Free DCP Player

Image processing
Multithreaded
Cross-platform (windows, Linux)
CPU and GPU

Johel Mitéran - 2022

Digital Cinema Package

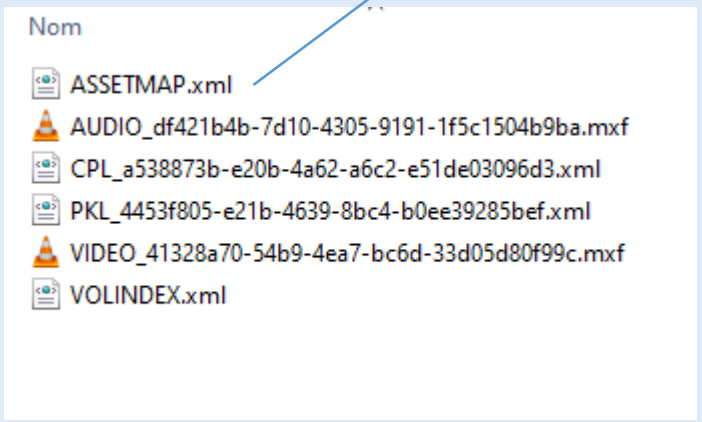
- A DCP is a set of files allowing the distribution of movies in cinemas.
- The DCP meets strict so-called INN standards, with many variants
- A DCP is a set of XML files (text with tags) that define the video, audio and subtitle files that will be broadcast (it is a kind of playlist)
- A DCP player is an equivalent of VLC or other media player, dedicated to the cinema standard.
- Difficulty: the videos are suites of compressed images in jpeg2000 format, very powerful, but complex to decode. To decode 2K images in real time (i.e. in less than 10 to 15 ms), it is essential to perform it on a dedicated component or GPU.

Digital Cinema Package

- Cinema projectors are equipped with ASICs dedicated to jpeg2000 real-time decompression (high cost)
- There are fast-paced GPU-accelerated paid solutions and slow, free CPU-based solutions like Dcp-o-matic.
- Free Dcp Player is a free alternative solution, based on a free library provided by Nvidia, allowing fast decoding of jpeg2000 on GPU. It allows a low-cost preview on PC, before a test in the cinema.

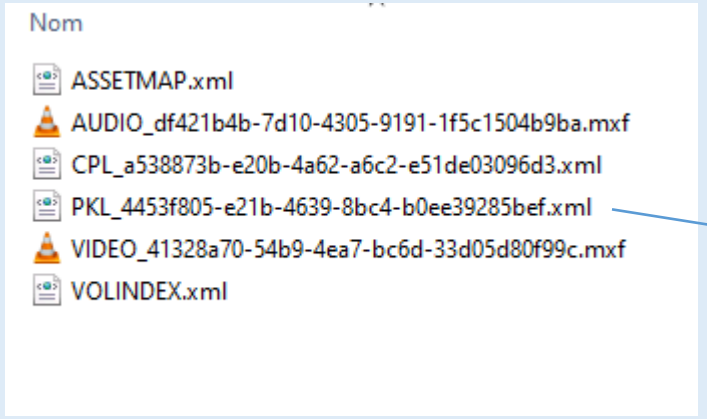
Digital Cinema Package

A DCP is a folder containing
multiple XML files (text)
And video/audio/subtitle/font files.
It can contain subfolders
The base file is ASSETMAP (Interop standard) or ASSETMAP.xml (SMPTE standard)




```
<?xml version="1.0" encoding="UTF-8"?>
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:8e6fdc42-61f6-4b83-81b2-4d7c0fe9ba4f</Id>
  <AnnotationText>CROQUIS_SHR-5-25_S-239_FR_20_2K</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.3.1.0005</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-01-13T17:20:15</IssueDate>
  <Issuer>KARLEENER</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:41328a70-54b9-4ea7-bc6d-33d05d80f99c</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_41328a70-54b9-4ea7-bc6d-33d05d80f99c.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>286503882</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:df421b4b-7d10-4305-9191-1f5c1504b9ba</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_df421b4b-7d10-4305-9191-1f5c1504b9ba.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>13472162</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:a538873b-e20b-4a62-a6c2-e51de03096d3</Id>
      <ChunkList>
        <Chunk>
          <Path>CPL_a538873b-e20b-4a62-a6c2-e51de03096d3.xml</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>1772</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:4453f805-e21b-4639-8bc4-b0ee39285bef</Id>
      <PackingList>true</PackingList>
      <ChunkList>
        <Chunk>
          <Path>PKL_4453f805-e21b-4639-8bc4-b0ee39285bef.xml</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>1516</Length>
        </Chunk>
      </ChunkList>
    </Asset>
  </AssetList>
</AssetMap>
```

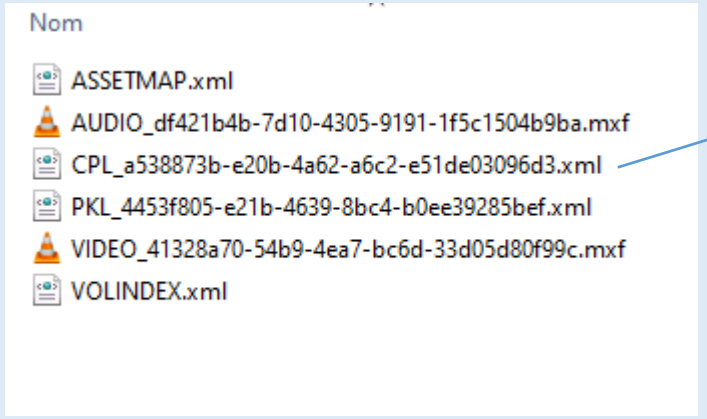
Digital Cinema Package



The "Packing List" file contains the names of the different files that make up the movies.
Its name begins with PKL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PackingList xmlns="http://www.smpte-ra.org/schemas/429-8/2007/PKL">
  <Id>urn:uuid:4453f805-e21b-4639-8bc4-b0ee39285bef</Id>
  <AnnotationText>CROQUIS_SHR-5-25_S-239_FR_20_2K</AnnotationText>
  <IssueDate>2022-01-13T17:20:15</IssueDate>
  <Issuer>KARLEENER</Issuer>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.3.1.0005</Creator>
  <AssetList>
    <Asset>
      <Id>urn:uuid:41328a70-54b9-4ea7-bc6d-33d05d80f99c</Id>
      <AnnotationText>VIDEO_41328a70-54b9-4ea7-bc6d-33d05d80f99c.mxf</AnnotationText>
      <Hash>JnDbYaJLUXul9X9/npSvKWX7ug4=</Hash>
      <Size>286503882</Size>
      <Type>application/mxf</Type>
      <OriginalFileName>VIDEO_41328a70-54b9-4ea7-bc6d-33d05d80f99c.mxf</OriginalFileName>
    </Asset>
    <Asset>
      <Id>urn:uuid:df421b4b-7d10-4305-9191-1f5c1504b9ba</Id>
      <AnnotationText>AUDIO_df421b4b-7d10-4305-9191-1f5c1504b9ba.mxf</AnnotationText>
      <Hash>jTcWJ2im47ZoK7/8hSIw4kBcoFs=</Hash>
      <Size>13472162</Size>
      <Type>application/mxf</Type>
      <OriginalFileName>AUDIO_df421b4b-7d10-4305-9191-1f5c1504b9ba.mxf</OriginalFileName>
    </Asset>
    <Asset>
      <Id>urn:uuid:a538873b-e20b-4a62-a6c2-e51de03096d3</Id>
      <AnnotationText>CROQUIS_SHR-5-25_S-239_FR_20_2K</AnnotationText>
      <Hash>uDYfXUHBlZlHNYGM+bEqtAxwMWg=</Hash>
      <Size>1772</Size>
      <Type>text/xml</Type>
      <OriginalFileName>CPL_a538873b-e20b-4a62-a6c2-e51de03096d3.xml</OriginalFileName>
    </Asset>
  </AssetList>
</PackingList>
```

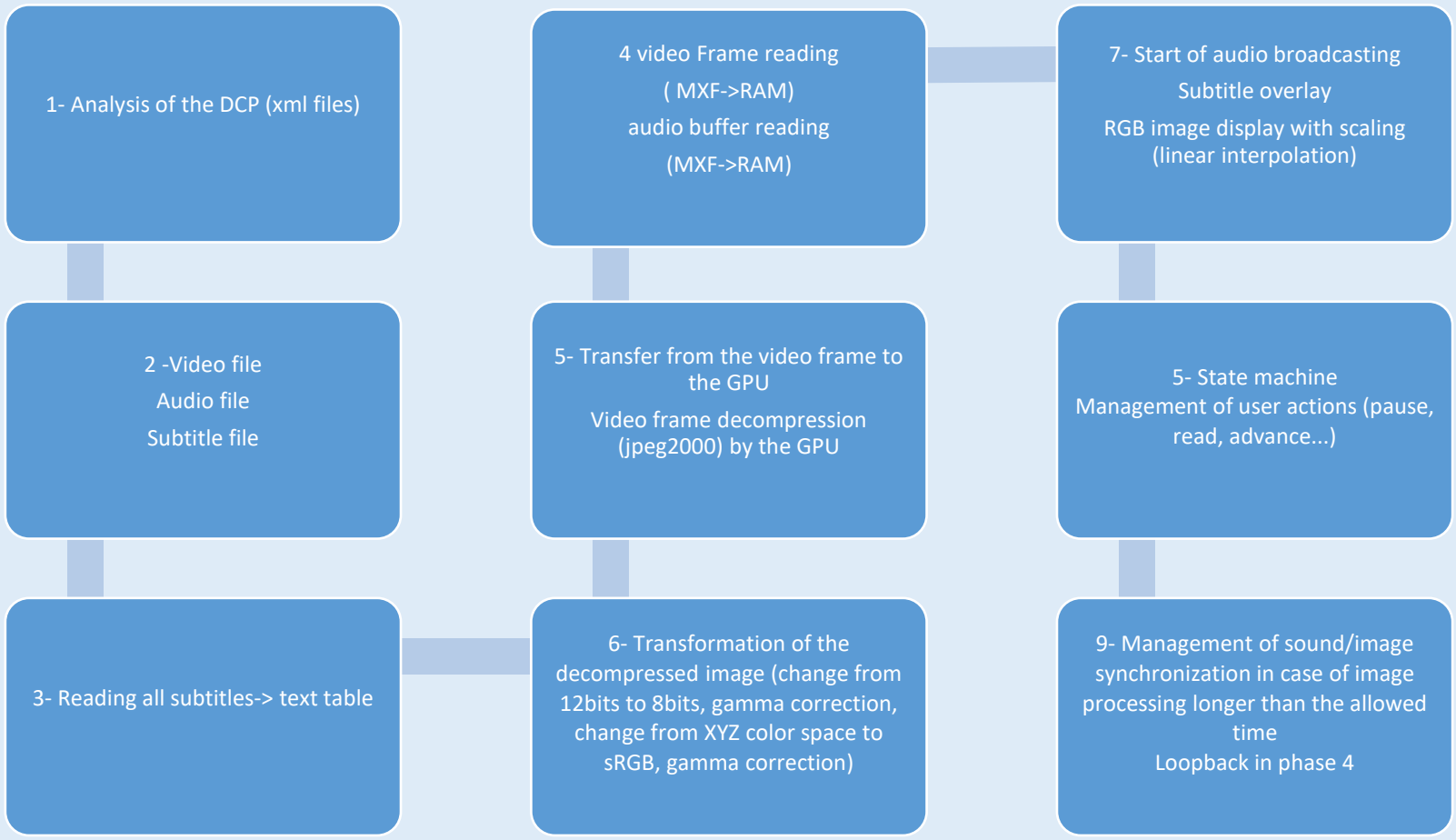
Digital Cinema Package




```
<?xml version="1.0" encoding="UTF-8"?>
<CompositionPlaylist xmlns="http://www.smpte-ra.org/schemas/429-7/2006/CPL">
  <Id>urn:uuid:a538873b-e20b-4a62-a6c2-e51de03096d3</Id>
  <AnnotationText>CROQUIS_SHR-5-25_S-239_FR_20_2K</AnnotationText>
  <IssueDate>2022-01-13T17:20:15</IssueDate>
  <Issuer>KARLEENER</Issuer>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.3.1.0005</Creator>
  <ContentTitleText>CROQUIS_SHR-5-25_S-239_FR_20_2K</ContentTitleText>
  <ContentKind>short</ContentKind>
  <ContentVersion>
    <Id>urn:uuid:fd93fbee-69d7-4203-ad90-a744a2303a74</Id>
    <LabelText>5</LabelText>
  </ContentVersion>
  <RatingList/>
  <ReelList>
    <Reel>
      <Id>urn:uuid:f1cfb7e1-8539-4414-a7aa-17e7dd79715b</Id>
      <AnnotationText>Reel-1</AnnotationText>
      <AssetList>
        <MainPicture>
          <Id>urn:uuid:41328a70-54b9-4ea7-bc6d-33d05d80f99c</Id>
          <AnnotationText>VIDEO_41328a70-54b9-4ea7-bc6d-33d05d80f99c.mxf</AnnotationText>
          <EditRate>25 1</EditRate>
          <IntrinsicDuration>382</IntrinsicDuration>
          <EntryPoint>0</EntryPoint>
          <Duration>382</Duration>
          <Hash>JnDbYaJLUXul9X9/npSvKWX7ug4=</Hash>
          <FrameRate>25 1</FrameRate>
          <ScreenAspectRatio>2048 858</ScreenAspectRatio>
        </MainPicture>
        <MainSound>
          <Id>urn:uuid:df421b4b-7d10-4305-9191-1f5c1504b9ba</Id>
          <AnnotationText>AUDIO_df421b4b-7d10-4305-9191-1f5c1504b9ba.mxf</AnnotationText>
          <EditRate>25 1</EditRate>
          <IntrinsicDuration>382</IntrinsicDuration>
          <EntryPoint>0</EntryPoint>
          <Duration>382</Duration>
          <Hash>jTcWJ2im47ZoK7/8hSIw4kBcoFs=</Hash>
        </MainSound>
      </AssetList>
    </Reel>
  </ReelList>
</CompositionPlaylist>
```

A DCP can contain multiple CPL (Composition Play List)
Each CPL can contain several "Reel" or clip.
A DCP can be encrypted, this is the case for all commercial films.
To decrypt it, you need a key called KDM,
generated by the distributor of the film, valid for a screening in a given room, on a specific date.
This version of the program does not take into account encrypted DCPs.

Free Dcp Player – general principle



Libraries used: free and compatible Windows / Linux

- Parsing the DCP (XML file): pugixml for reading XML
- pugixml.org – Home
- Audio Management and Image Display: SDL2
- https://www.libsdl.org/
- Developed for games, it also allows keyboard/mouse management
- Font management (font) for subtitles: SDL_TTF
- Playback of MXF files (audio, video, subtitles): ASDCPLIB
- GitHub - cinecert/asdcplib: AS-DCP and AS-02 File Access Library
- Jpeg2000 decoding on GPU: nvjpeg2k by Nvidia
- https://docs.nvidia.com/cuda/nvjpeg2000/userguide.html
- General GUI: Wxwidgets
- https://www.wxwidgets.org/downloads/

Preparing the Development Environment (Windows)

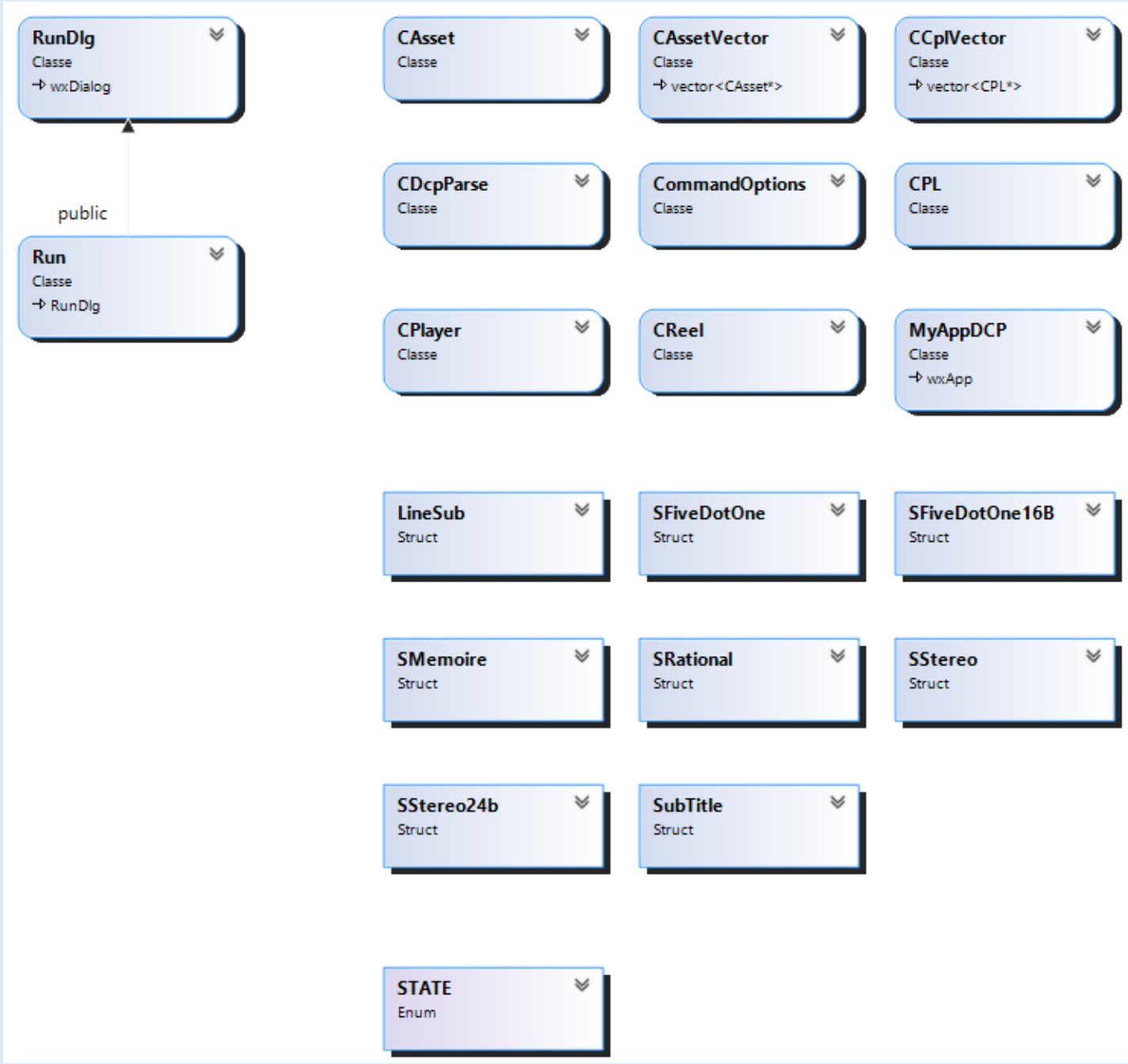
- Installation de Microsoft Visual Studio community
- Installing Nvidia Developer Tools (CUDA Toolkit 11.0 or later)
- Downloading the nvJPEG2000 library requires a free account at Nvidia
- https://developer.nvidia.com/nvjpeg/nvjpeg200-developer-survey
-
- To start and configure the basic project, it is advisable to start from an example provided by Nvidia, which has the advantage of configuring the Visual project for the use of Cuda
- In the Visual project, add the paths to the INCLUDE and LIB for all libraries used and for release and Debug modes
- Add the names of the libraries used (.lib extension file) in the link editor.
- During the first compilation, the Debug and Release directories will be created. Place the nvjpeg2k_0.dll file there. All other libraries will be statically linked to your executable, except those of the Wxwidgets that must also be put in these same folders.

Preparing the Development Environment (Windows)

- Download the ASDCPLIB library to a subfolder of your project.
- To compile it in Release and Debug mode, use Cmake in its version integrated with Visual (there is no project in the sense of Visual Studio). This creates .lib files that will be integrated into your executable.
- The .h files in this library must be accessible by your project (include path).

Program Structure

- The MyAppDCP class launches the program
- RunDlg and Run create and launch the interface (dialog box to choose the DCP/CPL to play, the output screen, the audio output, and some parameters).
- The freedcpplayer.cpp file contains the main_dcpplayer function that allows you to launch the player itself (independent of the graphical interface). This makes it easy to change the starting interface if necessary.
- The CDcpParse class allows deep analysis of XML files, to determine which video, audio, and subtitle files to play, as well as the fonts for these subtitles.



DCP parsing – CAsset and CAssetVector Classes

An Asset is one of the constituent elements of the DCP.
It can be a video, a sound, a subtitle
The CAssetVector vector will contain all the assets of the DCP.

```
class CAsset
{
public:

    CAsset() :
        bPackingList(false), uiSize(0),
        iDurationIntrisic(0), iEntryPoint(0), iDuration(0),uiLength(0),uiOffset(0),sID(""),sPath(""),
        sAnnotation(""),sTypeAsset(""),sFileName(""),sEditRate(""),iVol_Index(0),sKeyID(""),sFrameRate(""),sScreenAspectRatio(""),sFont("") {}
    ~CAsset() {};
    string      sID;
    string      sPath;
    string      sAnnotation;
    string      sHash;
    string      sKeyID;
    string      sFrameRate;
    string      sScreenAspectRatio;
    string      sLanguage;
    string      sFont;
    string      sTypeAsset;
    string      sFileName;
    string      sEditRate;
    int         iDurationIntrisic;
    int         iEntryPoint;
    int         iDuration;
    int         iVol_Index;
    uint32_t    uiOffset;
    uint32_t    uiLength;
    uint32_t    uiSize;
    bool        bPackingList;
    double       dFrameRate;
    void        LogAll();
};

class CAssetVector : public std::vector<CAsset*> {};
```

DCP parsing– Classes

CReel
CPL
CCplVector

The DCP to be read is a vector of CPL
Each CPL contains a vector of CReel
Each CReel contains 3 pointers
- ptrMainPicture
- ptrMainSound
- ptrSubtitle

```
class CReel
{
public:
    CReel()
        :ptrMainPicture(NULL), ptrMainSound(NULL), ptrSubtitle(NULL)
    {};

    string sID;
    string sAnnotation;
    CAsset* ptrMainPicture;
    CAsset* ptrMainSound;
    CAsset* ptrSubtitle;

};

class CPL
{
public:
    CPL() {};
    string sID;
    string sAnnotation;
    string sIcon_Id;
    string sIssueDate;
    string sIssuer;
    string sCreator;
    string sContentTitle;
    string sContentKind;
    std::vector<CReel*>   VecReel;
};

class CCplVector : public std::vector<CPL*> {};
```

DCPparsing
CDcpParse
(ASSETMAP files.XML)

```
void CDcpParse::ParseDCP(vector<string>& MxfFiles, string MyPath)
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

```
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:2213857f-d187-4cf3-b027-a609f3aeb1ec</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>313705658</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>3159142</Length>
        </Chunk>
      </ChunkList>
    </Asset>
```

DCP parsing

```
void CDcpParse::ParseDCP(vector<string>& MxfFiles, string MyPath)
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

```
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:2213857f-d187-4cf3-b027-a609f3aeb1ec</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>313705658</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>3159142</Length>
        </Chunk>
      </ChunkList>
    </Asset>
```

DCP parsing

```
void CDcpParse::ParseDCP(vector<string>& MxfFiles, string MyPath)
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

```
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:2213857f-d187-4cf3-b027-a609f3aeb1ec</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>313705658</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>3159142</Length>
        </Chunk>
      </ChunkList>
    </Asset>
```

DCP parsing

```
void CDcpParse::ParseDCP(vector<string>& MxfFiles, string MyPath)
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

```
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:2213857f-d187-4cf3-b027-a609f3aeb1ec</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>313705658</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>3159142</Length>
        </Chunk>
      </ChunkList>
    </Asset>
```

DCP parsing

```
void CDcpParse::ParseDCP(vector<string>& MxfFiles, string MyPath)
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

```
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:2213857f-d187-4cf3-b027-a609f3aeb1ec</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>313705658</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>3159142</Length>
        </Chunk>
      </ChunkList>
    </Asset>
```

DCP parsing

```
    <Id>urn:uuid:0a9b092c-57d6-4a45-b2c3-f010554f3f4e</Id>
    <ChunkList>
      <Chunk>
        <Path>CPL_0a9b092c-57d6-4a45-b2c3-f010554f3f4e.xml</Path>
        <VolumeIndex>1</VolumeIndex>
        <Offset>0</Offset>
        <Length>1788</Length>
      </Chunk>
    </ChunkList>
  </Asset>
  <Asset>
    <Id>urn:uuid:427ddd92-01ba-48aa-8aed-af4901f60974</Id>
    <PackingList>true</PackingList>
    <ChunkList>
      <Chunk>
        <Path>PKL_427ddd92-01ba-48aa-8aed-af4901f60974.xml</Path>
        <VolumeIndex>1</VolumeIndex>
        <Offset>0</Offset>
        <Length>1489</Length>
      </Chunk>
    </ChunkList>
  </Asset>
</AssetList>
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

DCP parsing

```
void CDcpParse::ParseDCP(vector<string>& MxfFiles, string MyPath)
```

```xml
<AssetMap xmlns="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <Id>urn:uuid:2213857f-d187-4cf3-b027-a609f3aeb1ec</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <VolumeCount>1</VolumeCount>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <ChunkList>
        <Chunk>
          <Path>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>313705658</Length>
        </Chunk>
      </ChunkList>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <ChunkList>
        <Chunk>
          <Path>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</Path>
          <VolumeIndex>1</VolumeIndex>
          <Offset>0</Offset>
          <Length>3159142</Length>
        </Chunk>
      </ChunkList>
    </Asset>
```

```
pugi::xml_node Asset = doc.child("AssetMap").child("AssetList").child("Asset");

if (res)
{
    pugi::xml_node assets = doc.child("AssetMap").child("AssetList");
    for (pugi::xml_node asset = assets.first_child(); asset; asset = asset.next_sibling())
    {
        CAsset* OneAsset = new CAsset();
        for (pugi::xml_node attr = asset.first_child(); attr; attr = attr.next_sibling())
        {
            //std::cout << " " << attr.name() << "=" << attr.value()<<endl;
            string AssetNodeName = attr.name();
            if (AssetNodeName == "Id")
            {
                OneAsset->sID = attr.first_child().value();
            }
            if (AssetNodeName == "PackingList")
            {
                string v = attr.first_child().value();
                if (v == "true") OneAsset->bPackingList; else OneAsset->bPackingList = false;
            }
            if (AssetNodeName == "ChunkList")
            {
                for (pugi::xml_node chunk = attr.first_child(); chunk; chunk = chunk.next_sibling())
                {

                    for (pugi::xml_node ChunkNode = chunk.first_child(); ChunkNode; ChunkNode = ChunkNode.next_sibling())
                    {
                        string cattr = ChunkNode.first_child().value();
                        string ChunkNodeName(ChunkNode.name());
                        if (ChunkNodeName == "Path")     OneAsset->sPath = cattr;
                        if (ChunkNodeName == "VolumeIndex") OneAsset->iVol_Index = atoi(cattr.c_str());
                        if (ChunkNodeName == "Offset")   OneAsset->uiOffset = atoi(cattr.c_str());
                        if (ChunkNodeName == "Length")   OneAsset->uiLength = atoi(cattr.c_str());
                        //if ()

                    }
                }
            }
        }
        AssetVector.push_back(OneAsset);
    }
}
```

DCP parsing– PKL files – can contain TTF files (fonts for subtitles)

```
void CDcpParse::ParsePKL(CAsset* OneAsset, string MyPath)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<PackingList xmlns="http://www.smpte-ra.org/schemas/429-8/2007/PKL">
  <Id>urn:uuid:427ddd92-01ba-48aa-8aed-af4901f60974</Id>
  <AnnotationText>accords nomade</AnnotationText>
  <IssueDate>2022-04-27T15:21:46</IssueDate>
  <Issuer>Blackmagic Design</Issuer>
  <Creator>Blackmagic Design DaVinci Resolve Studio 17.4.6.0004</Creator>
  <AssetList>
    <Asset>
      <Id>urn:uuid:2011ca17-662a-4f4d-8a01-778b00f2ac5e</Id>
      <AnnotationText>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</AnnotationText>
      <Hash>ZHpPWt5ipoVIYVjMPOX+e2ks+No=</Hash>
      <Size>313705658</Size>
      <Type>application/mxf</Type>
      <OriginalFileName>VIDEO_2011ca17-662a-4f4d-8a01-778b00f2ac5e.mxf</OriginalFileName>
    </Asset>
    <Asset>
      <Id>urn:uuid:bed40ed8-8aa6-4db7-a8b6-962f4df53b79</Id>
      <AnnotationText>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</AnnotationText>
      <Hash>ou2cQNzeaHWMBg/7OLdweyfbLKo=</Hash>
      <Size>3159142</Size>
      <Type>application/mxf</Type>
      <OriginalFileName>AUDIO_bed40ed8-8aa6-4db7-a8b6-962f4df53b79.mxf</OriginalFileName>
    </Asset>
    <Asset>
      <Id>urn:uuid:0a9b092c-57d6-4a45-b2c3-f010554f3f4e</Id>
      <AnnotationText>accords nomade</AnnotationText>
      <Hash>DDWDRkpWsZBAc43x5Ga6IfS+XwE=</Hash>
      <Size>1788</Size>
      <Type>text/xml</Type>
      <OriginalFileName>CPL_0a9b092c-57d6-4a45-b2c3-f010554f3f4e.xml</OriginalFileName>
    </Asset>
  </AssetList>
</PackingList>
```

DCP parsing –CPL files

```
void CDcpParse::ParseCPL(CAsset* OneAsset,  string MyPath)
```

```
<ReelList>
  <Reel>
    <Id>urn:uuid:8d4b50f2-15b9-4fc1-81a7-bf16fa574b75</Id>
    <AssetList>
      <MainPicture>
        <Id>urn:uuid:98f01473-cef5-4226-b1cb-e4c79c814f89</Id>
        <AnnotationText>j2c_98f01473-cef5-4226-b1cb-e4c79c814f89.mxf</AnnotationText>
        <EditRate>25 1</EditRate>
        <IntrinsicDuration>1487</IntrinsicDuration>
        <EntryPoint>0</EntryPoint>
        <Duration>1487</Duration>
        <Hash>4/OLrEP1nuVOl1hT01gduYnRWGw=</Hash>
        <FrameRate>25 1</FrameRate>
        <ScreenAspectRatio>2048 858</ScreenAspectRati
      </MainPicture>
      <MainSound>
        <Id>urn:uuid:cf3b2443-207f-4602-9d23-b37c0140
        <AnnotationText>pcm_cf3b2443-207f-4602-9d23-b
        <EditRate>25 1</EditRate>
        <IntrinsicDuration>1487</IntrinsicDuration>
        <EntryPoint>0</EntryPoint>
        <Duration>1487</Duration>
        <Hash>r6AHIYLRzjSh9oYvyxSJNR2Xseo=</Hash>
      </MainSound>
      <MainSubtitle>
        <Id>urn:uuid:2b69d715-27e9-412f-ae51-6f1622a5
        <AnnotationText>sub_2b69d715-27e9-412f-ae51-6
        <EditRate>25 1</EditRate>
        <IntrinsicDuration>1487</IntrinsicDuration>
        <EntryPoint>0</EntryPoint>
        <Duration>1487</Duration>
        <Hash>aKq3Tzqu9zG+HOpdnF5sO5ttA7Q=</Hash>
      </MainSubtitle>
    </AssetList>
  </Reel>
</ReelList>
```

```
CReel* OneReel = new CReel();
//MainPicture
pugi::xml_node MainPicture = reel.child("AssetList").child("MainPicture");
if (MainPicture)
{
    string mpid = MainPicture.child("Id").first_child().value();
    ki = -1;
    for (int i = 0; i < AssetVector.size(); i++)
    {
        if (mpid == AssetVector[i]->sID) {
            ki = i; break;
        }
    }
    if (ki != -1)
    {
        AssetVector[ki]->sEditRate = MainPicture.child("EditRate").first_child().value();
        AssetVector[ki]->iDurationIntrisic = atoi(MainPicture.child("IntrinsicDuration").first_child().value());
        AssetVector[ki]->iEntryPoint = atoi(MainPicture.child("EntryPoint").first_child().value());
        AssetVector[ki]->iDuration = atoi(MainPicture.child("Duration").first_child().value());
        AssetVector[ki]->sFrameRate = MainPicture.child("FrameRate").first_child().value();
        SRational Fr;
        if( DecodeRational(AssetVector[ki]->sFrameRate.c_str(), Fr)) AssetVector[ki]->dFrameRate=double(Fr.Numerator)/ double(Fr.Denominator);
        AssetVector[ki]->sScreenAspectRatio = MainPicture.child("ScreenAspectRatio").first_child().value();
        string Annotation = MainPicture.child("AnnotationText").first_child().value();
        if (Annotation != "") AssetVector[ki]->sAnnotation = Annotation;

        OneReel->ptrMainPicture = AssetVector[ki];
        VideoOk = true;
    }

}
```

SDL: Create a window and display an image

The basic tools are

- Window
- Renderer (final rendering tool)
- Surface (image RAM CPU)
- Texture (image RAM GPU)

Advantage: accelerated via OPENGL, compatible windows, Linux, Mac

- SDL_CreateWindowAndRenderer - SDL Wiki (libsdl.org)

SDL: Create a window and display an image

```c
#include "SDL.h"

int main(int argc, char *argv[])
{
    SDL_Window *window;
    SDL_Renderer *renderer;
    SDL_Surface *surface;
    SDL_Texture *texture;
    SDL_Event event;

    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't initialize SDL: %s", SDL_GetError());
        return 3;
    }

    if (SDL_CreateWindowAndRenderer(320, 240, SDL_WINDOW_RESIZABLE, &window, &renderer)) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create window and renderer: %s", SDL_GetError());
        return 3;
    }

    surface = SDL_LoadBMP("sample.bmp");
    if (!surface) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create surface from image: %s", SDL_GetError());
        return 3;
    }
    texture = SDL_CreateTextureFromSurface(renderer, surface);
    if (!texture) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create texture from surface: %s", SDL_GetError());
        return 3;
    }
    SDL_FreeSurface(surface);
```

SDL: Create a window and display an image

```
    while (1) {
        SDL_PollEvent(&event);
        if (event.type == SDL_QUIT) {
            break;
        }
        SDL_SetRenderDrawColor(renderer, 0x00, 0x00, 0x00, 0x00);
        SDL_RenderClear(renderer);
        SDL_RenderCopy(renderer, texture, NULL, NULL);
        SDL_RenderPresent(renderer);
    }

    SDL_DestroyTexture(texture);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);

    SDL_Quit();

    return 0;
}
```

SDL: Manage Sound

The basic tools are

- Open the audio device (sound card)
- Transmit sound to audio buffer
- Start playback
- Stop playback

Advantage: compatible with Windows, Linux, Mac

Disadvantage: does not support 24-bit sound, the standard for DCPs (only 16 or 32)

- SDL_OpenAudio - SDL Wiki (libsdl.org)

Program structure: CPlayer class

- The CPlayer class handles playback/decompression/display/synchronization.
- It contains mainly the following member functions
  - CPlayer::SelectAudioDeviceInitAudio()
  - CPlayer::win_init_render(int w, int h, SDL_Renderer** Renderer, bool BlackBackground = false, int NumDisplay = 0, bool FullScreen = false)
  - CPlayer::InitialisationReaders(CDcpParse &DcpParse, bool FirstTime, CReel *ptrReel_i)
  - CPlayer::InitialisationJ2K()
  - CPlayer::PrepareXYZ2RGBLUT()
  - CPlayer::Read_timed_text_file(const Kumu::IFileReaderFactory& fileReaderFactory, string inputFile, fs::path full_path)
  - CPlayer::PrepareFirstAudioBuffering(void *Param)
  - CPlayer::DecodeAndScreenFirstFrame(bool WaitAfterFirstFrame)
  - CPlayer::MainLoop(bool WaitAfterFirstFrame)
  - CPlayer::Synchronisation()
  - CPlayer::RenderImageWithSub(SDL_Renderer* Renderer, TTF_Font* Font, vector<SubTitle>& MySubTitles, int width, int height, vector<int>& IndiceSub, Uint32 i, SMemoire& Mem)
  - CPlayer::From51toStereo(const SFiveDotOne* GlobalBufferOneFrame, SStereo* AudioDeviceStereo, int NbSamples)
  - CPlayer::From51to51_16B(const SFiveDotOne* GlobalBufferOneFrame, SFiveDotOne16B* AudioDevice, int NbSamples)
  - CPlayer::FromStereotoStereo(const SStereo24b* GlobalBufferOneFrame, SStereo* AudioDeviceStereo, int NbSamples)
  - CPlayer::ThreadQuarter1(void* Param)
  - CPlayer::Swap(SDL_Surface* &out1, SDL_Surface* &out2)
  - CPlayer::EndAndClear(bool LastTime)

Calls to CPlayer functions in main_dcpplayer

Parsing XML files
Back: A CPL table
Each CPL contains a "reel" table

```
CDcpParse DcpParse(Options.verbose_flag);
string DcpPath = Options.input_filename;
DcpParse.ParseDCP(MxfFiles, Options.input_filename);
int CplIndex = Options.NumCpl;
int Totalduration = 0;
if (DcpParse.CplVector.size() <= CplIndex)
{
        fprintf(fp_log, "CPL index %d not found",CplIndex); IsPlaying = false; if (fp_log) fclose(fp_log); return 3;
}
if (!DcpParse.VideoOk || !DcpParse.SoundOk)
{
        fprintf(fp_log, "No Video found or no audio found"); IsPlaying = false; if (fp_log) fclose(fp_log); return 3;
}

for (int k = 0; k < DcpParse.CplVector[CplIndex]->VecReel.size(); k++)
{
        string Video = DcpParse.CplVector[CplIndex]->VecReel[k]->ptrMainPicture->sPath;
        Uint32 duration = DcpParse.CplVector[CplIndex]->VecReel[k]->ptrMainPicture->iDuration;
        Totalduration += duration;
}
vector<JP2K::MXFReader*> VectVideoReader;
VectVideoReader.resize(Totalduration);
CPlayer* pPlayer=NULL;
bool AudioSelectedOk=false;
bool WaitAfterFirstFrame=true;


pPlayer = new CPlayer(Options, defaultFactory, full_path);
AudioSelectedOk = pPlayer->SelectAudioDeviceInitAudio();
```

Calls to CPlayer functions in main_dcpplayer

```
CDcpParse DcpParse(Options.verbose_flag);
string DcpPath = Options.input_filename;
DcpParse.ParseDCP(MxfFiles, Options.input_filename);
int CplIndex = Options.NumCpl;
int Totalduration = 0;
if (DcpParse.CplVector.size() <= CplIndex)
{
        fprintf(fp_log, "CPL index %d not found",CplIndex); IsPlaying = false; if (fp_log) fclose(fp_log); return 3;
}
if (!DcpParse.VideoOk || !DcpParse.SoundOk)
{
        fprintf(fp_log, "No Video found or no audio found"); IsPlaying = false; if (fp_log) fclose(fp_log); return 3;
}

for (int k = 0; k < DcpParse.CplVector[CplIndex]->VecReel.size(); k++)
{
        string Video = DcpParse.CplVector[CplIndex]->VecReel[k]->ptrMainPicture->sPath;
        Uint32 duration = DcpParse.CplVector[CplIndex]->VecReel[k]->ptrMainPicture->iDuration;
        Totalduration += duration;
}
vector<JP2K::MXFReader*> VectVideoReader;
VectVideoReader.resize(Totalduration);
CPlayer* pPlayer=NULL;
bool AudioSelectedOk=false;
bool WaitAfterFirstFrame=true;


pPlayer = new CPlayer(Options, defaultFactory, full_path);
AudioSelectedOk = pPlayer->SelectAudioDeviceInitAudio();
```

Creating the "player", and selecting the audio device

Initializations for the first "reel"

Initializations for Jpeg2000 decoding

Main loop for reading the Real

```
if (AudioSelectedOk && DcpParse.VideoOk && DcpParse.SoundOk)
{
        // Process first reel
        Result_t Result = pPlayer->InitialisationReaders(DcpParse, true, DcpParse.CplVector[CplIndex]->VecReel[0]);
        if (ASDCP_SUCCESS(Result))      Result = pPlayer->InitialisationJ2K();
        else
        {
                IsPlaying = false;
                if (fp_log) fclose(fp_log);
                return RESULT_FAIL;
        }
        if (ASDCP_SUCCESS(Result))  Result = pPlayer->MainLoop(WaitAfterFirstFrame);
        else
        {
                IsPlaying = false;
                if (fp_log) fclose(fp_log);
                return RESULT_FAIL;
        }
        if (ASDCP_SUCCESS(Result))
        {
                if (pPlayer->OutEscape) pPlayer->EndAndClear(true);
                else
                {
                        if (DcpParse.CplVector[CplIndex]->VecReel.size() > 1) pPlayer->EndAndClear(false);
                        else pPlayer->EndAndClear(true);
                }
        }
        else
        {
                IsPlaying = false;
                if (fp_log) fclose(fp_log);
                return RESULT_FAIL;
        }
```

Reading the next "reels"

```
        for (int k = 1; k < DcpParse.CplVector[CplIndex]->VecReel.size() && !pPlayer->OutEscape; k++)
        {
                // compute global duration
                // for global navigation
                // to be added in future version
                //string Video = DcpParse.CplVector[ClpIndex]->VecReel[k]->ptrMainPicture->sPath;
                //JP2K::MXFReader *pReader = new JP2K::MXFReader(defaultFactory);
                //Result_t resultVideo = pReader->OpenRead(Video); // read video
                //Uint32 duration = DcpParse.CplVector[ClpIndex]->VecReel[k]->ptrMainPicture->iDuration;
                //for (int d = 0; d < duration; d++) VectVideoReader.push_back(pReader);

                Result_t Result = pPlayer->InitialisationReaders(DcpParse, false, DcpParse.CplVector[CplIndex]->VecReel[k]);
                if (ASDCP_SUCCESS(Result))      Result = pPlayer->InitialisationJ2K();
                else
                {
                        IsPlaying = false;
                        if (fp_log) fclose(fp_log);
                        return RESULT_FAIL;
                }
                if (ASDCP_SUCCESS(Result))  Result = pPlayer->MainLoop(false);
                else
                {
                        IsPlaying = false;
                        if (fp_log) fclose(fp_log);
                        return RESULT_FAIL;
                }
                if (k == DcpParse.CplVector[CplIndex]->VecReel.size() - 1) pPlayer->EndAndClear(true);
                else pPlayer->EndAndClear(false);
                if (pPlayer->OutEscape) break;
        }

} // if audio device is selected
delete pPlayer;
IsPlaying = false;
if (fp_log) fclose(fp_log);
return 0;
```

Error handling

```
for (int k = 1; k < DcpParse.CplVector[CplIndex]->VecReel.size() && !pPlayer->OutEscape; k++)
{
        // compute global duration
        // for global navigation
        // to be added in future version
        //string Video = DcpParse.CplVector[ClpIndex]->VecReel[k]->ptrMainPicture->sPath;
        //JP2K::MXFReader *pReader = new JP2K::MXFReader(defaultFactory);
        //Result_t resultVideo = pReader->OpenRead(Video); // read video
        //Uint32 duration = DcpParse.CplVector[ClpIndex]->VecReel[k]->ptrMainPicture->iDuration;
        //for (int d = 0; d < duration; d++) VectVideoReader.push_back(pReader);

        Result_t Result = pPlayer->InitialisationReaders(DcpParse, false, DcpParse.CplVector[CplIndex]->VecReel[k]);
        if (ASDCP_SUCCESS(Result))      Result = pPlayer->InitialisationJ2K();
        else
        {
                IsPlaying = false;
                if (fp_log) fclose(fp_log);
                return RESULT_FAIL;
        }
        if (ASDCP_SUCCESS(Result))  Result = pPlayer->MainLoop(false);
        else
        {
                IsPlaying = false;
                if (fp_log) fclose(fp_log);
                return RESULT_FAIL;
        }
        if (k == DcpParse.CplVector[CplIndex]->VecReel.size() - 1) pPlayer->EndAndClear(true);
        else pPlayer->EndAndClear(false);
        if (pPlayer->OutEscape) break;
}

} // if audio device is selected
delete pPlayer;
IsPlaying = false;
if (fp_log) fclose(fp_log);
return 0;
```

Error handlings (in asdcp.h et KM_error.h)

```
#define ASDCP_SUCCESS(v) (((v) < 0) ? 0 : 1)
#define ASDCP_FAILURE(v) (((v) < 0) ? 1 : 0)
```

```
class Result_t
  {
    int value;
    std::string label, symbol, message;
    Result_t();

  public:
    // Return registered Result_t for the given "value" code.
    static const Result_t& Find(int value);
    static Result_t Delete(int value);

    static unsigned int End();
    static const Result_t& Get(unsigned int);

    Result_t(int v, const std::string& s, const std::string& l);
    Result_t(const Result_t& rhs);
    const Result_t& operator=(const Result_t& rhs);
    ~Result_t();

    const Result_t operator()(const std::string& message) const;
    const Result_t operator()(const int& line, const char* filename) const;
    const Result_t operator()(const std::string& message, const int& line, const char* filename) const;

    inline bool        operator==(const Result_t& rhs) const { return value == rhs.value; }
    inline bool        operator!=(const Result_t& rhs) const { return value != rhs.value; }
    inline bool        Success() const { return ! ( value < 0 ); }
    inline bool        Failure() const { return ( value < 0 ); }

    inline int         Value() const { return value; }
    inline operator    int() const { return value; }
    inline const char* Label() const { return label.c_str(); }
    inline operator    const char*() const { return label.c_str(); }
    inline const char* Symbol() const { return symbol.c_str(); }
    inline const char* Message() const { return message.c_str(); }
  };
```

```
KM_DECLARE_RESULT(FALSE,       1,   "Successful but not true.");
  KM_DECLARE_RESULT(OK,          0,   "Success.");
  KM_DECLARE_RESULT(FAIL,       -1,   "An undefined error was detected.");
  KM_DECLARE_RESULT(PTR,        -2,   "An unexpected NULL pointer was given.");
  KM_DECLARE_RESULT(NULL_STR,   -3,   "An unexpected empty string was given.");
  KM_DECLARE_RESULT(ALLOC,      -4,   "Error allocating memory.");
  KM_DECLARE_RESULT(PARAM,      -5,   "Invalid parameter.");
  KM_DECLARE_RESULT(NOTIMPL,    -6,   "Unimplemented Feature.");
  KM_DECLARE_RESULT(SMALLBUF,   -7,   "The given buffer is too small.");
  KM_DECLARE_RESULT(INIT,       -8,   "The object is not yet initialized.");
  KM_DECLARE_RESULT(NOT_FOUND,  -9,   "The requested file does not exist on the system.");
  KM_DECLARE_RESULT(NO_PERM,    -10,  "Insufficient privilege exists to perform the operation.")
  KM_DECLARE_RESULT(STATE,      -11,  "Object state error.");
  KM_DECLARE_RESULT(CONFIG,     -12,  "Invalid configuration option detected.");
  KM_DECLARE_RESULT(FILEOPEN,   -13,  "File open failure.");
  KM_DECLARE_RESULT(BADSEEK,    -14,  "An invalid file location was requested.");
  KM_DECLARE_RESULT(READFAIL,   -15,  "File read error.");
  KM_DECLARE_RESULT(WRITEFAIL,  -16,  "File write error.");
  KM_DECLARE_RESULT(ENDOFFILE,  -17,  "Attempt to read past end of file.");
  KM_DECLARE_RESULT(FILEEXISTS, -18,  "Filename already exists.");
  KM_DECLARE_RESULT(NOTAFILE,   -19,  "Filename not found.");
  KM_DECLARE_RESULT(UNKNOWN,    -20,  "Unknown result code.");
  KM_DECLARE_RESULT(DIR_CREATE, -21,  "Unable to create directory.");
  KM_DECLARE_RESULT(NOT_EMPTY,  -22,  "Unable to delete non-empty directory.");
  // 23-100 are reserved

} // namespace Kumu
```

```
Result_t Result = pPlayer->InitialisationReaders(DcpParse, true, DcpParse.CplVector[CplIndex]->VecReel[0]);
if (ASDCP_SUCCESS(Result))Result = pPlayer->InitialisationJ2K();
```

Principle of decoding and display of the first image

4 -Video frame reading

(MXF->RAM)

Audio buffer reading

(MXF->RAM)

5- Transfer from the video frame to the GPU

Video frame decompression (jpeg2000) by the GPU

6- Transformation of the decompressed image (transition from 12bits to 8bits, Gamma 2.6, change from XYZ color space to sRGB, gamma 2.2)

Multithread

7- Start of audio broadcasting

Subtitle overlay

RGB image display with scaling (linear interpolation)

Principle of decoding and display of the first image

6- Transformation of the decompressed image (transition from 12bits to 8bits, Gamma 2.6, change from XYZ color space to sRGB, gamma 2.2)

Multithread

Principle of decoding and display of the first image

6- Transformation of the decompressed image (transition from 12bits to 8bits, Gamma 2.6, change from XYZ color space to sRGB, gamma 2.2)

Multithread

Transferring the audio buffer to the sound card

Video frame playback

j2k stream analysis

GPU memory allocation for decoding

```
631  Result_t CPlayer::DecodeAndScreenFirstFrame(bool WaitAfterFirstFrame)
632  {
633
634          //int octetsenv = NbSampleperImage * sizeof(SStereo) * NbBlock;
635          NbBlock = BUFFER_AUDIO;
636          thread *tPrepAudio = new thread(PrepareFirstAudioBuffering,this);
637
638          BlockOffset = NbBlock;
639          NbBlock = 1;
640          NumBlock = 0;
641          NbProcFrame = 1;
642          NumFrameAudio = start_frame;
643          //initial_start_frame = start_frame;
644
645          // start process for the first frame
646          Result_t resultVideo = pReader->ReadFrame(start_frame, *pFrameBuffer, Context, HMAC);
647          bitstream_buffer = pFrameBuffer->Data();
648          length = pFrameBuffer->Size();
649          nvjpeg2kStatus_t etat = nvjpeg2kStreamParse(nvjpeg2k_handle, bitstream_buffer, length, 0, 0, nvjpeg2k_stream);
650          if (etat != NVJPEG2K_STATUS_SUCCESS) {
651                  if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamParse failed\n Image format not supported\n");
652                  tPrepAudio->join(); delete tPrepAudio;  return RESULT_FAIL;
653          }
654          etat = nvjpeg2kStreamGetImageInfo(nvjpeg2k_stream, &image_info);
655          if (etat != NVJPEG2K_STATUS_SUCCESS) {
656                  if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamGetImageInfo failed\n");
657                  tPrepAudio->join(); delete tPrepAudio; return RESULT_FAIL;
658          }
659          for (unsigned int c = 0; c < image_info.num_components; c++)
660          {
661                  etat = nvjpeg2kStreamGetImageComponentInfo(nvjpeg2k_stream, &image_comp_info[c], c);
662                  if (etat != NVJPEG2K_STATUS_SUCCESS) { if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamGetImageComponentInfo failed\n");  tPrepAudio->join(
663          }
664          for (int c = 0; c < NUM_COMPONENTS; c++)
665          {
666                  cudaError_t er = cudaMallocPitch((void**)&decode_output[c], (size_t*)&pitch_in_bytes[c], image_comp_info[c].component_width * bytes_per_element, image_c
667                  if (er != cudaSuccess) {
668                          if (Options.verbose_flag) fprintf(fp_log, "\n cudaMallocPitch failed for component=%d\nArchitecure Nvidia PASCAL and above required\n", c);
669                          tPrepAudio->join(); delete tPrepAudio; return RESULT_FAIL;
670                  }
671          }
```

Transferring the audio buffer to the sound card

Video frame MXF reading

j2k stream analysis

Error handling

GPU memory allocation for decoding

```
631  Result_t CPlayer::DecodeAndScreenFirstFrame(bool WaitAfterFirstFrame)
632  {
633
634          //int octetsenv = NbSampleperImage * sizeof(SStereo) * NbBlock;
635          NbBlock = BUFFER_AUDIO;
636          thread *tPrepAudio = new thread(PrepareFirstAudioBuffering,this);
637
638          BlockOffset = NbBlock;
639          NbBlock = 1;
640          NumBlock = 0;
641          NbProcFrame = 1;
642          NumFrameAudio = start_frame;
643          //initial_start_frame = start_frame;
644
645          // start process for the first frame
646          Result_t resultVideo = pReader->ReadFrame(start_frame, *pFrameBuffer, Context, HMAC);
647          bitstream_buffer = pFrameBuffer->Data();
648          length = pFrameBuffer->Size();
649          nvjpeg2kStatus_t etat = nvjpeg2kStreamParse(nvjpeg2k_handle, bitstream_buffer, length, 0, 0, nvjpeg2k_stream);
650          if (etat != NVJPEG2K_STATUS_SUCCESS) {
651                  if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamParse failed\n Image format not supported\n");
652                  tPrepAudio->join(); delete tPrepAudio;  return RESULT_FAIL;
653          }
654          etat = nvjpeg2kStreamGetImageInfo(nvjpeg2k_stream, &image_info);
655          if (etat != NVJPEG2K_STATUS_SUCCESS) {
656                  if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamGetImageInfo failed\n");
657                  tPrepAudio->join(); delete tPrepAudio; return RESULT_FAIL;
658          }
659          for (unsigned int c = 0; c < image_info.num_components; c++)
660          {
661                  etat = nvjpeg2kStreamGetImageComponentInfo(nvjpeg2k_stream, &image_comp_info[c], c);
662                  if (etat != NVJPEG2K_STATUS_SUCCESS) { if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamGetImageComponentInfo failed\n");  tPrepAudio->join()
663          }
664          for (int c = 0; c < NUM_COMPONENTS; c++)
665          {
666                  cudaError_t er = cudaMallocPitch((void**)&decode_output[c], (size_t*)&pitch_in_bytes[c], image_comp_info[c].component_width * bytes_per_element, image_c
667                  if (er != cudaSuccess) {
668                          if (Options.verbose_flag) fprintf(fp_log, "\n cudaMallocPitch failed for component=%d\nArchitecure Nvidia PASCAL and above required\n", c);
669                          tPrepAudio->join(); delete tPrepAudio; return RESULT_FAIL;
670                  }
671          }
```

Creating the window for the image

Creation of surfaces for display (two surfaces are created to prepare the loop in ping-pong operation)

Initialization for autoscaling

Memory allocation for the 3 channels

Jpeg2000 decoding on the GPU

```
672         height = image_comp_info[0].component_height;
673         width = image_comp_info[0].component_width;
674
675         if (mywin == NULL)
676         {
677                 mywin = win_init_render(width, height, &Renderer, BlackBackground, Options.NumDisplay, Options.FullScreen);
678                 if (mywin == NULL) { if (Options.verbose_flag) fprintf(fp_log, "\n Unable to open display window=%d\n", Options.NumDisplay);   tPrepAudio->join(); delete tPr
679         }
680   // we create new surface since width and weight can change with cpl change
681         if (out) SDL_FreeSurface(out);
682         if (out_swap) SDL_FreeSurface(out_swap);
683         out = SDL_CreateRGBSurface(0, width, height, 32,rmask, gmask, bmask, amask);
684         SDL_SetSurfaceBlendMode(out, SDL_BLENDMODE_NONE);
685         out_swap = SDL_CreateRGBSurface(0, width, height, 32,rmask, gmask, bmask, amask);
686         SDL_SetSurfaceBlendMode(out_swap, SDL_BLENDMODE_NONE);
687
688         if (win_h > 1440) Font = Font64;
689         else Font = Font32;
690
691         float scalex = (float(width) / float(win_w));
692         float scaley = (float(height) / float(win_h));
693         scalef = max(scalex, scaley);
694         int hf = height / scalef;
695         int wf = width / scalef;
696         int linits = win_h / 2 - hf / 2;
697         int cinits = win_w / 2 - wf / 2;
698
699         vchanR.resize(width * height);
700         vchanG.resize(width * height);
701         vchanB.resize(width * height);
702         output_image.pixel_data = (void**)decode_output;
703         output_image.pixel_type = NVJPEG2K_UINT16;
704         output_image.pitch_in_bytes = pitch_in_bytes;
705         output_image.num_components = NUM_COMPONENTS;
706         bool loop = true;
707
708         nvjpeg2kStatus_t status = nvjpeg2kDecodeImage(nvjpeg2k_handle, decode_state, nvjpeg2k_stream, decode_params, &output_image, 0); // 0 corresponds to cudaStream_t
709         if (etat != NVJPEG2K_STATUS_SUCCESS) { if (Options.verbose_flag) fprintf(fp_log, "\n Cuda decoding failed\n"); tPrepAudio->join(); delete tPrepAudio; return RESULT_F
710         cudaError_t er = cudaDeviceSynchronize();
711         if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n Cuda synchronization error\n"); tPrepAudio->join();delete tPrepAudio; return RESULT_FAIL; }
712         unsigned short* chanR = (unsigned short*)vchanR.data();
713         unsigned short* chanG = (unsigned short*)vchanG.data();
```

Copy decoded GPU image to CPU RAM

Initializations for threads

Starting the 4 XYZ image to RGB transformation threads

```
712         unsigned short* chanR = (unsigned short*)vchanR.data();
713         unsigned short* chanG = (unsigned short*)vchanG.data();
714         unsigned short* chanB = (unsigned short*)vchanB.data();
715         er = cudaMemcpy2D(chanR, (size_t)width * sizeof(unsigned short), output_image.pixel_data[0], pitch_in_bytes[0], width * sizeof(unsigned
716         if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n cudaMemcpy2D Red failed\n");     tPrepAudio->join();     delete
717         er = cudaMemcpy2D(chanG, (size_t)width * sizeof(unsigned short), output_image.pixel_data[1], pitch_in_bytes[1], width * sizeof(unsigned
718         if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n cudaMemcpy2D  Green failed\n");  tPrepAudio->join();     delete
719         er = cudaMemcpy2D(chanB, (size_t)width * sizeof(unsigned short), output_image.pixel_data[2], pitch_in_bytes[2], width * sizeof(unsigned
720         if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n cudaMemcpy2D Blue failed\n");    tPrepAudio->join();     delete
721
722         Mem.mywin = mywin;
723         Mem.width = width;
724         Mem.height = height;
725         Mem.pLut26 = Lut26;
726         Mem.pLut22 = Lut22;
727         Mem.scr = out;
728
729         Mem.win_w = win_w;
730         Mem.win_h = win_h;
731         Mem.dstRect = { cinits, linits, wf, hf };
732         Mem.base = win_h - (win_h - (height) / scalef) / 2;
733         Mem.Scalef = scalef;
734         Mem.FrameCount = frame_count;
735         Mem.IncrustPosition = Options.IncrustPosition;
736         Mem.IncrustFps = Options.IncrustFps;
737         Mem.DisplayFps = 0.0;
738
739
740         Mem.chanB = chanB;
741         Mem.chanR = chanR;
742         Mem.chanG = chanG;
743         Af1 = new thread(ThreadQuarter1, &Mem);
744         Af2 = new thread(ThreadQuarter2, &Mem);
745         Af3 = new thread(ThreadQuarter3, &Mem);
746         Af4 = new thread(ThreadQuarter4, &Mem);
```

Starting the 4 XYZ image to RGB transformation threads

Waiting for the image processing to complete

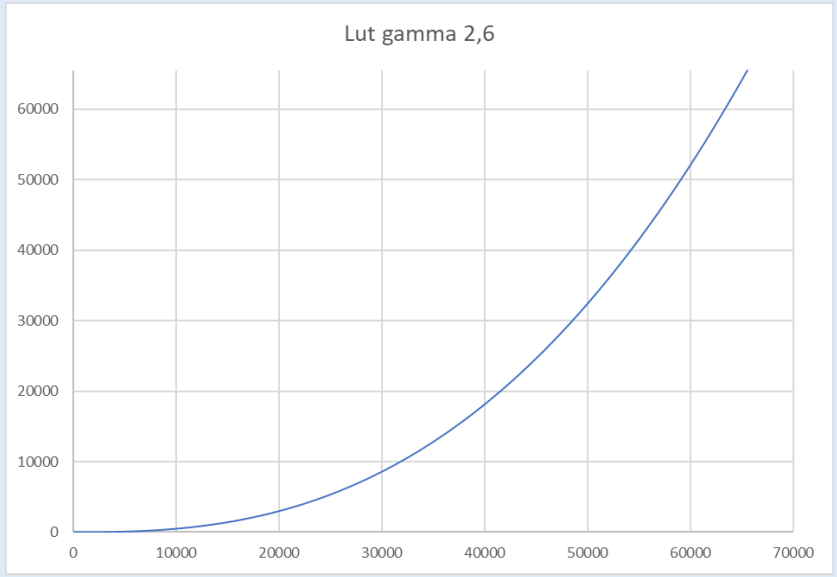
Rendering the image on the screen with subtitle overlay

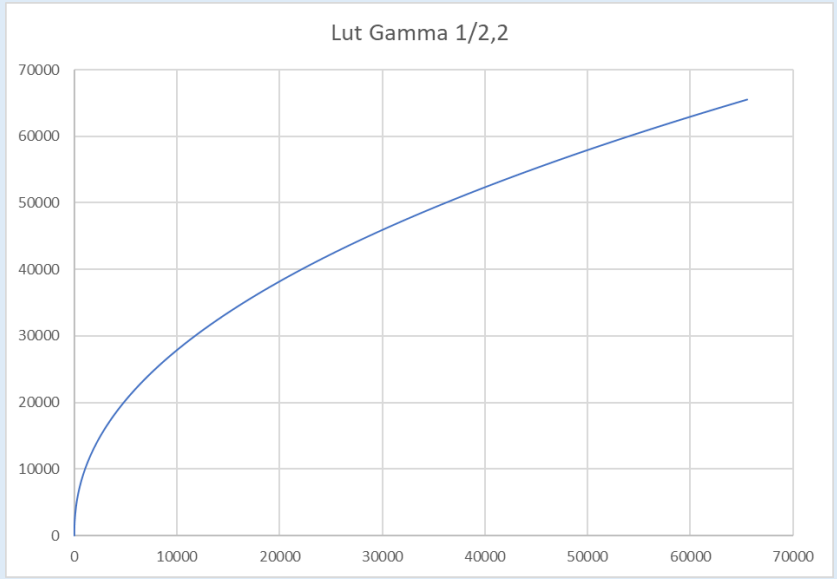
```
743             Af1 = new thread(ThreadQuarter1, &Mem);
744             Af2 = new thread(ThreadQuarter2, &Mem);
745             Af3 = new thread(ThreadQuarter3, &Mem);
746             Af4 = new thread(ThreadQuarter4, &Mem);
747             if (Af1 && Af2 && Af3 && Af4)
748             {
749                     //      // wait for end of image drawing
750                     Af1->join(); delete Af1; Af1 = NULL;
751                     Af2->join(); delete Af2; Af2 = NULL;
752                     Af3->join(); delete Af3; Af3 = NULL;
753                     Af4->join(); delete Af4; Af4 = NULL;
754             }
755
756             RenderImageWithSub(Renderer, Font, ref(MySubTitles), width, height, ref(IndiceSub), start_frame, ref(Mem));
757
758             tPrepAudio->join();
759             delete tPrepAudio;
760             return RESULT_OK;
761     }
762
```

Principle of multithreaded image processing

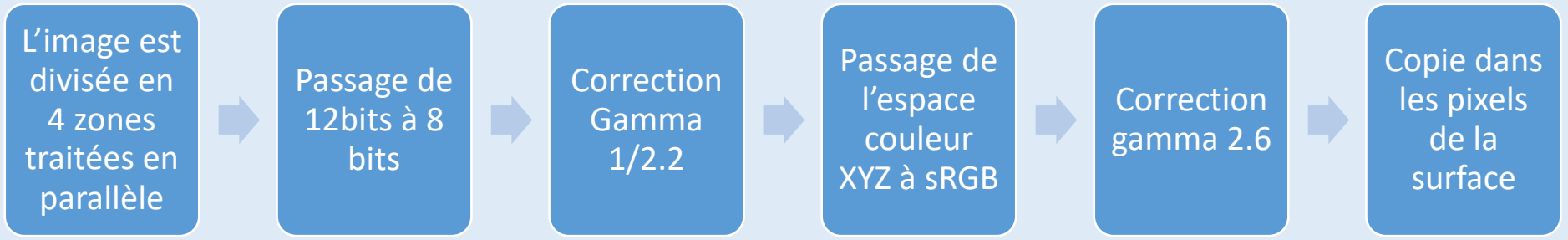
The image is divided into 4 areas processed in parallel → Moving from 12-bit to 8-bit → Correction Gamma 2,6 → Switching from XYZ color space to sRGB → Correction gamma 1/2,2 → Copy to surface pixels

```
bool CPlayer::PrepareXYZ2RGBLUT()
{
	if (Lut26 != NULL) free(Lut26);
	if (Lut22 != NULL) free(Lut22);
	Lut26 = (unsigned short*)malloc(65536 * sizeof(unsigned short));
	Lut22 = (unsigned short*)malloc(65536 * sizeof(unsigned short));
	if (Lut26 == NULL || Lut22 == NULL) return false;
	double gamma26 = 2.6;
	double gamma22 = 1.0 / 2.2;
	static unsigned short int maxs = 0XFFFF;
	for (int i = 0; i <= 0xFFFF; i++)
	{
		double p = /*saturate_cast<ushort>*/(maxs * pow((double)i / (double(maxs)), gamma26));
		Lut26[i] = (unsigned short int)p; // faire un cast avec verif
		p = /*saturate_cast<ushort>*/(maxs * pow((double)i / (double(maxs)), gamma22));
		Lut22[i] = (unsigned short int)p; // faire un cast avec verif
	}
	return true;
}
```





Principle of multithreaded image processing



XYZ to RGB (brucelindbloom.com)

```
float MyCoefXYZ[3][3] = { {3.2404542, -1.5371385, -0.4985314},
                          {-0.9692660, 1.8760108, 0.0415560},
                          {0.0556434, -0.2040259,  1.0572252 } };
```

```
red_cor_coul = int((float)red_co_gamma * MyCoefXYZ[0][0] + (float)green_co_gamma * MyCoefXYZ[0][1] + (float)blue_co_gamma * MyCoefXYZ[0][2]);
green_cor_coul = int((float)red_co_gamma * MyCoefXYZ[1][0] + (float)green_co_gamma * MyCoefXYZ[1][1] + (float)blue_co_gamma * MyCoefXYZ[1][2]);
blue_cor_coul = int((float)red_co_gamma * MyCoefXYZ[2][0] + (float)green_co_gamma * MyCoefXYZ[2][1] + (float)blue_co_gamma * MyCoefXYZ[2][2]);
```

Retrieving image parameters

Declarations and initializations

```
1479  void CPlayer::ThreadQuarter1(void* Param)
1480  {
1481          SMemoire* Mem = (SMemoire*)Param;
1482          SDL_Surface* scr = Mem->scr;
1483          if (Mem->mywin == NULL )
1484          {
1485                  return;
1486          }
1487          if ( scr==NULL)
1488          {
1489                  return;
1490          }
1491          int height = Mem->height;
1492          int width = Mem->width;
1493
1494          Uint8 bpp = scr->format->BytesPerPixel;
1495          Uint8 rs = scr->format->Rshift / 8;
1496          Uint8 rg = scr->format->Gshift / 8;
1497          Uint8 rb = scr->format->Bshift / 8;
1498          unsigned short int redbase = 3500;
1499          unsigned short int greenbase = 4000;
1500          unsigned short int bluebase = 4000;
1501          unsigned short int red_co_gamma;
1502          unsigned short int blue_co_gamma;
1503          unsigned short int green_co_gamma;
1504          int red_cor_coul;
1505          int green_cor_coul;
1506          int blue_cor_coul;
1507          unsigned char red_f, green_f, blue_f;
1508          static unsigned short int maxs = 0XFFFF;
1509
1510          int h2 = height >> 1;
1511          int w2 = width >> 1;
1512          int linit = (scr->h >> 1) - h2;
1513          int cinit = (scr->w >> 1) - w2;
1514
```

We treat the upper right quarter

First Gamma correction
Via precalculated LUTs

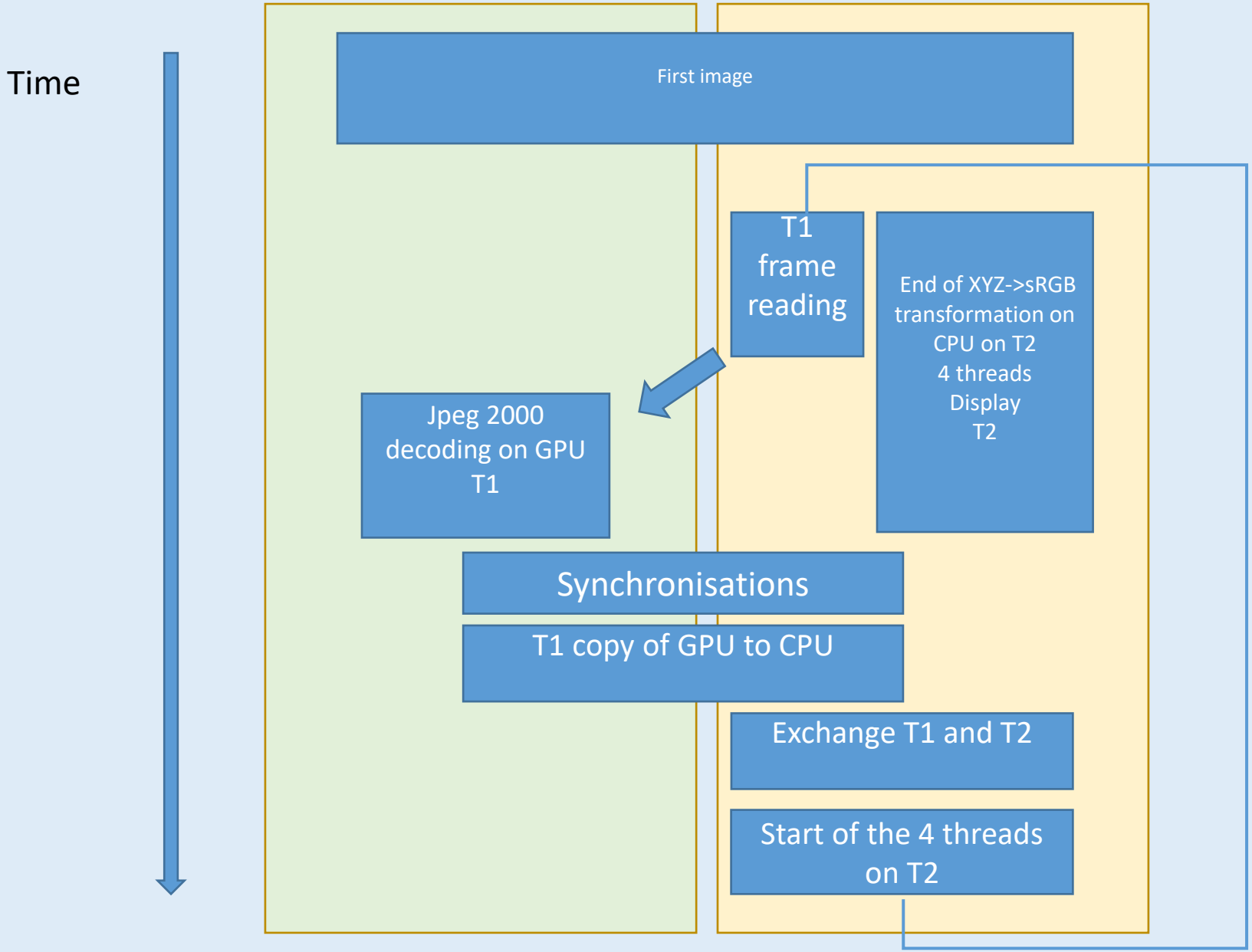
Switching from XYZ to RGB (standardized)

Second Gamma correction
Via precalculated LUTs

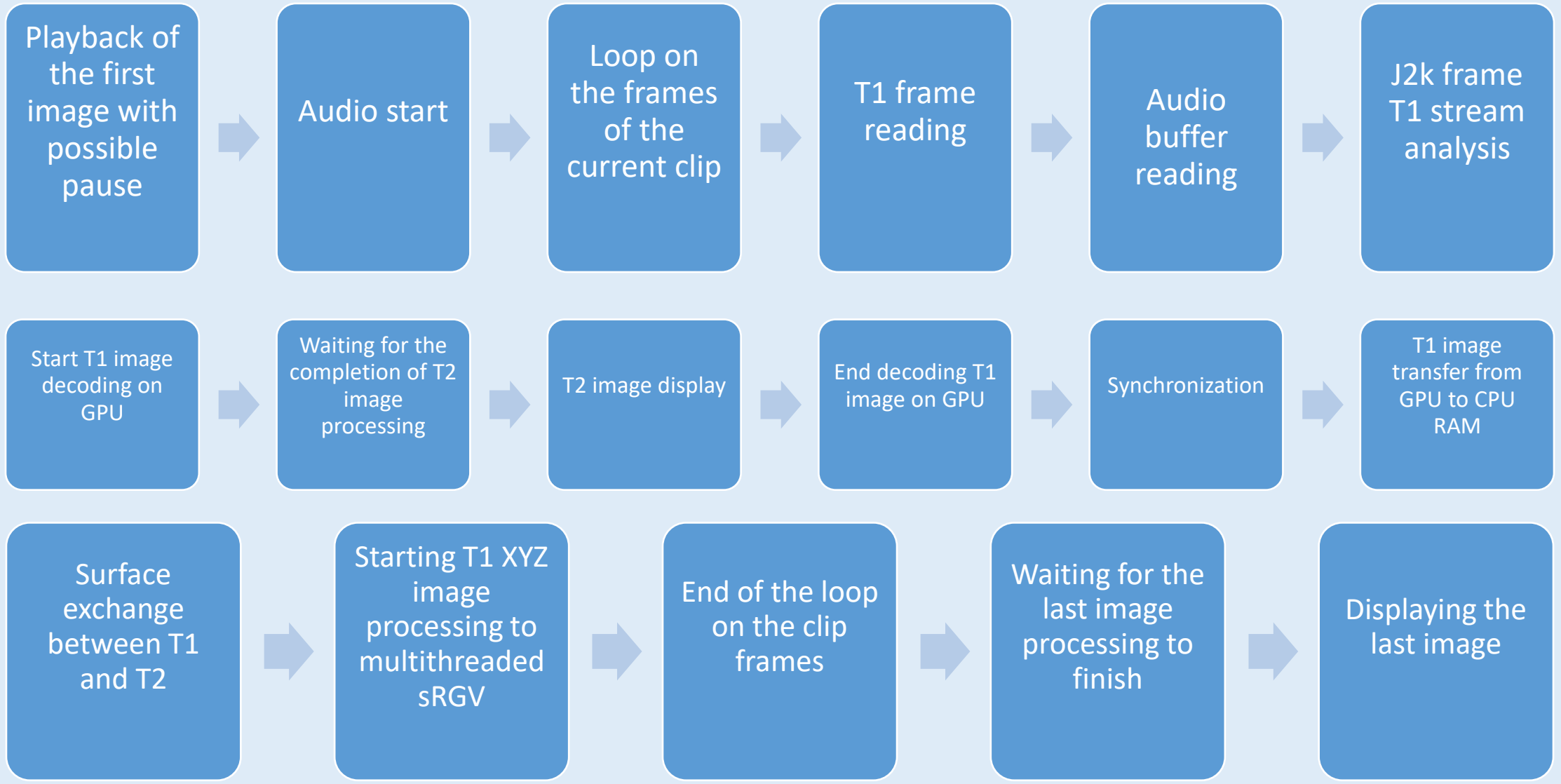
Copying to the current surface

```
int h2 = height >> 1;
int w2 = width >> 1;
int linit = (scr->h >> 1) - h2;
int cinit = (scr->w >> 1) - w2;

for (int li = 0; li < h2; li++)
{
        for (int col = 0; col < w2; col++)
        {
                int p = width * li + col;
                redbase = Mem->chanR[p];
                greenbase = Mem->chanG[p];
                bluebase = Mem->chanB[p];
                red_co_gamma = Mem->pLut26[redbase << 4];
                green_co_gamma = Mem->pLut26[greenbase << 4];
                blue_co_gamma = Mem->pLut26[bluebase << 4];
                red_cor_coul = int((float)red_co_gamma * MyCoefXYZ[0][0] + (float)green_co_gamma * MyCoefXYZ[0][1] + (float)blue_co_gamma * MyCoefXYZ[0][2]);
                green_cor_coul = int((float)red_co_gamma * MyCoefXYZ[1][0] + (float)green_co_gamma * MyCoefXYZ[1][1] + (float)blue_co_gamma * MyCoefXYZ[1][2]);
                blue_cor_coul = int((float)red_co_gamma * MyCoefXYZ[2][0] + (float)green_co_gamma * MyCoefXYZ[2][1] + (float)blue_co_gamma * MyCoefXYZ[2][2]);
                if (red_cor_coul < 0) red_cor_coul = 0;
                if (green_cor_coul < 0) green_cor_coul = 0;
                if (blue_cor_coul < 0) blue_cor_coul = 0;
                if (red_cor_coul > maxs) red_cor_coul = maxs;
                if (green_cor_coul > maxs) green_cor_coul = maxs;
                if (blue_cor_coul > maxs) blue_cor_coul = maxs;
                red_f = (unsigned char)(Mem->pLut22[red_cor_coul] >> 8);
                green_f = (unsigned char)(Mem->pLut22[green_cor_coul] >> 8);
                blue_f = (unsigned char)(Mem->pLut22[blue_cor_coul] >> 8);
                const int idx = ((li + linit) * scr->w + (col + cinit)) * bpp;
                Uint8* px = (Uint8*)scr->pixels + idx;
                *(px + rs) = (unsigned char)red_f;
                *(px + rg) = (unsigned char)green_f;
                *(px + rb) = (unsigned char)blue_f;
        }
}
return;
```

Main loop: ping-pong between two images T1 and T2



Main loop: ping pong between two images T1 and T2



We can be paused
or playing

In the main loop
Decoding and displaying the first image (allows the user to see the image before starting playback by the space key)

The sound is launched

Loops on all frames in the current clip

Managing user actions

```
Result_t CPlayer::MainLoop(bool WaitAfterFirstFrame)
{
        if (WaitAfterFirstFrame) NextState = PAUSE;
        else NextState = PLAY;

        do
        {
                Result_t resultDecod = DecodeAndScreenFirstFrame(WaitAfterFirstFrame);
                if (!ASDCP_SUCCESS(resultDecod)) return RESULT_FAIL;
                offset_frame = start_frame-1;

                AtimerinitialGlobal = MyGetCurrentTime();
                AtimePerimage = AtimerinitialGlobal;
                // start audio
                if (NextState == PLAY &&  RestartLoop ==true) SDL_PauseAudioDevice(Audiodev, SDL_FALSE);
                RestartLoop = false;
                cudaError_t er;

                CurrentFrameNumber = start_frame + 1;

                while (VideoSuccess && CurrentFrameNumber < last_frame && CurrentFrameNumber >= start_frame)
                {//SDL_RaiseWindow(mywin);
                        //RestartLoop = false;
                        StateMachine();

                        if (RestartLoop || OutEscape)
                        {
                                start_frame = CurrentFrameNumber;
                                break;
                        }
```

MainLoop

```
if (NextState == PLAY)
{
        //read one video frame
        Result_t resultVideo = pReader->ReadFrame(CurrentFrameNumber, *pFrameBuffer, Context, HMAC);
        if (ASDCP_SUCCESS(resultVideo)) VideoSuccess = true; else VideoSuccess = false;
        // read one audio frame and send it to the device
        if (NumFrameAudio + BlockOffset < last_frame)
        {
                Result_t resultAudio = pReaderPCM->ReadFrame(NumFrameAudio + BlockOffset, *pFrameBufferPCM, ContextPCM, HMACPCM);
                const byte_t* p = pFrameBufferPCM->RoData();
                memcpy(GlobalBufferOneFrame, p, pFrameBufferPCM->Size());
                if (ADesc.ChannelCount == 6 && Options.Output51 == false)       From51toStereo((SFiveDotOne*)GlobalBufferOneFrame, (SStereo*)AudioForDe
                if (ADesc.ChannelCount == 6 && Options.Output51==true )         From51to51_16B((SFiveDotOne*)GlobalBufferOneFrame, (SFiveDotOne16B*)Aud
                if (ADesc.ChannelCount == 2)    FromStereotoStereo((SStereo24b*)GlobalBufferOneFrame, (SStereo*)AudioForDevice, NbSampleperImage);
                SDL_QueueAudio(Audiodev, (const void*)AudioForDevice, NbSampleperImage * SizeAudioDeviceStruct);
        }
        // decode video
        NumFrameAudio++;
        bitstream_buffer = pFrameBuffer->Data();
        length = pFrameBuffer->Size();
        //FILE* temp = fopen("c:\\video\\temp.bin", "wb"); fwrite(bitstream_buffer, length, 1, temp); fclose(temp);
        nvjpeg2kStatus_t etat = nvjpeg2kStreamParse(nvjpeg2k_handle, bitstream_buffer, length, 0, 0, nvjpeg2k_stream);
        if (etat != NVJPEG2K_STATUS_SUCCESS) {
                if (Options.verbose_flag) fprintf(fp_log, "\n nvjpeg2kStreamParse in loop failed\n ");
                //return RESULT_FAIL;
        }
        nvjpeg2kStatus_t status = nvjpeg2kDecodeImage(nvjpeg2k_handle, decode_state, nvjpeg2k_stream, decode_params, &output_image, 0); // 0 correspond
        if (status != NVJPEG2K_STATUS_SUCCESS) { if (Options.verbose_flag) fprintf(fp_log, "\n Cuda decoding in loop failed\n");  return RESULT_FAIL; }
        // image is decoded, still in the GPU memory
```

MainLoop

```
if (Af1 && Af2 && Af3 && Af4)
{
        // wait for end of previous image processing from XYZ to RGB
        Af1->join(); delete Af1; Af1 = NULL;
        Af2->join(); delete Af2; Af2 = NULL;
        Af3->join(); delete Af3; Af3 = NULL;
        Af4->join(); delete Af4; Af4 = NULL;

        RenderImageWithSub(Renderer, Font, ref(MySubTitles), width, height, ref(IndiceSub), CurrentFrameNumber, ref(Mem));
} // if treads ok

// wait for the end of decoding
er = cudaDeviceSynchronize();
if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n Cuda synchronization in loop error\n");  return RESULT_FAIL; }



// Synchronisation
// wait if the process is too fast
// skip image is the process is too slow
Mem.DisplayFps=Synchronisation();
```

Suite MainLoop

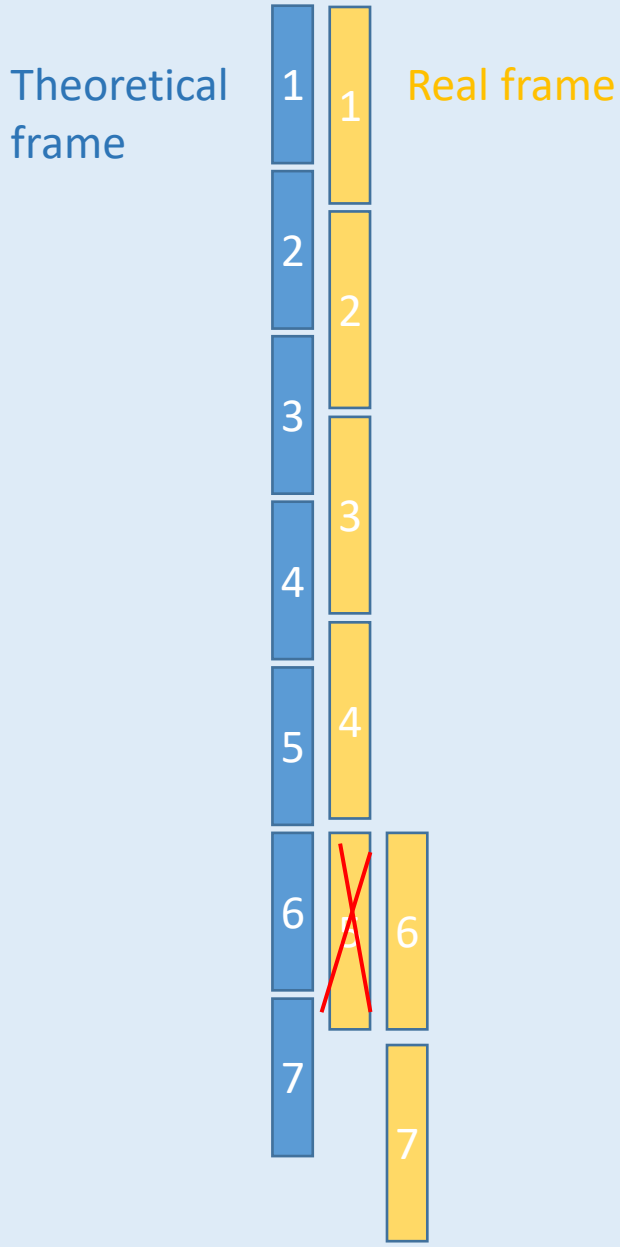
```
                        // copy decoded image from GPU yo Host memory
                        // and start new rendering threads
                        unsigned short* chanR = (unsigned short*)vchanR.data();
                        unsigned short* chanG = (unsigned short*)vchanG.data();
                        unsigned short* chanB = (unsigned short*)vchanB.data();
                        er = cudaMemcpy2D(chanR, (size_t)width * sizeof(unsigned short), output_image.pixel_data[0], pitch_in_bytes[0], width * sizeof(unsigned short),
                        if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n cudaMemcpy2D Red failed\n");  return RESULT_FAIL; }
                        er = cudaMemcpy2D(chanG, (size_t)width * sizeof(unsigned short), output_image.pixel_data[1], pitch_in_bytes[1], width * sizeof(unsigned short),
                        if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n cudaMemcpy2D Green failed\n");  return RESULT_FAIL; }
                        er = cudaMemcpy2D(chanB, (size_t)width * sizeof(unsigned short), output_image.pixel_data[2], pitch_in_bytes[2], width * sizeof(unsigned short),
                        if (er != cudaSuccess) { if (Options.verbose_flag) fprintf(fp_log, "\n cudaMemcpy2D Blue failed\n");  return RESULT_FAIL; }
                        Mem.chanB = chanB;
                        Mem.chanR = chanR;
                        Mem.chanG = chanG;
                        Swap(out, out_swap);
                        Mem.scr = out;
                        Af1 = new thread(ThreadQuarter1, &Mem);
                        Af2 = new thread(ThreadQuarter2, &Mem);
                        Af3 = new thread(ThreadQuarter3, &Mem);
                        Af4 = new thread(ThreadQuarter4, &Mem);

                        CurrentFrameNumber++;

                } // if nextstate = play

        } // end of while frames

        // wait for the last frame rendering
        if (Af1 && Af2 && Af3 && Af4)
        {
                Af1->join(); delete Af1; Af1 = NULL;
                Af2->join(); delete Af2; Af2 = NULL;
                Af3->join(); delete Af3; Af3 = NULL;
                Af4->join(); delete Af4; Af4 = NULL;
        }

        RenderImageWithSub(Renderer, Font, ref(MySubTitles), width, height, ref(IndiceSub), CurrentFrameNumber, ref(Mem));
}
while (CurrentFrameNumber < frame_count && !OutEscape);
return RESULT_OK;
```

Synchronization

- Two types of problems
- The CPU is very fast, the image processing time is less than the allotted time (40ms for 25fps): insert a wait
- The CPU is slow: the processing time of an image is greater than the time allotted, for example 45ms instead of 40ms: if the drift is important it is necessary to sacrifice from time to time one or more images (less annoying than slowing down or cutting the sound).

Synchronization

- The CPU is slow: processing time higher than the allotted time, for example 45ms instead of 40ms: if the drift is important it is necessary to sacrifice from time to time one or more images (less annoying than slowing down or cutting the sound).
- At the beginning of each frame, the theoretical frame number and the current calculated frame number are compared. If the latter is higher than the theoretical frame number, one or more images are eliminated (the sound must never be cut)
-



Controlling User Actions (StateMachine)

- Space: pause/play mode
- Left and right arrows: fast forward or backward
- Up and down arrows: frame by frame if pause mode
- Page Up and Page down: very fast forward and backward
- Esc: stop the player
- Mouse click: the horizontal position of the click gives the position in time
- Requires restarting the main loop with each type of action, and managing audio pause, and audio resynchronization

Traitement du son

- Les DCP sont en 5.1 (parfois plus de canaux, non géré ici), 24 bits par échantillon
- Les périphériques audio de PC sont en stéréo ou 5.1 (parfois plus, non géré)
- La librairie SDL ne gère que le son 16 bits
- D'où les fonctions de conversion suivantes
  - int CPlayer::From51toStereo(const SFiveDotOne* GlobalBufferOneFrame, SStereo* AudioDeviceStereo, int NbSamples)
  - int CPlayer::From51to51_16B(const SFiveDotOne* GlobalBufferOneFrame, SFiveDotOne16B* AudioDevice, int NbSamples)
  - int CPlayer::FromStereotoStereo(const SStereo24b* GlobalBufferOneFrame, SStereo* AudioDeviceStereo, int NbSamples)

5.1 24-bit to 16-bit stereo conversion example

```
int CPlayer::From51toStereo(const SFiveDotOne* GlobalBufferOneFrame, SStereo* AudioDeviceStereo, int NbSamples)
{
	if (GlobalBufferOneFrame == NULL) return -1;
	if (AudioDeviceStereo == NULL) return -2;
	if (NbSamples < 0) return -3;

	SFiveDotOne Sample51;
	float fechR, fechC, fechL, fechBR, fechBL; // discard LFE
	int echR, echC, echL, echBR, echBL;
	float fech;

	for (int j = 0; j < NbSamples; j++)
	{
		//int i = (j * BytePerSampleOutput * NbChannelOut); //  stereo output
		Sample51 = GlobalBufferOneFrame[j];
		echR = ((Sample51.R[2] << 16 | Sample51.R[1] << 8 | Sample51.R[0]) << 8) >> 8;
		fechR = (float)(echR) / float(16777216.0);
		fechR = (fechR * float(0x7fff));

		echC = ((Sample51.C[2] << 16 | Sample51.C[1] << 8 | Sample51.C[0]) << 8) >> 8;
		fechC = (float)(echC) / float(16777216.0);
		fechC = (fechC * float(0x7fff)) * 0.5F;

		echBR = ((Sample51.BR[2] << 16 | Sample51.BR[1] << 8 | Sample51.BR[0]) << 8) >> 8;
		fechBR = (float)(echBR) / float(16777216.0);
		fechBR = (fechBR * float(0x7fff));

		fech = (fechR + fechC + fechBR) / 2.5f;

		AudioDeviceStereo[j].R = short(fech);
		echL = ((Sample51.L[2] << 16 | Sample51.L[1] << 8 | Sample51.L[0]) << 8) >> 8;
		fechL = (float)(echL) / float(16777216.0);
		fechL = (fechL * float(0x7fff));
```

```
struct SFiveDotOne // 5.1
{
	uchar L[3];
	uchar R[3];
	uchar C[3];
	uchar LFE[3];
	uchar BL[3];
	uchar BR[3];
};
```

```
void CPlayer::RenderImageWithSub(SDL_Renderer* Renderer, TTF_Font* Font, vector<SubTitle>& MySubTitles, int width, int height, vector<int>& IndiceSub, Uint32 i, SMemoire& Mem)
{
	SDL_Rect MessageRect; //create a rect
	SDL_Texture* TextTexture;
	double ScaleFont = (Mem.win_h) / 2160.0; // 1.0 for 2160 and 0.5 for 1080

	Mem.Background_Tx = SDL_CreateTextureFromSurface(Renderer, Mem.scr);
	SDL_SetRenderDrawColor(Renderer, 0, 0, 0, 0);
	SDL_RenderClear(Renderer);
	SDL_RenderCopy(Renderer, Mem.Background_Tx, NULL, &Mem.dstRect);
	SDL_DestroyTexture(Mem.Background_Tx);

	if (Mem.IncrustPosition)
	{
		int PosFen = ((i * width) / Mem.FrameCount) / Mem.Scalef;
		SDL_SetRenderDrawBlendMode(Renderer, SDL_BLENDMODE_BLEND);
		SDL_SetRenderDrawColor(Renderer, 255, 0, 0, 127);
		SDL_Rect R1{ 0, Mem.win_h - 30, PosFen, 5 };
		SDL_RenderFillRect(Renderer, &R1);
		SDL_SetRenderDrawColor(Renderer, 255, 255, 0, 127);
		SDL_Rect R2{ PosFen, Mem.win_h - 30,  Mem.win_w-PosFen, 5 };
		SDL_RenderFillRect(Renderer, &R2);
		char buf[512];
		sprintf(buf,"Frame %d", i);
		bool bget = get_text_and_rect(Renderer, 0, 0, buf, Font, &TextTexture, &MessageRect);
		if (bget)
		{
			MessageRect.x = 10;
			MessageRect.y = 10;

			SDL_RenderCopy(Renderer, TextTexture, NULL, &MessageRect);
			SDL_DestroyTexture(TextTexture);
		}
		//else
		//	if (Options.verbose_flag) fprintf(fp_log, "Error in frame information printing\n");
	}
```

Display of the image with subtitles, fps, and possible progress bar

Fps overlay

Subtitle overlay

Image rendering

```
        //      if (Options.verbose_flag) fprintf(fp_log, "Error in frame information printing\n");
}
if (Mem.IncrustFps)
{
        char buf[512];
        sprintf(buf, "fps %3.2f", Mem.DisplayFps);
        bool bget = get_text_and_rect(Renderer, 0, 0, buf, Font, &TextTexture, &MessageRect);
        if (bget)
        {
                MessageRect.x = (width / Mem.Scalef) - MessageRect.w-10;
                MessageRect.y = 10;
                SDL_RenderCopy(Renderer, TextTexture, NULL, &MessageRect);
                SDL_DestroyTexture(TextTexture);
        }
        //else
        //      if (Options.verbose_flag) fprintf(fp_log, "Error in frame information printing\n");

}

if (&MySubTitles != NULL)
        if (MySubTitles.size() > 0)
                if (IndiceSub[i] != 0 && MySubTitles.size() > IndiceSub[i])
                {
                        Uint32 nbligne = MySubTitles[IndiceSub[i]].Line.size();
                        for (ui32_t line = 0; line < nbligne; line++)
                        {
                                //cout << MySubTitles[IndiceSub[i]].Line[line].Text << "    ";
                                bool bget=get_text_and_rect(Renderer, 0, 0, MySubTitles[IndiceSub[i]].Line[line].Text.c_str(), Font, &TextTexture, &MessageRect);
                                if (bget)
                                {
                                        MessageRect.x = (width / 2) / Mem.Scalef - MessageRect.w / 2;
                                        MessageRect.y = Mem.base - MessageRect.h - ((nbligne - line) * MessageRect.h * 1.5);// -MessageRect.h;
                                        SDL_RenderCopy(Renderer, TextTexture, NULL, &MessageRect);
                                        SDL_DestroyTexture(TextTexture);
                                }
                        }
                }

SDL_RenderPresent(Renderer);
```

Conclusion

- Allows smooth playback of the DCP in 2K, with real-time 4K resizing, including on older machines, if the GPU card is new (Pascal architectures and newer)
- Compatible with Windows and Linux
- For Linux (Ubuntu 20.4), see the makefile on the Github website
- Highlights the value of multithreaded CPU and GPU programming
- The Wxwidgets interface can very easily be replaced to launch the program, but it would be more complex to replace the SDL.
- Disadvantage: depends on the nvJpeg2k library developed by Nvidia, powerful, but not open source.
- 4K decoding should be available at the end of 2022