Revolution

THP Library Guide

Version 1.01

© 2006-2007 Nintendo
CONFIDENTIAL

RVL-06-0017-001-D
Released: October 22, 2007

Table of Contents

Code Examples

Tables

Figures

Equations

Revision History

Revision	Date Revised	Item	Description
1.01	2006/12/21	2.2	Updated source code to the newest version.
		3.2.1	Removed `-on` flag.
		3.2.2	Removed `-on` flag.
		3.3.2	Updated the caution regarding playback frequency for Revolution.
		3.3.3.1	Changed the file where the version number is defined.
		3.4.1	Changed path for `dsptoo.dll`.
		3.4.3	Removed section.
1.00	2006/03/01	-	First release by Nintendo of America Inc.

1 Introduction

1.1 Overview

The THP library is designed for the playback of movies on Revolution. By using the THP library, movie data comprising interleaved video data and audio data (THP movie data) can be played on Revolution.

The format of the video data handled by the THP library (THP video data) is customized for rapid decoding by Revolution. The THP library has been optimized to a high extent for decoding of this THP video data.

Note: For increased speed, the THP library uses functions unique to the Broadway CPU (that is, locked cache, as well as GQR). Exercise caution when using the THP library.

Because THP movie data can be decoded so rapidly, it is extremely well suited for applications inside the game. For example, a movie displayed to a screen size of 640 x 480 pixels at a frame rate of 60 frames per second that has been compressed to around 1 bit per pixel, requires about a 40 percent load on the CPU when it is being decoded. If the frame rate is 30 frames per second, the compression ratio can be lowered to 5 or 6 bits per pixel (raising picture quality).

Note: The actual compression ratio is influenced by limits on the speed at which data is read from the optical disc. The slowest optical disc read speed is 3 MB per second. Accordingly, for a movie displayed at 30 frames per second, compress data to 102.4 K, or less, per frame.

By utilizing high-speed THP movie data, you can decode and display a movie in real time, while rendering objects in front of the movie.

THP video data is created from the conversion of normal JPEG data. The JPEG data is converted into THP video very quickly and with no loss in picture quality from the original JPEG data.

Through the use of THP movie data, you can estimate how much time is needed to decode movie data and what kind of picture quality will be realized on Revolution when the data is decoded.

The THP movie audio data (THP audio data) is compressed in the same format as the Revolution audio system's `DSPADPCM`. This THP audio data is appropriately processed to synchronize the video data and the audio data as required for the movie.

In summary, with the THP library, you can playback high-quality video and audio movies with only a minimal burden on the CPU.

1.2 Sample Program

Every frame of THP video data is compressed independently of every other frame. Therefore, movie playback is easy with the THP library, compared to other movie libraries, which require that information from prior and subsequent frames be referenced when decoding the present frame.

However, a slightly complicated procedure is required to achieve appropriate synchronization of the video and audio data during movie playback.

To avoid complexity, two movie players have been prepared that hide this procedure in the THP library, making the playback of movies even simpler.

The first movie player is an extremely simple player designed to provide a basic understanding of how to use the THP library.

The second movie player is an advanced player that assumes use in applications. This advanced player has been created with flexibility to meet the needs of the game for movie playback.

The THP library was created so that movies could be actively played inside even ordinary game scenes. However, the game system affects the movie player when it is used in this way. Therefore, even though the THP movie player library is easy to use, modify the configuration of the player to suit the environment of your game.

Source code is distributed along with the movie players. You can modify this source to suit the THP movie data playback environment.

1.3 Organization of this Document

This document is organized as follows:

- "2 Using the Simple Player to Play Back THP Movie Data" on page 9 explains the minimal procedure required for playing THP movie data on Revolution
- "3 How to Create THP Movie Data" on page 17 explains how to create THP movie data
- "4 Cautions" on page 29 reviews the use of the THP library
- Function List includes a list of functions in the THP library

2 Using the Simple Player to Play Back THP Movie Data

The THP simple player (Simple Player) is a THP movie player that provides the minimal set of functions needed for playing THP movie data on Revolution.

This section explains how to use the Simple Player.

2.1 Source Files

The Simple Player is supplied as the sample program `THPSimple` in the THP library (`build/demos/thpdemo`). The Simple Player is comprised of the following source files:

- `src/THPSimple/THPSimple.c`

This file describes the functions of the Simple Player. The player calls the low-level `THPVideoDecode` function when decoding THP video data and the low-level `THPAudioDecode` function when decoding THP audio data.

- `src/THPSimple/THPDraw.c`

The `THPVideoDecode` function, which is called by the Simple Player when decoding THP video data, outputs the decoded data in YUV texture format. This file describes the functions for drawing this data to the EFB with the graphics processor.

- `src/THPSimple/main.c`

This is a simple sample of THP movie data for playback, using the Simple Player.

- `include/THPSimple.h`

This file contains the definitions for the functions used by the Simple Player.

- `include/THPDraw.h`

This file contains the definitions for the functions used for drawing the decoded YUV texture format data to the EFB with Hollywood.

2.2 How to Use the Simple Player

"Code 2-1 Example of Simple Player Use (main.c)" on page 10 is a simple application using the Simple Player for playback of THP movie data. This application is described in `build/demos/thpdemo/src/THPSimple/main.c`.

Code 2-1 Example of Simple Player Use (main.c)

```
#include <stdlib.h>
#include <string.h>
#include <demo.h>
#include <revolution/mem.h>

#include "THPSimple.h"
#include "axseq.h"

MEMHeapHandle __ExpHeap;

void main(void)
{
    u32             size, x, y, count;
    s32             frame, start, vol, audioTrack;
    u16             buttonDown, button;
    u8              *buffer;
    GXRenderModeObj *rmode;
    THPVideoInfo    videoInfo;
    THPAudioInfo    audioInfo;
    void            *arenaMem2Lo;
    void            *arenaMem2Hi;

    DEMOInit(&GXNtsc480Int);

    arenaMem2Lo = OSGetMEM2ArenaLo();
    arenaMem2Hi = OSGetMEM2ArenaHi();
    _ExpHeap    = MEMCreateExpHeap(arenaMem2Lo, (u32)arenaMem2Hi - (u32) arenaMem2Lo);

    AIInit(NULL);

    AXSeqSetup();

    THPSimpleInit(THP_MODE_WITH_AX);

    GXSetDispCopyGamma(GX_GM_1_0);

    // Open THP file
    if (THPSimpleOpen("thpdemo/fish_mlt.thp") == FALSE)
    {
        OSHalt("THPSimpleOpen fail");
    }

    THPSimpleGetVideoInfo(&videoInfo);
    THPSimpleGetAudioInfo(&audioInfo);

    rmode = DEMOGetRenderModeObj();

    x = (rmode->fbWidth   - videoInfo.xSize) / 2;
    y = (rmode->efbHeight - videoInfo.ySize) / 2;

    // Allocate work area
    size = THPSimpleCalcNeedMemory();

    buffer = MEMAllocFromExpHeapEx(__ExpHeap, size, 32);
```

```

        if (!buffer)
        {
            OSHalt("Can't allocate the memory\n");
        }

        THPSimpleSetBuffer(buffer);

        OSReport("\n######################################\n");
        OSReport("Push A button     : restart the movie, select audio track at random\n");
        OSReport("Push B button     : play/stop midi file using AX\n");
        OSReport("Push Start button : application end\n");
        OSReport("Pad up            : volume up\n");
        OSReport("Pad down          : volume down\n");
        OSReport("######################################\n");

    RESTART:
        // Read ahead for playback
        if (THPSimplePreLoad(THP_PLAY_LOOP) == FALSE)
        {
            OSHalt("THPSimplePreLoad fail");
        }

        audioTrack = (s32)(OSGetTick() % audioInfo.sndNumTracks);

        start = 1;

        count = VIGetRetraceCount();

        while(1)
        {
            DEMOPadRead();

            buttonDown = DEMOPadGetButtonDown(PAD_CHAN0);
            button     = DEMOPadGetButton(PAD_CHAN0);

            DEMOBeforeRender();

            // Decode single frame of THP data
            if (THPSimpleDecode(audioTrack) == VIDEO_DECODE_FAIL)
            {
                OSHalt("Fail to decode video data");
            }

            // Render decoded THP video data
            frame=THPSimpleDrawCurrentFrame(rmode,x,y, videoInfo.xSize, videoInfo.ySize);

            while (VIGetRetraceCount() < count + 1)
            {
                VIWaitForRetrace();
            }

            DEMODoneRender();

            count = VIGetRetraceCount();

            if (start)
            {
                // Allow audio output
                THPSimpleAudioStart();
                start = 0;
            }

            if (buttonDown & PAD_BUTTON_A)
            {
                // Pause playback and restart
                THPSimpleAudioStop();
                THPSimpleLoadStop();
                goto RESTART;
            }

            if (buttonDown & PAD_BUTTON_START)
            {
                // Terminate playback and leave main loop
                THPSimpleAudioStop();
                THPSimpleLoadStop();
                THPSimpleClose();

                MEMFreeToExpHeap(__ExpHeap, buffer);

                break;
            }

            if (buttonDown & PAD_BUTTON_B)
            {
                if (GetSeqState())
                {
                    SeqStop();
                }
                else
                {
                    SeqPlay();
                }
            }

            if (button & PAD_BUTTON_UP)
            {
                vol = THPSimpleGetVolume() + 1;
                if (vol > 127)
                {
                    vol = 127;
                }
                THPSimpleSetVolume(vol, 0);
            }

            if (button & PAD_BUTTON_DOWN)
            {
                vol = THPSimpleGetVolume() - 1;
                if (vol < 0)
                {
                    vol = 0;
                }
                THPSimpleSetVolume(vol, 0);
            }
        }

        THPSimpleQuit();

        VIWaitForRetrace();

        OSHalt("application end\n");

        return;
    }
```

Line 18

The `THPVideoInfo` structure is declared. This structure maintains the vertical and horizontal size of the THP video data. The members of this structure are set by the `THPSimpleGetVideoInfo` function (see line 43).

Line 29:

Before initializing one of the Revolution audio libraries, AX library, the `AIInit` function is called to initialize the audio interface.

Line 31:

The `AXSeqSetup` function initializes AX internally and prepares for use of the AX sequencer.

Line 33:

The `THPSimpleInit` function must be called first, before using any of the Simple Player functions. In addition to initializing the Simple Player control structure (the `THPSimple` structure), the `THPSimpleInit` function enables locked cache and calls the THP library low-level function `THPInit`. It also registers a callback function in the Revolution audio interface for the playback of THP audio data. When used at the same time as AX, initialization functions for the audio library (`AXInit`) must be called before `THPSimpleInit` is called. When used at the same time as AX, `THPSimpleInit` must be called while sound output of AX is set to produce no sound. This sample program specifies `THP_MODE_WITH_AX` for the argument of `THPSimpleInit` for simultaneous use of AX.

Line 38:

This opens the THP movie data (`fish_mlt.thp`) specified by the argument. The `THPSimpleOpen` function calls the `DVDOpen` function to open the THP movie data specified by the argument, and calls the `DVDRead` function to read the header portion of that data. After that, the `THPSimpleOpen` function analyzes the header to check that the data specified by the argument is valid THP movie data.

Line 43:

This obtains information about THP video data from the header of the THP movie data that was loaded into main memory in line 38, and stores it in the `THPVideoInfo` structure specified by the argument. The information acquired here is utilized by the application (in this case, the sample program described in `main.c`).

Line 44:

This obtains information regarding THP audio data from the header of the THP movie data that was loaded into main memory in line 38 and stores it in the `THPAudioInfo` structure specified by the argument. The information acquired here is utilized by the application (in this case, the sample program described in `main.c`).

Lines 48, 49:

Calculates the render location (upper-left corner coordinates) of the decoded data so the THP video data is rendered centered on the screen when the THP movie data is played back.

Line 52:

Calculates the size of the work region to be used by the Simple Player. The `THPSimpleCalcNeedMemory` function returns the following:

- A value that is the sum of the size of the buffer used when the THP movie data is read from the optical disc.
- The size of the YUV texture buffer used when the THP library low-level function decodes the THP video data.
- The size of the buffer used by the THP audio data.
- The size of the work region used by the THP library low-level functions.

The formula used for this calculation is shown in Equation 2-1:

Equation 2-1 Size of the Work Region for the THP Simple Player

```
// Size of buffer for reading the THP movie data
size  = OSRoundUp32B (Maximum data size of THP frame data) * READ_BUFFER_NUM;

// Size of Y texture buffer
+ OSRoundUp32B (THP video data horizontal size x vertical size)

// Size of U texture buffer
+ OSRoundUp32B (THP video data horizontal size x vertical size/4)

// Size of V texture buffer
+ OSRoundUp32B (THP video data horizontal size x vertical size/4)

// Size of buffer for THP audio data
+ OSRoundUp32B (THP audio data's maximum number of samples x 4) x  AUDIO_BUFFER_NUM

// Size of work region used by THP library's low level API functions
+ THP_WORK_SIZE;
```

Notes:

- The `THPConv` tool, which is used to create THP movie data, automatically sets the maximum size of the THP frame data when it is creating THP movie data. This maximum size differs for every set of THP movie data. In addition, for movies that play back a long time or have screens that change frequently, this value differs largely from the average size that can be calculated from the THP movie data file size. For example, in the sample data that comes with the THP library (`rebirth.thp` (`dvddata/thpdemo/rebirth.thp`), the average size is 39,552 bytes, and the maximum size is set to 77,728 bytes. This type of localized fluctuation in data size can have a large influence, not only on the required buffer size for the Simple Player, but also on the data transfer speed from the optical disc and the CPU load required for decoding. Therefore, try to create well-balanced movies.
- The `THPConv` tool also automatically sets the maximum size for the THP audio data. Unlike for the video value, this value for the THP audio data does not fluctuate largely in every frame.
- The values for `READ_BUFFER_NUM` and `AUDIO_BUFFER_NUM` are defined in the THP Simple Player header file, `THPSimple.h` (`build/demos/thpdemo/include/THPSimple.h`).

Code 2-2 Definition of `READ_BUFFER_NUM / AUDIO_BUFFER_NUM` in THP Simple Player

```
#define READ_BUFFER_NUM  10
#define AUDIO_BUFFER_NUM 3
```

Note: The value of `THP_WORK_SIZE` is defined in the THP library header file, `thp.h` (`include/revolution/thp.h`).

Code 2-3 Definition of `THP_WORK_SIZE` in THP Library

```
#define THP_WORK_SIZE 0x1000
```

Line 54:

Allocates a work region for the Simple Player.

Line 61:

Registers the work region that was allocated in line 54 to the Simple Player.

Line 73:

Several frames of THP movie data are read from the optical disc before the THP movie data begins to play. The `THPSimplePreLoad` function calls the `DVDRead` function to read the `READ_BUFFER_NUM` number of frames of THP movie data. The method of playback for the THP movie data is set in the argument of the `THPSimplePreLoad` function. `THP_PLAY_LOOP` is specified to loop the THP movie data, and `THP_PLAY_ONESHOT` is specified to play the THP movie data only once in **one playing**. Both `THP_PLAY_LOOP` and `THP_PLAY_ONESHOT` are defined in the header file of the Simple Player `THPPlayerCommon.h` (`build/demos/thpdemo/include/THPCommon.h`).

Code 2-4 Definitions of the Argument of the `THPSimplePreLoad` Function

```
#define THP_PLAY_ONESHOT 0x00
#define THP_PLAY_LOOP    0x01
```

Line 78:

Determines the audio track played back from the time base register value.

Line 80:

The `start` flag is set to obtain the timing for the start of THP audio data playback (see lines 111-116).

Line 94:

Decodes the THP video data. The decoded data is stored in YUV texture format in a buffer maintained internally by the Simple Player. If audio data is interleaved in the THP movie data, the audio data from the audio track specified with `audioTrack` is also decoded.

Line 100:

Renders the THP video data that was decoded in line 94. For rendering, the `THPSimpleDrawCurrentFrame` function takes the YUV texture-format data decoded by the `THPSimpleDecode` function and pastes it onto polygons.

Lines 102-107:

Writes from the EFB to the XFB, and waits for two retraces.

When using the Simple Player, the loop portion in which the various Simple Player API functions are called and the frame buffer is rendered (that is, the portion between lines 84 and 169), must have the same frame rate as that of the THP movie data being played. For example, if the Simple Player is incorporated into a loop where objects are being rendered at a frame rate of 60 frames per second, the frame rate for that object rendering must drop to 30 frames per second, which is the frame rate for playback of THP movie data.

Note: This sample program is for the playback of THP movie data at a frame rate of 29.97 frames per second.

Lines 111-116:

Enables start of playback of THP audio data. If start of playback is enabled with the `THPSimpleAudioStart` function, decoded THP audio data is sent to the Revolution audio interface. When audio data plays back-interleaved THP movie data, `THPSimpleAudioStart` must be called once immediately after the start of playback.

The best location for the call to this function is immediately after the first frame (line 100) of THP movie data rendered by the `THPSimpleDrawCurrentFrame` function has been displayed on the screen (at or after line 107).

Lines 118 - 124

Stops playback of THP movie data, and restarts playback of that data at the beginning. To do this, first the `THPSimpleAudioStop` function (line 121) is called to cancel permission to play the audio data. When the denial is received, decoded THP audio data is not sent to the Revolution audio interface. Next, the `THPSimpleLoadStop` function (line 122) is called to stop preloading of THP movie data and to initialize the `THPSimple` structure. Doing so returns the Simple Player to its initial status.

Lines 126 - 136:

Terminates playback of THP movie data. First the `THPSimpleAudioStop` function (line 129) is called to stop playback of audio data. Next the `THPSimpleLoadStop` function (line 130) is called to return the Simple Player to its initial status. Then, the `THPSimpleClose` function (line 131) is called to close the THP file.

Lines 138 - 148:

Uses AX to play back or stop a MIDI file.

Lines 150 - 158:

Turns up the playback volume of THP audio data.

Lines 160 - 168:

Turns down the playback volume of THP audio data.

Line 171:

Terminates the Simple Player. The `THPSimpleQuit` function calls the `LCDisable` function to disable locked cache and to return the Simple Player internal state to the status that it was in before the `THPSimpleInit` function was called. To use the Simple Player again, the process must begin with a call to the `THPSimpleInit` function. When using AX simultaneously, call the `THPPlayerQuit` function while sound output of AX is set to not produce any sound.

3 How to Create THP Movie Data

3.1 Overview

The format of THP movie data has specifications unique to Revolution. THP movie data is composed of THP video data and THP audio data, and both types of data are interleaved in every frame. THP movie data has high extensibility and can also incorporate a third type of data besides THP video data and THP audio data.

A unique format is employed for the THP video data. It has been customized for rapid decoding by Revolution. The JPEG data is converted into THP video data very quickly, with no loss in picture quality from the original JPEG data.

The THP audio data is compressed in the Revolution audio system DSP ADPCM format. There are no restrictions on playback frequency. This THP audio data supports both monaural and stereo playback.

The `THPConv` utility is provided for creating THP movie data. `THPConv` supports sequential JPEG files (see Converting Serial JPEG files into THP Movie Data). The tool can also be used to replace the audio data that is incorporated in the THP movie data (see Replacing or Adding THP Audio Data in THP Movie Data).

3.2 Usage

`THPConv` is a Win32 console application. Its use depends on the format of the input data. The following sections explain how to use the THPConv tool for each input data format. The usage can be displayed by running the `THPConv` tool without any options specified.

3.2.1 Converting Serial JPEG files into THP Movie Data

Use `THPConv` as shown in Code 3-1 when converting serial JPEG files to THP movie data.

Code 3-1 `THPConv` Syntax when Converting Serial JPEG Files

```
THPConv.exe -j <*.jpg> -d <outputfile> [-<option> <argument>]
```

Table 3–1 `THPConv` Arguments for Serial JPEG Files

Argument	Description
-j `<*.jpg>`	Specifies the input files (serial JPEG files). Wildcard characters (`*`) can be used. This argument is required.
-d `<outputfile>`	Specifies the output file (the THP file). This argument is required.

The `THPConv` supports the options shown in Table 3–2:

Table 3–2 `THPConv` Options for Conversion of Serial JPEG Files

Option	Description
-s `<wavefile> <wavefile> <wavefile>`	Specifies an audio file (a WAV file) to convert into THP audio data. If multiple files are specified in the argument, the data is allocated to audio tracks in the order specified.
-r `<framerate>`	Specifies the movie frame rate. The frame rate can be set in the range from 1.0 through 59.94. The default value is 29.97).

Option	Description
-o	With THP movie data, the offset to the data for each frame can be stored as a table (see THPFrameOffsetData). When this argument is specified, THPConv creates an offset table in the THP data. If this argument is not specified, no such table is created.
-non or **-odd** or **-even**	Specifies the format of the THP video data (see "THPVideoInfo"). If the video data is in interlaced format and starts from an odd-numbered field, specify –odd. If the video data is in non-interlaced format and starts from an even-numbered field, specify –even. If this argument is not specified, the THP video data format is automatically set as non-interlaced (-non).
-v	Enables verbose mode.

In the example in Code 3-2, the THPConv tool is used to convert sequential JPEG files into THP movie data.

Code 3-2 Converting Sequential JPEG Files into THP Movie Data

```
THPConv -j test*.jpg -d output.thp
```

When this is run on the command line, the sequential JPEG files located in the current directory (`test001.jpg`, `test002.jpg`, `test003.jpg` ...) are individually converted into THP video data, and then collected in one THP movie data file named `output.thp`. See "3.3.1 Serial JPEG Files" on page 21 for information about sequential JPEG files.

Note: When converting sequential JPEG files to THP movie data, make sure the file name of the specified input file has a `.jpg` or `.jpeg` extension.

3.2.2 Replacing or Adding THP Audio Data in THP Movie Data

Code 3-3 `THPConv` Syntax when Replacing or Adding THP Audio Data

```
THPConv.exe -c <inputfile> -s <wavefile> <wavefile>... -d <outputfile> [-<option> <argument>]
```

Table 3–3 `THPConv` Arguments when Replacing or Adding THP Audio Data

Argument	Description
`-c` `<inputfile>`	Specifies the input file (THP file). This argument is required.
`-s` `<wavefile> <wavefile> <wavefile> ... ... ...`	Specifies the sound file (WAV file) to be substituted. If multiple files are specified in the argument, the data is allocated to the audio tracks in the order specified. If a THP file format not supporting multi-tracks is input, the format is changed to provide this support.
`-d` `<outputfile>`	Specifies the output file (a THP file). If this argument is not specified, the THPConv tool *overwrites the input file*.

`THPConv` supports the options shown in Table 3–4.

Table 3–4 `THPConv` Options when Replacing or Adding THP Audio Data

Option	Description
`-r` `<framerate>`	Specifies the movie frame rate. The frame rate can be set in the range from 1.0 through 59.94. The default value is 29.97.
`-o`	With THP movie data, the offset to the data for each frame can be stored as a table (see THPFrameOffsetData). When this argument is specified, the `THPConv` tool creates an offset table in the THP data. If this argument is not specified, no such table is created.
`-non` or **`–odd`** or **`-even`**	Specifies the format of the THP video data (see "(1) THPVideoInfo" on page 25). If the video data is in interlaced format and starts from an odd-numbered field, specify `–odd`. If the video data is in non-interlaced format and starts from an even-numbered field, specify `–even`. If this argument is not specified, the THP video data format is automatically set as non-interlaced (`–non`).
`-trk` `<track No.> <wavefile>`	Use this option instead of –s when you want to substitute only one audio data track. In the first argument, you specify the number of the audio track that you want to replace (0 or higher). In the second argument, the sound file (WAV file) that carries out the substitution is specified. If this option is used, `–r`, `–o`, `–non`, `–odd`, and `–even` are disabled. In addition, the –on option is ignored, and the playback frequency conversion is automatically set to match the other audio tracks. This option can be used only with THP file formats that support multiple tracks.
`-v`	Turns verbose mode on.

Notes:

- When `THPConv` replaces the audio data in the THP movie data, the old THP audio data is deleted and cannot be recovered. Moreover, if an output file is not specified when the audio data is replaced, the `THPConv` tool will overwrites the existing THP movie data. Therefore, when replacing audio data, create a backup file or specify an output file so that you do not lose data.
- If the playback duration of the WAV file is longer than that of the THP movie data, the portion of the WAV file that is longer than the THP movie data is not converted when the WAV file is substituted for the existing audio data. Conversely, if the playback duration of the THP movie data is longer than that of the WAV file, there will be no sound during that extra time.
- If the THP movie data that has been specified as the input file does not contain audio data, this operation adds THP audio data to the specified file.
- If you are including multiple audio data files in the THP movie data, the number of channels for the audio data and the playback frequencies must be consistent.

3.3 Data Formats

3.3.1 Serial JPEG Files

The `THPConv` tool can convert serial JPEG files (a group of JPEG files with consecutive numbers at the end of the files that indicate the ordered sequence for playback) into THP movie data. If audio data is interleaved in the serial JPEG files, either a WAV file must be specified with the `-s` option during the conversion process, or the WAV file must be added after the sequential JPEG files have been converted.

The following restrictions apply to the serial JPEG files that can be handled by the `THPConv` tool:

- Consecutive numbers corresponding to the frame numbers must be at the end of the serial JPEG files
- The JPEG file that corresponds to the first frame of the THP movie data can have any number, but the files must be numbered consecutively
- The frame number at the end of the file name must have the same number of digits as the final frame, with zeros attached to the front
- All the JPEG files that in a set of THP movie data must have the same number of vertical pixels and the same number of horizontal pixels

In Code 4, sequential JPEG files are used to create 4 seconds of THP movie data at a frame rate of 30 frames per second. The sequential JPEG files (`testxxxx.jpg`) are numbered as shown:

Code 3-4 Using Serial JPEG Files to Create THP Movie Data

```
     Frame     1 : test001.jpg
     Frame     2 : test002.jpg
     Frame     3 : test003.jpg
        :             :
     Frame    51 : test051.jpg
     Frame    52 : test052.jpg
        :             :
     Frame   118 : test118.jpg
     Frame   119 : test119.jpg
     Frame   120 : test120.jpg
```

The following restrictions apply to the individual JPEG files that comprise the serial JPEG file group:

- Only the JPEG baseline DCT format is supported
- Only sequential encoding is supported
- Only 4:2:0 sub-sampling is supported
- The pixel count must be a multiple of 16 both vertically and horizontally
- The maximum number of pixels in the horizontal direction is 672. There is no restriction in the vertical direction.

If the serial JPEG files do not fully meet these restrictions, `THPConv` raises an error, and processing stops.

Note: Serial JPEG files must have the extension `.jpg` or `.jpeg`.

3.3.2 WAV Files

THPConv can convert a common WAV file into THP audio data and interleave it with the THP movie data. However, there are the following restrictions on what kinds of WAV files THPConv can convert:

- Files must be uncompressed 16-bit PCM data
- Monaural (1 channel) and stereo (2 channels) are supported

If the WAV file does not fully meet these restrictions, THPConv raises an error, and processing stops.

Although there are no playback frequency restrictions on wav files converted with `THPConv`, keep the following in mind:

- Wii, unlike the Nintendo GameCube, outputs audio data at the configured playback frequency (48,000 Hz or 32,000 Hz)
- WAV files with playback frequencies other than 32,000 Hz and 48,000 Hz can also be converted with `THPConv`. When playing THP movie data created with this type of WAV file, use `THPPlayerStrmAX`.
  Even when you create THP movie data with a WAV file having a playback frequency of 32,000 or 48,000 Hz, if the Revolution playback frequency setting differs from the WAV file playback frequency, use `THPPlayerStrmAX`.

Note: If you are including multiple audio data files in the THP movie data, the number of channels for the audio data and the playback frequencies must be consistent.

Note: WAV files must have the extension `.wav`.

3.3.3 THP Movie Data

The following sections explain the format of the THP movie data that is output by `THPConv`.

3.3.3.1 THPHeader

`THPHeader` is situated at the beginning of the THP movie data. It holds information for the proper initialization of the THP Player.

`“THP\0” (magic[4])` and the version number are stored at the start of `THPHeader` to identify the THP movie data. The upper 2 bytes of the version number indicate the major version number, and the lower 2 bytes the minor version number. `THPConv` automatically sets the version number. The version number is defined in the header file (`include/revolution/thpfile.h`) as `THP_VERSION`.

Following this information in `THPHeader` is information about the maximum frame data size (`bufSize`) and the maximum number of audio samples (`audioMaxSamples`), which are used by the THP Player to calculate the size of the work region.

After this comes other information, including the THP movie data frame rate (`frameRate`) and the total number of frames (`numFrames`).

The THP movie data format is designed with extensibility in mind and can accommodate additional data. The THP offset to this additional data can also be stored in `THPHeader` (after `finalFrameDataOffsets`).

3.3.3.2 THPFrameCompInfo

Data can be interleaved in frames and stored inside the THP movie data. In the THP library, this interleaved data is referred to as components. THP video data and THP audio data are also components. Designed with extensibility in mind, the THP movie data can store components other than video and audio.

`THPFrameCompInfo` holds `numComponents`, which is the number of components included in the THP movie data, and `frameComp[]`, an array showing the order of the components stored in each frame. This array stores the THP Components descriptors (see Table 3–5) in the order in which the components are interleaved.

Table 3–5 THP Component Descriptors

Descriptor	Order
Video Comp	0
Audio Comp	1
Undefined	2
Undefined	3
Undefined	4
Undefined	5
Undefined	6
Undefined	7
-	8
.	9
.	10
.	11
.	12
.	13
.	14
.	15
Nothing	FF

For example, if the first component of the frame is THP video data and the second component is THP audio data, `THPFrameCompInfo` looks like Code 3-5:

Code 3-5 THPFrameCompInfo

```
      numComponents       = 2    : Number of components
      frameComp[0]        = 0    : Video data descriptor
      frameComp[1]        = 1    : Audio data descriptor
      frameComp[2..15]    = FF   : No data
```

After `THPFrameCompInfo,` comes information on each component (`THPVideoInfo`, `THPAudioInfo`, etc.). This component information must also be in the order of the THP Components descriptors stored in the `frameComp` array.

(1) THPVideoInfo

`THPVideoInfo` holds the THP video data vertical and horizontal size information (`xSize`, `ySize`). This information is used for proper playback of the THP video data.

The THP video data format (videoType) is stored next. The values shown in Table 3–6 are stored for the video format, based on the options when using `THPConv`.

Table 3–6 THP Video Data Format Descriptors

Video Format	THPConv Option	videoType
Non-interlaced	None or [-non]	0
Interlaced starting from odd-numbered fields	[-odd]	1
Interlaced starting from even-numbered fields	[-even]	2

(2) THPAudioInfo

`THPAudioInfo` holds the number of channels (`sndChannels`), the playback frequency (`sndFrequency`), and the total number of audio samples (`sndNumSamples`) of THP audio data. It also holds the number of audio data tracks (`sndNumTracks`) included in the THP audio data. This data is used for proper playback of the THP audio data.

3.3.3.3 THPFrameOffsetData

The THP movie data can also maintain a data table with the offset to the start of every frame. When `THPConv` is used with the `-o` option, `THPFrameOffsetData` is created in the THP movie data. If the `-o` option is not specified, `THPFrameOffsetData` is not created. Use this offset data table for special playback purposes, such as to start movie playback from any frame.

3.3.3.4 MovieData

`MovieData` holds the interleaved component data for each frame.

(1) FrameHeader

`FrameHeader` is situated at the front of the `MovieData` data for each frame.

Stored at the start of `FrameHeader` are the size of the previous frame (`frameSizePrevious`) and the size of the next frame (`frameSizeNext`). For the first frame, the size of the last frame is stored in `frameSizePrevious`. Similarly, for the last frame, the size of the first frame is stored in `frameSizeNext`.

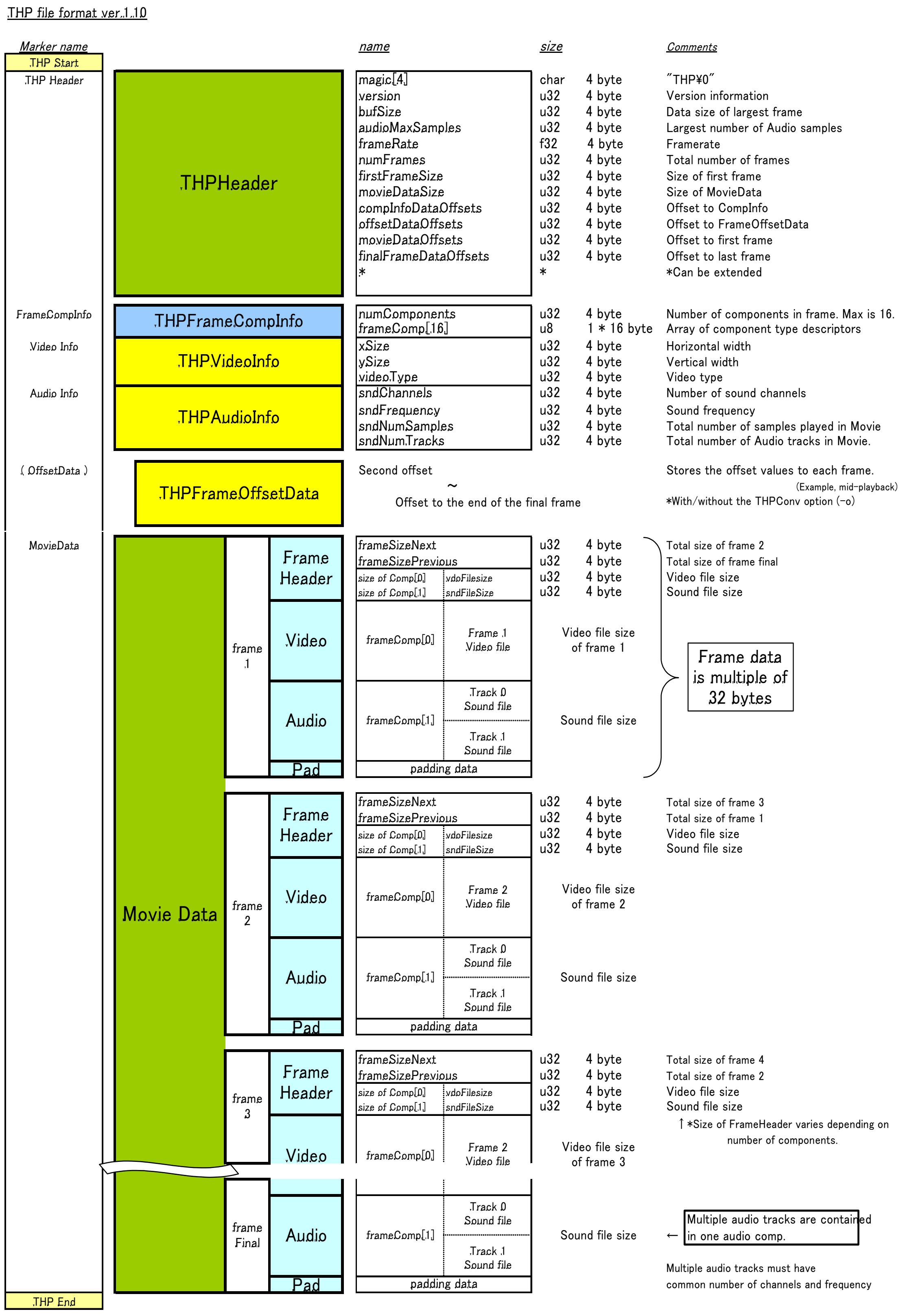
Following these two values, `FrameHeader` stores the data size information in each component interleaved in that frame. This size information must be stored in the order in which the components are interleaved, as defined in `THPFrameCompInfo`.

The size of each frame must be a multiple of 32 bytes. Frames must be padded with 0, as needed, at the end of the frame data to meet this requirement.

Notes:

- The component data sizes included in `FrameHeader` store the size information for each component. However, in the case of audio components, the data size of one audio track is stored.
- The THP audio data is compressed in the Revolution audio system DSP ADPCM format. Therefore, the number of audio data samples stored in each frame (with the exception of the last frame) must be a multiple of 14.

Figure 3-1 THP File Format



name	size		Comments
magic[4]	char	4 byte	"THP¥0"
version	u32	4 byte	Version information
bufSize	u32	4 byte	Data size of largest frame
audioMaxSamples	u32	4 byte	Largest number of Audio samples
frameRate	f32	4 byte	Framerate
numFrames	u32	4 byte	Total number of frames
firstFrameSize	u32	4 byte	Size of first frame
movieDataSize	u32	4 byte	Size of MovieData
compInfoDataOffsets	u32	4 byte	Offset to CompInfo
offsetDataOffsets	u32	4 byte	Offset to FrameOffsetData
movieDataOffsets	u32	4 byte	Offset to first frame
finalFrameDataOffsets	u32	4 byte	Offset to last frame
*	*		*Can be extended
numComponents	u32	4 byte	Number of components in frame. Max is 16.
frameComp[16]	u8	1 * 16 byte	Array of component type descriptors
xSize	u32	4 byte	Horizontal width
ySize	u32	4 byte	Vertical width
videoType	u32	4 byte	Video type
sndChannels	u32	4 byte	Number of sound channels
sndFrequency	u32	4 byte	Sound frequency
sndNumSamples	u32	4 byte	Total number of samples played in Movie
sndNumTracks	u32	4 byte	Total number of Audio tracks in Movie.

Second offset
~
Offset to the end of the final frame

Stores the offset values to each frame.
(Example, mid-playback)
*With/without the THPConv option (-o)

name	size		Comments
frameSizeNext	u32	4 byte	Total size of frame 2
frameSizePrevious	u32	4 byte	Total size of frame final
size of Comp[0] vdoFilesize	u32	4 byte	Video file size
size of Comp[1] sndFileSize	u32	4 byte	Sound file size
frameSizeNext	u32	4 byte	Total size of frame 3
frameSizePrevious	u32	4 byte	Total size of frame 1
size of Comp[0] vdoFilesize	u32	4 byte	Video file size
size of Comp[1] sndFileSize	u32	4 byte	Sound file size
frameSizeNext	u32	4 byte	Total size of frame 4
frameSizePrevious	u32	4 byte	Total size of frame 2
size of Comp[0] vdoFilesize	u32	4 byte	Video file size
size of Comp[1] sndFileSize	u32	4 byte	Sound file size

↑ *Size of FrameHeader varies depending on number of components.

← Multiple audio tracks are contained in one audio comp.

Multiple audio tracks must have common number of channels and frequency

3.4 When Creating THP Movie Data

The following items should be noted when using `THPConv`.

3.4.1 Specifying the Path

`THPConv` makes use of `dsptool(D).dll`, which comes with the Revolution SDK. It is stored in the following location: `$REVOLUTION_SDK_ROOT\x86\bin`.

3.4.2 Sufficient Available Hard-disk Space

THP movie data output by `THPConv` is nearly the same size as the original data before the conversion. When `THPConv` converts video data and audio data into THP movie data, it creates temporary files in the current directory. These temporary files are also about the same size as the original data.

Be sure to confirm that there is sufficient available space on the hard disk when using `THPConv`.

Equation 3-1 Required Available Hard-disk Space for THPConv

```
<Necessary free hard-disk space> = <Size of original data> x 3
```

When `THPConv` creates temporary files, it names the temporary file for video data `__tmp_VD` and the temporary file for audio data `__tmp_AD`. If files with these file names exist in the current directory, `THPConv` overwrites them.

4 Cautions

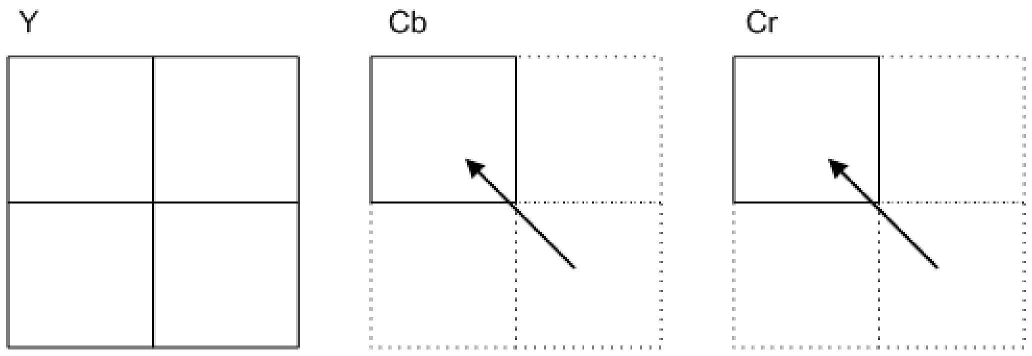
4.1 General Cautions

4.1.1 Sub-sampling

The THP library supports only 4:2:0 sub-sampling.

For a JPEG file with 4:2:0 sub-sampling, the Cb and Cr components are culled as shown in Figure 4-1 in regards to the Y component:

Figure 4-1 4:2:0 Sub-Sampling



If the input JPEG data does not have 4:2:0 sub-sampling, the `THPConv` tool raises an error, and conversion processing stops. If it is not clear what kind of sub-sampling is performed on the JPEG data output by a commercial graphics application, you can determine whether the THP library can decode the data by seeing what happens when the file is input to `THPConv`.

4.1.2 The Players and the GX settings

The THP Simple Player and the THP Player both utilize the THP library low-level function `THPVideoDecode` to decode THP video data. This function outputs the decoded data in YUV texture format. Both players render to the screen by pasting this YUV texture format decoded data to polygons.

Both players make major changes to the GX settings when rendering the decoded data. For applications that use THP movie data, and especially for applications that play movies and draw objects at the same time, be sure to rectify the GX settings that are used for rendering objects, fixing them in each frame after the movie has been rendered and before the object is rendered.

4.1.3 Output of THP Audio Data for Players

When the THP Simple Player and THP Player (`THPPlayer`) are used at the same time as AX, they mix audio output data from the audio libraries with THP audio data, using the CPU, and send it to the audio interface.

In contrast, the THP player that uses AX audio streaming (`THPPlayerStrmAX`) sends the THP audio data to the audio library, and any subsquent processing (until audio data is output from Revolution) is handled by the audio library.

4.1.4 Sampling Rate of THP Audio Data with Simultaneous Use of Audio Libraries

The sampling rate for THP audio data must be 32 KHz when the THP Simple Player and the THP Player are used with AX simultaneously.

When using THP Player utilizing AX audio streaming (THPPlayerStrmAX), the audio library internally converts the sampling rate to 32KHz, so there are no limitations on the THP audio data sampling rate.

4.1.5 Monitoring the Optical Disc Drive during Streaming Playback

When streaming THP movie data from the optical disc for playback, monitor the state of the disc drive from inside the main loop that calls the THP Player functions and renders the frame buffer, and perform the proper process for the specified disc drive state. (The code shown in Example: Monitoring the Drive within the Main Loop is an example of such processes; the exact procedure need not be followed.)

Code 4-1 Example: Monitoring the Drive within the Main Loop

```
switch(DVDGetDriveStatus())
{
    case DVD_STATE_FATAL_ERROR:
        Stop playback
        break;
    case DVD_STATE_NO_DISK:
        Stop playback
        break;
    case DVD_STATE_COVER_OPEN:
        Pause playback
        break;
    case DVD_STATE_WRONG_DISK:
        Stop playback
        break;
    case DVD_STATE_RETRY:
        Stop playback
        break;
 }
```

4.1.6 Handling Sample Data

The sample data that accompanies this library (`rebirth.thp`) is provided strictly for reference purposes. The misappropriation of this sample data and its duplication or alteration without the permission of Nintendo of America is prohibited.

Sample data[a]	`rebirth.thp`	Copyright © 2000 Nintendo
	Created by:	mix core: http://www.mix-core.com
	CG Designers:	Shinichi Shirai, Yutaka Nishikawa, Makoto Nishibori, Takahiro Onishi, Hiroyuki Kusukawa
	Sound Composer:	Masaya Tsunemoto
	Violinist:	Yoko Yoshida

a. All queries regarding this sample data should be directed to Nintendo of America, Inc. (support@noa.com).

4.2 Regarding the THP Player

4.2.1 Main Loop

When playing THP movie data, program your application so that the main loop that renders to the screen and calls the THP Player loops once per `vsync`.

Code 4-2 Restriction on Main Loop When Using the THP Player

```
THPPlayerPlay()

while(1)
{
           .
           .

    THPPlayerDrawCurrentFrame();
           .
           .

    VISetNextFrameBuffer();
    VIFlush();
    VIWaitForRetrace();
}
```

THP movie data may not play smoothly if the main loop is set to loop at anything other than once per `vsync`.

4.2.2 THPPlayerDrawCurrentFrame function

Sometimes, the call to the `THPPlayerDrawCurrentFrame` function may fail (return –1), even after the call to the `THPPlayerPlay` function has succeeded. This is because, internally, playback has been delayed past the proper timing for the start of play.

Essentially, the first frame of decoded data is fetched by the THP Player VI post callback function at the time of the `vsync,` right after the `THPPlayerPlay` function is called (although, in the case of an interlaced movie, this action could be delayed due to restrictions on the first field). Rendering of video data by the `THPPlayerDrawCurrentFrame` function becomes possible after this process.

If you are rendering movies and objects at the same time, check the value returned by the `THPPlayerDrawCurrentFrame` function and confirm that it is not –1 (failure) before displaying the object.

If it is not an interlaced movie, you can use the procedure in Making Certain the Call to the THPPlayer-DrawCurrentFrame Function Succeeds to ensure that the `THPPlayerDrawCurrentFrame` function is called successfully:

Code 4-3 Making Certain the Call to the THPPlayerDrawCurrentFrame Function Succeeds

```
THPPlayerPlay();

VIWaitForRetrace(); <------- At the time of this vsync, the first frame is
                       fetched by the VI post callback.
while(1)
{
       .
       .

    THPPlayerDrawCurrentFrame();
       .
       .

    VISetNextFrameBuffer();
    VIFlush();
    VIWaitForRetrace();
}
```

4.2.3 VI Post Callback

The THP Player utilizes VI post callbacks to control movie playback. Callbacks that are registered before the registration of the THP Player's VI post callback function are called by the player's VI post callback. At the end of playback (that is, when `THPPlayerStop` is called), the player returns the VI post callback to its origin.

4.2.4 Interlaced Movies

The THP Player supports the playback of interlaced movies, but only if the format is such that the data in every frame contains even fields and odd fields that are alternately interleaved in each scan. The even and odd fields should be set up so that each field comes 1/59.94 seconds after the previous field for NTSC and MPAL, and 1/25 seconds for PAL

When the data is in this format, the timing for the start of playback is affected by whether the data in each frame begins with the even field or the odd field.

The THP Player uses the second argument of the `THPPlayerPrepare` function to get information on the scanning order of the interlaced movie and to automatically adjust the timing of the start of playback.

There are no restrictions on the screen size for interlaced movies that can be played by the THP Player. However, whether each frame begins with an even field or an odd field places restrictions on the rendering location when the movie is drawn to the screen. You must adjust the rendering location.

When an interlaced movie is scaled, the position of the even and odd lines in the frame is misaligned, and it may not be possible to play the movie back correctly. Therefore, do not scale interlaced movies for play-back.

The following restrictions apply to interlaced movies that can be played with the THP Player:

- The even fields and the odd fields must be interleaved
- The frame rate must be 29.97 frames per second for NTSC and MPAL, and 25 frames for PAL
- You must specify in the THP Player whether the frame data that comprises the interlaced movie is scanned in the order of even field—odd field, or odd field—even field
- There are no restrictions on screen size, but the rendering location must be adjusted
- The decoded data cannot be scaled

4.2.5 Displaying THP Movie Data Created for NTSC on a PAL System

When creating software versions for markets that use PAL, you may want to repurpose THP movie data created in the NTSC format for release in a PAL format.

When THP movie data created for NTSC televisions at a rate of 29.97 FPS is displayed on an NTSC television, the movie plays correctly if the screen is refreshed every other `Vsync`. If the same data is displayed on a PAL television and the screen is refreshed every other `Vsync`, because PAL televisions display 25 frames (50 fields) per second, not all of the frames are displayed, and the movie does not appear smooth (when using THP player).

The easiest way to avoid this lack of smoothness in display is to change the screen refresh rate to every `Vsync`. This allows all the frames in the THP movie data to be displayed on the television and reduces the loss of fluidity.

The critical thing here is when to display each frame. The THP Player monitors the time since the playback started and determines the frame to draw with the `THPPlayerDrawCurrentFrame` function. Accordingly, the user does not need to worry about the timing and can simply call the `THPPlayerDrawCurrentFrame` function each `Vsync`.

In contrast, the THP Simple Player does not internally monitor the time since the start of playback. The user must monitor the time and decode and display the frame at the right time.

Note: The above method can be applied only to NTSC movie data that is not interlaced and has a frame rate below 50 fps.

Appendix A. Function List

A.1 THP Low-level Functions

API	Process
`THPInit`	Prepares for decoding with the THP library.
`THPVideoDecode`	Decodes THP video data.
`THPAudioDecode`	Decodes THP audio data.

A.2 THP Simple Player Functions

API	Process
`THPSimpleInit`	Initializes the THP Simple Player.
`THPSimpleQuit`	Quits the THP Simple Player.
`THPSimpleOpen`	Opens THP movie data.
`THPSimpleClose`	Closes THP movie data.
`THPSimpleCalcNeedMemory`	Obtains the size of the work area for the THP Simple Player.
`THPSimpleSetBuffer`	Sets the work area in the THP Simple Player.
`THPSimplePreLoad`	Starts the pre-load of THP movie data.
`THPSimpleLoadStop`	Stops the pre-load of THP movie data.
`THPSimpleDecode`	Decodes THP video and audio data.
`THPSimpleDrawCurrentFrame`	Renders decoded THP video data immediately after calling the `THPSimpleDecode` function.
`THPSimpleAudioStart`	Starts playback of THP audio data.
`THPSimpleAudioStop`	Stops playback of THP audio data.
`THPSimpleSetVolume`	Sets the playback volume for THP audio data.
`THPSimpleGetVolume`	Obtains the current volume setting of THP audio data.
`THPSimpleGetVideoInfo`	Obtains information for THP video data.
`THPSimpleGetAudioInfo`	Obtains information for THP audio data.
`THPSimpleGetFrameRate`	Obtains the frame rate for THP movie data.
`THPSimpleGetTotalFrame`	Obtains the total number of frames for THP movie data.

A.3 THP Player Functions

API	Process
`THPPlayerInit`	Initializes the THP Player.
`THPPlayerQuit`	Quits the THP Player.
`THPPlayerOpen`	Opens THP movie data.
`THPPlayerClose`	Closes THP movie data.
`THPPlayerCalcNeedMemory`	Obtains the size of the work area for the THP Player.
`THPPlayerSetBuffer`	Sets the work area in the THP Player.
`THPPlayerPrepare`	Prepares playback of THP movie data.
`THPPlayerPlay`	Starts playback of THP movie data.
`THPPlayerStop`	Stops playback of THP movie data.
`THPPlayerPause`	Pauses playback of THP movie data.
`THPPlayerSkip`	Skips one frame of THP movie data.
`THPPlayerDrawCurrentFrame`	Renders THP video data, synchronizing with playback of THP audio data.
`THPPlayerDrawDone`	Use this function instead of `GXDrawDone` when playing back THP movie data.
`THPPlayerSetVolume`	Sets the playback volume for THP audio data.
`THPPlayerGetVolume`	Obtains the current volume setting of THP audio data.
`THPPlayerGetVideoInfo`	Obtains information for THP video data.
`THPPlayerGetAudioInfo`	Obtains information for THP audio data.
`THPPlayerGetFrameRate`	Obtains the frame rate for THP movie data.
`THPPlayerGetTotalFrame`	Obtains the total number of frames for THP movie data.
`THPPlayerGetState`	Obtains the current status of the THP Player.