTransform);
hr = mEncoderNode->SetObject(encoderTransform);

//////////////////////////////////////////////////////////////////////
// Build & Load the Topology
// Add Source & Sink Node to the Topology
hr = mPartialTopology->AddNode(mSourceNode);
RETURNIFFAILED(hr);
hr = mPartialTopology->AddNode(mSinkNode);
RETURNIFFAILED(hr);

// Add decoder, VQ & encoder node to the topology
hr = mPartialTopology->AddNode(mDecoderNode);
RETURNIFFAILED(hr);
hr = mPartialTopology->AddNode(mVqNode);
RETURNIFFAILED(hr);
hr = mPartialTopology->AddNode(mEncoderNode);
RETURNIFFAILED(hr);

//////////////////////////////////////////////////////////////////////
// Connect sourceNode->DecoderNode->Vqnode->EncoderNode->sinkNode
hr = mSourceNode->ConnectOutput(0, mDecoderNode, 0);
RETURNIFFAILED(hr);
hr = mDecoderNode->ConnectOutput(0, mVqNode, 0);
RETURNIFFAILED(hr);
hr = mVqNode->ConnectOutput(0, mEncoderNode, 0);
RETURNIFFAILED(hr);
hr = mEncoderNode->ConnectOutput(0, mSinkNode, 0);
RETURNIFFAILED(hr);

// Bind the sink node and activate
CComPtr<IMFCollection> nodeCollection;
hr = topology->GetOutputNodeCollection(&nodeCollection);
RETURNIFFAILED(hr);

DWORD nodeCount = 0;
hr = nodeCollection->GetElementCount(&nodeCount);
RETURNIFFAILED(hr);

for (DWORD i = 0; i < nodeCount; i++)
{
CComPtr<IUnknown> nodeUnknown;
hr = nodeCollection->GetElement(i, &nodeUnknown);
RETURNIFFAILED(hr);
```

```
CComPtr<IMFTopologyNode> node;
hr = nodeUnknown->QueryInterface(&node);
RETURNIFFAILED(hr);

hr = bindNode(node);
RETURNIFFAILED(hr);
}
RETURNIFFAILED(hr);

//////////////////////////////////////////////////////////////////
// Create topoloder, load the topology & resolve the same
CComPtr<IMFTopoLoader> topoLoader;
hr = MFCreateTopoLoader(&topoLoader);
RETURNIFFAILED(hr);

hr = topoLoader->Load(mPartialTopology, &mTopology, nullptr);
RETURNIFFAILED(hr);
```

See the following links to the MFT Programming Guide and Architecture document.

http://msdn.microsoft.com/en-us/library/windows/desktop/ms697062(v=vs.85).aspx

http://msdn.microsoft.com/en-us/library/windows/desktop/ms696219(v=vs.85).aspx

3.5 Using MFT

The MFT solution from AMD lets developers use:

- VCE for encoding.
- UVD for decoding.
- GPU compute units (CU) for video processing.

Table 3.1 shows the complete Media Foundation with the different compute elements on the AMD platform.

Table 3.1 Media Foundation Use Based on Platform

Component	Windows 7		Windows 8/8.1	
	Microsoft	AMD	Microsoft	AMD
MPEG4 Part 2 Decoder	CPU	UVD	CPU	UVD
MPEG2 Decoder	N/A	N/A	CPU, UVD[1]	N/A
WMV9/VC-1 Decoder	CPU, UVD	N/A	CPU, UVD	N/A
H.264 (AVC) Decoder	CPU, UVD	N/A	CPU, UVD	N/A
MJPEG Decoder	CPU	CU	CPU	CU
H.264 (AVC) Encoder	CPU	VCE	CPU	VCE
H.264 (SVC) Encoder	N/A	VCE	N/A	VCE
Video Quality	N/A	CU	N/A	CU

1. Not available by default in Windows 8/8.1.

Here, some MFTs are included with the Windows OS, and the rest are supplied by AMD. All MFTs, except the Video Quality (VQ) MFT, are used with the AMD Catalyst driver. The VQ MFT is included in the Media SDK package and must be bundled with the developer application. Note that VQ MFT is only supported on Southern Islands and Sea Islands GPU families.

AMD has developed several Media SDK samples, ranging from Video Playback, Transcode, Video Edit with OpenCl, Video Edit with C++ AMP, Image Overlay to Video Processor MFTs. These samples deploy several MFT components which are either developed by AMD or by Microsoft and are connected based on the sample topology.

See the following link to understand the supported media formats in Media Foundation:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd757927(v=vs.85).aspx

Table 3.2 describes the various MFT components used across all the samples.

Table 3.2 Sample Use of MFT Components

Component	Codec	Source	CLSID	MFT Component Used	Source
Decoder	H.264	Microsoft	CLSID_CMSH264DecoderMFT	CPU/UVD	Production Quality
	MPEG4	AMD	CLSID_AMD_D3D11HW_DecoderMFT	UVD	Production Quality
	VC1/WMV	Microsoft	CLSID_CWMVDecMediaObject	CPU/UVD	Production Quality
Encoder	H.264	AMD	CLSID_AMD_H264_HW_EncoderMFT	VCE	Production Quality
Video Processor	Contrast, Brightness, False Contour	AMD	See Chapter 4 for more details	GPU	Production Quality
Resizer	Scaling	AMD	CLSID_OpenCLMFTDx11 and CLSID_OpenCLMFTDx9	GPU	Sample MFT
Overlay	Image Overlaying	AMD	CLSID_Overlay_OpenCLMFTDx11 and CLSID_Overlay_OpenCLMFTDx9	GPU	Sample MFT
Color Converter	RBG ←→ YUV	AMD	CLSID_ColorConvert_OpenCLMFTDx11 and CLSID_ColorConvert_OpenCLMFTDx9	GPU	Sample MFT
SVC Splitter		AMD	CLSID_SvcMFT		Sample MFT

3.6 Decoders and Encoders

3.6.1 H.264 Decoder

The Media Foundation H.264 video decoder is a Media Foundation Transform that supports decoding of Baseline, Main, and High profiles. This decoder supports an attribute to enable, or disable, hardware acceleration. If DirectX Video acceleration is enabled, UVD is used to decode the stream; otherwise, decoding is done with software.

An instance of the decoder can be created in two ways:

- Call the `MFTEnumEx` function.
- Call `CoCreateInstance`. The CLSID for the decoder is `CLSID_CMSH264DecoderMFT`.

The following code shows how to create a H.264 decoder.

```
HRESULT findVideoDecoder(REFCLSID inputVideoSubtype,
        BOOL *useHwCodec,
        UINT32 inWidth,
        UINT32 inHeight,
        IMFTransform **ppDecoder)
{
    HRESULT hr;
    UINT32 i;
    UINT32 flags = MFT_ENUM_FLAG_SORTANDFILTER;

    // Input Type
    MFT_REGISTER_TYPE_INFO inputRegisterTypeInfo = { MFMediaType_Video,
        inputVideoSubtype };

    flags |= MFT_ENUM_FLAG_SYNCMFT;

    // If UVD is to be used, enable flag so that MS MFT can use UVD
        internally
    if (*useHwCodec)
    {
        flags |= MFT_ENUM_FLAG_HARDWARE;
    }

    IMFActivate **ppActivate = NULL;
    UINT32 registeredDecodersNumber = 0;

    const CLSID supportedOutputSubtypes[] = { MFVideoFormat_NV12,
        MFVideoFormat_YUY2 };
    bool foundDecoder = false;
    for (i = 0; !foundDecoder && i < ARRAYSIZE(supportedOutputSubtypes);
i++)
    {
        MFT_REGISTER_TYPE_INFO outputRegisterTypeInfo = {
MFMediaType_Video, supportedOutputSubtypes[i] };

        // Invoke function to get list of registered MFTs
        hr = MFTEnumEx( MFT_CATEGORY_VIDEO_DECODER,
                        flags,
                        &inputRegisterTypeInfo,
                        &outputRegisterTypeInfo,
                        &ppActivate,// Pointer to list of supported MFTs
                        &registeredDecodersNumber); // No. of reg MFTs
        RETURNIFFAILED(hr);

        if (SUCCEEDED(hr) && (registeredDecodersNumber == 0))
```

```
        {
            hr = MF_E_TOPO_CODEC_NOT_FOUND;
        }

        if (SUCCEEDED(hr))
        {
            hr = ppActivate[0]->ActivateObject(IID_PPV_ARGS(ppDecoder));
            foundDecoder = true;
        }

        for (i = 0; i < registeredDecodersNumber; i++)
        {
            ppActivate[i]->Release();
        }
        CoTaskMemFree(ppActivate);
    }

    if (!foundDecoder)
    {
        printf("There is no registered decoder for given video subtype\n");
        return E_FAIL;
    }
    return S_OK;
}

// Creating MFT using CoCreateInstance
CComPtr < IUnknown > transformUnknown;
hr = CoCreateInstance(CLSID_CMSH264DecoderMFT, nullptr, CLSCTX_INPROC,
IID_IMFTransform, (void**) &transformUnknown);
RETURNIFFAILED(hr);
```

3.6.2 MPEG4 Part 2 Video Decoder

The MPEG4 Part 2 Video decoder decodes video streams that were encoded according to the MPEG4 Part 2 standard. An instance of the MPEG4 Part 2 Video decoder can be created by calling the `CoCreateInstance` or `MFTEnumEx` function. To create an instance of the Microsoft decoder that behaves as a Media Foundation Transform (MFT), use the class identifier `CLSID_CMpeg4sDecMFT`. The AMD hardware MFT Decoder can be created using the class identifier `CLSID_AMD_D3D11HW_DecoderMFT`, which currently supports only Mpeg4 Part 2 video with frame sizes 1280x720 or higher.

3.6.3 Windows Media Video 9 Decoder

The Windows Media Video 9 decoder decodes WMV and VC1 video streams. The class identifier (CLSID) for the Windows Media Video decoder is represented by the constant `CLSID_CWMVDecMediaObject`. An instance of the video decoder can be created by calling the `CoCreateInstance` or `MFTEnumEx` function. Similar to the H.264 decoder, the Windows Media Video decoder supports a property that specifies whether the decoder uses DirectX video acceleration hardware. If it does, UVD is used to decode the stream; otherwise, software-based decoding is used. The following code snipped provides an example of this.

```
PROPVARIANT var;
hr = propertyStore->GetValue(MFPKEY_DXVA_ENABLED, &var);
RETURNIFFAILED(hr);

bool isHWAccelerated;
isHWAccelerated = var.boolVal == VARIANT_TRUE && var.vt == VT_BOOL;
```

If `HWAccelerated` is True, UVD is used to decode the stream; otherwise, software-based decoding is used.

3.6.4 Microsoft H.264 Video Encoder

You can select the Microsoft H.264 video encoder, in which case a software encoder is used to encode the stream; or you can select the AMD H.264 video encoder, in which case a VCE-based hardware video encoder is used to encode the stream. Both video encoders support setting of various attributes on the output media type such as bitrate, frame rate, frame size, aspect ratio, etc. Just as with the video decoder, an instance of the H.264 video encoder can be created by invoking `MFTEnumEx` function or by calling `CoCreateInstance`. The CLSID for the AMD hardware encoder is `CLSID_AMD_H264_HW_EncoderMFT`.

3.7 MFT Components

In addition to the video encoders and decoders, other MFT components developed by AMD also are used across various samples. The following sections describe each of these components. Other than the Video Processor MFT (which is a production-quality MFT), the rest of the MFTs are sample MFTs developed by AMD provided for reference only.

3.7.1 Video Processor (VQ MFT Component)

The Video Quality (VQ) Media Foundation Transform (MFT) lets developers enhance the quality of raw video. The VQ MFT can be used in Windows Store and classic desktop applications. There are many attributes supported by the VQ MFT, such as steady video, deblocking, mosquito noise removal, denoise, edge enhancement, de-interlacing, dynamic contrast, color vibrance effect, skin-tone correction, gamma correction, brighter-whites effect, brightness, contrast, saturation, tint, false contour reduction, scaling, and frame rate conversion.

All the video quality filters mentioned above run the kernel on the GPU. These kernels must be compiled during the run time. To minimize runtime costs, a Cache Builder is used.

All kernels used in the Video Quality filter are precompiled to the host GPU ISA for the first run and cached on the disk.

All video processing parameters are set through the standard IMFAttributes interface that is mandatory for every MFT. An application can get this interface through the QueryInterface() method. The application can set these properties statically before the MF session starts, or dynamically during the playback or video conferencing session.

The following code snippet shows this for two Video Processing parameters.

```
// Set VQ settings using user configuration file
CComPtr<IMFAttributes> attributes;
hr = vqTransform->GetAttributes(&attributes);
RETURNIFFAILED(hr);
```

```
//Enable VQ feqtures available for this system
CComPtr<IAMFCapabilityManager> capabilityManager;
hr = AMFCreateCapabilityManagerMFT(IID_PPV_ARGS(&capabilityManager));
RETURNIFFAILED(hr);

// Showcasing for couple of parameters. Similarly done for rest of VQ
      parameters.
if (S_OK == capabilityManager->IsEnabled(requestType,
AMF_EFFECT_STEADY_VIDEO))
{
  vqAttributes->SetUINT32(AMF_EFFECT_STEADY_VIDEO,
      vqParams->streadyVideoEnable);
}

if (S_OK == capabilityManager->IsEnabled(requestType,
AMF_EFFECT_DEINTERLACING))
{
  vqAttributes->SetUINT32(AMF_EFFECT_DEINTERLACING,
      vqParams->deinterlacing);
}


RETURNIFFAILED(hr);
```

All parameters represent pairs of Globally Unique Identifier (GUID) and Value; they are defined in the `VqMft.h` header file (`<installDirectory>/inc/VqMft.h`). All parameters can be set to, or queried from, the MFT.

The Video Quality filter provides a C++ interface that lets you Query the capability manager for the real-time playback capabilities of the current GPU, based on the Video Quality feature set.

In WinRT mode, an application does not have direct access to the Video Processor MFT. The MFT is inserted as an effect using the `MediaElement:AddVideoEffect()` or `MediaCapture::AddEffectAsync()` method. See Section 7.1.2, “The AddVideoEffect Method,” page 7-2, for more information.

3.7.2 Color Converter

AMD also provides an MFT to convert color between NV12 and RGBA DX surfaces. This MFT supports color conversion using OpenCL kernels, with and without DX11-OpenCL inter-operations (interops) and DX9-OpenCL interops.

You can create an instance of this component by invoking `CoCreateInstance` with CLSID `CLSID_ColorConvert_OpenCLMFTDx11` for DX11 surfaces or `CLSID_ColorConvert_OpenCLMFTDx9` for DX9 surfaces.

3.7.3 Overlay

An Overlay MFT is provided to help you overlay images on base images. The MFT supports overlaying in RGBA using OpenCL kernels, with and without DX11-OpenCL inter-operations (interops) and DX9-OpenCL interops.

You can create an instance of this component by invoking `CoCreateInstance` with CLSID `CLSID_Overlay_OpenCLMFTDx11` for DX11 surfaces or `CLSID_Overlay_OpenCLMFTDx9` for DX9 surfaces.

3.7.4 SVC Splitter

The SVC splitter MFT is a custom MFT to split the temporal layers into individual H.264 streams.

You can create an instance of this component by invoking `CoCreateInstance` with CLSID `CLSID_SvcMFT`.

3.7.5 Resizer

A Resizer MFT is provided to demonstrate how to write and use a custom MFT for resizing using DX11-OpenCL inter-operations (interops) and DX9-OpenCL interops. The resizer MFT supports resizing to the following resolutions

- 1920 x 1080
- 1280 x 720
- 800 x 480
- 640 x 480
- 320 x 240

On Windows 8/8.1, the OpenCL Resizer MFT uses the DirectX 11-OpenCL interop to get the OpenCL image from D3D11Texture2D, on which the resizer kernel is applied.

On Windows 7, the OpenCL Resizer MFT uses the DirectX9-OpenCL interop to obtain the OpenCL image from Direct3DSurface9, on which the resizer kernel is applied.

You can create an instance of this component by invoking `CoCreateInstance` with CLSID `CLSID_OpenCLMFTDx11` for DX11 surfaces or `CLSID_OpenCLMFTDx9` for DX9 surfaces.

3.8 Custom Functions

Custom Source: The AMD custom source is implemented to accept H.264 elementary stream as input. The custom source is provided as part of the Simple Transcode sample.

Custom Sink: The AMD custom sink produces a H.264 elementary stream as on output. The custom sink writes out elementary H.264 stream. Similar to the custom source, the custom sink is also provided as part of the Simple Transcode sample.

The AMD custom source and sink components can be used in their topology to parse elementary H.264 stream or to write elementary H.264 data.

3.9 DX9/11 – OpenCL Interoperability

The computing power of GPUs can improve the interactivity of 3D graphics. This requires optimum interoperability between compute (OpenCL) and graphics APIs (DirectX). Achieving this optimum interoperability requires:

- Marking the buffers and textures that need to be shared.
- Enqueuing, acquiring, and releasing commands into an OpenCL command queue to synchronize the graphics and compute code. This makes it possible to efficiently execute the compute kernels on the GPU without synchronizing with the host.

The ImageOverlayOpenCL sample has taken advantage of this DirectX – OpenCL interop. Enabling the interop involves translating the input from DirectX to OpenCl and the generated output from OpenCL to DirectX land. This enhances performance because there is no need to memory copy buffers from the GPU to the CPU and back to the GPU. Figure 3.6 illustrates the advantage of using interop.

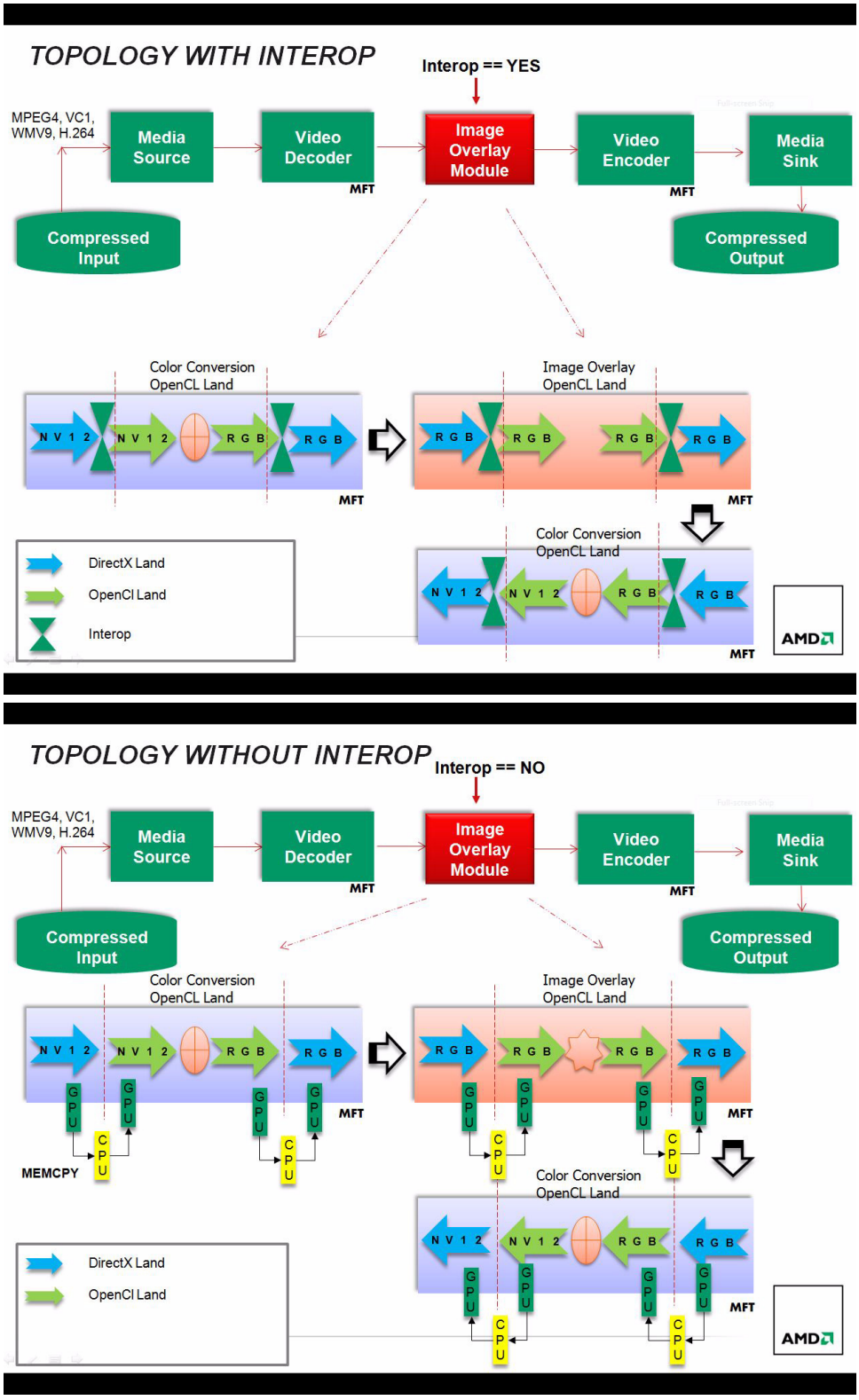


Figure 3.6 Topology With and Without Interop

The following is an outline of the code flow for using interop on D3D11.

1. Output OpenCL Image

   a. Create an output texture of type `ID3D11Texture2D`.

   ```
   CreateTexture2D(&desc, NULL, texture);
   ```

   Where `desc` is of type `D3D11_TEXTURE2D_DESC`, and the fields of this structure are populated by the user.

   b. From the texture, create an output surface.

   ```
   CComPtr<IDXGISurface> outputSurface;
   outputTexture->QueryInterface(&outputSurface);
   ```

   c. Get the output surface buffer.

   ```
   hr = MFCreateDXGISurfaceBuffer(IID_ID3D11Texture2D,
   outputSurface, 0, TRUE, &outputMediaBuffer);
   ```

   d. Gets OpenCL image object from output D3D11 texture using interop.

   ```
   imageOut = clCreateFromD3D11Texture2DKHR(context(),
   memAccess, outputTexture, textureViewIndex, &errNum);
   ```

2. Input OpenCL Image

   a. Convert Input media buffer to texture.

   ```
   CComPtr<IMFDXGIBuffer> spD11Buffer;
          hr = mediaBuffer->QueryInterface(IID_IMFDXGIBuffer,
   (void**) &spD11Buffer);
   if (SUCCEEDED(hr))
          {
              hr = spD11Buffer-
   >GetSubresourceIndex(textureViewIndex);
              RETURNIFFAILED(hr);

              hr = spD11Buffer->GetResource(IID_ID3D11Texture2D,
   (void**) ppTexture);
              RETURNIFFAILED(hr);
           }
   ```

   b. Gets OpenCL image object from output D3D11 texture using interop.

   ```
   imageIn = clCreateFromD3D11Texture2DKHR(context(), memAccess,
   inputTexture, textureViewIndex, &errNum);
   ```

3. Acquire OpenCL memory objects.

```
errNum = clEnqueueAcquireD3D11ObjectsKHR(mClQueue(), 1, &imageIn(), 0,
NULL, NULL);
OPENCL_RETURNIFFAILED(errNum);

errNum = clEnqueueAcquireD3D11ObjectsKHR(mClQueue(), 1, &imageOut(), 0,
NULL, NULL);
OPENCL_RETURNIFFAILED(errNum);
```

4. Perform the OpenCL operation. Invoke the kernel.

```
hr = openCL_kernel(imageIn, widthIn, heightIn, imageOut,
widthOut, heightOut);
RETURNIFFAILED(hr);
```

5. Release OpenCL memory objects.

```
errNum = clEnqueueReleaseD3D11ObjectsKHR(mClQueue(), 1, &imageIn(), 0,
NULL, NULL);
OPENCL_RETURNIFFAILED(errNum);
```

```
errNum = clEnqueueReleaseD3D11ObjectsKHR(mClQueue(), 1, &imageOut(), 0,
NULL, NULL);
OPENCL_RETURNIFFAILED(errNum);

errNum = mClQueue.finish();
OPENCL_RETURNIFFAILED(errNum);
```

The following is the outline for the code flow to use Interop on D3D9.

1. Output OpenCL Image.
   a. Create an output surface of type `IDirect3DSurface9`.

```
device->CreateOffscreenPlainSurface(outputWidth,
outputHeight, D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT,
&outputSurface, nullptr);
```

   b. Get the output surface buffer.

```
hr = MFCreateDXSurfaceBuffer(IID_IDirect3DSurface9,
outputSurface, TRUE, &outputMediaBuffer);
```

   c. Gets OpenCL image object from output D3D11 texture using interop.

```
_cl_dx9_surface_info_khr surfaceInfo = {oututSurface, 0};
imageOut = clCreateFromDX9MediaSurfaceKHR(context(), memAccess,
       CL_ADAPTER_D3D9EX_KHR, &surfaceInfo, surfaceIdx,
&errNum);
```

2. Input OpenCL Image.
   a. Convert Input media buffer to surface.

```
hr = MFGetService(mediaBuffer, MR_BUFFER_SERVICE,
IID_PPV_ARGS(inputSurface));
```

   b. Gets OpenCL image object from output D3D9 surface using interop.

```
_cl_dx9_surface_info_khr surfaceInfo = {inputSurface, 0};
imageIn = clCreateFromDX9MediaSurfaceKHR(context(), memAccess,
CL_ADAPTER_D3D9EX_KHR, &surfaceInfo, surfaceIdx, &errNum);
```

3. Acquire OpenCL memory objects.

```
errNum = clEnqueueAcquireDX9MediaSurfacesKHR(mClQueue(), 1,
&imageIn(), 0, nullptr, nullptr);
OPENCL_RETURNIFFAILED(errNum);

errNum = clEnqueueAcquireDX9MediaSurfacesKHR(mClQueue(), 1,
&imageOut(), 0, nullptr, nullptr);
OPENCL_RETURNIFFAILED(errNum);
```

4. Perform the OpenCL operation. Invoke the kernel.

```
hr = openCL_kernel(imageIn, widthIn, heightIn, imageOut, widthOut,
heightOut);
RETURNIFFAILED(hr);
```

5. Release OpenCL memory object.

```
errNum = clEnqueueReleaseDX9MediaSurfacesKHR(mClQueue(), 1,
&imageIn(), 0, nullptr, nullptr);
OPENCL_RETURNIFFAILED(errNum);

errNum = clEnqueueReleaseDX9MediaSurfacesKHR(mClQueue(), 1,
&imageOut(), 0, nullptr, nullptr);
```

```
OPENCL_RETURNIFFAILED(errNum);

errNum = mClQueue.finish();
OPENCL_RETURNIFFAILED(errNum);
```

Chapter 4
Video Quality Filters

The suite of AMD video quality processing algorithms lets users improve the resulting perceptual quality of video playback, transcode, and conferencing applications. Earlier, these video processing features were implemented in the multi-media driver: users could access and customize these features through the Catalyst Control Center (CCC). However, settings applied through the CCC were global across all invocations of the video pipeline, including transcode, video conferencing, and playback, so they could not be applied to only one instance.

The AMD Media SDK lets users independently select the video processing features to be applied to each invocation of the video pipeline. The AMD Media SDK also provides a user-friendly way to interact with, and use, video processing features.

Video processing algorithms are integral to the viewing experience. In some cases, these algorithms target objectionable video artifacts. In other cases, they can be used to adjust for display limitations or personal preferences. For example, skewing a color wheel to boost a particular region of the color matrix can compensate for a display limitation. In Figure 4.1, the left image shows a color wheel representing the color space; the right image shows the same color wheel with a five-degree shift in the hue.

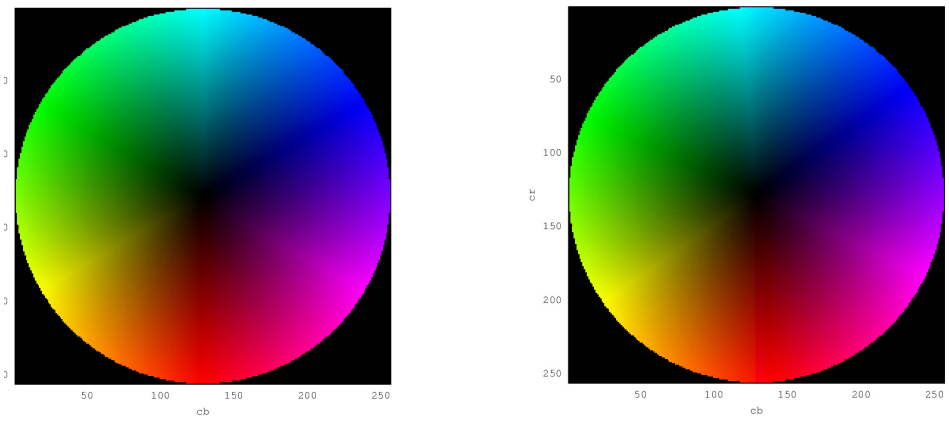


Figure 4.1 Example of Hue Shift on Color Wheel

4.1 Generating Video Pipelines

Using the AMD Media SDK, an application developer can generate a video pipeline which chains processing elements incorporating both video decoding

and encoding. For example, a video decoder can be instantiated and its output channeled through a series of video processing elements, as shown in Figure 4.2.

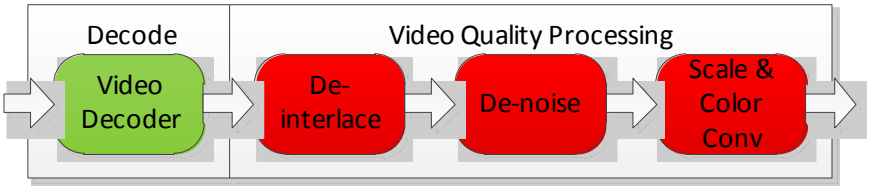


Figure 4.2 Example of Video Pipeline Instantiation

Similarly, an application developer can string together a video transcoding use case, as illustrated in Figure 4.3.

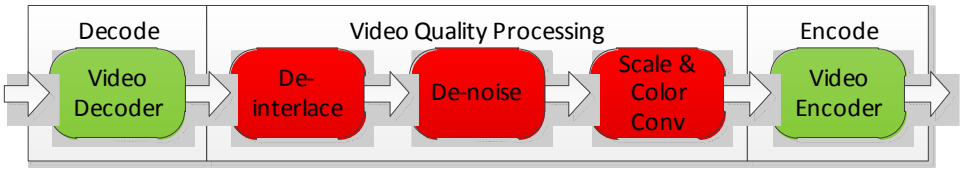


Figure 4.3 Example of Video Transcoding Use Case

The video processing algorithms also let developers control the user experience based on content characteristics.

To familiarize themselves with the video quality processing features in the AMD Media SDK, users can generate a simple playback application and insert the needed video quality features. The video quality processing features are exposed through the Video Quality MFT. This MFT can be used in Windows 7 and Windows 8/8.1 applications.

4.2 Video Processing Features

The AMD Media SDK provides direct access to the latest video processing techniques which are accelerated on AMD GPUs, ranging from high-end discrete GPUs to low-end APUs. This lets users directly tune pipeline parameters for specific use cases.

Using the video processing features provided in the AMD Media SDK, users can:

- Attenuate undesirable artifacts, such as compression or other artifacts (de-blocking, mosquito noise removal, false contour, de-noise).
- Enhance the video perception using features such as sharpening (edge enhancement, super resolution detail enhancement).
- Adjust for display limitations (such as by adjusting color mappings).
- Manage colors (color vibrance, brighter whites, brightness control, saturation, tint, dynamic contrast).

- Stabilize the image (steady video).
- Reduce temporal noise (super resolution motion compensation, temporal noise reduction)

The AMD Media SDK also provides a “Demo mode” option. Using this option, users can view the effect of applying video quality effects to only a vertical (right) half of the image. For example, Figure 4.4 shows the effect of applying false contour reduction to the right half of the image.

Figure 4.4 Demo Mode Example of False Contour Reduction

The video processing features in the AMD Media SDK include:

- Steady video
  A video de-shaking algorithm (AMD Steady Video™) that reduces blurring, such as that caused by camera motion. Specifying a higher level provides better image stabilization.
- Compression artifact removal
  Algorithms that reduce the video compression-induced artifacts, such as block noise, mosquito noise, and false contour noise, as well as those that allow up-scaled artifact removal.
  - De-blocking
    Improves the visual quality of decoded pictures by smoothing the sharp edges between macroblocks induced by block-coding techniques. Specifying a higher level provides better de-blocking effect.
    Figure 4.5 and Figure 4.6 show the effects of applying the de-blocking filter. Figure 4.5 shows the original image, in which the blocks and their edges are visible. In the corrected image, Figure 4.6, the edges have been smoothed out.

Figure 4.5 Example Image Without De-Blocking

Figure 4.6 De-Blocking Applied to Example Image

- Mosquito de-noise
  Reduces the “ringing” effect in successive still images that appear in sequence as a shimmering blur of dots around edges. Specifying a higher level provides greater noise reduction.
- False contour reduction
  In a digital video signal, the intensity values of components are represented by a limited number of discrete levels, so certain differences in intensity in continuous gradations are perceived instead as a series of discrete steps (false contours). Specifying a higher level provides greater false contour reduction.
  To see the effects of applying false contour reduction to an image, see Figure 4.4. Demo mode, which shows the effect of video filters as applied to the right half of the source image, has been enabled. The left half of the figure shows the original image with a series of discrete steps (false contours). The right half of the figure shows how applying false contour reduction helps smooth those gradations.

″ De-noise
Attenuates temporal noise patterns. Specifying a higher level provides greater de-noise effect.

″ Sharpness
Enhances the edge contrast of an image or video in an attempt to improve its sharpness. It works by identifying sharp edge boundaries in the image, such as the edge between a subject and a background of a contrasting color, and increasing the image contrast in the area immediately around the edge. Specifying a higher level provides greater edge enhancement.

″ Super resolution effects
Enhances video using motion-compensated temporal noise reduction, and enhances detail with enhanced frequency response.

″ Color management
Provides controls for a variety of video color effects:

- Hue and saturation controls.
- Color vibrance and brighter whites: Color vibrance saturates already intense colors (bright reds, blues, greens, etc.) while ignoring skin tones and other normal colors. This tends to produce more natural-looking images.
- Gamma correction: Gamma encoding optimizes the bits or bandwidth for human visual perception of light and color.
- Brightness, contrast, and tint.
- Dynamic range: Determines the range of the pixel values.

″ Skin tone correction
Corrects skin tone inaccuracies due to lighting effects. The effect of applying skin tone correction is shown in the difference between Figure 4.7 and Figure 4.8. Figure 4.7 shows the original image, in which lighting effects have obscured the original skin tone. In the corrected image, Figure 4.8, a more natural skin tone is visible.

Figure 4.7 Example Image Without Skin Tone Correction

Figure 4.8 Skin Tone Correction Applied to Example Image

- Dynamic contrast
  Adjusts the color histogram to improve overall image contrast.
- Scaling
  Provides both bilinear and four-tap poly-phase scaling.
- De-interlacing
  Provides a variety of scalable techniques, including motion adaptive, vector adaptive, directional, cadence detection, and pull-down detection. The following figure shows the de-interlace effect of applying the adaptive technique.

Figure 4.9 Example Image Without De-Interlacing

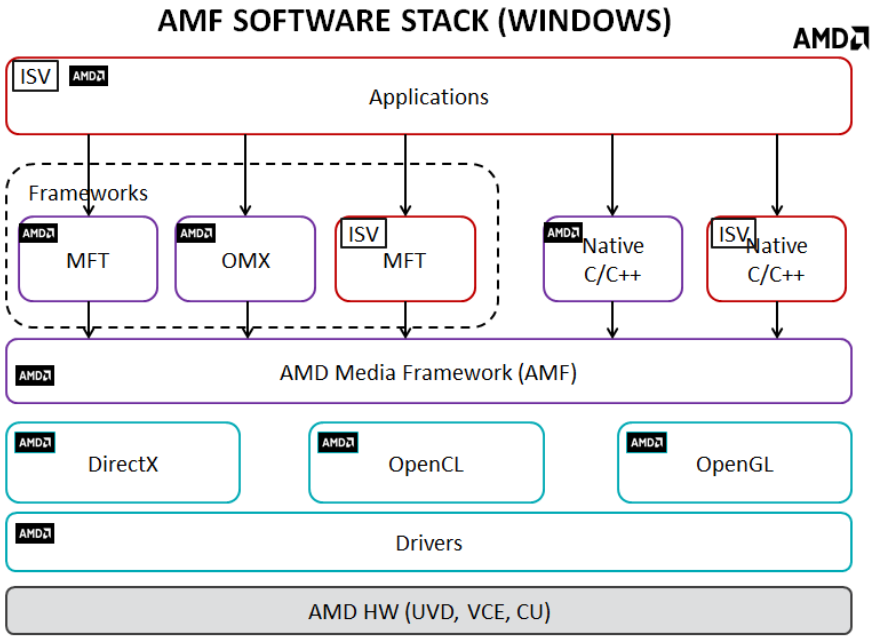
Figure 4.10 De-Interlacing Applied to Example Image

Chapter 5
AMF Library

The AMD Media Framework (AMF) is a subset of the AMD Media SDK. The AMF provides a framework for media processing on AMD platforms, consisting of OS- and framework-agnostic C++ APIs. The C++ API interface allows AMF to be easily adapted to other multimedia frameworks such as MFT, Gstreamer, FFmpeg, and OpenMAX. AMF accepts input from and provides inter-op with DirectX, OpenCL and OpenGL.

The AMF interface allows ISVs to perform video encoding, decoding, and pre- or post-processing using AMD's VCE and UVD hardware blocks and GPU shader-based acceleration.

The following diagram depicts the AMD Media SDK software stack:



The AMF comprises the following components:

- Full Decoder
  - UVD - H264 AVC, H264 VC1, WMV9, MPEG2, MPEG4 P2, H264 MVC, MJPEG, H265 HEVC
  - Shader-based MJPEG for legacy hardware

- Encoder:
  - VCE: H264 AVC, SVC
- Video Converter (Shader based)
  - Color Space Conversion, Scale, Composition
- AMF DEM Screen capture encoder (VCE)
  - Desktop only
- Capability Manager
  - Information about hardware and SDK capabilities

The AMF supports SI and later platforms at full performance and NI platforms at reduced performance. It supports 32- and 64-bit Windows 7 and Windows 8.1 Desktop applications.

For more information about the AMF library and APIs, see the *AMD AMF Reference Manual*.

Chapter 6
AMF-DEM Library

The AMD Media Framework (AMF) Display Encode Mode (DEM) is a subset of the AMD Media SDK. It is built on top of the AMF components. The AMF-DEM interface allows an Independent Software Vendor (ISV) to encode the audio and video contents of the display for streaming to a remote device.

Figure 6.1 shows a system overview of the AMF DEM mechanism.

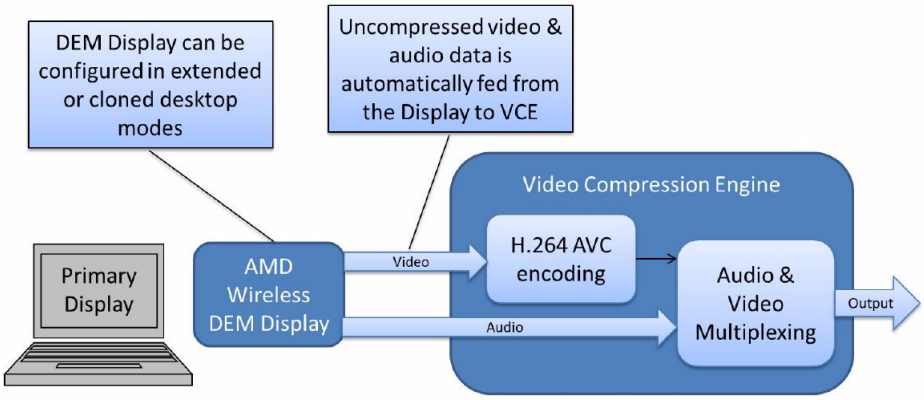


Figure 6.1 System Overview of the AMF Display Encode Mode

AMD provides a hardware connection from the Display Controller Engine to the Video Coding Engine. It functions as a physical display connection to the Windows operating system. Like a standard display, it can be configured through the Windows Control Panel | Change Display Settings panel. For more details, see Section 6.1, “Configuring the Display”.

The AMF DEM interface can be configured to get both video and audio data from the display (similar to HDMI); it also can control the video encoding parameters for the video stream. Streams can be received as either a WiFi Display compliant MPEG-2 transport stream, or as elementary video and/or audio streams.

After the display is configured and the DEM session started, the video and audio data rendered to the display is automatically fed to the Video Coding Engine and can be retrieved through the DEM API. This approach allows very low-latency when compared with screen-scraping plus encoding.

6.1 Configuring the Display

The source display can be configured through the Windows Control Panel -> Change Display Settings panel. This virtual display appears as AMD Wireless in the panel. Figure 6.2 and Figure 6.3 show how to set the display in Extended Desktop Mode and Cloned Mode (Duplicate displays).

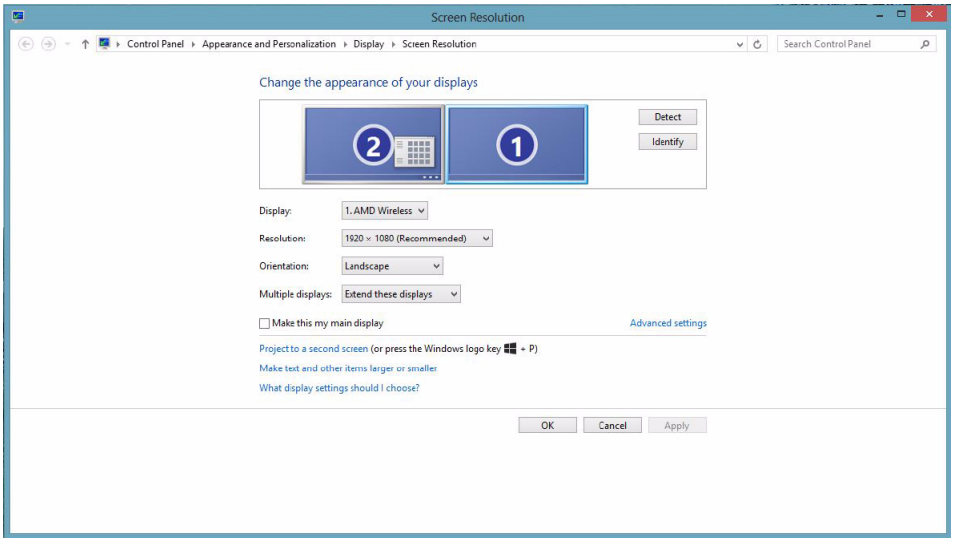


Figure 6.2 Setting AMD Wireless in Extended Desktop Mode

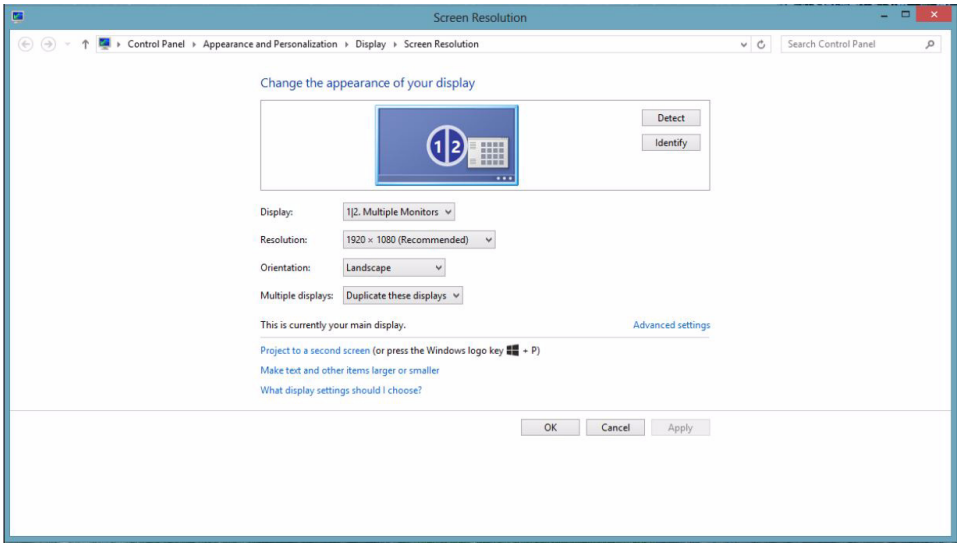


Figure 6.3 Setting AMD Wireless in Cloned Mode

You also can configure the display by writing code, as shown below.

```
DISPLAY_DEVICE dispDev;
```

```
/*****************************************************************
* Acquiring Remote Display*
*****************************************************************/
mLastAmfErrorCode = remoteDisplayCreate();
if (AMF_OK != mLastAmfErrorCode)
{
mSetupErrorCode = ERR_REMOTEDISPLAYCREATE_FAILED;
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}

/*****************************************************************
* Changing Display Resolution*
*****************************************************************/
dispDev.cb = sizeof(DISPLAY_DEVICE);

if (!EnumDisplayDevices(NULL, 0, &dispDev, 0))
{
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED;
status = GetLastError();
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}

DISPLAY_DEVICE monitor;
monitor.cb = sizeof(DISPLAY_DEVICE);
if (!EnumDisplayDevices(dispDev.DeviceName, 0, &monitor, 0))
{
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED;
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}

DEVMODE devMode;
devMode.dmSize = sizeof(DEVMODE);
if (!EnumDisplaySettings(dispDev.DeviceName, ENUM_CURRENT_SETTINGS,
&devMode))
{
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED;
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}

devMode.dmPelsWidth = pConfig->width;
devMode.dmPelsHeight = pConfig->height;
status = ChangeDisplaySettings(&devMode, CDS_TEST);
if(status == DISP_CHANGE_FAILED)
{
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED;
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}
else
{
status = ChangeDisplaySettings(&devMode, CDS_UPDATEREGISTRY);

switch(status)



{
case DISP_CHANGE_SUCCESSFUL:
break;
case DISP_CHANGE_RESTART:
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED; return (mLastAmfErrorCode |
ERR_SETUP_FAILED); break;
default:
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED;
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}
}



/*****************************************************************
* Cloning the Remote Display*
```

```
*********************************************************************/
status = SetDisplayConfig(0, NULL, 0, NULL, (SDC_APPLY |
SDC_TOPOLOGY_CLONE));
if(status != ERROR_SUCCESS)
{
mSetupErrorCode = ERR_CHANGERESOLUTION_FAILED;
return (mLastAmfErrorCode | ERR_SETUP_FAILED);
}
```

6.2 Supported Resolutions

The DEM supports 1920x1080, 1280x720, and 720x480 resolutions. The Windows Desktop can be set to additional display resolutions. These resolutions are automatically scaled (done in hardware) and encoded to the nearest supported resolution. For example, a desktop of 800x600 is scaled to 1280x720 and generates a 1280x720 encoded stream.

Note that the display resolutions also vary, depending on whether the desktop is in Extended mode or Cloned mode, as well as on the displays that are connected.

Table 6.1 shows examples of possible supported display modes and target timing in a single display configuration paired with a 1080p HDTV.

Table 6.1 Supported Modes (Single Display)

Display Resolution	Encoding Resolution	Capture Frame Rates (fps)
640x480	720x480	60
720x480	720x480	60
800x600	1280x720	50, 60
1024x768	1920x1080	24, 50, 60
1280x720	1280x720	50, 60
1280x1024	1920x1080	24, 50, 60
1680x1050	1920x1080	24, 50, 60
1920x1080	1920x1080	24, 50, 60

For the clone case, the timings used on the target displays can vary based on what the monitor supports. Table 6.2 shows examples of possible supported display modes and target timing in a 1600*1200 CRT cloned display configuration with a 1080p HDTV.

Table 6.2 Supported Modes (Clone Case)

Display Resolution	Encoding Resolution	CRT	Capture Frame Rates (fps)
640x480	720x480	640x480	60
720x480	720x480	720x480	60
800x600	1280x720	800x600	60
1024x768	1920x1080	1024x768	60
1280x720	1280x720	1280x720	60
1280x1024	1920x1080	1280x1024	60
1600x1200	1920x1080	1600x1200	60

Note: Some lower Thermal Design Power (low TDP) AMD platforms might not support full 1080p resolution.

6.3 Supported Operating Systems and Hardware

The DEM solution is supported on Windows 7 and Windows 8/8.1. VCE-DEM is not supported on WinRT.

VCE-DEM requires AMD Video Coding Engine (VCE) hardware. It is only available in AMD APU and GPU products that contain VCE.

The DEM is limited to one DEM session per system.

6.4 Using DEM

Figure 6.4 shows the high-level sequence for using DEM.

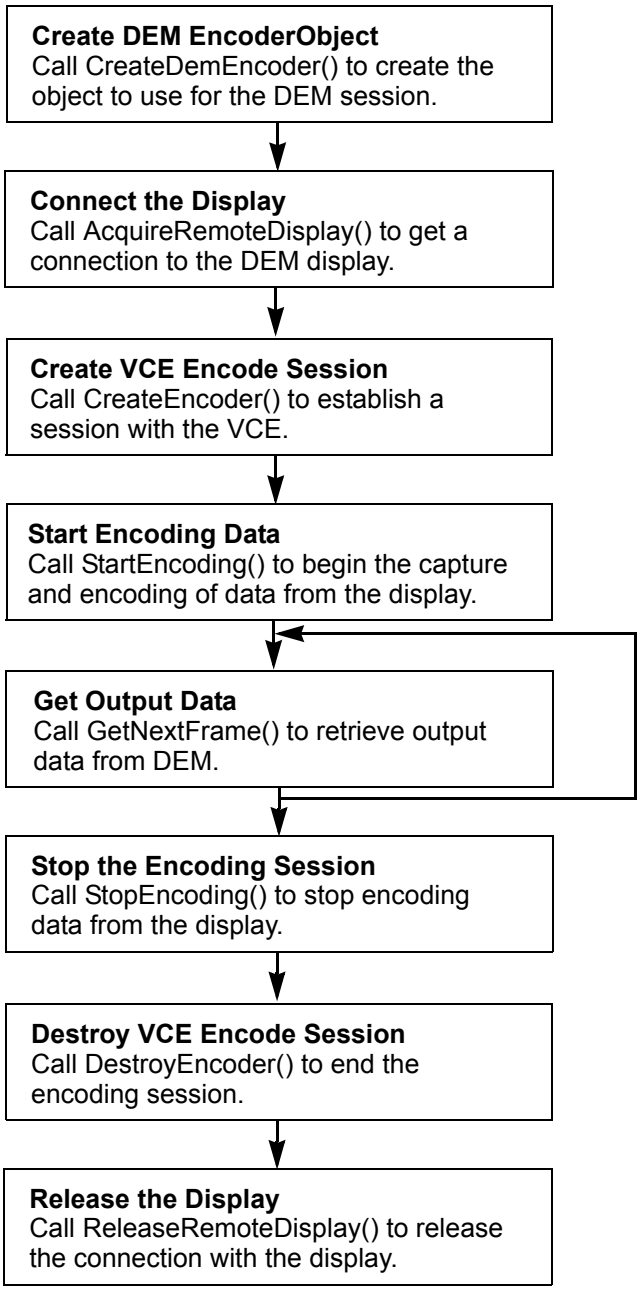


Figure 6.4 DEM High-Level Sequence

6.5 Using the VCE-DEM Library

For details about using the VCE-DEM library, see the *MediaSDK\docs\displayEncodeMode\MediaSDK_DEMAPI.pdf* document.

6.6 Desktop Screen Capture Use Case

For information on how to build and run, supported configuration parameters, and supported platforms of the use case, see
`MediaSDK\samples\dem\screenCapture\docs\MediaSDK_DEM_ScreenCapture.pdf`

Chapter 7
Developing Windows Store Applications

There are many ways of developing Windows Store applications in Windows 8/8.1, using any of these programming languages:

- JavaScript and HTML5
- C# and Extensible Application Markup Language (XAML)
- Microsoft Visual Basic and XAML
- Visual C++ component extensions (C++/CX) and XAML
- C++/CX and Microsoft DirectX

The AMD Media SDK uses Visual C++ component extensions (C++/CX) and the XAML programming language. For details, see:
http://msdn.microsoft.com/en-us/library/windows/apps/br229566.aspx

For example, C# can be used to write Windows Store Apps using MFTs. To write a Windows Store device application in C# or JavaScript that interacts with an MFT, a wrapper must be created. This wrapper exposes the MFT interfaces in a Windows Runtime Component that is visible to the Windows Store device app. See:
http://msdn.microsoft.com/en-us/library/windows/hardware/dn394064(v=vs.85).aspx

The Wrapper subproject in the Windows Store device app for camera sample provides an example of how to expose MFT to the Windows Runtime, so that it can be used from a Windows Store device app written in C# or JavaScript. See:
http://code.msdn.microsoft.com/windowsapps/Metro-style-device-app-for-4f39b7bf

It is designed to work with the Driver MFT sample. See the Driver MFT sample for a step-by-step guide to installing, running, and testing the samples:
http://code.msdn.microsoft.com/windowshardware/Driver-MFT-Sample-34ecfecb

7.1 Adding Multimedia

This section contains information about how to add multimedia components and custom video processing filters. Filters can be Video Quality filters by AMD or any C++ AMP accelerated filters.

To capture, play, and render media in Windows Store applications, you can use the CaptureElement, MediaElement, and Image controls classes defined in the `Windows.UI.Xaml.Controls` namespace.

7.1.1 Commonly Used APIs

These include:

- `Windows.Media.Capture` – contains the majority of the APIs used for capturing media.
- `Windows.Media`, `Windows.Media.MediaProperties`, and `Windows.Media.Play` – contain the majority of the Windows Store APIs used for media playback.
- `Windows.Graphics.Imaging` – namespace that lets you decode images and edit the pixels and metadata.
- `Windows.Media.Transcoding` – lets you transcode video files from one format to another.
- `MediaTranscoder.PrepareFileTranscodeAsync` – Asynchronously initializes transcode operations on the specified file and returns a PrepareTranscodeResult object that can be used to invoke the transcode operation.
- `MediaElement::Play` – Plays media from the current position.
- `MediaElement::Pause` – Pauses media at the current position.
- `MediaElement::Stop` – Stops, and resets, media to be played from the beginning.

7.1.2 The AddVideoEffect Method

Video Effects can be applied to playback and transcode sessions using `MediaElement:AddVideoEffect` and `MediaTranscoder.AddVideoEffect`, respectively.

AMD Video Processing Filters (VQ filters) can be applied using the AddVideoEffect method. The playbackVqWinStore and transcodeVqWinStore samples show how to apply these filters using this method. Also see section 5.2 in `MediaSDK\docs\videoQualityMft\MediaSDK_MFT_VideoQualityAPI.pdf` for details.

Similarly, you can add C++ AMP written effects to Windows Store applications. This is done in the videoEditAmpWinStore sample. Also see http://msdn.microsoft.com/en-us/library/jj856977.aspx for more details.

Chapter 8
Performance Optimization

This chapter describes how to:

- build optimized pipelines for targeted application scenarios,
- use the buffer-sharing solution with DirectShow components,
- use the DX9 to OpenCL interop,
- use MFTs effectively to build optimized applications, and
- develop applications by using the C++ AMP and OpenCL filters.

8.1 DirectX Buffer Sharing

One of the critical bottlenecks in optimal application performance is memory transfers across devices. There is a high performance cost associated with applications that have huge memory transfers between CPU and GPU devices.

To reduce this cost, it is best to have solutions with minimal memory transfer across devices. This can be done by allocating and reusing buffers located in the GPU across all the MFT components in the pipeline. This ensures the buffers remain in the GPU memory throughout the pipeline and are transferred just once to the CPU after the buffer is last updated.

Buffers allocated in the GPU can be shared across different MFT components using the same Direct3D device across both the components.

For example, as shown in Figure 8.1, if the pipeline contains Decoder and Encoder Transform objects of type IMFTransform, the buffers can be shared by associating the same Direct3D device across the two objects.

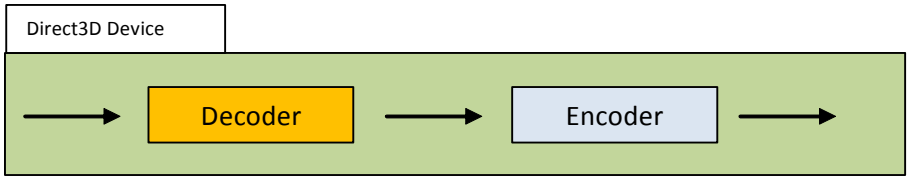


Figure 8.1 Buffer Sharing Example

The Direct3D device manager enables two or more objects to share the same Direct3D9 device. One object acts as the owner of the device and creates the Direct3D device manager by calling `DXVA2CreateDirect3DDeviceManager9`.

Other objects can obtain a pointer to the device manager from the device owner and use the device manager to get a pointer to the Direct3D device.

The following code snippet provides an example.

```
// Create the D3D object, which is needed to create the D3DDevice
CComPtr<IDirect3D9Ex> d3d9;
hr = Direct3DCreate9Ex(D3D9b_SDK_VERSION, &d3d9);

// Create the Direct3D device
CComPtr<IDirect3DDevice9Ex> d3dDevice;
hr = d3d9->CreateDeviceEx(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
BehaviorFlags>, &presentParameters, NULL, &d3dDevice);
CComPtr<IDirect3DDeviceManager9> d3dDeviceManager;

// Create the Direct3D device manager
hr = DXVA2CreateDirect3DDeviceManager9(&resetToken, &d3dDeviceManager);
hr = d3dDeviceManager->ResetDevice(d3dDevice, resetToken);
```

The Microsoft link
http://msdn.microsoft.com/en-us/library/windows/desktop/aa965267(v=vs.85).aspx
explains in detail how to create, initialize, and share a device among several objects.

The Direct3D device manager mentioned above supports only Direct3D9 devices. It does not support DXGI devices. For DXGI devices, the Direct3D11 device is created by invoking `D3D11CreateDevice`, and the instance of a DirectX Graphics Manager is created by invoking `MFCreateDXGIDeviceManager`.

See
http://msdn.microsoft.com/en-us/library/windows/desktop/ff476082(v=vs.85).aspx and
http://msdn.microsoft.com/en-us/library/windows/desktop/hh162750(v=vs.85).aspx
for further details.

8.2 DirectX to OpenCL Interop

Application developers can use OpenCL instead of DirectCompute APIs to program on the GPU. Sharing Direct3D buffers with OpenCL context result in memory copy of the buffers from the GPU to the CPU, then back from the CPU to the GPU.

Figure 3.6 shows how DirectX buffers are first copied from GPU to CPU and then copied back to OpenCL buffers on the GPU. This memory copy has a cost associated with it and greatly impacts performance if the buffers copied are large. Performance in such cases can be optimized by eliminating the copy of buffers. This can be done using the OpenCL interoperability feature, which lets applications use DX9 media surfaces or Direct3D11 media textures as OpenCL memory objects.

Figure 3.6 shows how interoperability between DirectX and OpenCL helps eliminate the need for memory copies, hence boosting performance.

The use of OpenCL - DirextX9/11 interop APIs is explained in detail in the imageOverlayOpenCl sample.

8.3 Developing applications with C++ AMP and OpenCL

C++ Accelerated Massive Parallelism (C++ AMP) accelerates the execution of C++ code on the GPU. C++ AMP also allows the coding of multi-dimensional data algorithms to harness parallelism on heterogeneous hardware. C++ AMP language extensions can be used to control how data is moved from the CPU to the GPU and back, thereby improving performance.

C++ AMP samples are provided with the AMD Media SDK to increase the speed of applications in Windows and in Windows Store environments.

The videoEditAmp and videoEditAmpWinStore samples feature the use of MFTs and C++ AMP-based video processing. The videoEditAmp sample uses a C++ AMP-based resizer to resize the decoded video frames; the videoEditeAmpWinStore sample uses a C++ AMP-based greyscale convertor to color convert the decoded video frames. Figure 8.2 shows the topology used in the videEditAmp sample.

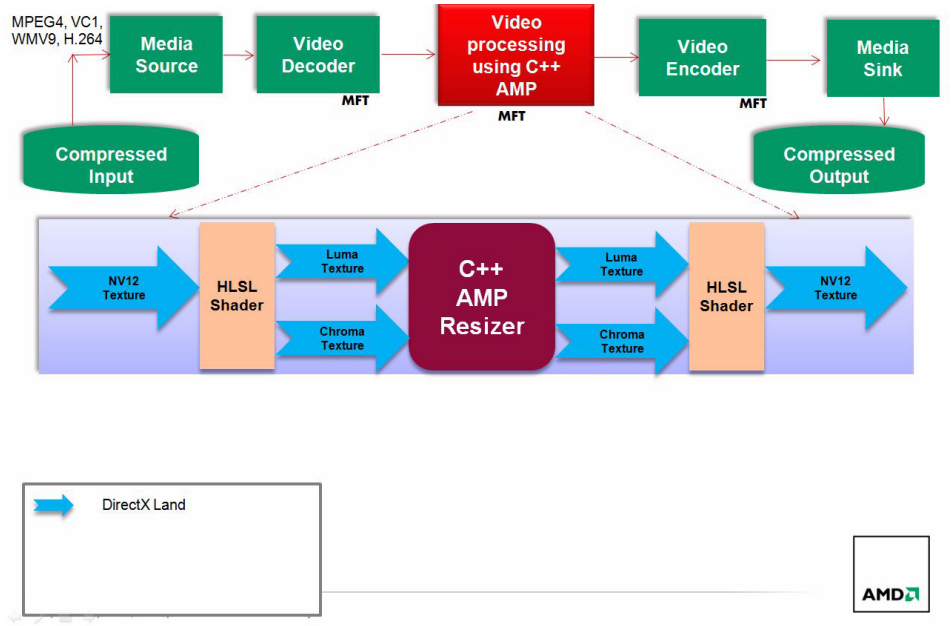


Figure 8.2 Video Editing Using C++ AMP

Input and output data to the video processing component is in NV12 format; however, C++ AMP works only with the RGBA format. Thus, the NV12 texture is split into Y and UV planes and processed as R8 and R8G8. The following code snippet provides an example. (See videoEditAmp and videoEditAmpWinStore for details of each of the functions.)

```
//Split NV12 into Y and UV planes:
hr = applyShader(nv12Texture, DXGI_FORMAT_R8_UNORM, luminanceTarget,
       DXGI_FORMAT_R8_UNORM, mCopyRshader);
hr = applyShader(nv12Texture, DXGI_FORMAT_R8G8_UNORM, chrominanceTarget,
       DXGI_FORMAT_R8G8_UNORM, mCopyRGshader);

//Process Y and UV planes as R8 and R8G8 textures:
```

```
hr = mMftBuilderObjPtr->createTexture(dstDesc.Width, dstDesc.Height,
        D3D11_BIND_SHADER_RESOURCE | D3D11_BIND_UNORDERED_ACCESS,
        DXGI_FORMAT_R8_UNORM, mD3d11Device, &spOutLuma);
hr = mMftBuilderObjPtr->createTexture(dstDesc.Width / 2,
        dstDesc.Height / 2, D3D11_BIND_SHADER_RESOURCE
            | D3D11_BIND_UNORDERED_ACCESS,
        DXGI_FORMAT_R8G8_UNORM, mD3d11Device, &spOutChroma);

//Resize process using C++ AMP:
hr = doResize(spOutLuma, spOutChroma, lumaTex, chromTex);

//Merge luma and chroma planes as NV12 texture:
hr = applyShader(pInY, DXGI_FORMAT_R8_UNORM, pOutText,
        DXGI_FORMAT_R8_UNORM, mCopyRshader);
hr = applyShader(pInUV, DXGI_FORMAT_R8G8_UNORM, pOutText,
        DXGI_FORMAT_R8G8_UNORM, mCopyRGshader);
```

8.4 Optimizing MFTs

The MFT encoder and decoder have certain configuration parameters that can be altered for performance gains. Some of these parameters and their values for optimal performance are described below.

″ AVEncCommonQualityVsSpeed

This parameter represents the trade-off between encoding quality and speed. A value of zero indicates lower quality but faster encoding; a value of 100 indicates higher quality but slower encoding. This parameter can be set to a lower value, at the cost of lower quality, for faster encoding.

For details, see:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd317840(v=vs.85).aspx

″ AVEncCommonLowLatency

Setting the value to TRUE structures the encoded stream, resulting in low latency decoding. Decoding latency is defined as the amount of data the decoder must buffer. The lower the amount of data the decoder must buffer, the faster is the decoding.

For details, see:

http://msdn.microsoft.com/en-us/library/windows/desktop/dd317656(v=vs.85).aspx

″ AVDecVideoAcceleration_H264

Enables or disables the hardware acceleration for H.264 video decoding. If the value is zero, the decoder does not use the DirectX Video Acceleration (DXVA) for H.264 video decoding. For faster decoding, set this parameter to TRUE. For details, see:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd742712(v=vs.85).aspx

″ AVDecVideoAcceleration_MPEG2

Enables or disables the hardware acceleration for MPEG2 video decoding. If the value is zero, the decoder does not use the DirectX Video Acceleration (DXVA) for MPEG2 video decoding. For faster decoding, set this parameter

to TRUE. For details, see:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd742713(v=vs.85).aspx

″ AVDecVideoAcceleration_VC1

This enables or disables the hardware acceleration for VC-1 video decoding. If the value is zero, the decoder does not use the DirectX Video Acceleration (DXVA) for VC-1 video decoding. For faster decoding, set this parameter to TRUE. For details, see:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd742714(v=vs.85).aspx

″ AVDecVideoFastDecodeMode

This gets or sets the video decoding speed. Values can range from 0 to 32, where 0 indicates normal decoding, and 32 indicates the fastest decoding available. The eAVFastDecodeMode enumeration specifies the video decoding speed. For details, see:

http://msdn.microsoft.com/en-us/library/windows/desktop/hh184791(v=vs.85).aspx

Chapter 9
Debugging Tips and Tricks

This chapter describes the use of logging in Media SDK components.

9.1 Using Log Files

The `-l` command-line option generates log files for each console-based sample. If `-l` is not provided in the command line, then no log file is generated.

- `-l 0` generates no log.
- `-l 1` generates the log at the API level.
- `-l 2` generates the log at the session level.

For example, running the transcodeVq sample with the command-line option `-l 1` or `-l 2` generates the `TranscodeVqErrorlog.txt` log file in the following directory

`$<installDirectory>\samples\mediaFoundation\transcodeVq\build\windows\.`

Figure 9.1 provides an example. The log file indicates any errors that occurred during the sample run; it also specifies the file and line number of the source file in which the error occurred.

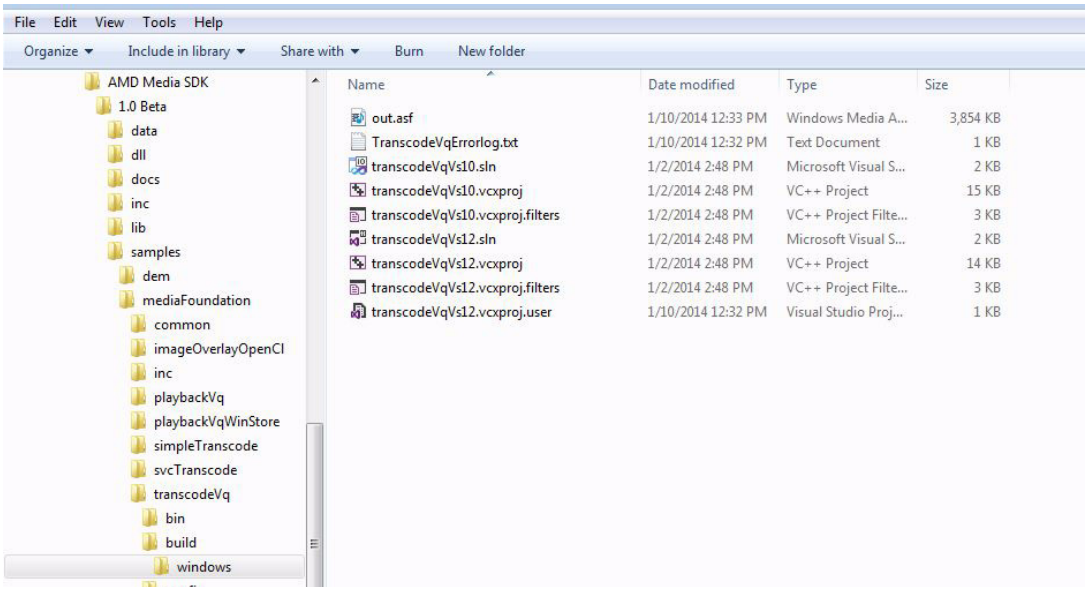


Figure 9.1 Example of Log File Generation

9.2 Using MFT Error Codes

When an error is returned in an asynchronous event, the failing HRESULT is stored in a member variable of the IMFAsyncResult object, which is sent to the `IMFAsyncCallback::Invoke()` method. The relevant values stored in the asynchronous result object are:

- The type of the event, which provides a context to the failure. The event type can be extracted using `IMFAsyncResult::GetType()`.
- The failing HRESULT that signals the cause of the failure. To get the HRESULT, use `IMFAsyncResult::GetStatus()`. MF HRESULTs are specific and can provide good clues to the cause of the failure.

Unfortunately, MF error codes are not commonly stored in various HRESULT lookup tools. For example, the lookup tool that comes with Microsoft Visual Studio 2010 does not recognize these error codes. The fastest way to determining what an MF error means is to looking it up inside the `mferror.h` header file. All standard MF errors start with a C00D hex value; that is, all of the common MF error numbers are between 0xC00D0000 and 0xC00DFFFF. The following table contains a list of the standard MF error ranges.

Table 9.1 Media Foundation Error Code Types

Error Range	Error Type
14000 - 14999	General MF
15000 - 15999	ASF parsing
16000 - 16999	Media source
17000 - 17999	MF network
18000 - 18999	MF Windows Media Container
19000 - 19999	Media sink
20000 - 20999	Renderer
21000 - 21999	Topology
25000 - 25999	Timeline
28000 - 28999	Transform
29000 - 29999	Content protection
40000 - 40999	Clock
41000 - 41999	MF quality management
42000 - 42999	MF transcode API

For example, general MF errors fall in the 0xC00D0000+14000 to 0xC00D0000+14999 range (0xC00D36B0 to 0xC00D3A97). The `mferror.h` header also contains brief one-sentence descriptions of the errors.

Similarly, to map the event type from the number returned by `IMFAsyncResult::GetType()`, look in the `mfobjects.h` header file, in the MediaEventType enumeration. The media event type names are fairly descriptive and usually provide some context for the error code.

9.3 Using MFTrace

MFTrace generates trace logs for Microsoft Media Foundation applications. It uses the Detours library to hook into Media Foundation API calls and generate trace logs. MFTrace also can record traces from any component that uses Event Tracing for Windows (ETW) or the software trace preprocessor (WPP) to generate traces. Trace logs can be generated by starting a new process from MFTrace, or by attaching MFTrace to an existing process.

Usage:

```
mftrace [-a Process][-c ConfigurationFile][-dc][-es][-k KeyWords][-l
Level][-o OutputFile][-v][-?] [{COMMAND|ETL_FILE}]
```

For details, see:
http://msdn.microsoft.com/en-us/library/windows/desktop/ff685370(v=vs.85).aspx

For information about using MFTrace to trace Media Foundation, see:
http://blogs.msdn.com/b/mf/archive/2010/08/11/using-mftrace-to-trace-media-foundation.aspx

For information about automating Trace Analysis, see:
http://blogs.msdn.com/b/mf/archive/2010/10/28/automating-trace-analysis.aspx

9.4 Ensuring that the transcodeVq sample works for H.264 Encoded Video

The transcodeVq sample should work for H.264, Mpeg4 Part 2 and VC1. If this sample fails for H.264 video, use the simpleTranscode sample for the same H.264 elementary stream, perform the transcode, and verify that the simpleTranscode sample runs as expected.

You can generate an H.264 elementary stream in one of the two following ways.

- Use Yamb-2.1.0.0 beta 2 with installer from http://yamb.unite-video.com/download.html
- Use FFmpeg:

  ```
  ffmpeg -i {some file} -vcodec copy -bsf h264_mp4toannexb -an -f
  {rawvideo|h264|or_other} out.h264:
  ```

  Download FFmpeg from: http://ffmpeg.zeranoe.com/builds/

9.5 Identifying Video Quality (VQ) Filter Issues by Bypassing VQ

If you encounter issues while playing any stream using the playbackVq or transcodeVq sample, bypass the VQ filter and run the same stream. If the stream runs normally, the VQ filter is the problem; otherwise, some other part of the sample is the problem. To bypass the VQ from the topology, remove the following code snippet from the playbackVq sample:

```
hr = mDecoderNode->ConnectOutput(0, mVqNode, 0);
LOGIFFAILED(mLogFile,hr,"Failed to connect decodernode->vqnode node @ %s
%d \n", FILE , LINE );
```

```
hr = mVqNode->ConnectOutput(0, mSinkNode, 0);
LOGIFFAILED(mLogFile,hr,"Failed to connect vqnode -> sink node @ %s %d
\n", FILE , LINE );
```

Replace the above snippet with the following code snippet

```
hr = mDecoderNode->ConnectOutput(0, mSinkNode, 0);
LOGIFFAILED(mLogFile,hr,"Failed to connect decodernode->sink node @ %s %d
\n", FILE , LINE );
```

9.6 Using the DirectX Video Acceleration (DXVA) Checker

By using the DXVA checker, you can confirm the following details.

- Decoder Device and Processor Device that can be used with GPU. See Figure 9.2 and Figure 9.3.
- Usage condition in other applications by trace log (see Figure 9.4).
- Supported mode and usage condition in DirectShow/Media Foundation codecs.
- Setting change in some DirectShow/Media Foundation codecs.
- The effect by changing the Video Processor parameter and changing Deocoder/Processor Device while playing the video.
  For details, see: http://bluesky23.yu-nagi.com/en/DXVAChecker.html

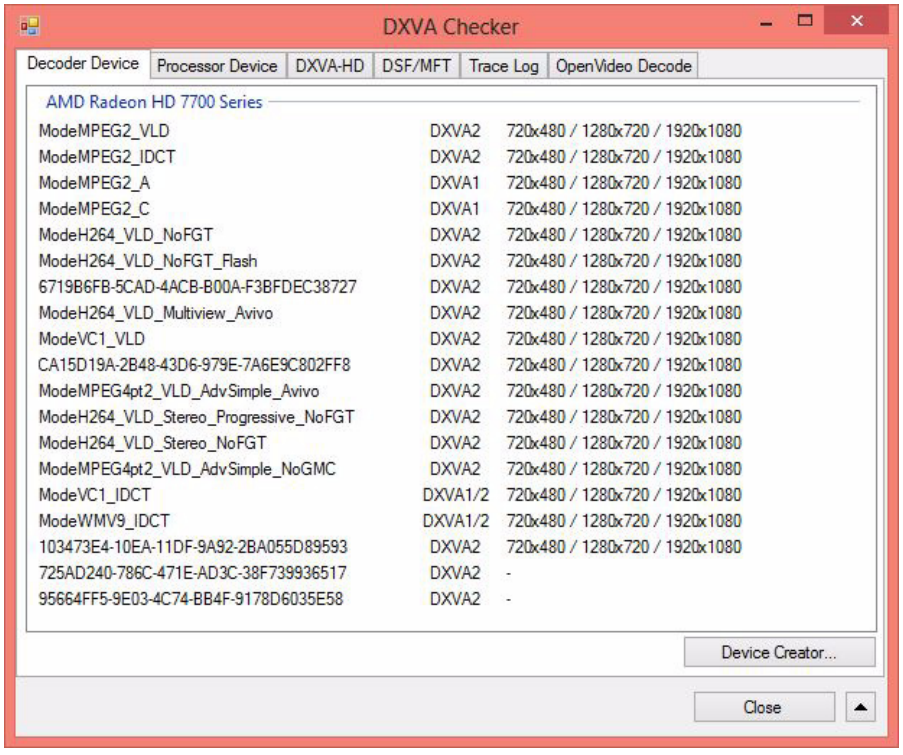


Figure 9.2 DXVA Checker Devices View

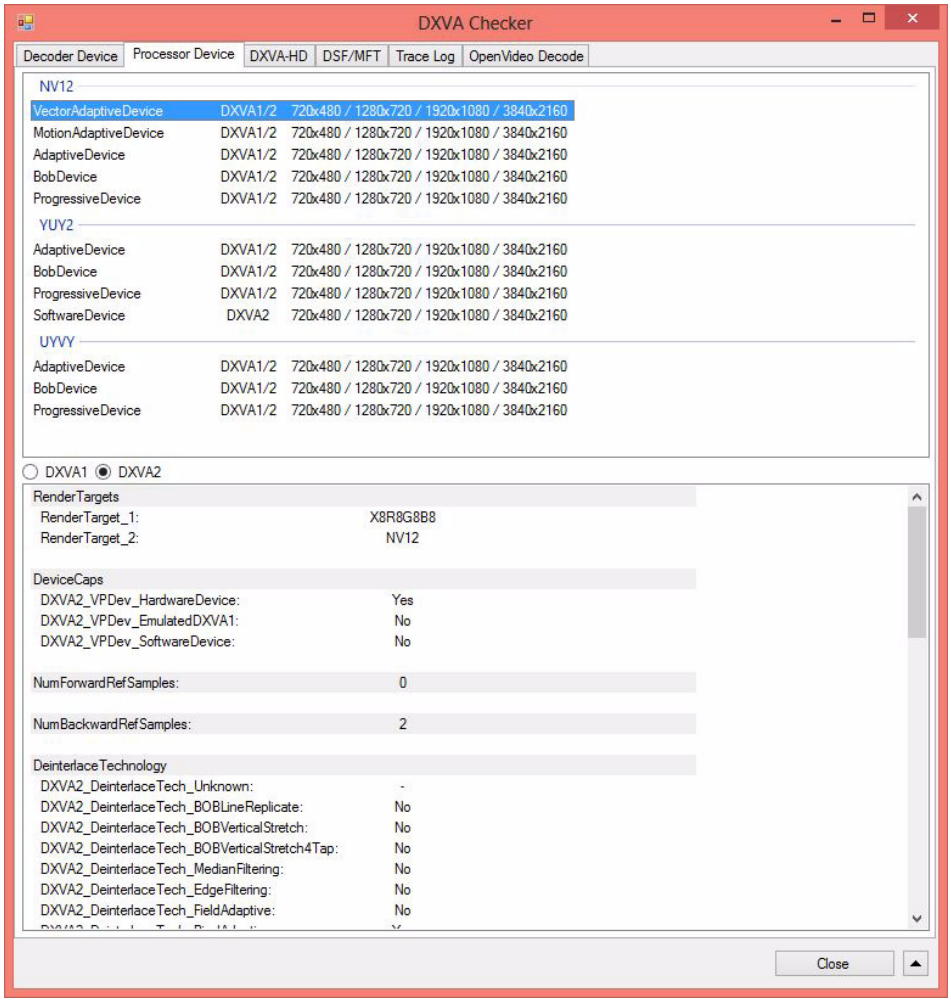


Figure 9.3 DXVA Checker Processor View

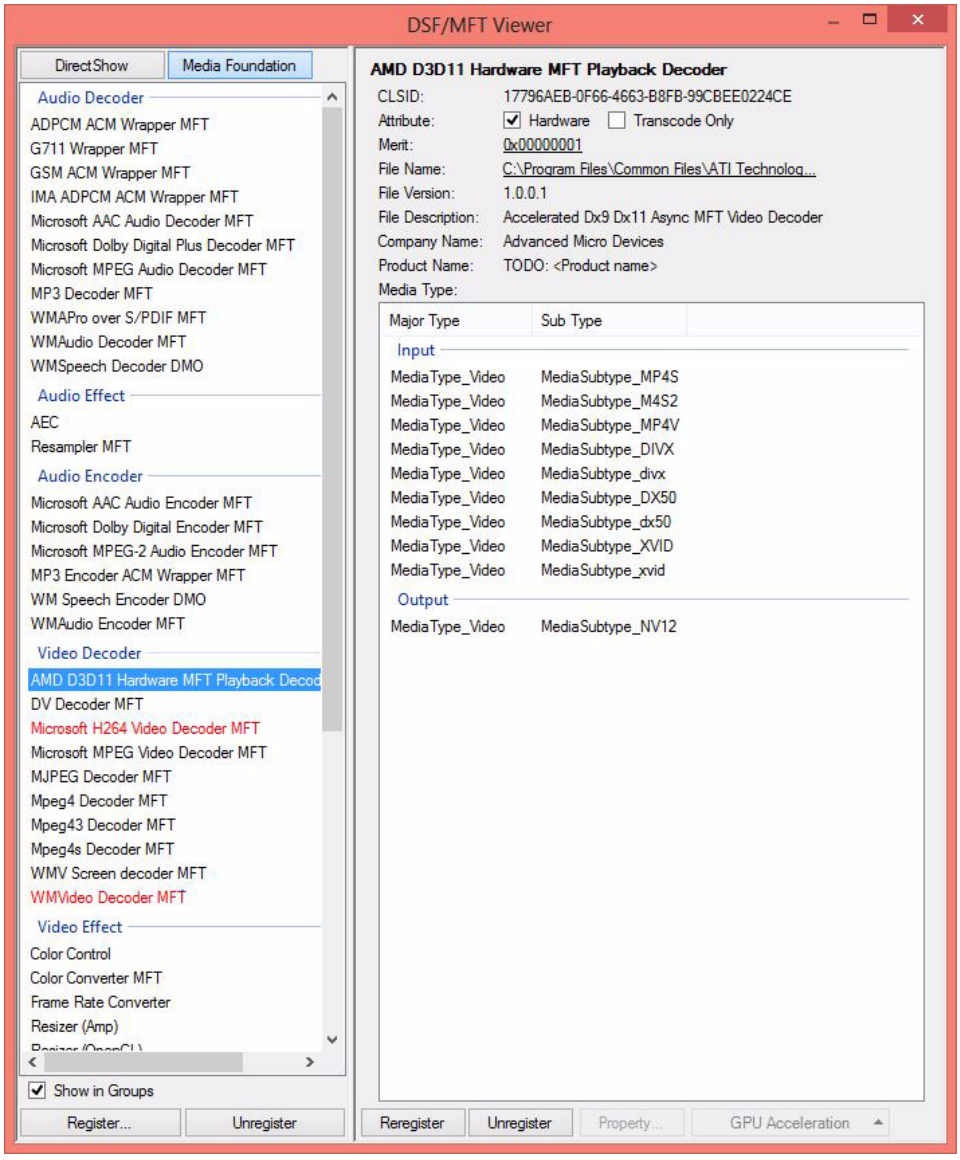


Figure 9.4 List of Available Media Foundation Components

Chapter 10
Programming with OpenCL

For information about programming applications using OpenCL, see the following links:

http://developer.amd.com/resources/heterogeneous-computing/opencl-zone/programming-in-opencl/

http://developer.amd.com/resources/heterogeneous-computing/opencl-zone/programming-in-opencl/introductory- exercises-and-tutorials/

For information on using OpenCL for heterogeneous computing, see "Heterogeneous Computing with OpenCL by Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry and Dana Schaa": http://developer.amd.com/partners/university-programs/heterogeneous-computing-with-opencl/.