**文章目录**

* 一、ffmpeg使用dxva2
* 二、解码
* 三、渲染

**前言**

ffplay本身是支持设置解码器的，比如设置h264\_qsv、hevc\_cuvid等就可以使用硬解功能，实际测试确实是有效的，cpu使用率也是有所下降。但是这并不是最佳的方案，在Windows上更好的方案是使用dxva2解码然后使用d3d9渲染，这种方法不仅极大降低cpu使用率、gpu使用率也有所下降、同时解码速度也比较快。但是ffplay本身是不支持使用dxva2的，所以这个时候就要我们进行拓展了。

**一、ffmpeg使用dxva2**

dxva2解码渲染包含2个步骤：解码和渲染。之所以是很优的方案是因为，解码和渲染都是显卡中处理，解码的数据不需要取出到内存，直接在显存转换然后渲染。ffmpeg有包含dxva2的示例代码，但没有显卡渲染功能，性能还是和设置解码器没有区别。我们需要参考的是项目名称叫Win32Project1的ffmpeg\_dxva2解码渲染的博文（暂时没找到当时那篇，就不贴其他类似链接了）

头文件如下:

/\*

\* This file is part of FFmpeg.

\*

\* FFmpeg is free software; you can redistribute it and/or

\* modify it under the terms of the GNU Lesser General Public

\* License as published by the Free Software Foundation; either

\* version 2.1 of the License, or (at your option) any later version.

\*

\* FFmpeg is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

\* Lesser General Public License for more details.

\*

\* You should have received a copy of the GNU Lesser General Public

\* License along with FFmpeg; if not, write to the Free Software

\* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

\*/

#ifndef FFMPEG\_DXVA2\_H

#define FFMPEG\_DXVA2\_H

//#include "windows.h"

extern "C"{

#include "libavcodec/avcodec.h"

#include "libavutil/pixfmt.h"

#include "libavutil/rational.h"

}

enum HWAccelID {

HWACCEL\_NONE = 0,

HWACCEL\_AUTO,

HWACCEL\_VDPAU,

HWACCEL\_DXVA2,

HWACCEL\_VDA,

HWACCEL\_VIDEOTOOLBOX,

HWACCEL\_QSV,

};

typedef struct AVStream AVStream;

typedef struct AVCodecContext AVCodecContext;

typedef struct AVCodec AVCodec;

typedef struct AVFrame AVFrame;

typedef struct AVDictionary AVDictionary;

typedef struct InputStream {

int file\_index;

AVStream \*st;

int discard; /\* true if stream data should be discarded \*/

int user\_set\_discard;

int decoding\_needed; /\* non zero if the packets must be decoded in 'raw\_fifo', see DECODING\_FOR\_\* \*/

#define DECODING\_FOR\_OST 1

#define DECODING\_FOR\_FILTER 2

AVCodecContext \*dec\_ctx;

AVCodec \*dec;

AVFrame \*decoded\_frame;

AVFrame \*filter\_frame; /\* a ref of decoded\_frame, to be sent to filters \*/

int64\_t start; /\* time when read started \*/

/\* predicted dts of the next packet read for this stream or (when there are

\* several frames in a packet) of the next frame in current packet (in AV\_TIME\_BASE units) \*/

int64\_t next\_dts;

int64\_t dts; ///< dts of the last packet read for this stream (in AV\_TIME\_BASE units)

int64\_t next\_pts; ///< synthetic pts for the next decode frame (in AV\_TIME\_BASE units)

int64\_t pts; ///< current pts of the decoded frame (in AV\_TIME\_BASE units)

int wrap\_correction\_done;

int64\_t filter\_in\_rescale\_delta\_last;

int64\_t min\_pts; /\* pts with the smallest value in a current stream \*/

int64\_t max\_pts; /\* pts with the higher value in a current stream \*/

int64\_t nb\_samples; /\* number of samples in the last decoded audio frame before looping \*/

double ts\_scale;

int saw\_first\_ts;

int showed\_multi\_packet\_warning;

AVDictionary \*decoder\_opts;

AVRational framerate; /\* framerate forced with -r \*/

int top\_field\_first;

int guess\_layout\_max;

int autorotate;

int resample\_height;

int resample\_width;

int resample\_pix\_fmt;

int resample\_sample\_fmt;

int resample\_sample\_rate;

int resample\_channels;

uint64\_t resample\_channel\_layout;

int fix\_sub\_duration;

struct { /\* previous decoded subtitle and related variables \*/

int got\_output;

int ret;

AVSubtitle subtitle;

} prev\_sub;

struct sub2video {

int64\_t last\_pts;

int64\_t end\_pts;

AVFrame \*frame;

int w, h;

} sub2video;

int dr1;

/\* decoded data from this stream goes into all those filters

\* currently video and audio only \*/

//InputFilter \*\*filters;

//int nb\_filters;

//int reinit\_filters;

/\* hwaccel options \*/

enum HWAccelID hwaccel\_id;

char \*hwaccel\_device;

/\* hwaccel context \*/

enum HWAccelID active\_hwaccel\_id;

void \*hwaccel\_ctx;

void(\*hwaccel\_uninit)(AVCodecContext \*s);

int(\*hwaccel\_get\_buffer)(AVCodecContext \*s, AVFrame \*frame, int flags);

int(\*hwaccel\_retrieve\_data)(AVCodecContext \*s, AVFrame \*frame);

enum AVPixelFormat hwaccel\_pix\_fmt;

enum AVPixelFormat hwaccel\_retrieved\_pix\_fmt;

/\* stats \*/

// combined size of all the packets read

uint64\_t data\_size;

/\* number of packets successfully read for this stream \*/

uint64\_t nb\_packets;

// number of frames/samples retrieved from the decoder

uint64\_t frames\_decoded;

uint64\_t samples\_decoded;

} InputStream;

int dxva2\_init(AVCodecContext \*s, HWND hwnd);

int dxva2\_retrieve\_data\_call(AVCodecContext \*s, AVFrame \*frame);

#endif /\* FFMPEG\_DXVA2\_H \*/

**二、解码**

修改ffplay解码功能需要在stream\_component\_open中进行：

**1、添加字段**

引用Win32Project1的ffmpeg\_dxva2.h头文件

#include "ffmpeg\_dxva2.h"

定义一个枚举说明硬件加速类型

/// <summary>

/// 硬件加速选项

/// </summary>

typedef enum

{

AC\_HARDWAREACCELERATETYPE\_DISABLED,

AC\_HARDWAREACCELERATETYPE\_AUTO,

//使用dxva解码,仅在Windows有效,成功启动：started、display事件的pixformat为AC\_PIXELFORMAT\_DXVA2\_VLD，render事件的data[3]为d3d9的surface对象。

AC\_HARDWAREACCELERATETYPE\_DXVA

}ACHardwareAccelerateType;

在VideoState中添加如下字段硬件加速类型，以及Win32Project1的InputStream对象

ACHardwareAccelerateType hwaccel;

InputStream\* ist;

添加相应接口

//设置硬件加速类型

void ac\_play\_setHardwareAccelerateType(ACPlay play, ACHardwareAccelerateType value) {

VideoState\* s = (VideoState\*)play;

s->hwaccel = value;

}

**2、初始化**  
在stream\_component\_open的avcodec\_open2上一行，加入判断hwaccel初始化dxva逻辑。dxva2\_init就是Win32Project1中的方法，此方法一定要有hwnd，这个hwnd必须是渲染窗口的。如果不想设置hwnd达到相同性能则需要另外做修改，本文就不深入讨论了。

if (is->hwaccel == AC\_HARDWAREACCELERATETYPE\_AUTO || is->hwaccel == AC\_HARDWAREACCELERATETYPE\_DXVA)

{

switch (codec->id)

//dxva2支持的格式

{

case AV\_CODEC\_ID\_MPEG2VIDEO:

case AV\_CODEC\_ID\_H264:

case AV\_CODEC\_ID\_VC1:

case AV\_CODEC\_ID\_WMV3:

case AV\_CODEC\_ID\_HEVC:

case AV\_CODEC\_ID\_VP9:

//while (1)

{

avctx->thread\_count = 1; // Multithreading is apparently not compatible with hardware decoding

is->ist = av\_mallocz(sizeof(InputStream));

is->ist->hwaccel\_id = HWACCEL\_AUTO;

is->ist->active\_hwaccel\_id = HWACCEL\_AUTO;

is->ist->hwaccel\_device = "dxva2";

is->ist->dec = codec;

is->ist->dec\_ctx = avctx;

avctx->opaque = is->ist;

if (dxva2\_init(avctx, is->hwnd) == 0)

{

avctx->get\_buffer2 = is->ist->hwaccel\_get\_buffer;

avctx->get\_format = GetHwFormat;

avctx->thread\_safe\_callbacks = 1;

avctx->pix\_fmt = AV\_PIX\_FMT\_DXVA2\_VLD;

}

else

{

av\_free(is->ist);

is->ist = NULL;

}

}

break;

}

}

将解码的avframe的格式设置为AV\_PIX\_FMT\_DXVA2\_VLD，上述代码中的GetHwFormat具体如下：

static enum AVPixelFormat GetHwFormat(AVCodecContext\* s, const enum AVPixelFormat\* pix\_fmts)

{

InputStream\* ist = (InputStream\*)s->opaque;

ist->active\_hwaccel\_id = HWACCEL\_DXVA2;

ist->hwaccel\_pix\_fmt = AV\_PIX\_FMT\_DXVA2\_VLD;

return ist->hwaccel\_pix\_fmt;

}

**3、反初始化**

在stream\_close中加入如下反初始化代码，其中dxva2\_uninit2是Win32Project1中的dxva2\_uninit将其参数类型改为了InputStream\*。

if (is->ist)

{

dxva2\_uninit2(is->ist);

av\_free(is->ist);

is->ist = NULL;

}

**三、渲染**

有了上述的解码设置之后，解码出来的数据将是d3d9的surface，这个对象在avframe.data[3]中，我们需要对它进行处理，将其显示到界面上。幸运的是Win32Project1包含了这部分功能，我们只需要调用方法就可以了。  
在video\_display的SDL\_RenderClear上一行加入如下代码。

Frame\* vp;

vp = frame\_queue\_peek\_last(&is->pictq);

if (vp->format == AV\_PIX\_FMT\_DXVA2\_VLD)

{

dxva2\_retrieve\_data\_call(is->viddec.avctx, vp->frame);

return;

}

**总结**

以上就是今天要讲的内容，通过上述方法实现的dxva2硬解渲染性能非常好，直观的感受就是渲染4k视频cpu使用率不超过1%,当然此时gpu使用率可能是50%左右的，但是设置硬解编码器的效果就是10%的cpu使用率，gpu使用率也是50%左右。原因在Win32Project1\_ffmpeg\_dxva2的博文中有说明，这里就不重复了。总的来说，ffplay支持xva2硬解渲染后就有更广泛和实际的应用了，比如直接解决了实时流多路渲染性能不足的问题等。

原文链接：https://blog.csdn.net/u013113678/article/details/124773187