

CRI Sofdec
PLAYSTATION3 Tutorials



CRI Middleware Co., LTD. (CRI-MW)

## Table of Contents

1. Preparation.....	1
1.1 Environment.....	1
1.2 Folder and files.....	1
2. Getting start.....	2
2.1 Build.....	2
2.2 Copy files to root directory.....	2
2.3 Run.....	2
3. Review of playback process .....	4
3.1 Data flow.....	4
3.2 Source code .....	5
3.2.1 sfd_t01_simple_play.c.....	5
3.2.2 smp_frag.cg .....	15
3.2.3 smp_vert.cg .....	15
4. Making your own Sofdec file .....	16
4.1 Sofdec CRAFT.....	16
5. Appendix.....	17
5.1 Frame format.....	17
5.1.1 YUV420 plain.....	17
5.1.2 YUVA8 .....	18

# 1. Preparation

## 1.1 Environment

Sofdec tutorial program is assumed to run on a certain develop environment as follows.

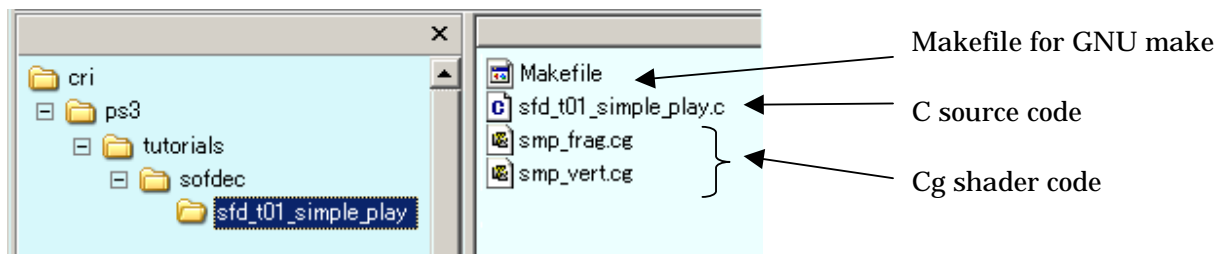
items	availables
Target machine	CEB-201x, DEH-R10xx or later
Sony SDK	0.4.0, 0.5.0, 0.6.0 and 0.8.0
Audio output	Creative USB SoundBlaster (Audigy2) w/ CEB
Video output	HDTV 1280x720

## 1.2 Folder and files

### - Programs files

Sofdec tutorial program is in cri/ps3/tutorial/sofdec.

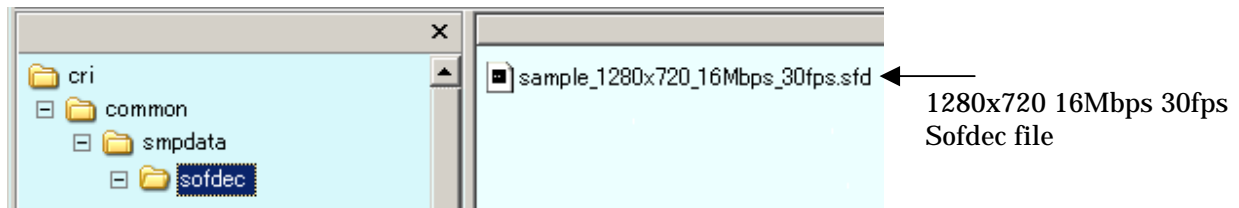
A tutorial is packed each other, it can be built by itself.



### - Data files

Sofdec files are prepared as common data in CRI SDK.

They're in cri/common/smpdata/sofdec.



### - System files

Some system binary file is necessary to run tutorials.

They're as follows.

[Sony SDK 0.4.0]

file name	purpose	folder
brio.elf	PSGL	usr/local/cell/0_4_0/target-ceb/images
fs.elf	cell file system	usr/local/cell/0_4_0/target-ceb/images
shaders.bin	Cg shader	usr/local/cell/0_4_0/samples/graphics/common

[Sony SDK 0.5.0/0.6.0/0.8.0]

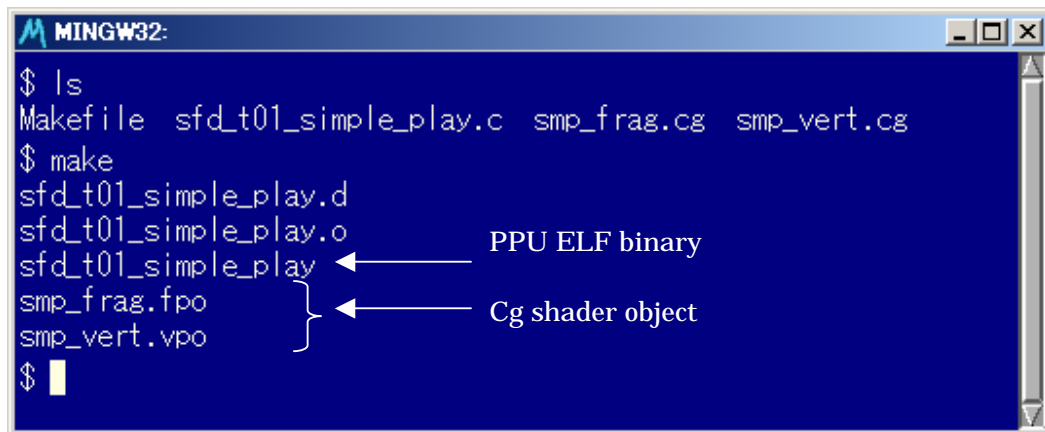
file name	purpose	folder
shaders.bin	Cg shader	usr/local/cell/0_?_0/sample_data/graphics

## 2. Getting start

### 2.1 Build

It is need to be able to call Sony SDK tools (eg. compiler), so please check your "PATH".  
Also check your environment variable of "CRISDK\_PS3\_ROOT", because Makefile refers to it.

To build, type 'make' on prompt as follows.

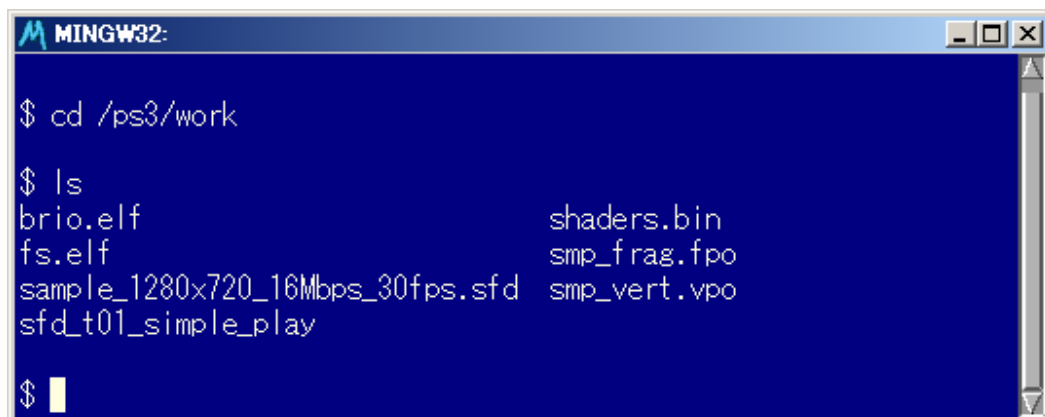


```
MINGW32:
$ ls
Makefile  sfd_t01_simple_play.c  smp_frag.cg  smp_vert.cg
$ make
sfd_t01_simple_play.d
sfd_t01_simple_play.o
sfd_t01_simple_play ← PPU ELF binary
smp_frag.fpo         } ← Cg shader object
smp_vert.vpo         }
$
```

### 2.2 Copy files to root directory

All files to be required is assumed to be in root directory as "lcns1srv" specified.  
Please copy not only PPU ELF binary and Cg shader object but also data files, system files.

For example, in the case of MSYS prompt, if you set root directory typing as "lcns1srv -at  
-hd /ps3/work -ip 192.168.X.XXX", the files in root directory is to be as follows.



```
MINGW32:
$ cd /ps3/work
$ ls
brio.elf          shaders.bin
fs.elf            smp_frag.fpo
sample_1280x720_16Mbps_30fps.sfd  smp_vert.vpo
sfd_t01_simple_play
$
```

### 2.3 Run

Finally, run PPU ELF binary through ppu-lv2-gdb, Eclipse or ProDG.  
Followings are running process through ppu-lv2-gdb on prompt and debug prints of cterm.

```
MINGW32:
$ ppu-lv2-gdb -x /usr/local/cell/0_4_0/host-win32/etc/gdbinit.ppu
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=mingw32 --target=ppu-lv2".
The target architecture is assumed to be powerpc:a35
(ppu-gdb) target remote localhost:12000 ← connect gdb server
Remote debugging using localhost:12000
0xbad00badbad00bad in ?? ()
(ppu-gdb) monitor load sfd_t01_simple_play 0x3e9 ← load ELF file
(ppu-gdb) Loading ELF...
LOAD result = 0x00000000
Process ID = 0x01000300
continue ← Type "continue" !
Continuing.
A new PPU thread has been generated.
Process ID = 0x01000300
```

```
lcterm lpid=3 lcid=1
File Edit Help
lv2(3): ===== Start agent =====
lv2(3): Thr13 <PU_THR13> is going to STOP from sched
lv2(3): VFS_Entry::VFS_Entry
lv2(3): bus: 0.6.6
lv2(3): NV: Using 4 IOIF thread
lv2(3): USB Audio probe (dev_id:2)
lv2(3): USB Audio probe (dev_id:2) ... done.
lv2(3): USB Audio attach (dev_id:2)
lv2(3): USB Audio attach (dev_id:2) ... done.
lv2(3): *** [USB] device can not be allocated.
lv2(3): USB Audio is ready.
lv2(3): [CRI ADX] ADX created the thread : name=cric_adxm_vv_proc, id=23
(0x17), priority=100(0x64)
lv2(3): [CRI ADX] ADX created the thread : name=cric_adxm_vsync_proc, id=24
(0x18), priority=500(0x1f4)
lv2(3): [CRI ADX] ADX created the thread : name=cric_adxm_fs_proc, id=25
(0x19), priority=600(0x258)
lv2(3): [CRI ADX] ADX created the thread : name=cric_adxm_idle_proc, id=26
(0x1a), priority=1500(0x5dc)
lv2(3): MFS:0000000021400000.00000000063E7000
lv2(3): VFS_Entry::~VFS_Entry
```

### 3. Review of playback process

#### 3.1 Data flow

Now Sofdec can play sofdec movie file on system memory, so application has to read file previously. It's shown as `tutor_load_file()` in tutorial program.

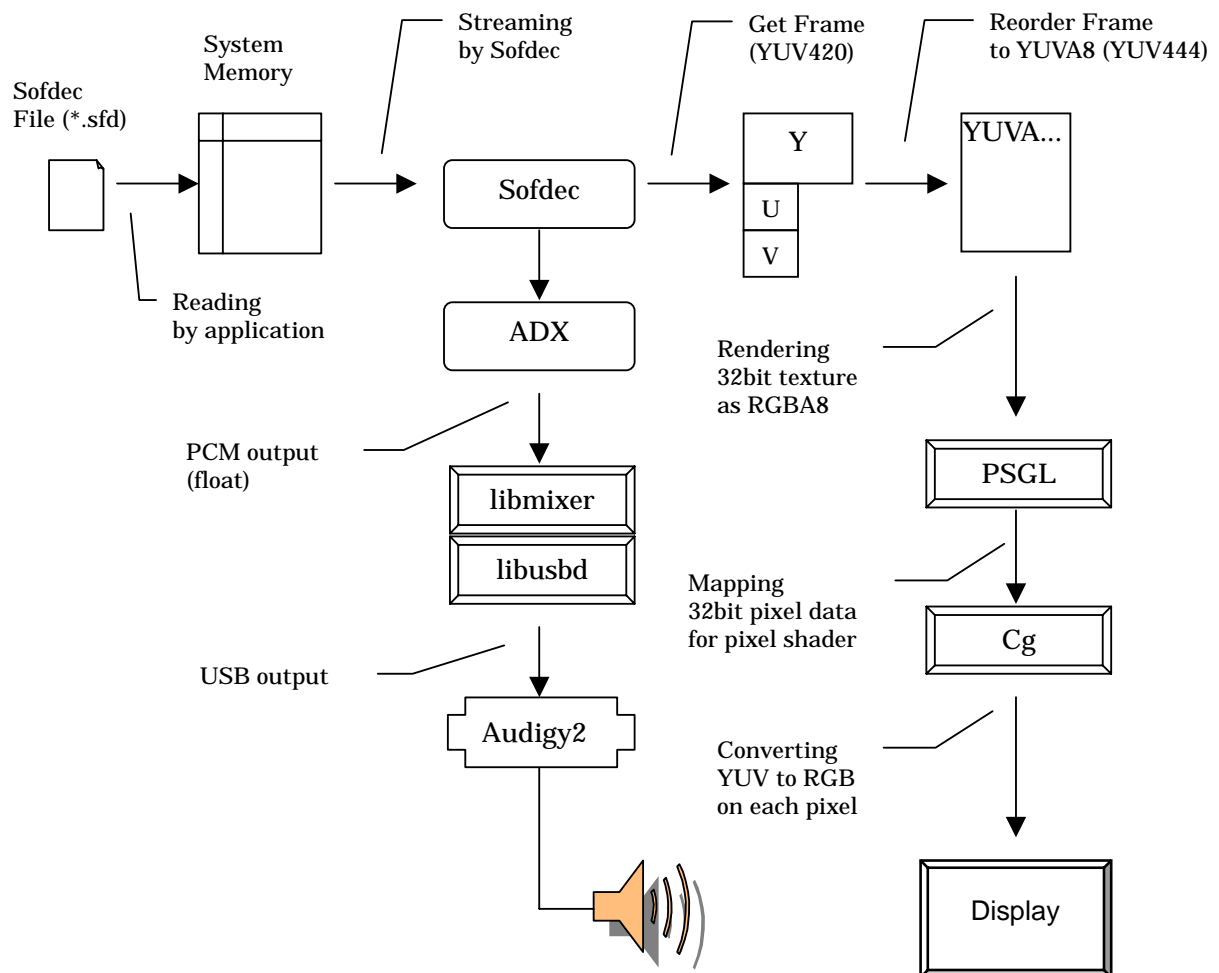
Sofdec reads data as stream from system memory and decodes to YUV420 plain frame, which application can get via `mwPlyGetCurFrm()`. That has to be converted to RGB format but it takes a lot of time, so we propose 2 step frame conversion with pixel shader (Cg).

At first, reordering YUV420 plain frame to YUVA8 format via `mwPlyFxCnvFrmYuva8_PS3()`, then rendering it as 32bit texture specified by `GL_RGBA` with `GL_UNSIGNED_INT_8_8_8_8`.

Secondly, pixel shader gets 32bit pixel data but it consists of YUVA8, so converts to RGB via `main()` function of `smp_frag.cg`.

This is temporary proposal, so it may be replaced to the other way in the near future.

By the way, audio data from Sofdec is going to USB output automatically through ADX.



## 3.2 Source code

### 3.2.1 sfd\_t01\_simple\_play.c

```
/*
 *
 *      CRI Middleware SDK
 *
 *      Copyright (c) 2005-2006 CRI-MW
 *
 *      Appli.      : Sofdec Tutorial
 *      Module      : Sofdec Tutorial No.1(Playing a single Sofdec file)
 *      File        : sfd_t01_simple_play.c
 *
 */
/*****
 *
 *      DESC. : Playing a single Sofdec from memory.
 *
 *      This program is for ...
 *              PLAYSTATION(R)3 Programmer Tool Runtime Library 080.006 or later
 *              DEH-R10xx
 *
 *      Please set necessary sound file in home directory of logical
 *      console server.
 *****/

/*--- standard ---*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*--- SCE ---*/
#include <sdk_version.h>
#include <PSGL/pvgl.h>
#include <PSGL/pvglu.h>
#include <sys/process.h>
#include <sys/spu_initialize.h>
#include <sys/sys_time.h>
#include <sys/timer.h>
#include <sys/memory.h>

#if (CELL_SDK_VERSION >= 0x080000)
#include <cell/gcm.h>
#include <cell/cell_fs.h>
#include <cell/audio.h>
#else
#include <cell/fs/cell_fs_file_api.h>
#include <cell/usbd.h>
#endif

#include <cell/mixer.h>

#if (CELL_SDK_VERSION >= 0x050000)
#include <sys/paths.h>
#endif

/*--- CRI-MW ---*/
#include <cri_mw.h>

#if (CELL_SDK_VERSION < 0x050000)
#define DATA_DIR      "/"
#define SHADER_DIR     ""
#else
#define DATA_DIR      SYS_APP_HOME "/"
#define SHADER_DIR     SYS_APP_HOME "/"
#endif
#endif
```

```

#define SAMPLE_SFD          DATA_DIR "sample_1280x720_16Mbps_30fps.sfd"
#define SHADERS_BIN         SHADER_DIR "shaders.bin"
#define VERT_PROG           SHADER_DIR "smp_vert.vpo"
#define FRAG_PROG           SHADER_DIR "smp_frag.fpo"

#define WINDOW_WIDTH        (1280)
#define WINDOW_HEIGHT       (720)

Char8          tutor_data_dir[512];          // Data directory
const char  *tutor_fs_option[1];
void          *tutor_mem_ptr = NULL;
PSGLdevice    *tutor_psgl_device = NULL;    // PSGL device
CGcontext     tutor_cg_context = NULL;      // Cg context
CGprogram     tutor_vert_prog;              // Cg loaded vertex program
CGprogram     tutor_frag_prog;              // Cg loaded fragment program
GLuint        tutor_texobj;                  // PSGL texture object
GLfloat       tutor_vertex[12] = {
-1.f,  1.f, 0.f,    1.f,  1.f, 0.f,    -1.f, -1.f, 0.f,    1.f, -1.f, 0.f
};
GLfloat       tutor_tc[8] = {
0.f, 0.f,    1.f, 0.f,    0.f, 1.f,    1.f, 1.f
};
#ifdef (CELL_SDK_VERSION < 0x050000)
GLuint        tutor_surface[3] = {0, 0, 0};  // PSGL surfaces
GLuint        tutor_cur_surface = 0;         // PSGL current surface
#endif

CellAANHandle tutor_sur_handle;
#ifdef (CELL_SDK_VERSION >= 0x080000)
const CellSurMixerConfig tutor_surmix_config = { // surround mixer config
400,          // thread priority
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE1A
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE2A
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE6A
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE8A
};
#else
const CellSurMixerConfig tutor_surmix_config = { // surround mixer config
400,          // max samples
400,          // thread priority
CELL_SURMIXER_OUTPUT_2CH, // number of output channel
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE1A
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE2A
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE6A
0,            // Number of CELL_SURMIXER_CHSTRIP_TYPE8A
};
#endif

Char8          tutor_texture[1280 * 720 * 4] __attribute__((aligned(128)));

void tutor_init(void);
void tutor_finish(void);
void tutor_init_video(void);
void tutor_finish_video(void);
void tutor_init_sound(void);
void tutor_finish_sound(void);
void tutor_setup_middleware();
void tutor_shutdown_middleware();
void tutor_err_func(void *obj, Char8 *msg);
void *tutor_load_file(const char *fname, char *mem);
int tutor_sound_callback(void *arg, Uint32 counter, Uint32 num_samples);
void tutor_draw_flip(void);

int main(void)
{
    MwsfdInitPrm      iprm;          // Initialization parameter
    MwsfdCrePrm       cprm;          // Creation parameter
    MwsfdFrmObj       frm;           // Decoded movie frame data

```



```

MWPLY                                ply;                                // Player handle
Char8                                memfile[64];

// Initialize the PSGL and more
tutor_init();

// Setup configurations of the file system, the sound system and the thread mechanism
tutor_setup_middleware();

// Initialize the Sofdec library
memset(&iprm, 0, sizeof(iprm));
iprm.vhz                             = MWSFD_VHZ_60_00;
iprm.dec_svr                         = MWSFD_DEC_SVR_IDLE;
mwPlyInitSfdFx(&iprm);

// Create the Sofdec player handle
memset(&cprm, 0, sizeof(cprm));
cprm.compo_mode                      = MWSFD_COMPO_AUTO;
cprm.ftype                           = MWSFD_FTYPE_SFD;
// cprm.ftype                         = MWSFD_FTYPE_VONLYSFD;
cprm.max_bps                         = 30*1000*1000;
cprm.max_width                       = 1280;
cprm.max_height                      = 720;
cprm.nfrm_pool_wk                    = 5;
cprm.max_stm                         = 1;
cprm.wksize                          = mwPlyCalcWorkCprmSfd(&cprm);
cprm.work                           = (Sint8 *)malloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);

// Start playing the Sofdec file
#if 0
mwPlyStartFname(ply, SAMPLE_SFD);
#else
tutor_mem_ptr = tutor_load_file(SAMPLE_SFD, memfile);
mwPlyStartFname(ply, memfile);
#endif

// ATTN: pause off (this is temporary)
if (cellSurMixerPause(CELL_SURMIXER_CONT_PAUSE_OFF) < 0){
    tutor_err_func(NULL, "cellSurMixerPause");
}
sys_timer_usleep(10000);           // wait 10msec

// Wait for PLAYEND of the Sofdec player handle
for (;;) {
    // Wait for V-sync and decode the image at the same time
    ADXM_WaitVsync();
    // Call the server function
    ADXM_ExecMain();           // Decode the image here when you select MAIN decoding mode

    // Get decoded current movie image
    mwPlyGetCurFrm(ply, &frm);
    if (frm.bufadr != NULL) {
        // Set the size of the output buffer (pitch is equal to texture width)
        mwPlyFxSetOutBufPitchHeight(ply, frm.width * 4, frm.height);
        // Set to load the movie image onto YUVA texture
        mwPlyFxCnvFrmYuva8_PS3(ply, &frm, (UInt8*)tutor_texture);
        // Release the movie image
        mwPlyRelCurFrm(ply);
        // Draw the movie image
        tutor_draw_flip();
    }
    // Check the playing state
    if (mwPlyGetStat(ply) == MWSFD_STAT_PLAYEND) {
        break;
    }
}

```

```

        // Destroy the Sofdec player handle
        mwPlyDestroy(ply);
        free(cprm.work);
        // Cleanup the Sofdec library
        mwPlyFinishSfdFx();

        // Shutdown configurations of the file system, the sound system and the thread mechanism
        tutor_shutdown_middleware();

        // Cleanup DirectSound and more
        tutor_finish();

        return 0;
}

void tutor_init(void)
{
    // Setting for data path
    strcpy(tutor_data_dir, DATA_DIR);

#ifdef (CELL_SDK_VERSION < 0x050000)
    CellFsErrno fs_err;
    CellFsUtilLsp lsp;
    int          ret;
    pid_t        pid;

    ret = sys_process_spawn (&pid, "brio.elf", NULL, 0, 1001, 0);
    if (ret != SUCCEEDED) {
        tutor_err_func(NULL, "sys_process_spawn #1");
        return;
    }

    ret = sys_process_spawn(&pid, "fs.elf", NULL, 0, CELL_FS_SERVER_PRIORITY, 0);
    if (ret != SUCCEEDED) {
        tutor_err_func(NULL, "sys_process_spawn #2");
        return;
    }

    strcpy(lsp.device, "CELL_FS_PSEUDO:");
    lsp.partition_id = 0;
    tutor_fs_option[0] = NULL;

    fs_err = cellFsUtilMount(&lsp, "CELL_FS_PSEUDO", tutor_data_dir, CELL_FS_UTIL_BLOCK,
                             CELL_FS_UTIL_MOUNT_RO, CELL_FS_UTIL_MOUNT_EXCLUSIVE,
tutor_fs_option);
    if ( fs_err != CELL_FS_SUCCEEDED ) {
        printf("cellFsUtilMount %d\n", fs_err);
        return;
    }
#endif

    // Initialize other system device
    tutor_init_video();
    tutor_init_sound();
}

void tutor_finish(void)
{
    if (tutot_mem_ptr != NULL) {
        sys_memory_free((sys_addr_t)tutot_mem_ptr);
    }

    // Finalize other system device
    tutor_finish_sound();
    tutor_finish_video();

#ifdef (CELL_SDK_VERSION < 0x050000)
    // Cell file system unmount

```

```

        cellFsUtilUmount(tutor_data_dir, CELL_FS_UTIL_BLOCK);
#endif

        // Exit this thread
        sys_process_exit(0);
    }

void tutor_init_video(void)
{
    PSGLinitOptions options;
    sys_spu_initialize(6, 2);
    options.enable = PSGL_INIT_MAX_SPUS | PSGL_INIT_INITIALIZE_SPUS;
    options.maxSPUs = 1;
    options.initializeSPUs = GL_FALSE;
    psglInit(&options);

    // Create a PSGL device
    PSGLbufferParameters params = {
        width:WINDOW_WIDTH,
        height:WINDOW_HEIGHT,
        colorBits:24,
        alphaBits:8,
        depthBits:24,
        stencilBits:8,
        deviceType:PSGL_DEVICE_TYPE_TV,
        TVStandard:PSGL_TV_STANDARD_HD720P,
        TVFormat:PSGL_TV_FORMAT_YCRCB,
#if (CELL_SDK_VERSION >= 0x050000)
        bufferingMode:PSGL_BUFFERING_MODE_DOUBLE,
#endif
        antiAliasing:GL_TRUE
    };
    if ((tutor_psgl_device = psglCreateDevice(&params)) == NULL) {
        printf("Failed to create a PSGL device.");
        for (;;)
    }

    // Create a PSGL context
    PSGLcontext *tutor_psgl_context;
    if ((tutor_psgl_context = psglCreateContext()) == NULL) {
        printf("Failed to create a PSGL device.");
        for (;;)
    }
    psglMakeCurrent(tutor_psgl_context, tutor_psgl_device);

    psglLoadShaderLibrary(SHADERS_BIN);
    if (glGetError() != GL_NO_ERROR) {
        tutor_err_func(NULL, "Failed to load shaders.bin");
    }

    // Create a CG context
    tutor_cg_context = cgCreateContext();

    psglResetCurrentContext();

#if (CELL_SDK_VERSION < 0x050000)
    // Set up surfaces
    glGenSurfacesSCE(3, tutor_surface);
    glSetupSurfaceSCE(tutor_surface[0], GL_RGBA8,
        GL_DRAWABLE_BIT_SCE | GL_ALLOW_SCAN_OUT_BIT_SCE | GL_ANTIALIASED_BIT_SCE,
        WINDOW_WIDTH, WINDOW_HEIGHT);
    glSetupSurfaceSCE(tutor_surface[1], GL_RGBA8,
        GL_DRAWABLE_BIT_SCE | GL_ALLOW_SCAN_OUT_BIT_SCE | GL_ANTIALIASED_BIT_SCE,
        WINDOW_WIDTH, WINDOW_HEIGHT);
    glSetupSurfaceSCE(tutor_surface[2], GL_DEPTH24_STENCIL8_SCE,
        GL_DRAWABLE_BIT_SCE | GL_ANTIALIASED_BIT_SCE,
        WINDOW_WIDTH, WINDOW_HEIGHT);
    glBindSurfaceSCE(GL_DRAW_BUFFER0_ATI, tutor_surface[0]);
#endif

```

```

glBindSurfaceSCE(GL_DRAW_DEPTH_SCE, tutor_surface[2]);
glBindSurfaceSCE(GL_DRAW_STENCIL_SCE, tutor_surface[2]);
tutor_cur_surface = 0;
#endif

// Reset the screen
glViewport(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);
glScissor(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);
glClearColor(0.f, 0.f, 0.f, 1.f);
glEnable(GL_DEPTH_TEST);
glFlush();

// Load vertex and fragment program
tutor_vert_prog = cgCreateProgramFromFile(tutor_cg_context,
    CG_BINARY, VERT_PROG, CG_PROFILE_SCE_VP_TYPEC, NULL, NULL);
tutor_frag_prog = cgCreateProgramFromFile(tutor_cg_context,
    CG_BINARY, FRAG_PROG, CG_PROFILE_SCE_FP_TYPEC, NULL, NULL);
if ((tutor_vert_prog == NULL) || (tutor_frag_prog == NULL)) {
    tutor_err_func(NULL, (void*)cgGetErrorString(cgGetError()));
}

// Bind and enable the vertex and fragment programs
cgGLBindProgram(tutor_vert_prog);
cgGLBindProgram(tutor_frag_prog);
cgGLEnableProfile(CG_PROFILE_SCE_VP_TYPEC);
cgGLEnableProfile(CG_PROFILE_SCE_FP_TYPEC);

// YUVA8
glGenTextures(1, &tutor_texobj);
glBindTexture(GL_TEXTURE_2D, tutor_texobj);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 1280, 720, 0, GL_RGBA, GL_UNSIGNED_INT_8_8_8_8, tutor_texture);

// Set to Cg parameter
cgGLSetTextureParameter(cgGetNamedParameter(tutor_frag_prog, "diffuseMapY"), tutor_texobj);
cgGLSetStateMatrixParameter(cgGetNamedParameter(tutor_vert_prog, "ModelViewProj"),
    CG_GL_MODELVIEW_PROJECTION_MATRIX, CG_GL_MATRIX_IDENTITY);
cgGLSetStateMatrixParameter(cgGetNamedParameter(tutor_vert_prog, "ModelView"),
    CG_GL_MODELVIEW_MATRIX, CG_GL_MATRIX_IDENTITY);
cgGLSetStateMatrixParameter(cgGetNamedParameter(tutor_vert_prog, "ModelViewIT"),
    CG_GL_MODELVIEW_MATRIX, CG_GL_MATRIX_INVERSE_TRANSPOSE);
cgGLSetParameterPointer(cgGetNamedParameter(tutor_vert_prog, "Pobject"),
    3, GL_FLOAT, 0, tutor_vertex);
cgGLSetParameterPointer(cgGetNamedParameter(tutor_vert_prog, "YTexUV"),
    2, GL_FLOAT, 0, tutor_tc);

// create vertex buffer
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(tutor_vertex), NULL, GL_STREAM_DRAW);
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(tutor_vertex), tutor_vertex);
glEnableClientState(GL_VERTEX_ARRAY);

// set the background color
glClearColor(0.f, 0.f, .120f, 1.f);

glEnable(GL_TEXTURE_2D);
}

void tutor_finish_video(void)
{
    PSGlContext *tutor_psgl_context = psglGetCurrentContext();
    if (tutor_psgl_context != NULL) {
        #if (CELL_SDK_VERSION < 0x050000)
            glDeleteSurfacesSCE(3, tutor_surface);
        #endif
    }
}

```

```

#endif
        psglDestroyContext(tutor_psgl_context);
    }

    psglDestroyDevice(tutor_psgl_device);
    psglExit();
}

void tutor_init_sound(void)
{
    #if (CELL_SDK_VERSION >= 0x080000)
        CellAudioPeripheralConfig peripheralConfig;

        if (cellAudioInit() < 0){
            tutor_err_func(NULL, "cellAudioInit");
            return;
        }

        peripheralConfig.resolution = CELL_GCM_DISPLAY_720P;
        peripheralConfig.output     = 2;
        if (cellAudioPeripheralConfig(&peripheralConfig) < 0) {
            tutor_err_func(NULL, "cellAudioPeripheralConfig");
            return;
        }
    #else
        CellUsbdInitOps usbd_init_ops;
        usbd_init_ops.dev = 0; // 0 means max number of USB devices
        usbd_init_ops.ed  = 2;
        usbd_init_ops.ldd = 1;
        if (cellUsbdInit(&usbd_init_ops) != CELL_USBD_SUCCEEDED) {
            tutor_err_func(NULL, "cellUsbdInit");
            return;
        }
    #endif

    if (cellSurMixerCreate(&tutor_surmix_config) < 0){
        tutor_err_func(NULL, "cellSurMixerCreate");
        return;
    }

    if (cellSurMixerGetAANHandle(&tutor_sur_handle) < 0){
        tutor_err_func(NULL, "cellSurMixerGetAANHandle");
        return;
    }

    if (cellSurMixerSetNotifyCallback(tutor_sound_callback, NULL) < 0){
        tutor_err_func(NULL, "cellSurMixerSetNotifyCallback");
        return;
    }

    // HEADPHONE setting for level of all channel
    if (cellSurMixerSetParameter(CELL_SURMIXER_PARAM_TOTALLEVEL, -24.0) < 0) {
        tutor_err_func(NULL, "cellSurMixerSetParameter");
    }

    return;
}

void tutor_finish_sound(void)
{
    if (cellSurMixerPause(CELL_SURMIXER_CONT_PAUSE_OFF) < 0){
        tutor_err_func(NULL, "cellSurMixerSetParameter");
    }

    if (cellSurMixerFinalize() < 0){
        tutor_err_func(NULL, "cellSurMixerFinalize");
    }
}

```

```

#if (CELL_SDK_VERSION >= 0x080000)
    /* AudioServer の終了処理 */
    /* Finalize AudioServer */
    if (cellAudioQuit() < 0) {
        tutor_err_func(NULL, "cellAudioQuit");
    }
#else
    if (cellUsbdEnd() != CELL_USBD_SUCCEEDED) {
        tutor_err_func(NULL, "cellUsbdEnd");
    }
#endif
}

void tutor_setup_middleware(void)
{
    Adxps3SoundConfig config;

    memset(&config, 0, sizeof(config));
    config.mixer_handle = tutor_sur_handle;
    // config.mixer_handle = NULL; // NULL handle for no audio playback
    config.num_output_channels = 2;
    config.num_max_voices = 8;
    config.work_size = ADXPS3_CalcSoundWorkSize(&config);
    config.work = malloc(config.work_size);
    if (config.work == NULL) {
        tutor_err_func(NULL, "Failed to allocate memory for sound system");
        return;
    }

    // Set up middleware sound system
    ADXPS3_SetupSound(&config);

    // Set up middleware file streaming I/O
    ADXPS3_SetupPs3Fs(NULL);

    // Set up middleware thread system
    ADXM_SetupFramework(ADXM_FRAMEWORK_PS3PPU_MULTI_THREAD, NULL);

    // ATTN: This must be called after all of ADX*_SetupXxx().
    if (cellSurMixerStart() < 0){
        tutor_err_func(NULL, "cellSurMixerStart");
    }

    // ATTN: pause until playback start (this is temporary)
    if (cellSurMixerPause(CELL_SURMIXER_CONT_PAUSE_ON) < 0){
        tutor_err_func(NULL, "cellSurMixerPause");
    }
    sys_timer_usleep(10000); // wait 10msec to complete pause
}

void tutor_shutdown_middleware(void)
{
    // Pause mixer immediately
    if (cellSurMixerPause(CELL_SURMIXER_CONT_PAUSE_ON_IMMEDIATE) < 0){
        tutor_err_func(NULL, "cellSurMixerPause");
    }

    // Shutdown middleware thread system
    ADXM_ShutdownFramework();

    // Shutdown middleware streaming file I/O
    ADXPS3_ShutdownPs3Fs();

    // Shutdown middleware sound system
    ADXPS3_ShutdownSound();
}

void tutor_err_func(void *obj, Char8 *msg)

```

```

{
    // Show the error message and quit
    printf("ERROR: %s\n", msg);
}

// load movie file
void *tutor_load_file(const char *fname, char *memfile)
{
    ADXF adxf;
    sys_addr_t addr;
    void *buf;
    Sint32 fsize, fsct, asize, stat;

    adxf = ADXF_Open(fname, NULL);
    if (adxf == NULL) {
        tutor_err_func(NULL, "ADXF_Open");
        return NULL;
    }

    fsize = ADXF_GetFsizeByte(adxf);
    fsct = ADXF_GetFsizeSct(adxf);
    asize = (fsize / 0x100000 + 1) * 0x100000;    // (round up by 1MB) + 1MB

    if (sys_memory_allocate(asize, SYS_MEMORY_PAGE_SIZE_1M, &addr) != SUCCEEDED) {
        tutor_err_func(NULL, "sys_memory_allocate");
    }
    if (addr == NULL) {
        tutor_err_func(NULL, "Failed to allocate memory for movie file");
        ADXF_Close(adxf);
        return NULL;
    }

    buf = (void*)((long)addr+127)&~127;    // 128byte aligned
    ADXF_ReadNw(adxf, fsct, buf);
    do {
        ADXM_ExecMain();
        stat = ADXF_GetStat(adxf);
    } while (stat != ADXF_STAT_READEND);

    ADXF_Close(adxf);

#ifdef CELL_SDK_VERSION >= 0x080000
    sprintf(memfile, "MFS:%08X.%08X", (UIntPtr)buf, fsize);
#else
    sprintf(memfile, "MFS:%016X.%016X", (UIntPtr)buf, fsize);
#endif
    printf("%s\n", memfile);

    return (void*)addr;
}

// callback func for surround mixer
int tutor_sound_callback(void *arg, UInt32 counter, UInt32 num_samples)
{
    Sint32 ret;
    ret = ADXPS3_SoundNotifyCallback(arg, counter, num_samples);

    return ret;
}

void tutor_draw_flip(void)
{
    // Enable swap synchronized to Vsync
    glEnable(GL_VSYNC_SCE);
    // Clear buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
    glColor4f(1.f, 1.f, 1.f, 1.f);
}

```

```

// YUVA8
glBindTexture(GL_TEXTURE_2D, tutor_texobj);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 1280, 720, 0, GL_RGBA, GL_UNSIGNED_INT_8_8_8_8, tutor_texture);

// Enable parameter for vertex program
cgGLEnableClientState(cgGetNamedParameter(tutor_vert_prog, "Pobject"));
cgGLEnableClientState(cgGetNamedParameter(tutor_vert_prog, "YTexUV"));
// Enable parameter for fragment program
glActiveTexture(GL_TEXTURE0);
cgGLEnableTextureParameter(cgGetNamedParameter(tutor_frag_prog, "diffuseMapY"));

// Texture coordinates
glMultiTexCoord4f(GL_TEXTURE0, 1.0f, 1.0f, 0.0f, 1.0f); // 1st
glMultiTexCoord4f(GL_TEXTURE0, 1.0f, 0.0f, 0.0f, 1.0f); // 2nd
glMultiTexCoord4f(GL_TEXTURE0, 0.0f, 0.0f, 0.0f, 1.0f); // 3rd
glMultiTexCoord4f(GL_TEXTURE0, 0.0f, 1.0f, 0.0f, 1.0f); // 4th

// Draw the quad
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

// Disable parameters
cgGLDisableClientState(cgGetNamedParameter(tutor_vert_prog, "Pobject"));
cgGLDisableClientState(cgGetNamedParameter(tutor_vert_prog, "YTexUV"));
cgGLDisableTextureParameter(cgGetNamedParameter(tutor_frag_prog, "diffuseMapY"));

// Swap surface
#if (CELL_SDK_VERSION < 0x050000)
    glSwapScanToSurfaceSCE(tutor_surface[tutor_cur_surface]);
    tutor_cur_surface = 1 - tutor_cur_surface;
    glBindSurfaceSCE(GL_DRAW_BUFFER0_ATI, tutor_surface[tutor_cur_surface]);
#else
    psglSwap();
#endif
}

/* end of file */

```



### 3.2.2 smp\_frag.cg

```
/*-----*
 * Caluculation of the table of YUV -> RGB conversion (ITU-R BT.601)
 * R = 1.164(Y - 16) + 1.596(Cr - 128)
 * G = 1.164(Y - 16) - 0.392(Cb - 128) - 0.813(Cr - 128)
 * B = 1.164(Y - 16) + 2.017(Cb - 128)
 *-----*/

half4 main(float3 Peye          : TEXCOORD0,
           float2 ycoord        : TEXCOORD1,
           uniform sampler2D diffuseMapY) : COLOR
{
    half ColorY = (half)tex2D(diffuseMapY, ycoord).x;    // Y
    half ColorU = (half)tex2D(diffuseMapY, ycoord).y - 0.5f; // U
    half ColorV = (half)tex2D(diffuseMapY, ycoord).z - 0.5f; // V

    half3 Color = half3(ColorY, ColorY, ColorY) - half3(0.0625f, 0.0625f, 0.0625f) * 1.164;

    Color.r += ColorV * 1.596f;
    Color.g += -ColorU * 0.392f - ColorV * 0.813f;
    Color.b += ColorU * 2.017f;

    return half4(Color, 1);
}

/* end of file */
```

### 3.2.3 smp\_vert.cg

```
void main(float4 Pobject      : POSITION,
          float2 YTexUV       : TEXCOORD0,
          uniform float4x4 ModelViewProj,
          uniform float4x4 ModelView,
          uniform float4x4 ModelViewIT,

          out float4 HPosition : POSITION,
          out float3 Peye      : TEXCOORD0,
          out float2 ycoord     : TEXCOORD1)
{
    HPosition = mul(ModelViewProj, Pobject);
    Peye = mul(ModelView, Pobject).xyz;
    ycoord = YTexUV;
}

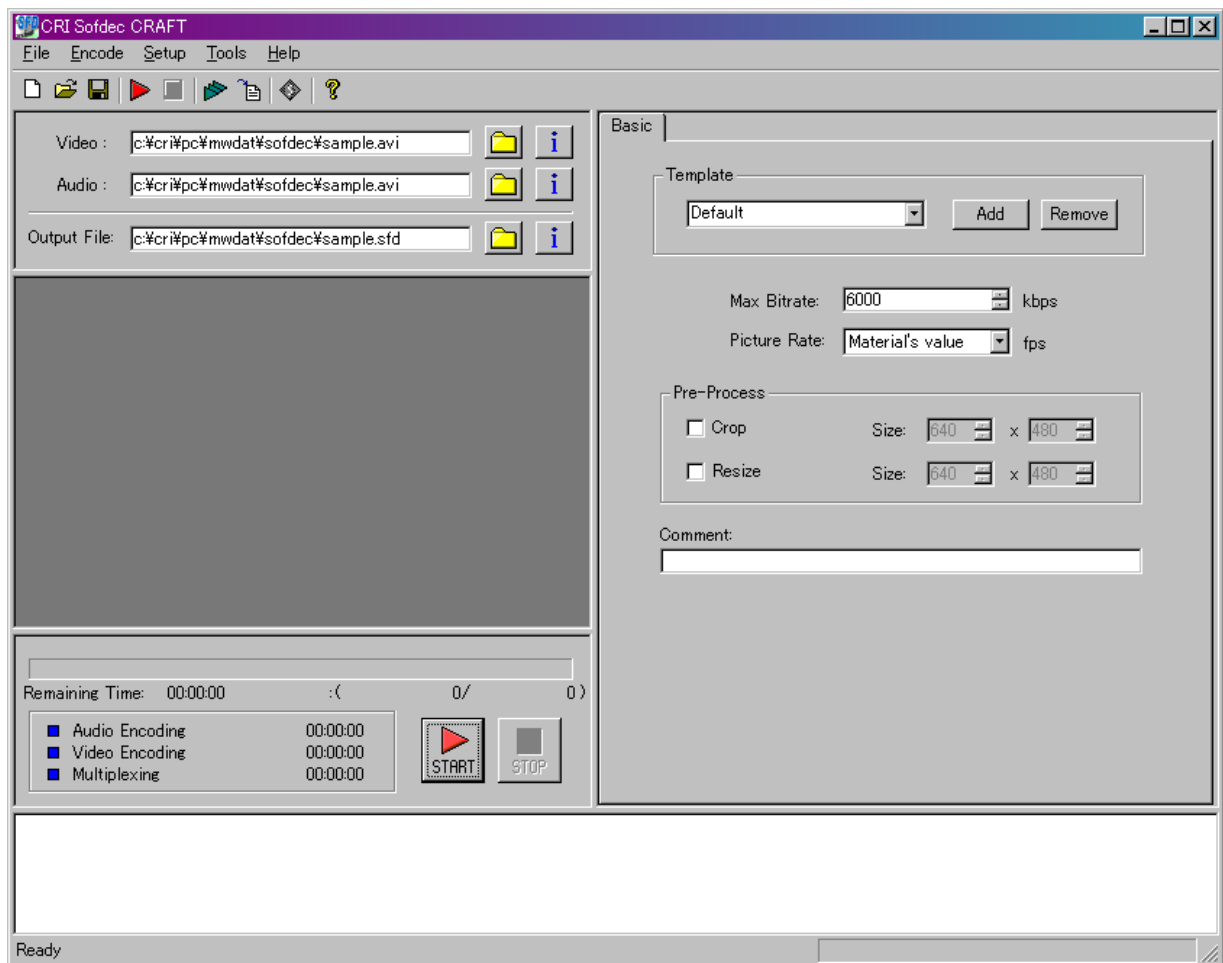
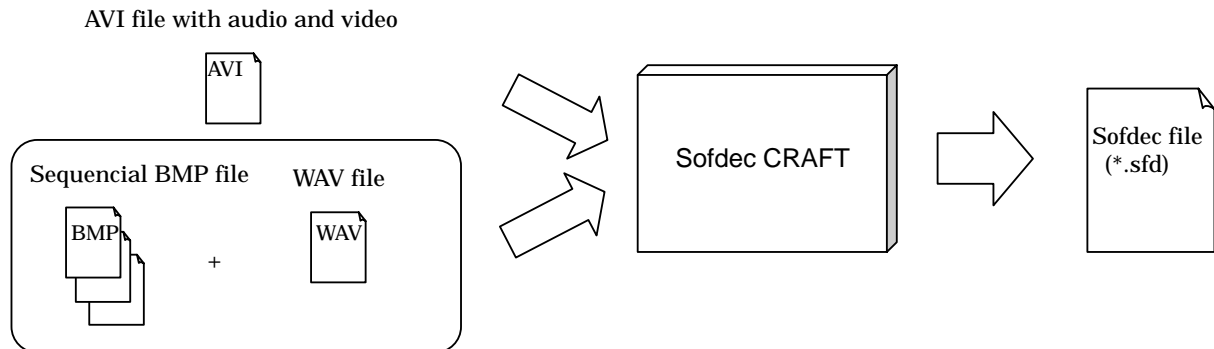
/* end of file */
```

## 4. Making your own Sofdec file

You can find encode tool, `cri/common/tools/windows/sofdec_tools/sfdcrfg.exe`. That's called as Sofdec CRAFT.

### 4.1 Sofdec CRAFT

Sofdec CRAFT converts AVI file to Sofdec file. Instead of AVI file, you can specify sequential BMP file and a WAV file.



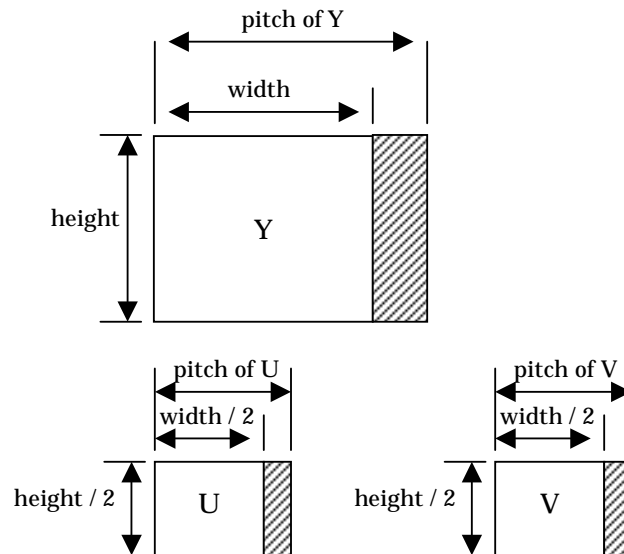
## 5. Appendix

### 5.1 Frame format

#### 5.1.1 YUV420 plain

The width and height of Y is equal to original RGB frame. The width and height of U and V is a half of Y. The pitch means total length of width and a certain margin.

In the case of PS3 sofdec, it's decided to round up by 128 byte, eg. "pitch = Round Up (width, 128)". So pitch of Y is not always equal to one of U or V.

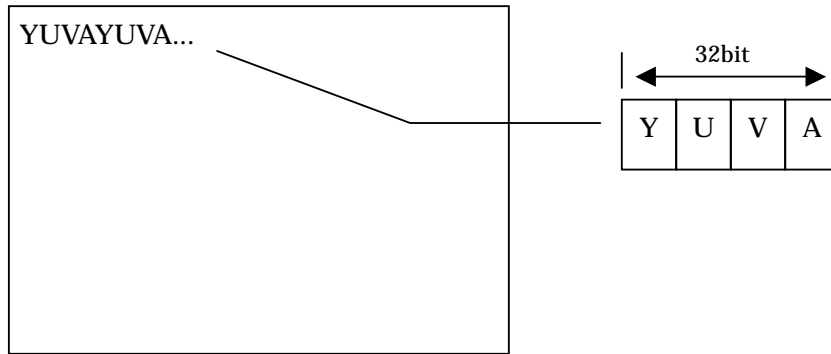


You can get the pitch from `mwPlyCalcYccPlane()` function specified structure `MwsfdFrmObj`, eg. `mwPlyCalcYccPlane(frm->bufadr, frm->width, frm->height, &ycc);`

```
void mwPlyCalcYccPlane(UInt8 *bufptr, Sint32 width, Sint32 height, MwsfdYccPlane *ycc)
typedef struct {
    UInt8      *y_ptr;          /* Y Buffer Pointer */
    UInt8      *cb_ptr;         /* U Buffer Pointer */
    UInt8      *cr_ptr;         /* V Buffer Pointer */
    Sint32      y_width;        /* pitch of Y buffer */
    Sint32      cb_width;       /* pitch of U buffer */
    Sint32      cr_width;       /* pitch of V buffer */
} MwsfdYccPlane, MWS_PLY_YCC_PLANE;
```

### 5.1.2 YUVA8

YUVA8 format consists of 8bit YUVA linearly as follows.



/\* end of file \*/