



## 第6部

# *Middleware ライブラリ編*

---

この章では、ムービー等を取り扱う  
ミドルウェアライブラリの説明を、新しい機能を  
中心に内部構造や使用上の注意点も含め  
ライブラリ全般を解説します。

---



# 目次

<b>1</b>	<b>ミドルウェアライブラリについて</b>	<b>6</b>
1.1	ミドルウェアとは	6
1.2	ミドルウェアライブラリの構成	6
1.3	ミドルウェアライブラリの機能	6
1.3.1	TrueMotion	6
1.3.2	CRI SofdecF/X	7
1.3.3	MPEG/Audio	7
1.3.4	WAVE 再生デコーダ	7
1.3.5	CRI ADX	7
1.3.6	DualSpeech	7
<b>2</b>	<b>ミドルウェアライブラリの概要</b>	<b>8</b>
2.1	ソフトウェア構成	8
2.2	共通ヘッダファイル・ライブラリファイル	9
2.3	ストリーム再生対応形式ファイルについて	9
2.4	ミドルウェアライブラリの内部構成	10
2.5	ミドルウェアの使用方法	11
2.5.1	ミドルウェアライブラリの初期化	11
2.5.2	再生ハンドル	12
2.5.3	ミドルウェアライブラリのサーバ処理	13
2.5.4	ミドルウェア再生ハンドルの動作状態	15
2.6	各再生について	16
2.6.1	SofdecF/X の再生	16
2.6.2	WAVE の再生	18
2.6.3	TrueMotion の再生	19
2.6.4	MPEG/Audio の再生	20
2.6.5	DualSpeech の再生	21
2.6.6	ストリームジョイントによる再生	21
2.6.7	メモリ上のデータからの再生	23
2.7	SIP ライブラリとの関係	24
2.7.1	SIP のサンプリング周波数	24
2.8	SofdecF/X と ADX との並行再生	25
<b>3</b>	<b>ミドルウェアライブラリのストリームの仕組み</b>	<b>26</b>
3.2	コントロールフロー	27
3.3	モジュール構成	28
3.3.1	ミドルウェア再生ライブラリ	28
3.3.2	ミドルウェア録音ライブラリ	29
<b>4</b>	<b>TrueMotion</b>	<b>30</b>
4.1	概要	30
4.2	特徴	30
4.3	ソフトウェア構成	30
4.4	関数一覧	30
<b>5</b>	<b>SofdecF/X</b>	<b>31</b>
5.1	SofdecF/X の概要	31
5.2	ライブラリ構成	31
5.3	SofdecF/X ライブラリ	31
5.3.1	MPEG SofdecF/X ライブラリの特長	31



5.3.2 システム構成 .....	32
5.3.3 モジュール構成 .....	32
5.3.4 コントロールフロー .....	33
5.3.5 表示タイプ .....	34
5.3.6 マルチウィンドウ再生 .....	39
5.3.7 合成 .....	40
5.3.8 テクスチャムービー .....	47
5.3.9 マルチストリーム再生 .....	50
5.3.10 シームレス連続再生 .....	51
5.3.11 動画再生時の注意点 .....	52
5.3.12 関数一覧 .....	56
5.4 SAN ライブラリ .....	57
5.4.1 SAN ライブラリの概要 .....	57
5.4.2 モジュール構成 .....	57
5.4.3 SAN の仕組み .....	58
5.4.4 マルチウィンドウ .....	59
5.4.5 SAN テクスチャ .....	60
5.4.6 合成機能 .....	62
5.4.7 アルファ合成の仕組み .....	62
5.4.8 データフォーマット .....	63
5.4.9 データの作成方法 .....	63
5.4.10 アルファ合成 .....	65
5.4.11 関数一覧 .....	66
6 MPEG/Audio について .....	67
6.1 MPEG/Audio の概要 .....	67
6.2 MPEG/Audio 利用時の処理の流れ .....	68
6.3 MPEG/Audio エンコード処理の流れ .....	69
6.4 MPEG/Audio デコード処理の流れ .....	70
6.5 関数一覧 .....	71
7 ADX 再生ライブラリ .....	72
7.1 概要 .....	72
7.2 ADX 再生ライブラリの特長 .....	72
7.3 ADX 再生システム .....	73
7.3.1 システム構成 .....	73
7.4 モジュール構成 .....	74
7.5 ADX データの流れ .....	75
7.6 アプリケーションインタフェース (API) .....	76
7.7 ADX データの作成方法 .....	76
7.8 ライブラリ関数の使用方法 .....	78
7.8.1 ライブラリの初期化と終了 .....	78
7.8.2 ADXT ハンドルの生成 .....	78
7.8.3 メモリ上の ADX データの再生 (メモリ再生) .....	79
7.8.4 メモリ上の ACX データの再生 (メモリインデックス再生) .....	79
7.8.5 GD 上の ADX データの再生 (GD ストリーム再生) .....	80
7.8.6 GD 上の AFS データの再生 (GD インデックス再生) .....	81
7.8.7 再生状態の取得 .....	82
7.9 サウンドデータ作成上の注意 .....	83
7.10 データ読み込みの高速化について .....	84
7.11 シーク音の軽減方法 .....	85
7.12 MPEG Sofdec との並行再生について .....	86



7.13 GDFS・Ninja ライブラリとの併用について .....	87
7.14 ADX マルチストリームシステムの仕組み .....	88
7.15 ループ再生 .....	89
7.16 シームレス連続再生 .....	90
7.17 音声出力までのタイムラグをなくす方法 .....	90
7.18 クロスフェード .....	91
7.19 サンプルプログラムについて .....	92
7.20 関数一覧 .....	93
<b>8 ADX ファイルシステムライブラリ .....</b>	<b>94</b>
8.1.1 ADX ファイルシステムライブラリの概要 .....	94
8.1.2 ADX ファイルシステムライブラリの特徴 .....	94
8.1.3 モジュール構成 .....	95
8.1.4 ADX ファイルシステムの使用方法 .....	95
8.1.5 ISO9660 ファイルの読み込み方 .....	96
8.1.6 AFS ファイルへのアクセス方法 .....	97
8.1.7 AFS ファイルの作成方法 .....	97
8.1.8 インサイドファイルの読み込み方 .....	98
8.1.9 追記したインサイドファイルの読み込み方 .....	98
8.1.10 関数一覧 .....	99
<b>9 DualSpeech .....</b>	<b>100</b>
9.1 概要 .....	100
9.2 動作環境 .....	100
9.3 DUS データ作成の流れ .....	101
9.4 素材の準備 .....	102
9.4.1 素材のフォーマットと準備方法 .....	102
9.4.2 素材準備の注意点 .....	102
9.5 DUS ファイルの作成方法 .....	103
9.6 DUS ファイルの確認方法 .....	104
9.7 関数一覧 .....	105
<b>10 Sofdec 用動画データ作成ツール .....</b>	<b>106</b>
10.1 動作環境 .....	106
10.2 素材の準備 .....	107
10.2.1 素材ファイルのフォーマットと方法 .....	107
10.2.2 フレームレート .....	108
10.2.3 ピクセル縦横比 .....	109
10.2.4 輝度と色 .....	110
10.2.5 画像サイズ .....	110
10.2.6 インタレース .....	111
10.2.7 NTSC / PAL 両用素材 .....	112
10.3 Sofdec データの作成方法 .....	114
10.3.1 エンコードパラメータ推奨値 .....	114
10.3.2 Sofdec データ作成例 .....	115
10.3.3 ビデオエンコード .....	118
10.3.4 オーディオエンコード .....	119
10.3.5 マルチプレクス .....	119
10.3.6 長尺素材 .....	120
10.3.7 長尺ムービー作成手順 .....	121
10.4 CRI MPEG CRAFT コマンド一覧 .....	122
<b>11 ADPCM エンコーダ .....</b>	<b>124</b>



11.1 ツール概要.....	124
11.2 動作環境.....	124
11.3 用語説明.....	124
11.4 データ作成の流れ.....	124
11.5 素材の準備.....	125
11.5.1 素材のフォーマットと準備方法.....	125
11.6 音声データの作成方法.....	125
11.6.1 音声データ作成方法.....	125
<b>12 ミドルウェアの移行手順.....</b>	<b>126</b>
12.1 ライブラリの変更.....	126
12.2 プログラムの変更.....	126
12.3 フレームワーク.....	129
12.4 インターレースドムービー.....	131
12.5 その他.....	131
12.6 付録.....	132
12.6.1 ドアオープンチェック.....	132
12.6.2 リードエラー.....	132
12.6.3 シーク音の軽減.....	133
12.6.4 データ読み込みの高速化.....	134
12.6.5 ストリームジョイントによる読み込み.....	135
<b>用語集.....</b>	<b>137</b>



# 1 ミドルウェアライブラリについて

ここでは、ミドルウェアライブラリについて基本的な部分の説明をします。

## 1.1 ミドルウェアとは

アプリケーションとハードウェアの間に位置するソフトウェア群のことを総称して言います。

Dreamcast のミドルウェアは本書に記述されている物のみとします。

SEGA から提供される Dreamcast 用ミドルウェアのツールやライブラリによって、アプリケーションの開発者は、簡易的に圧縮された映像や音声を再生したり、外部から取り込んだ音声を圧縮することが可能になります。

## 1.2 ミドルウェアライブラリの構成

ミドルウェアライブラリの構成図は、以下のようになります。

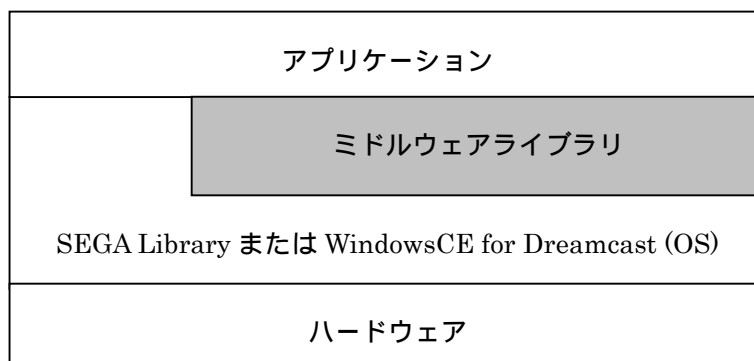


図 1 - 1 ミドルウェアの構成図

## 1.3 ミドルウェアライブラリの機能

ミドルウェアライブラリを構成する機能について説明します。

### 1.3.1 TrueMotion

TrueMotion は Duck 社が開発したムービー圧縮コーデックです。デコード時の CPU 負荷が少なく、画角サイズの大きなムービーに適しています。

デメリットとして、ムービーのデータサイズが大きくなる点に、注意して下さい。

**注 意** 現在、TrueMotion は SEGA Library Ver.2 には対応していませんので、注意して下さい。



---

### 1.3.2 CRI SofdecF/X

---

CRI SofdecF/X は、以下の 2 つのライブラリから構成されています。

(1) CRI SofdecF/X (以下 SofdecF/X)

SofdecF/X は、映像を MPEG1 圧縮し、音声を SofdecAudio 圧縮した動画を再生できます。ビットレートは、150KByte/Sec ~ 600KByte/Sec で、450KByte/Sec 程度で映像の劣化を感じることなく再生できます。

CPU への負荷は、解像度とビットレートに依存します。320 × 240 の解像度で 300KByte/Sec 程度のビットレートであれば、CPU 負荷は 50%程度です。

また、国際標準である MPEG 規格の映像データを再生できますので、市販の MPEG エンコーダを用いて作成したデータの場合でも映像の再生は可能です。

更に、ビジュアルエフェクト、マルチウィンドウ表示、シームレス連続再生、マルチストリーム再生等の特殊な再生が可能です。

(2) CRI SAN (以下 SAN)

SAN は、BMP ファイルを YUV420 フォーマットに変換して、連結したデータを容易に表示することができます。また、表示画面をピクセル単位で合成したり、テクスチャとして表示することができます。

---

### 1.3.3 MPEG/Audio

---

ISO11172 に規定される MPEG1/Audio Layer II フォーマットで圧縮された音声を再生します。CPU 負荷は、ステレオで 15%程度になります。

---

### 1.3.4 WAVE 再生デコーダ

---

非圧縮 PCM、もしくは DreamcastADPCM 圧縮された WAVE ファイルフォーマットの音声データを、GD-ROM から再生します。単独のストリームのみの再生も可能です。

---

### 1.3.5 CRI ADX

---

CRI ADX (以下 ADX) は、以下の 3 つの特徴を持ちます。

- (1) GD-ROM から複数のファイルを同時に再生 (マルチストリーム再生) することが可能です。
- (2) GD-ROM からストリーム再生中に、ゲームデータを読み出すことができます。
- (3) ADX 圧縮、DreamcastADPCM 圧縮、PCM 非圧縮の音声データを再生できます。

ADX 圧縮された音声データ (独自フォーマットを使用したものを指す) は、CD 音質並の再生が可能です。CPU 負荷は、サンプリング周波数 44.1kHz のモノラル音声で約 0.7%です。シームレスループ再生をすることも可能です。

また、この ADX にリンクすることにより SofdecF/X、WAVE デコーダ、MPEG/Audio 等のマルチストリーム再生も可能になります。

---

### 1.3.6 DualSpeech

---

NTT の開発した人間の声に特化した、音声コーデックです。8kHz のモノラルの音声を 6 ~ 8Kbps に圧縮します。CPU 負荷は、エンコードの場合で 10%、デコードの場合で 3%程度になります。



## 2 ミドルウェアライブラリの概要

ミドルウェアライブラリは、動画や音声を容易に再生するためのライブラリです。

### 2.1 ソフトウェア構成

ミドルウェアライブラリのソフトウェア構成は、以下のとおりです。

(1) ミドルウェア API

アプリケーションの開発者は、この API ( Application Program Interface ) を用いることで様々なコーデックの再生・録音を、それぞれ同じインタフェースで実行することが可能です。

現在は、SofdecF/X、WAVE 再生デコーダ、MPEG/Audio、DualSpeech ( デコーダ・エンコーダ ) がこの API に対応しています。ADX は、独自の API により音声の再生を行います。

(2) 各種ミドルウェア

各種コーデックで圧縮された映像や音声を、再生し録音するためのソフトウェアライブラリです。

(3) ミドルウェア基本ライブラリ

ミドルウェアマネージャ、オーディオレンダラ、ビデオレンダラ、オーディオキャプチャ、ビデオキャプチャ等のミドルウェアを再生・録音するための、ミドルウェアライブラリの基本となるライブラリです。

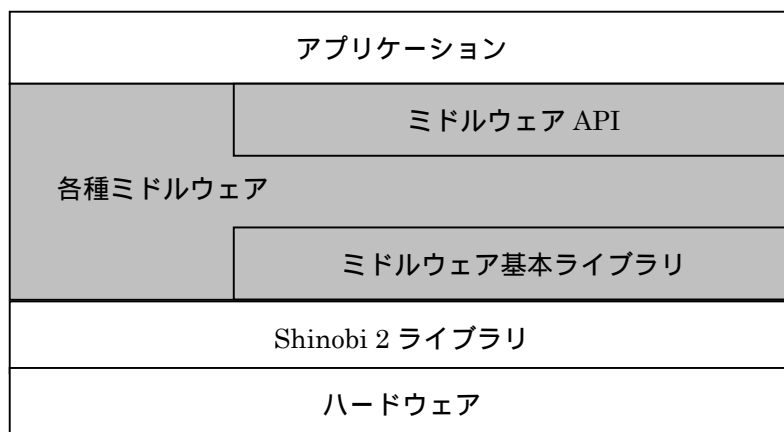


図 2 - 1 ミドルウェアライブラリの構成



## 2.2 共通ヘッダファイル・ライブラリファイル

---

ミドルウェアを再生、録音するために必要なヘッダファイルとライブラリファイルは、Dreamcast SDK に含まれる資料をご参照下さい。

## 2.3 ストリーム再生対応形式ファイルについて

---

- (1) TrueMotion データファイル  
映像を TrueMotion、音声を DK3/DK4 により圧縮しマルチプレクスしたデータ。
- (2) SofdecF/X データファイル  
映像を MPEG/Video、音声を Sofdec/Audio により圧縮しマルチプレクスされたデータです。
- (3) WAVE データファイル  
WAVE フォーマットの非圧縮 PCM 音声データ、Dreamcast ADPCM 圧縮された音声データです。
- (4) MPEG/Audio データファイル  
MPEG1/Audio Layer 方式で圧縮された音声データです。
- (5) ADX データファイル  
ADX 圧縮された音声データです。
- (6) DualSpeech データファイル  
DualSpeech で圧縮された音声データです。



## 2.4 ミドルウェアライブラリの内部構成

ミドルウェアライブラリは、以下の基本モジュールにより構成されています。他に専用のモジュール等もありますが、ここでは割愛させていただきます。

表 2 - 1 ミドルウェアライブラリの内部モジュール

モジュール名	説 明
モジュールマネージャ	デコードに CPU タイムを割り当てます。
ストリームコントローラ	インターリーブされた映像と音声データを読み込みます。
ストリームスプリッタ	インターリーブされた映像と音声データを分離し、デコードに渡します。
ビデオデコーダ	圧縮された映像データを展開し、ビデオレンダラに渡します。
オーディオデコーダ	圧縮された音声データを展開し、オーディオレンダラに渡します。
ビデオレンダラ	展開された映像データを出力します。
オーディオレンダラ	展開された音声データを出力します。

以下に、ミドルウェアライブラリの内部構成図を示します。

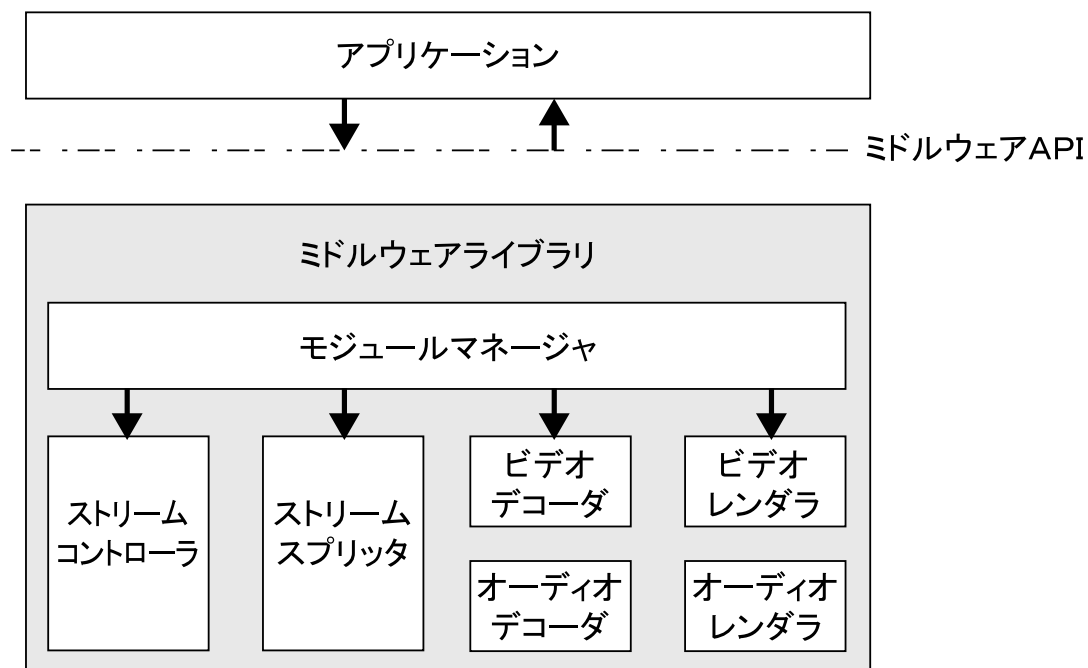


図 2 - 2 ミドルウェアライブラリの内部構成



## 2.5 ミドルウェアの使用方法

ここでは、具体的なミドルウェアの使用方法について順を追って説明します。

### 2.5.1 ミドルウェアライブラリの初期化

ミドルウェアライブラリの初期化は、各コーデックに対応した初期化関数を呼び出さなければなりません。

- **SofdecF/X の場合**

SEGA Library Ver.1 では、割込みスタックを設定するために mwPlyPreInitSofdec 関数を実行する必要があります。

```
mwPlyPreInitSofdec();
sbInitSystem(NJD_RESOLUTION_640x480_NTSCNI, NJD_FRAMEBUFFER_MODE_ARGB8888, 1);
/* サウンドの初期化 */
mwPlyInitSofdec(&iprm);
```

SEGA Library Ver.2 では、割込みスタックの設定が sbInitSystem 関数内で行われますので、mwPlyPreInitSofdec 関数を実行する必要はありません。

```
sbInitSystem();
/* ビデオの初期化 */
/* サウンドの初期化 */
mwPlyInitSofdec(&iprm);
```

- **WAVE の場合**

```
/* システムの初期化・ビデオの初期化・サウンドの初期化 */
mwPlyInitWav(MWD_PLY_SVR_VSYNC);
```

- **TrueMotion の場合**

```
/* システムの初期化・ビデオの初期化・サウンドの初期化 */
mwPlyInitTM();
```

- **MPEG/Audio の場合**

```
/* システムの初期化・ビデオの初期化・サウンドの初期化 */
mwPlyInitMpa(MWD_PLY_SVR_VSYNC);
```

- **DualSpeech の場合**

```
/* システムの初期化・ビデオの初期化・サウンドの初期化 */
mwPlyInitDlsd(MWD_PLY_SVR_VSYNC);
```



## 2.5.2 再生ハンドル

各コーデックによって圧縮された音声やデータは、ミドルウェア再生ハンドルを用いて再生します。手順としては、最初に各コーデック用のミドルウェア再生ハンドルを生成します（このハンドルの生成には、使用するコーデックに対応した API を用います）。

```

MWPLY          plysfd, plywav, plytm, plympa, plydlsd;
MWS_PLY_CPRM_SFD  cprm_sfd;
MWS_PLY_CPRM_TM   cprm_tm;
MWS_PLY_CPRM      cprm_wav, cprm_mpa, cprm_dlsd;

plysfd = mwPlyCreateSofdec(&cprm_sfd);      /* SofdecF/X 再生用ハンドルの生成 */
plywav = mwPlyCreateWav(&cprm_wav);         /* WAVE データ再生用ハンドルの生成 */
plytm  = mwPlyCreateTM(&cprm_tm);          /* TrueMotion データ再生用ハンドルの生成 */
plympa = mwPlyCreateMpa(&cprm_mpa);        /* MPEG/Audio データ再生用ハンドルの生成 */
plydlsd = mwPlyCreateDlsd(&cprm_dlsd);     /* DualSpeech データ再生用ハンドルの生成 */

```

ハンドルの生成には異なる API を使用しますが、再生指示には同じ API を使用することが可能です。

### a. GD-ROM 上のデータを再生する場合

mwPlyStartFname 関数により、動画や音声の再生開始を制御します。

```

mwPlyStartFname(plysfd, "SAMPLE.SFD");      /* SofdecF/X データの再生開始 */
mwPlyStartFname(plywav, "SAMPLE.WAV");     /* WAVE データの再生開始 */
mwPlyStartFname(plytm, "SAMPLE.AVI");      /* TrueMotion データの再生開始 */
mwPlyStartFname(plympa, "SAMPLE.MP2");     /* MPEG/Audio データの再生開始 */
mwPlyStartFname(plydlsd, "SAMPLE.DUS");    /* DualSpeech データの再生開始 */

```

### b. ストリームジョイントでストリーミングしながら再生する場合

mwPlyStartSj 関数により、動画や音声の再生開始を制御します。

```

mwPlyStartSj(plysfd, sj);      /* SofdecF/X データの再生開始 */
mwPlyStartSj(plywav, sj);     /* WAVE データの再生開始 */
mwPlyStartSj(plympa, sj);     /* MPEG/Audio データの再生開始 */
mwPlyStartSj(plydlsd, sj);    /* DualSpeech データの再生開始 */

```

### c. メモリ上のデータを再生する場合

mwPlyStartMem 関数により、動画や音声の再生停止を制御します。

```

mwPlyStartMem(plysfd, addr, len);      /* SofdecF/X データの再生開始 */
mwPlyStartMem(plywav, addr, len);     /* WAVE データの再生開始 */
mwPlyStartMem(plympa, addr, len);     /* MPEG/Audio データの再生開始 */
mwPlyStartMem(plydlsd, addr, len);    /* DualSpeech データの再生開始 */

```

再生方法に関わらず、mwPlyStop 関数により動画や音声の再生停止を制御します。



### 2.5.3 ミドルウェアライブラリのサーバ処理

メイン処理内で実行するサーバ関数として、以下の3つがあります。

- **mwExecMainServer**：音声コーデック用サーバ関数
- **mwPlyExecTexSvr**：テクスチャ転送用サーバ関数
- **mwPlyExecDrawSvr**：映像描画用サーバ関数

上記関数を以下のように呼び出すことにより、ミドルウェアライブラリの内部状態が更新されます。

- 音声コーデック使用時 (WAVE、MPEG/Audio、DualSpeech)

```
while (1) {
    UsrBeginDraw();                // 描画開始
    /* ペリフェラル情報の取得処理 */
    :
    mwExecMainServer();            // ミドルウェアサーバ関数
    /* ポリゴン処理 */
    :
    UsrEndDraw();                  // 描画終了
    UsrWaitVBlank();               // V-blank 待ち
}
```

- 映像コーデック使用時 (MPEG Sofdec、TrueMotion)

**参 照** 以下、フレームワークについては、SEGA Library 2.0 のドキュメントを参照して下さい。

```
UsrInitSync(2);
for (;;) {
    UsrSyncFrame();
    mwPlyExecTexSvr();             // テクスチャ転送用サーバ関数
    UsrBeginDraw();               // 描画開始
    mwPlyExecDrawSvr();           // 映像描画用サーバ関数
    /* ペリフェラル情報の取得処理 */
    :
    /* ポリゴン処理 */
    :
    UsrEndDraw();                 // 描画終了
}
```

使用するグラフィックライブラリ (Ninja や Kamui) のバージョンによって、UsrBeginDraw、UsrEndDraw、UsrWaitVBlank 関数の内容は異なります。

以下に、その例を示します。

- **Ninja Ver.1 の場合**

```
void UsrBeginDraw(void)
{
}

void UsrEndDraw(void)
{
}

void UsrWaitVBlank(void)
{
    njWaitVSync();
}
```



- **Ninja Ver.2 の場合**

```
void UsrBeginDraw(void)
{
    pdExecPeripheralServer();
    njStartDraw();
}

void UsrEndDraw(void)
{
    UsrRenderId[1] = UsrRenderId[0];
    UsrRenderId[0] = njRender();
}

void UsrWaitVBlank(void)
{
    for (;;) {
        stat = njGetRenderStatus(UsrRenderId[1]);
        if (stat == NJD_STATUS_UNDER_RENDER) {
            njWaitVBlank();
        } else {
            return;
        }
    }
}
```

- **Kamui Ver.2 の場合**

```
void UsrBeginDraw(void)
{
    kmBeginScene(&SystemConfig);
    kmBeginPass(&VertexBufferDesc);
}

void UsrEndDraw(void)
{
    kmEndPass(&VertexBufferDesc);
    UsrRenderId[1] = UsrRenderId[0];
    UsrRenderId[0] = kmRender(KM_RENDER_FLIP);
    kmEndScene(&SystemConfig);
}

void UsrWaitVBlank(void)
{
    for (;;) {
        stat = kmGetRenderStatus(UsrRenderId[1]);
        if (stat == KMSTATUS_UNDER_RENDER) {
            kmWaitVBlank();
        } else {
            return;
        }
    }
}
```



### 2.5.4 ミドルウェア再生ハンドルの動作状態

ミドルウェア再生ハンドルの動作状態を以下に示す。

生成した直後は、STOP 状態となる。再生開始後の状態は、PREP -> PLAYING -> PLAYEND と移行する。GD リードエラーが発生すると、ERROR に移行する。

表 2 - 2 ハンドルの状態

状 態	説 明
STOP	再生が停止している状態
PREP	再生の準備をしている状態
PLAYING	再生中の状態
PLAYEND	再生が終了した状態
ERROR	エラーが発生した状態

状態遷移図を以下に示します。

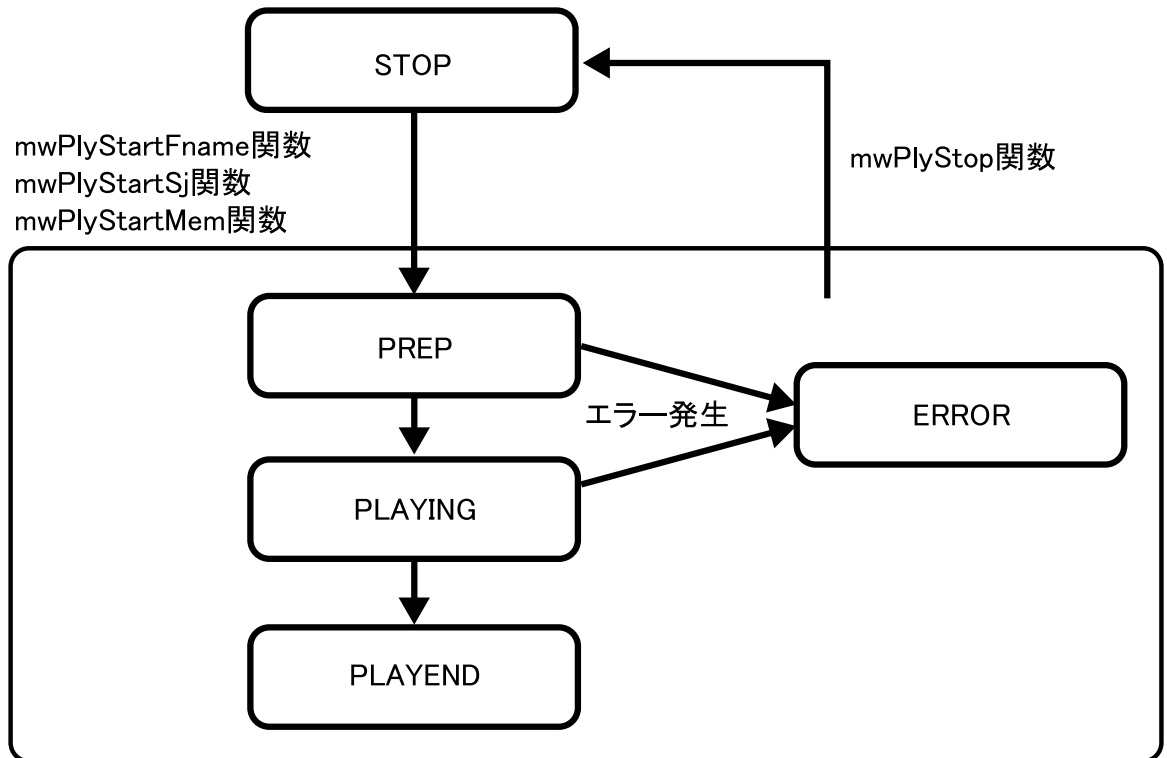


図 2 - 3 状態遷移図



## 2.6 各再生について

ここでは、各再生の方法について説明します。

### 2.6.1 SofdecF/X の再生

以下に、SofdecF/X を再生するサンプルプログラムを示す。

#### ● サンプルプログラム (SEGA Library Ver.1 の場合)

```

/* 表示モード */
#define SYS_MODE      NJD_RESOLUTION_640x480_NTSCNI
#define SYS_FRAME     NJD_FRAMEBUFFER_MODE_ARGB8888
#define SYS_COUNT     1

/* 頂点バッファサイズ(1つをマイナスにすると2Vレーテンシモード) */
#define SFD_VB_OP      -500000
#define SFD_VB_OM      0
#define SFD_VB_TP      20000
#define SFD_VB_TM      0
#define SFD_VB_PT      0

/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY              ply;                /* ミドルウェア再生ハンドル */
    MWE_PLY_STAT       stat;              /* ハンドルの状態 */
    MWS_PLY_CPRM_SFD   cprm;              /* ハンドル生成のパラメータ構造体 */

    mwPlyPreInitSofdec();                  /* 割り込みスタック等の設定 */
    sbInitSystem(SYS_MODE, SYS_FRAME, SYS_COUNT);
    njInitVertexBuffer(VB_OP, VB_OM, VB_TP, VB_TM, VB_PT);
    /* サウンドの初期化 */

    memset(&iprm, 0, sizeof(iprm));        /* 予約メンバのゼロ設定のため */
    iprm.mode          = SYS_MODE;
    iprm.frame         = SYS_FRAME;
    iprm.count         = SYS_COUNT;
    iprm.latency       = MWD_PLY_LATENCY(VB_OP, VB_OM, VB_TP, VB_TM, VB_PT);
    mwPlyInitSofdec(&iprm);                /* ライブラリの初期化 */

    memset(&cprm, 0, sizeof(cprm));        /* 予約メンバのゼロ設定のため */
    cprm.ftype         = MWD_PLY_FTYPE_SFD; /* 映像 + 音声 */
    cprm.dtype         = MWD_PLY_DTYPE_AUTO; /* 画質優先 */
    cprm.max_bps       = 450*1024*8;       /* ビットレート: 450 Kbyte/sec */
    cprm.max_width     = 320;              /* 画像サイズ: 320x480 */
    cprm.max_height    = 480;
    cprm.nfrm_pool_wk  = 3;                /* フレームバッファの枚数 */
    cprm.wksize        = mwPlyCalcWorkCprmSfd(&cprm); /* 作業領域サイズの計算 */
    cprm.work          = syMalloc(cprm.wksize);
    ply = mwPlyCreateSofdec(&cprm);        /* ハンドルの生成 */

    mwPlyStartFname(ply, "SAMPLE.SFD");    /* 再生開始 */
    UsrInitSync(2);
    for (;;) {
        UsrSyncFrame();
        mwPlyExecTexSvr();                /* ミドルウェアライブラリの実行 */
        UsrBeginDraw();                  /* 描画開始 */
        mwPlyExecDrawSvr();              /* ミドルウェアライブラリの実行 */
        stat = mwPlyGetStat(ply);         /* ハンドルの状態の取得 */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
    }
}

```



```

        UshrEndDraw();          /* 描画終了 */
    }
    mwPlyStop(ply);              /* 再生停止 */
    mwPlyDestroy(ply);          /* ハンドルの消去 */
    syFree(cprm.work);
    mwPlyFinishSofdec();         /* ライブラリの初期化 */
}

```

### ● サンプルプログラム (SEGA Library Ver.2 の場合)

```

/* 表示モード */
#define SYS_MODE      NJD_RESOLUTION_640x480_NTSCNI
#define SYS_FRAME     NJD_FRAMEBUFFER_MODE_ARGB8888
#define SYS_COUNT     1

/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY              ply;          /* ミドルウェア再生ハンドル */
    MWE_PLY_STAT       stat;         /* ハンドルの状態 */
    MWS_PLY_CPRM_SFD   cprm;        /* ハンドル生成のパラメータ構造体 */

    sbInitSystem();
    njInitDevice();
    njInitSystem(SYS_MODE, SYS_FRAME, SYS_COUNT);
    :
    njInitVertexBufferEx(&gVbinfo);
    /* サウンドの初期化 */

    memset(&iprm, 0, sizeof(iprm));  /* 予約メンバのゼロ設定のため */
    iprm.mode = SYS_MODE;
    iprm.frame = SYS_FRAME;
    iprm.count = SYS_COUNT;
    iprm.latency = 2;
    mwPlyInitSfDFx(&iprm);          /* ライブラリの初期化 */
    mwPlySetVertexBuffer(njGetVertexBuffDesc());

    memset(&cprm, 0, sizeof(cprm));  /* 予約メンバのゼロ設定のため */
    cprm.compo_mode = MWD_PLY_COMPO_OPEQ; /* 不透明 */
    cprm.ftype = MWD_PLY_FTYPE_SFD;      /* 映像 + 音声 */
    cprm.dtype = MWD_PLY_DTYPE_AUTO;     /* 画質優先 */
    cprm.max_bps = 450*1024*8;           /* ビットレート: 450 Kbyte/sec */
    cprm.max_width = 320;                /* 画像サイズ: 320x448 */
    cprm.max_height = 448;
    cprm.nfrm_pool_wk = 3;               /* フレームバッファの枚数 */
    cprm.wksize = mwPlyCalcWorkCprmSfd(cprm); /* 作業領域サイズの計算 */
    cprm.work = syMalloc(cprm.wksize);
    ply = mwPlyCreateSofdec(&cprm);      /* ハンドルの生成 */

    mwPlyStartFname(ply, "SAMPLE.SFD"); /* 再生開始 */
    UshrInitSync(2);
    for (;;) {
        UshrSyncFrame();
        mwPlyExecTexSvr();              /* ミドルウェアライブラリの実行 */
        UshrBeginDraw();                /* 描画開始 */
        mwPlyExecDrawSvr();             /* ミドルウェアライブラリの実行 */
        stat = mwPlyGetStat(ply);       /* ハンドルの状態の取得 */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        UshrEndDraw();                  /* 描画終了 */
    }
    mwPlyStop(ply);                    /* 再生停止 */
    mwPlyDestroy(ply);                 /* ハンドルの消去 */
    syFree(cprm.work);
}

```



<pre>mwPlyFinishSofdec(); }</pre>	<pre>/* ライブラリの初期化 */</pre>
---------------------------------------	----------------------------

---

## 2.6.2 WAVE の再生

---

以下に、WAVE ファイルを再生するサンプルプログラムを示します。

### ● サンプルプログラム

```
/* アプリケーションメイン関数 */  
void main(void)  
{  
    MWPLY          ply;                /* ミドルウェア再生ハンドル */  
    MWE_PLY_STAT    stat;              /* ハンドルの状態 */  
    MWS_PLY_CPRM    cprm;             /* ハンドル生成のパラメータ構造体 */  
  
    /*  
     * 画面やサウンドの初期化  
     */  
    mwPlyInitWav(MWD_PLY_SVR_VSYNC);  /* ライブラリの初期化 */  
    memset(&cprm, 0, sizeof(cprm));    /* 予約メンバのゼロ設定のため */  
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);  
    cprm.buf = syMalloc(cprm.size);  
    cprm.sample = MWD_PLY_SAMPLE_NUM;  
    ply = mwPlyCreateWav(&cprm);       /* ハンドルの生成 */  
    mwPlyStartFname(ply, "SAMPLE.WAV"); /* 再生開始 */  
    for (;;) {  
        UsrBeginDraw();               /* 描画開始 */  
        stat = mwPlyGetStat(ply);     /* ハンドルの状態の取得 */  
        if ( stat == MWE_PLY_STAT_PLAYEND ) {  
            break;  
        }  
        UsrEndDraw();                 /* 描画終了 */  
        mwExecMainServer();           /* ミドルウェアライブラリの実行 */  
        UsrWaitVBlank();              /* V-Blank 待ち */  
    }  
    mwPlyStop(ply);                   /* 再生開始 */  
    mwPlyDestroy(ply);                /* ハンドルの消去 */  
    syFree(cprm.buf);  
    mwPlyFinishWav();                 /* ライブラリの初期化 */  
}
```



### 2.6.3 TrueMotion の再生

以下に、SEGA Library Ver.1 で TrueMotion を再生するサンプルプログラムを示します。

**注意** SEGA Library Ver.2 では、TrueMotion の再生に対応していませんので注意して下さい。

#### ● サンプルプログラム

```

/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY          ply;          /* ミドルウェア再生ハンドル */
    MWE_PLY_STAT    stat;        /* ハンドルの状態 */

    /*
     * 画面やサウンドの初期化
     */
    mwPlyInitTM();              /* ライブラリの初期化 */
    ply = mwPlyCreateTM(NULL);   /* ハンドルの生成 */
    mwPlyStartFname(ply, "SAMPLE.AVI"); /* 再生開始 */
    for (;;) {
        mwPlyExecTexSvr();       /* テクスチャ転送用サーバ関数 */
        UsrBeginDraw();          /* 描画開始 */
        mwPlyExecDrawSvr();       /* 映像描画用サーバ関数 */

        stat = mwPlyGetStat(ply); /* ハンドルの状態の取得 */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }

        UsrEndDraw();            /* 描画終了 */
        UsrWaitVBlank();         /* V-Blank 待ち */
    }
    mwPlyStop(ply);             /* 再生開始 */
    mwPlyDestroy(ply);          /* ハンドルの消去 */
    mwPlyFinishTM();            /* ライブラリの初期化 */
}

```



## 2.6.4 MPEG/Audio の再生

以下に、MPEG/Audio を再生するサンプルプログラムを示します。

### ● サンプルプログラム

```
#define NUM_HNDL    (1)                                /* 生成するハンドル数          */
/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY          ply;                                /* ミドルウェア再生ハンドル    */
    MWE_PLY_STAT    stat;                              /* ハンドルの状態              */
    MWS_PLY_CPRM    cprm;                              /* ハンドル生成のパラメータ構造体 */

    /*
     * 画面やサウンドの初期化
     */
    mwPlyInitMpa(MWD_PLY_SVR_VSYNC);                  /* ライブラリの初期化          */
    memset(&cprm, 0, sizeof(cprm));                    /* 予約メンバのゼロ設定のため  */
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.libwork = syMalloc(MWD_MPA_CALC_WORK(NUM_HNDL));
    cprm.extsize[MWD_CH_L] = MWD_MPA_EXTRA_SIZE;
    cprm.extsize[MWD_CH_R] = MWD_MPA_EXTRA_SIZE;
    cprm.sample = MWD_PLY_SAMPLE_NUM;
    ply = mwPlyCreateMpa(&cprm);                        /* ハンドルの生成              */
    mwPlyStartFname(ply, "SAMPLE.MP2");                /* 再生開始                    */
    for (;;) {
        UsrBeginDraw();                                /* 描画開始                    */
        stat = mwPlyGetStat(ply);                      /* ハンドルの状態の取得        */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        UsrEndDraw();                                  /* 描画終了                    */
        mwExecMainServer();                             /* ミドルウェアライブラリの実行 */
        UsrWaitVBlank();                                /* V-Blank 待ち                */
    }
    mwPlyStop(ply);                                    /* 再生開始                    */
    mwPlyDestroy(ply);                                 /* ハンドルの消去              */
    syFree(cprm.libwork);
    syFree(cprm.buf);
    mwPlyFinishMpa();                                  /* ライブラリの初期化          */
}
```



## 2.6.5 DualSpeech の再生

以下に、DualSpeech を再生するサンプルプログラムを示します。

### ● サンプルプログラム

```
#define NUM_HNDL    (1)                                /* 生成するハンドル数          */

/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY           ply;                                /* ミドルウェア再生ハンドル    */
    MWE_PLY_STAT     stat;                              /* ハンドルの状態              */
    MWS_PLY_CPRM     cprm;                              /* ハンドル生成のパラメータ構造体 */

    /*
     * 画面やサウンドの初期化
     */
    mwPlyInitDlsd(MWD_PLY_SVR_VSYNC);                  /* ライブラリの初期化          */
    memset(&cprm, 0, sizeof(cprm));                    /* 予約メンバのゼロ設定のため  */
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.libwork = syMalloc(MWD_DLSD_CALC_WORK(NUM_HNDL));
    cprm.extsize[MWD_CH_L] = MWD_DLSD_EXTRA_SIZE;
    cprm.sample = MWD_DLSD_SAMPLE_NUM;
    cprm.limit = MWD_DLSD_MEMORY_LIMIT;
    ply = mwPlyCreateDlsd(&cprm);                      /* ハンドルの生成              */
    mwPlyStartFname(ply, "SAMPLE.DUS");                /* 再生開始                    */
    for (;;) {
        UsrBeginDraw();                                /* 描画開始                    */
        stat = mwPlyGetStat(ply);                      /* ハンドルの状態の取得        */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        UsrEndDraw();                                  /* 描画終了                    */
        mwExecMainServer();                             /* ミドルウェアライブラリの実行 */
        UsrWaitVBlank();                                /* V-Blank 待ち                */
    }
    mwPlyStop(ply);                                    /* 再生開始                    */
    mwPlyDestroy(ply);                                  /* ハンドルの消去              */
    syFree(cprm.libwork);
    syFree(cprm.buf);
    mwPlyFinishDlsd();                                  /* ライブラリの初期化          */
}
```

## 2.6.6 ストリームジョイントによる再生

以下に、ストリームジョイントを使用して再生するサンプルプログラムを示します。

### ● サンプルプログラム

```
#define NUM_HNDL    (1)                                /* 生成するハンドル数          */

/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY           ply;                                /* ミドルウェア再生ハンドル    */
    MWE_PLY_STAT     stat;                              /* ハンドルの状態              */
    MWS_PLY_CPRM     cprm;                              /* ハンドル生成のパラメータ構造体 */
    SJ              sj;                                 /* ストリームジョイントハンドル */
    SJCK             ck, ck2;                          /* チャンク                    */
    Sint8            *buf;                             /* ストリームジョイント作業領域 */
}
```



```

Sint8      *data;                /* データへのポインタ      */
Sint32      size;                /* データサイズ            */
Sint32      pos;                /* データ位置              */
Sint32      len;                /* データ転送量            */
Sint32      nroom;              /* パッファの空き容量      */

/*
 * 画面やサウンドの初期化
 */
mwPlyInitWav(MWD_PLY_SVR_VSYNC);          /* ライブラリの初期化      */
buf = syMalloc(2048 * 24);
sj = SJRBF_Create(buf, 2048 * 24, 0);
memset(&cprm, 0, sizeof(cprm));
cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
cprm.buf = syMalloc(cprm.size);
cprm.sample = MWD_PLY_SAMPLE_NUM;
ply = mwPlyCreateWav(&cprm);              /* ハンドルの生成          */
/*
 * この間で再生するデータを準備する
 */
mwPlyStartSj(ply, sj);                  /* 再生開始                */
pos = 0;

for (;;) {
    UsrBeginDraw();                    /* 描画開始                */
    If (pos < size) {
        nroom = SJ_GetNumData(sj, SJ_LIN_FREE);
        SJ_GetChunk(sj, SJ_LIN_FREE, nroom, &ck); /* 空きチャンクを取得      */
        len = MIN(ck.len, (size - pos));
        memcpy(&ck.data, data[pos], ck.len);
        SJ_SplitChunk(&ck, len, &ck, &ck2);
        SJ_PutChunk(sj, SJ_LIN_DATA, &ck);      /* データチャンクを渡す    */
        SJ_UngetChunk(sj, SJ_LIN_FREE, &ck2);    /* 空きチャンクを戻す      */
        pos += len;
    }
    stat = mwPlyGetStat(ply);          /* ハンドルの状態の取得    */
    if ( stat == MWE_PLY_STAT_PLAYEND ) {
        break;
    }
    UsrEndDraw();                    /* 描画終了                */
    mwExecMainServer();              /* ミドルウェアライブラリの実行 */
    UsrWaitVBlank();                /* V-Blank 待ち            */
}
mwPlyStop(ply);                      /* 再生開始                */
mwPlyDestroy(ply);                  /* ハンドルの消去          */
syFree(buf);
SJ_Destroy(sj);
syFree(cprm.buf);
mwPlyFinishWav();                  /* ライブラリの初期化      */
}

```



## 2.6.7 メモリ上のデータからの再生

以下に、メモリ上のデータを再生するサンプルプログラムを示します。

### ● サンプルプログラム

```
#define NUM_HNDL    (1)                                /* 生成するハンドル数          */

/* アプリケーションメイン関数 */
void main(void)
{
    MWPLY           ply;                                /* ミドルウェア再生ハンドル    */
    MWE_PLY_STAT    stat;                               /* ハンドルの状態              */
    MWS_PLY_CPRM    cprm;                               /* ハンドル生成のパラメータ構造体 */
    Sint8           *data;                             /* データへのポインタ          */
    Sint32           size;                             /* データサイズ                */

    /*
     * 画面やサウンドの初期化
     */
    mwPlyInitWav(MWD_PLY_SVR_VSYNC);                   /* ライブラリの初期化          */
    memset(&cprm, 0, sizeof(cprm));                    /* 予約メンバのゼロ設定のため  */
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.libwork = NULL;
    cprm.sample = MWD_PLY_SAMPLE_NUM;
    ply = mwPlyCreateWav(&cprm);                        /* ハンドルの生成              */
    /*
     * この間で再生するデータを準備する
     */
    mwPlyStartMem(ply, (void *)data, size);            /* 再生開始                    */
    for (;;) {
        UsrBeginDraw();                                /* 描画開始                    */
        stat = mwPlyGetStat(ply);                      /* ハンドルの状態の取得        */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        UsrEndDraw();                                  /* 描画終了                    */
        mwExecMainServer();                            /* ミドルウェアライブラリの実行 */
        UsrWaitVBlank();                               /* V-Blank 待ち                */
    }
    mwPlyStop(ply);                                    /* 再生開始                    */
    mwPlyDestroy(ply);                                 /* ハンドルの消去              */
    syFree(cprm.buf);
    mwPlyFinishWav();                                  /* ライブラリの初期化          */
}
```



## 2.7 SIP ライブラリとの関係

---

セガ・ライブラリ環境において、ミドルウェア録音ライブラリは SIP ( Sound Input Peripheral ) ライブラリ上に実装されています。

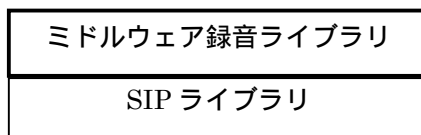


図 2 - 4 SIP ライブラリとの関係

---

### 2.7.1 SIP のサンプリング周波数

---

SIP にはサンプリング周波数のモードとして、8kHz と 11kHz の 2 種類があります。

SIP の正確なサンプリング周波数を以下に示します。

- 8kHz モード ... 8085 Hz
- 11kHz モード ... 11025 Hz

SIP は上記サンプリング周波数に対して、 $\pm 0.5\%$ の個体差があります。



## 2.8 SofdecF/X と ADX との並行再生

ADX 再生ライブラリを使用することにより、SofdecF/X と ADX との並行再生が実現できます。

例えば、BGM を CD ストリーム再生しながらムービーを非同期に再生することで、ゲームシーンからムービーシーンへのシームレスに途切れのない移り変わりを実現することが可能です。しかしながら、音声を並行再生するときに ADX ハンドルの入力バッファ容量を増やさなければなりません。それと同様に、ムービー用のバッファ容量を増やす必要もあります。

SofdecF/X 再生用ハンドルを生成する時に、最大ビットストリーム量を実際のストリーム量よりも大きい値に設定して下さい。具体的な目安として、約 1.5 倍の数値を指定して下さい。これは、シークタイムを約 1 秒として見込んでいるので、データの配置の仕方によっては減らせる可能性があります。また、ムービーが途切れる場合は、この値を増やすことで対処して下さい。

```

/* 44kHz ステレオを 2 ストリーム並行再生する場合の作業領域サイズ */
/* 2 ストリームは、BGM とムービーという意味 */
#define WKSIZ44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)

char    wk44[WKSIZ44S];                /* 作業領域 */
MWPLY   ply;                          /* MWPLY ハンドル */
ADXT     adxt_bgm;                    /* ADXT ハンドル */

/* アプリケーション */
:
:
:
adxt_bgm = ADXT_Create(2, wk44, WKSIZ44S); /* ADXT ハンドルの生成 */
ADXT_SetReloadSct(adxt_bgm, 25);          /* シーク音の軽減のため */

/* MWPLY ハンドルの生成 */
memset(&cprm, 0, sizeof(cprm));           /* 予約メンバのゼロ設定のために必要 */
cprm.ftype = MWD_PLY_FTYPE_MP3;
cprm.dtype = MWD_PLY_DTYPE_AUTO;
cprm.max_bps = 900*1024*8;               /* 通常 600Kbyte/sec の 1.5 倍を指定 */
cprm.max_width = 320;
cprm.max_height = 240;
cprm.nfrm_pool_wk = 3;
cprm.wksize = mwPlyCalcWorkCprmSfd(cprm);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);

ADXT_StartFname(adxt_bgm, "BGM.ADX");    /* 音声再生開始 */

if ( /* イベント発生 */ )
    mwPlyStartFname(ply, "SCENE01.M1V"); /* ムービー再生 */
:
:

```



## 3

## ミドルウェアライブラリのストリームの仕組み

アプリケーションは、ミドルウェア API を使用する事で簡易的に動画や音声の再生と録音を行うことができます。

ミドルウェアには、以下の 2 つのライブラリがあります。

## (1) ミドルウェア再生ライブラリ

ミドルウェア再生ライブラリのシステム構成は、以下のとおりです。

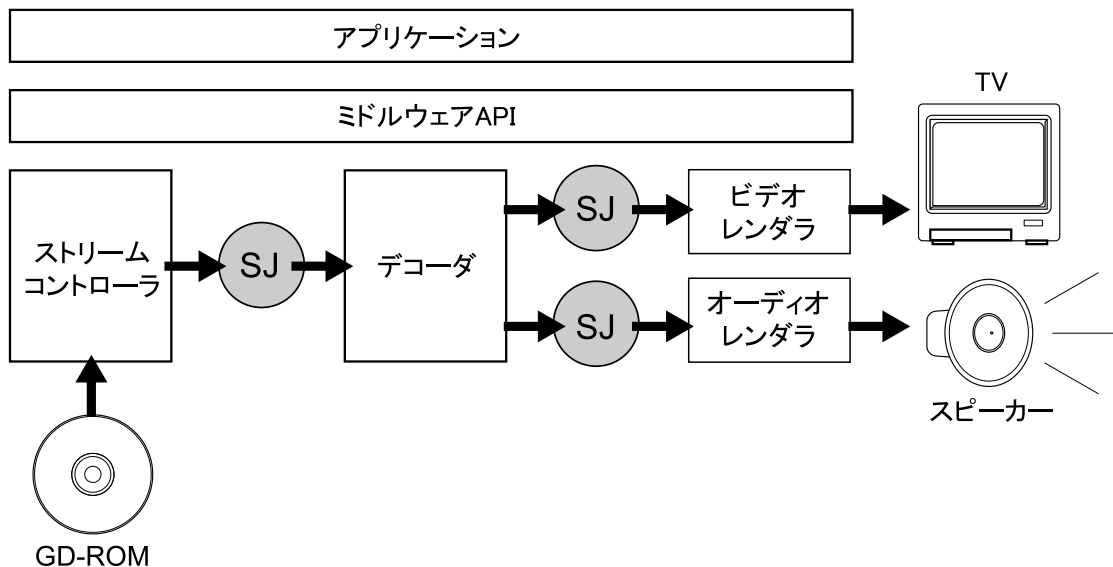


図 3 - 1 ミドルウェア再生ライブラリのシステム構成

## (2) ミドルウェア録音ライブラリ

ミドルウェア録音ライブラリのシステム構成は、以下のとおりです。

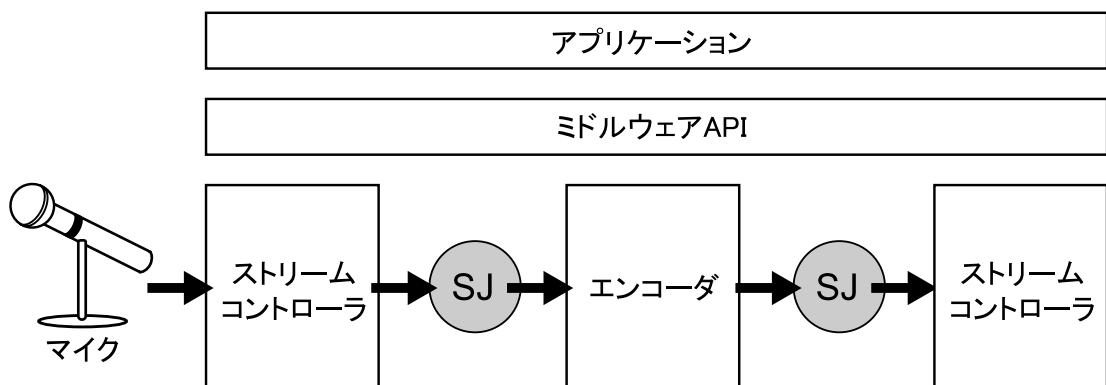


図 3 - 2 ミドルウェア録音ライブラリのシステム構成



## 3.2 コントロールフロー

ミドルウェアの処理は、以下の3つのタイミングで行われます。

表 3 - 1 サーバ処理の種類

サーバ処理	説 明
V-Sync 割込サーバ処理	V-Sync 割込み時に実行されます。 音声データのデコードや音声の出力等、必ず定期的に行う処理を実行します。
メインサーバ処理	メイン処理内で、サーバ関数が呼び出された時に実行されます。 映像データの転送や描画を行います。
アイドルサーバ処理	フレームフリップ関数内で、次のゲームフレームまでの時間に実行されます。 映像データのデコード等、CPU 負荷が変動する処理を行います。

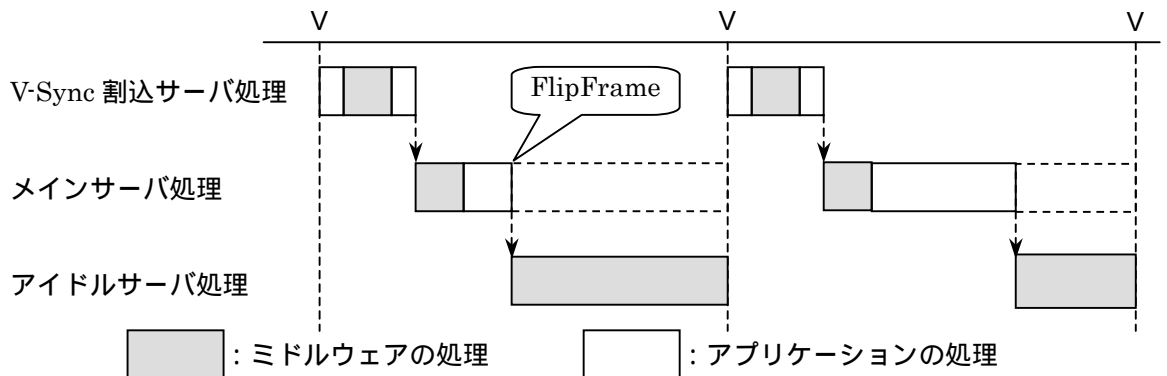


図 3 - 3 コントロールフロー

コーデックによって、処理の行われているタイミングが異なります。

表 3 - 2 各コーデックによるミドルウェアの処理

	Sofdec	TrueMotion	WAV	ADX
V-Sync 割込サーバ処理				
メインサーバ処理			×	×
アイドルサーバ処理		×	×	×

MPEG/Audio、DualSpeech は、WAV と同じになります。

Sofdec における各サーバ処理の内訳は、以下のとおりです。

- ：音声データのデコードと音声出力処理
- ：映像データのテクスチャメモリへの転送と描画処理
- ：映像データのデコード

TrueMotion における各サーバ処理の内訳は、以下のとおりです。

- ：音声出力処理
- ：映像データのデコードと映像出力処理



### 3.3 モジュール構成

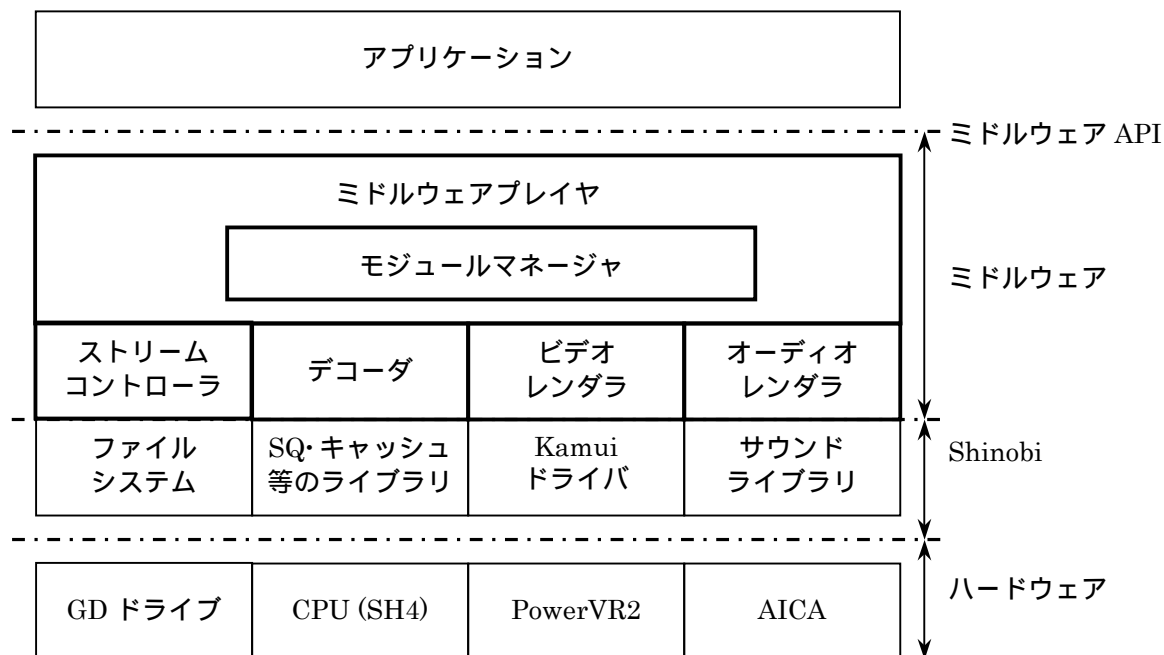
#### 3.3.1 ミドルウェア再生ライブラリ

ミドルウェア再生ライブラリは、以下のモジュールから構成されています。

表 3 - 3 ミドルウェア再生ライブラリの内部モジュール

モジュール	略 称	説 明
ミドルウェアプレイヤー	MWPLY	データの読み込みから出力までを管理し、映像や音声の再生を行います。
モジュールマネージャ	MWMNG	デコーダの CPU タイムを割り当てます。
ストリームコントローラ	MWGSC	映像や音声のデータを読み込みます。
デコーダ		圧縮されたデータを展開します。
ビデオレンダラ	MWRNV	展開された映像データを出力します。
オーディオレンダラ	MWRNA	展開された音声データを出力します。

以下に、ミドルウェア再生ライブラリの内部構成を示します。



SQ：ストアキュー

図 3 - 4 ミドルウェア再生ライブラリの内部構成



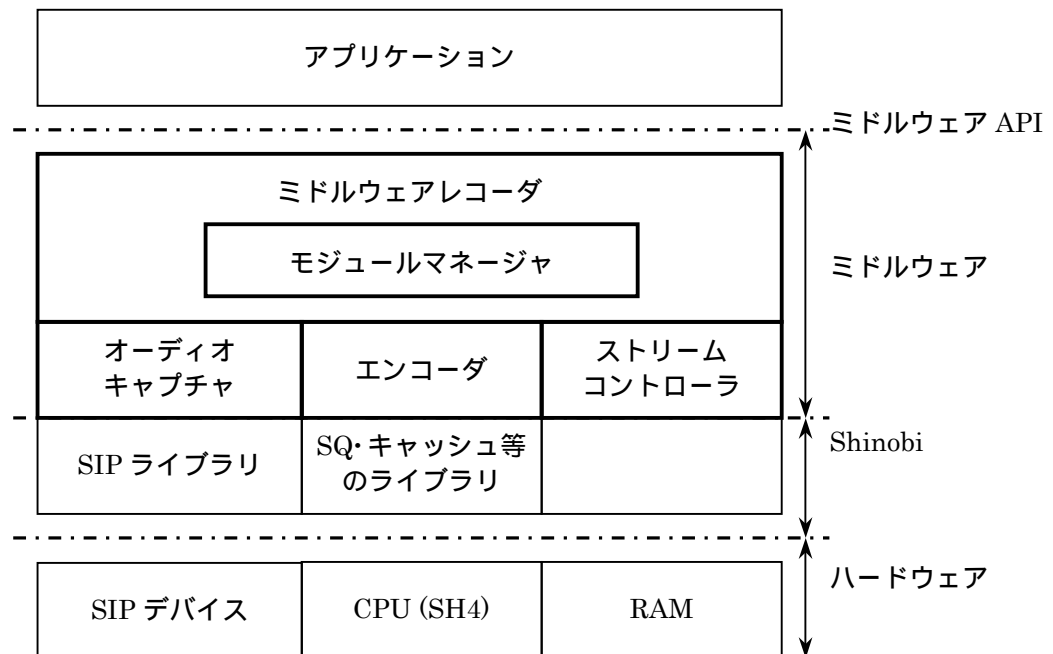
### 3.3.2 ミドルウェア録音ライブラリ

ミドルウェア録音ライブラリは、以下のモジュールから構成されています。

表 3 - 4 ミドルウェア録音ライブラリの内部モジュール

モジュール	略 称	説 明
ミドルウェアレコーダ	MWREC	データの取り込みから出力までを管理し、音声の録音を行います。
モジュールマネージャ	MWMNG	デコーダの CPU タイムを割り当てます。
オーディオキャプチャ	MWCPA	音声データを取り込みます。
エンコーダ		入力されたデータを圧縮します。
ストリームコントローラ	MWGSC	圧縮されたデータを転送します。

以下に、ミドルウェア録音ライブラリの内部構成を示します。



SQ : ストアキュー

図 3 - 5 ミドルウェア録音ライブラリの内部構成



## 4 TrueMotion

ここでは、Truemotion について説明します。

### 4.1 概要

PC 上で圧縮した映像と音声を、Dreamcast 上で再生します。ビットレートは、600KByte/Sec ~ 1.2MByte/Sec です。320×240 で 900Byte/Sec 程度、640×480 で 1.2Byte/Sec 程度のビットレートで比較的良好な画質が得られます。CPU 負荷が低く、320×240 で 25%程度です。

### 4.2 特徴

TrueMotion は、PC 上で圧縮された映像と音声を、Dreamcast 上で再生します。基本的な再生はミドルウェア API によって行うことができますが、より複雑なコントロールを行いたい場合は Duck 社の提供する TruePlay API を使用して下さい。

### 4.3 ソフトウェア構成

ソフトウェアの構成を下記に示します。TrueMotion を再生するためには、ミドルウェアライブラリのマニュアルをご覧ください。

アプリケーション
ミドルウェア API
TrueMotion デコードライブラリ
ミドルウェア基本ライブラリ
Shinobi ライブラリ
ハードウェア

図 4 - 1 TrueMotion のソフトウェア構成

### 4.4 関数一覧

TrueMotion では、以下の関数がサポートされています。  
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
mwPlyCreateTM	TrueMotion 用ハンドルの生成
mwPlyFinishTM	TrueMotion ライブラリの終了処理
mwPlyInitTM	TrueMotion ライブラリの初期化



## 5 SofdecF/X

ここでは、SofdecF/X について説明します。

### 5.1 SofdecF/X の概要

SofdecF/X は、ビジュアルエフェクト等の特殊再生機能を持っています。

### 5.2 ライブラリ構成

SofdecF/X は、以下の 2 つのライブラリから構成されます。

- (1) MPEG SofdecF/X ライブラリ  
MPEG 動画を、ポリゴンと合成し再生することが可能です。
- (2) SAN ライブラリ  
非圧縮のフルカラー静止画を、ピクセル単位で合成することが可能です。

### 5.3 SofdecF/X ライブラリ

#### 5.3.1 MPEG SofdecF/X ライブラリの特長

SofdecF/X ライブラリの特長は、以下のとおりです。

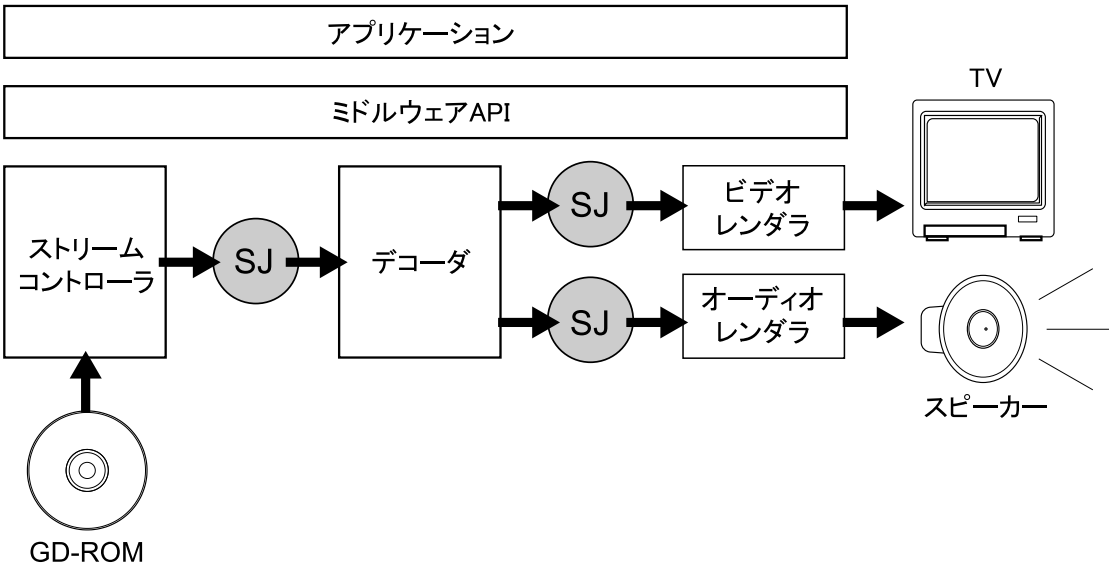
- (1) MPEG Sofdec の上位互換  
MPEG SofdecF/X ライブラリは、MPEG Sofdec ライブラリの上位互換です。  
基本的な API は、ミドルウェア API に準拠しています。ソースプログラムを修正することなく、リコンパイルするだけで MPEG SofdecF/X に移行できます。また、従来の MPEG Sofdec データも再生が可能です。
- (2) ビジュアルエフェクト機能  
アルファ合成や加算合成の機能を持ち、ピクセル単位でポリゴンと合成することができます。
- (3) マルチウィンドウ表示機能  
te 動画イメージの任意位置を切り出して、ディスプレイ上の任意の位置に変形させて表示することが可能です。その際に、複数のウィンドウで表示することができます。
- (4) テクスチャ表示機能  
動画をポリゴンのテクスチャとして表示することが可能です。アルファ合成することができるので、任意の形に抜くことが可能です。
- (5) マルチストリーミング機能  
複数の動画ファイルを同時に並行再生が可能です。最大で 4 ファイルまで、並行再生が可能です。
- (6) シームレス連続再生機能  
別々の動画ファイルをシームレスに接続し、再生することが可能です。また、同じファイルをシームレスにループ再生することも可能です。

**ノート** 現在、映像ファイルのみをサポートしています。



5.3.2 システム構成

MPEG SofdecF/X のシステム構成は以下のとおりです。



SJ：ストリームジョイント

図 5 - 1 MPEG SofdecF/X ライブラリのシステム構成

5.3.3 モジュール構成

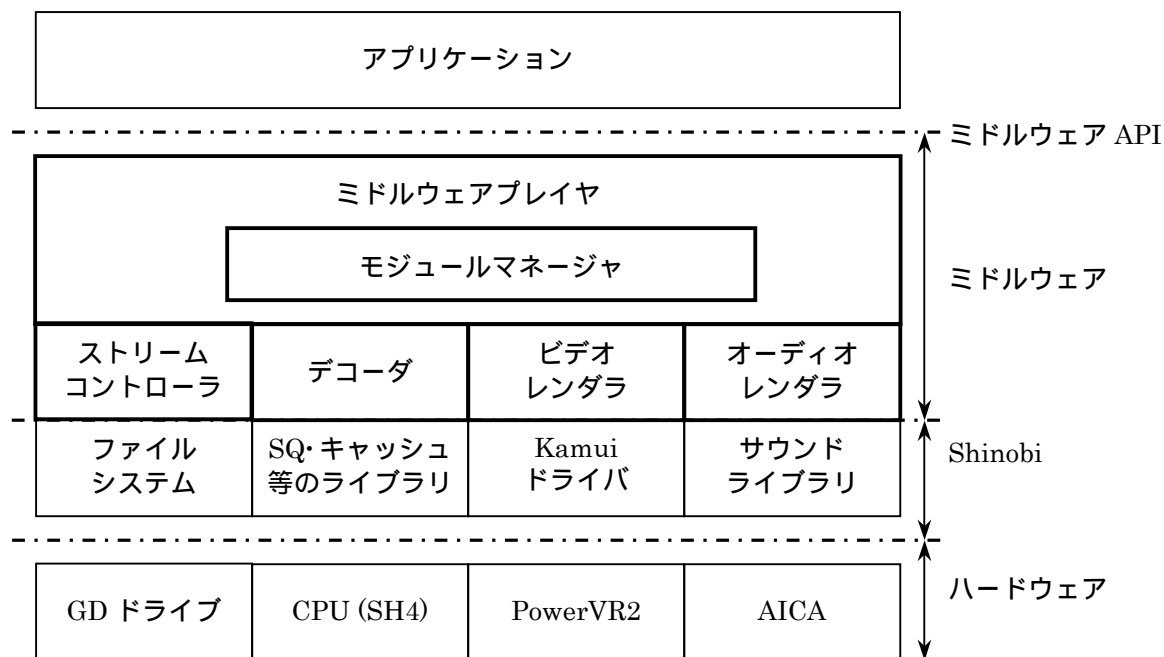
MPEG SofdecF/X は、以下のモジュールから構成されています。

表 5 - 1 内部モジュール

モジュール	略 称	説 明
ミドルウェアプレイヤ	MWPLY	データの読み込みから出力までを管理し、映像や音声の再生を行います。
モジュールマネージャ	MWMNG	デコーダの CPU タイムを割り当てます。
ストリームコントローラ	MWGSC	映像や音声のデータを読み込みます。
デコーダ	CRI_SFD	圧縮されたデータを展開します。
ビデオレンダラ	MWRNV	展開された映像データを出力します。
オーディオレンダラ	MWRNA	展開された音声データを出力します。



以下に、MPEG SofdecF/X の内部構成を示します。



SQ : ストアキュー

図 5 - 2 MPEG SofdecF/X の内部構成

### 5.3.4 コントロールフロー

MPEG SofdecF/X は、以下の 3 つのタイミングで処理が実行されます。

表 5 - 2 サーバ処理の種類

サーバ処理	説 明
V-Sync 割込サーバ処理	V-Sync 割込みによって実行されます。音声データのデコードや音声の出力など、必ず定期的に行う処理を実行します。
メインサーバ処理	メイン処理内で、サーバ関数が呼び出された時に実行されます。映像データの転送や描画を行います。
アイドルサーバ処理	フレームフリップ関数内で、次のゲームフレームまでの待ち時間に実行されます。映像データのデコードを行います。

以下に、コントロールフローを示します。

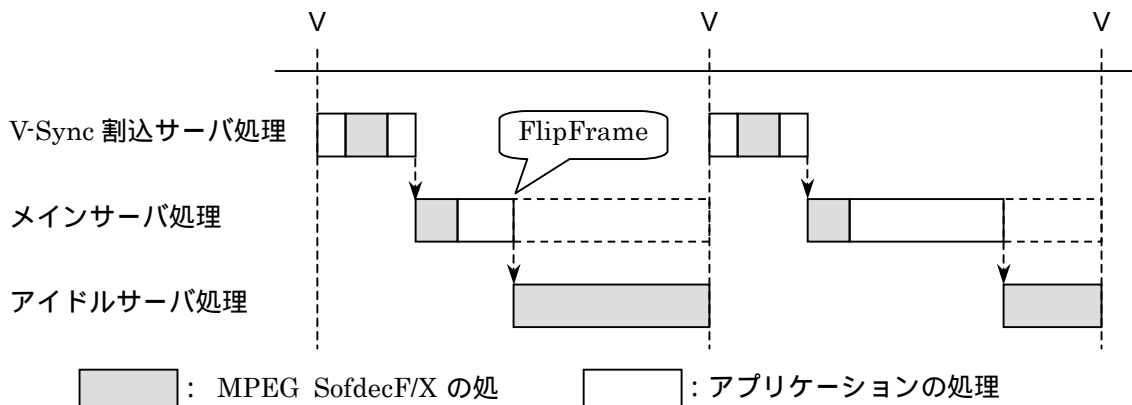


図 5 - 3 コントロールフロー



MPEG SofdecF/X における各サーバ処理の内訳は以下のとおりです。

- : 音声データのデコードと音声出力処理
- : 映像データのテクスチャメモリへの転送と描画処理
- : 映像データのデコード

### 5.3.5 表示タイプ

ハンドル生成パラメータの表示タイプによって表示方法を指定できます。表示タイプには、以下のような型があります。指定された表示タイプによって使用できる関数が異なります。

- 自動表示型            MWD\_PLY\_DTYPE\_AUTO
- ウィンドウ表示型    MWD\_PLY\_DTYPE\_WND
- サーフェス表示型    MWD\_PLY\_DTYPE\_SRF

**ノート** MWD\_PLY\_DTYPE\_WND は従来の MWD\_PLY\_DTYPE\_FULL と同じです。MPEG SofdecF/X から名前が変更されました。

- サンプルプログラム

```
cprm.dtype = MWD_PLY_DTYPE_AUTO;
ply = mwPlyCreateSofdec(&cprm);
```

表 5 - 3 表示タイプ

表示タイプ	表示方法	使用可能な関数
自動 AUTO	動画の解像度から自動的に表示領域の大きさを決定します。バイリニアフィルタによるボケが少なくなるように、UV 値を調節します。	mwPlySetDispPos
ウィンドウ WND	矩形のウィンドウ表示を行います。ウィンドウ全体に対しパラメータを設定します。扱えるウィンドウは 1 つです。	mwPlySetDispPos mwPlySetDispSize mwPlySetBright mwPlySetBrightOfst
サーフェス SRF	ウィンドウの 4 頂点を独立に制御できます。マルチウィンドウ表示を行うためのモードです。	MwPlySetSrfPos MwPlyGetSrfPos MwPlySetSrfBright MwPlyGetSrfBright mwPlySetSrfBrightOfst mwPlyGetSrfBrightOfst mwPlySetImgPos mwPlyGetImgPos



## (1) 自動表示型

自動表示型は、バイリニアフィルタによるボケを軽減し、シャープな映像が得られるように自動的に表示サイズを調整します。例えば、320×240のムービーは、639×479ドットで表示されます。従って、バックグラウンドカラーが設定されていると、画面の右端と下に1ドット分にバックグラウンドカラーが表示されてしまいます。以下に動画の解像度と表示サイズの関係を示します。

表 5 - 4 X 方向の動画サイズと表示サイズの関係

動画 X サイズ	320	352	480	640	左記以外
表示 X サイズ	639	640	640	640	640

← 左右が 16 ドットずつ表示されない

表 5 - 5 Y 方向の動画サイズと表示サイズの関係

動画 Y サイズ	224	240	320	336	448	480	左記以外
表示 Y サイズ	447	479	320	448	448	480	480

## ● バイリニアフィルタ処理について

バイリニアフィルタ処理は、UV 値を 0.0～1.0 として設定すると、下図のような演算が行われ全体的にぼかした映像になります。

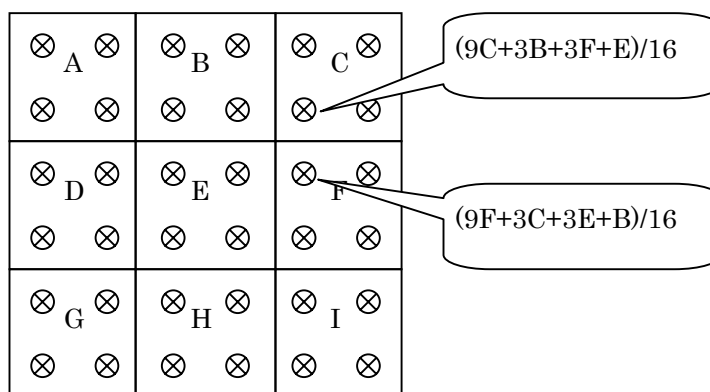


図 5 - 4 通常のバイリニアフィルタ処理

MPEG SofdecF/X では、2 倍に拡大するときは UV 値を下図のように設定し、ボケのすくない映像で再生できます。この方法では (2 倍 - 1) ドットの点しか生成できないため、320×240 の画像は 639×479 として表示されます。

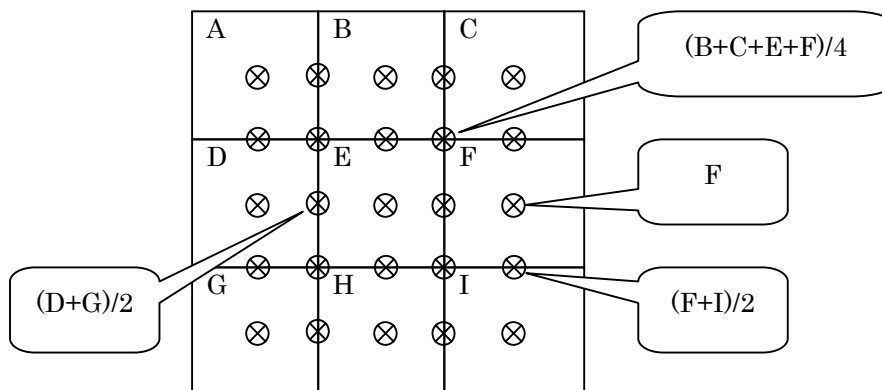


図 5 - 5 MPEG SofdecF/X におけるバイリニア処理



## (2) ウィンドウ表示型

ウィンドウ表示型は、矩形のウィンドウ単位で制御できます。

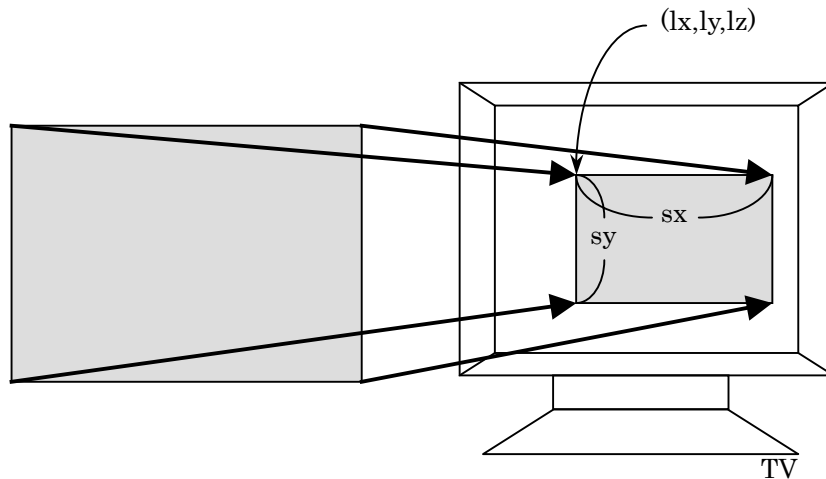


図 5 - 6 ウィンドウの制御

### ● サンプルプログラム

```

ply = mwPlyCreateSofdec(&cprm);
mwPlySetDispPos(ply, lx, ly);           // 表示位置の設定
mwPlySetDispSize(ply, sx, sy);         // 表示サイズの設定
mwPlySetDispZ(ply, lz);                // 奥行きの設定
mwPlySetBright(ply, bright);           // 輝度の設定
mwPlySetBrightOfst(ply, bofst);        // 輝度オフセットの設定
mwPlyStartFname(ply, "sample.sfd");
UsrInitSync(2);                        // フレーム同期システムの初期化
for (;;) {
    UsrSyncFrame();                    // フレーム同期
    mwPlyExecTexSvr();                 // テクスチャ転送用サーバ関数
    UsrBeginDraw();                   // 描画開始
    mwPlyExecDrawSvr();                // 映像描画用サーバ関数
    if ( ウィンドウの移動 ) {
        mwPlySetDispPos(ply, lx++, ly++);
        mwPlySetDispZ(ply, lz++);
    }
    if ( ウィンドウサイズの変更 )
        mwPlySetDispSize(ply, sx++, sy++);
    if ( 輝度の変更 )
        mwPlySetBright(ply, bright++);
    if ( 輝度オフセットの変更 )
        mwPlySetBrightOfst(ply, bofst++);

    /* ポリゴンの描画 */

    UsrEndDraw();                     // 描画終了
}

```



## (3) サーフェス表示型

サーフェス表示型は、ウィンドウの各4頂点を独立に以下のパラメータを設定できます。この表示タイプは映像を複数のウィンドウに表示できます。

## a. 表示位置

X座標、Y座標、Z座標を設定できます。X座標、Y座標はフレームバッファの座標値です。左上が(0.0, 0.0)、右下が(639.0, 479.0)になります。Z座標を指定しますが、透視変換は行いません。

## b. 頂点輝度

アルファ、赤、緑、青の強さを指定します。1.0から0.0までの値を設定します。映像データに対して、この設定値が乗じられます。これらの値を0.0から1.0まで変化させると黒からのフェードイン、1.0から0.0まで変化させるとフェードアウトできます。また、アルファ値は、背後の映像との混合比率になります。これを変化させることによりディゾルブ効果が得られます。

## c. 頂点輝度オフセット

アルファ、赤、緑、青の加算成分を指定します。1.0から0.0までの値を設定します。映像データに対しこの設定値と255を乗じた値が加算されます。これらの値を0.0から1.0まで変化させると白へのフェードアウト、1.0から0.0まで変化させると白からフェードインできます。

## d. 映像切り出し位置

各頂点对応する映像データの位置を指定します。UV座標に相当しますが、指定する値は、映像データに対するピクセル単位での位置です。

また、各頂点のパラメータは‘Z’を描くような順序で設定します。

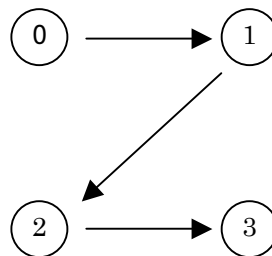


図5-7 サーフェス型の各頂点の設定順序

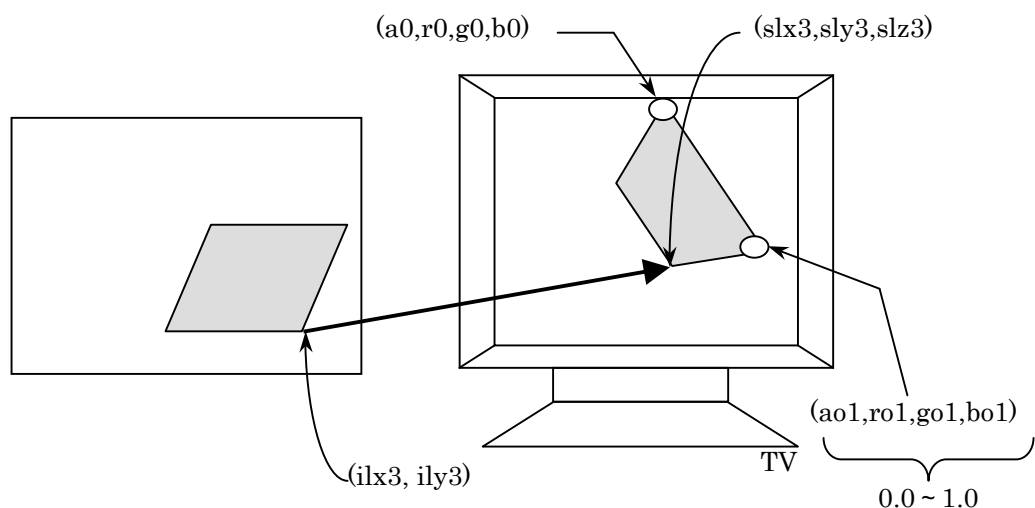


図5-8 サーフェスの制御



- サンプルプログラム

```
ply = mwPlyCreateSofdec(&cprm);

/* イメージ位置の設定 */
mwPlySetImgPos(ply, 0, ilx0, ily0);
mwPlySetImgPos(ply, 1, ilx1, ily1);
mwPlySetImgPos(ply, 2, ilx2, ily2);
mwPlySetImgPos(ply, 3, ilx3, ily3);

/* サーフェス位置の設定 */
mwPlySetSrfPos(ply, 0, slx0, sly0, slz0);
mwPlySetSrfPos(ply, 1, slx1, sly1, slz1);
mwPlySetSrfPos(ply, 2, slx2, sly2, slz2);
mwPlySetSrfPos(ply, 3, slx3, sly3, slz3);

/* 輝度の設定 */
mwPlySetSrfBright(ply, 0, a0, r0, g0, b0);
/* 輝度オフセットの設定 */
mwPlySetSrfBrightOfst(ply, 1, ao1, ro1, go1, bo1);

mwPlyStartFname(ply, "sample.sfd");
```



## 5.3.6 マルチウィンドウ再生

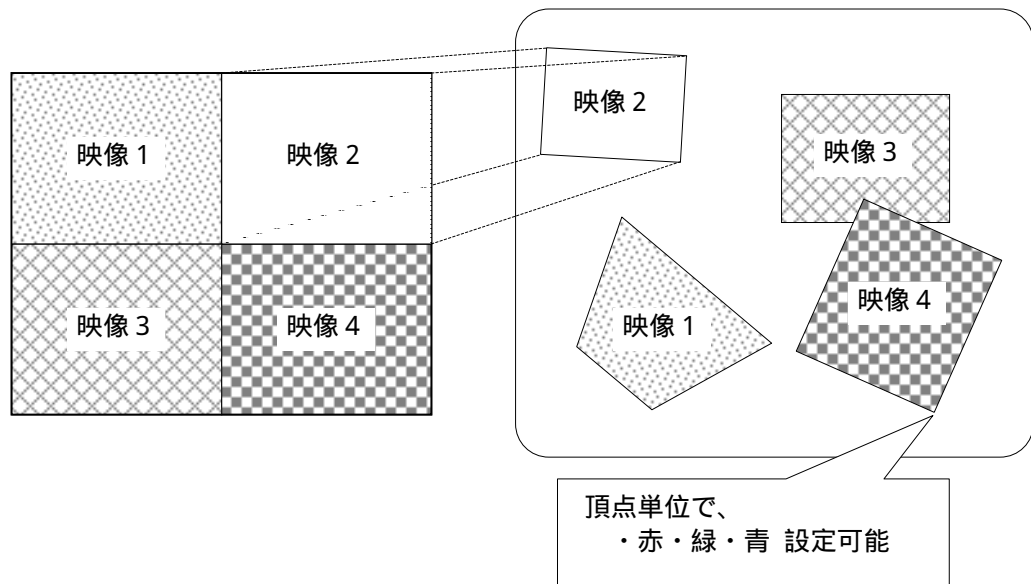


図 5 - 9 マルチウィンドウ表示

サーフェス表示型によりマルチウィンドウ表示を行うことができます。以下の手順でマルチウィンドウ表示を行います。

## (1) サーフェスモードの設定

ハンドル生成時に表示モードをサーフェスモードにします。

```
cprm.dtype = MWD_PLY_DTYPE_SRF;
ply = mwPlyCreateSofdec(&cprm);
```

## (2) サーフェスポイント用バッファの確保と設定

サーフェスポイントとは、ウィンドウの各頂点を示し、上記のパラメータを持ちます。ウィンドウは、4つのサーフェスポイントから構成されます。例えば、25個のウィンドウを表示するためには、100個のサーフェスのポイントが必要になります。

```
size = mwPlyCalcSrfBufSize(ply, 4*25); // ポイントバッファサイズの計算
buf = syMalloc(size);
mwPlySetSrfPntBuf(ply, 4*25, buf, bsize); // ポイントバッファの設定
:
/* 10 番目のウィンドウの設定 */
mwPlySetSrfPos(ply, 10*4 + 0, lx0, ly0, lz0); // ポイント番号に注意
mwPlySetSrfPos(ply, 10*4 + 1, lx0, ly0, lz0);
mwPlySetSrfPos(ply, 10*4 + 2, lx0, ly0, lz0);
mwPlySetSrfPos(ply, 10*4 + 3, lx0, ly0, lz0);
```



### 5.3.7 合成

#### (1) 特 長

MPEG SofdecF/X は、ビジュアルエフェクトのための合成機能を持ちます。

- アルファ付きムービーをポリゴンと合成  
アルファ付きのムービーをポリゴンの前景に表示し、ピクセル単位で合成することができます。
- 様々な合成方式をサポート（半透明合成 / 加算合成 / ルミナンスキー合成 / アルファ合成）  
状況に応じた複数の合成方式をサポートしています。方式の違いによってCPU負荷や機能が異なります。
- 合成モードの途中切り替え可能  
ムービーの途中で合成モードを切り換えることができます。カットごとに適切な合成方式を選択し切り換えることができます。
- フェードイン・アウト可能  
映像全体にアルファ値を持つことができます。CPU 増加させることなくフェードイン・フェードアウトすることができます。この機能を用いてスムーズにビジュアルエフェクトを開始することができます。
- 専用ツールにより AVI ファイルを変換  
アルファ付き AVI ファイルから CRI MPEG CRAFT を用いて合成用のムービーデータを作成します。

#### (2) 合成方式

MPEG SofdecF/X の合成方法として以下の方法があります。

- a. 不透明表示  
映像を合成せずに描画します。
- b. 半透明表示（ウィンドウ全体でのアルファ合成）  
ウィンドウ単位でアルファブレンディングします。
- c. 加算合成  
ポリゴンの RGB 値とムービーの RGB 値を単純に加算します。
- d. アルファ合成（ピクセル単位のアルファ合成）  
ポリゴンの RGB 値とムービーの RGB 値をピクセル単位でアルファブレンディングします。

表 5 - 6 SofdecF/X の使用リソースと負荷

	使用リソース	レンダリング負荷	CPU 負荷の増加
半透明・加算合成	なし	半透明 1 枚	なし
アルファ合成	256 個のパレット (ARGB8888)	半透明 2 枚	5 ~ 10%



- アルファブレンディングとは

アルファブレンディングは、混合値 により以下のように表現されます。

$$\text{output} = \alpha \cdot s + (1 - \alpha) \cdot d$$

output : 合成後の RGB 値

$\alpha$  : 混合比率

s : ムービーの RGB 値

d : ポリゴンの RGB 値

ポリゴンの RGB 値とは、ムービーよりも奥にあるオブジェクトが描画されたフレームバッファの RGB です。

(3) アルファ合成の仕組み

ピクセル単位のアルファ合成は、以下のように 2 枚の半透明ポリゴンを描画することによって実現されています。マスクサーフェスを描画（乗算処理）後、映像サーフェスを加算しています。

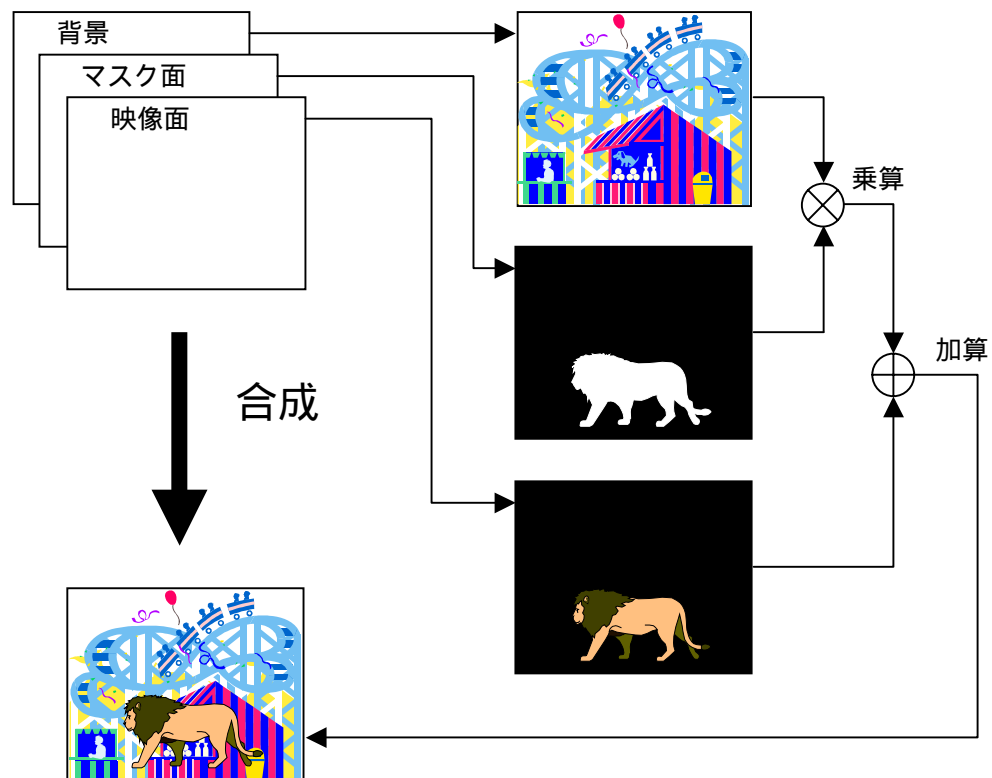


図 5 -10 合成の仕組み



SofdecF/X では、ハンドルから以下の 2 枚のテクスチャサーフェスを取得できます。

- 映像サーフェス  
矩形形式の YUV422 テクスチャ。デコードされた映像を格納。
- マスクサーフェス  
Twiddle 形式の 8bit/pixel のパレットテクスチャ。マスクイメージを格納。

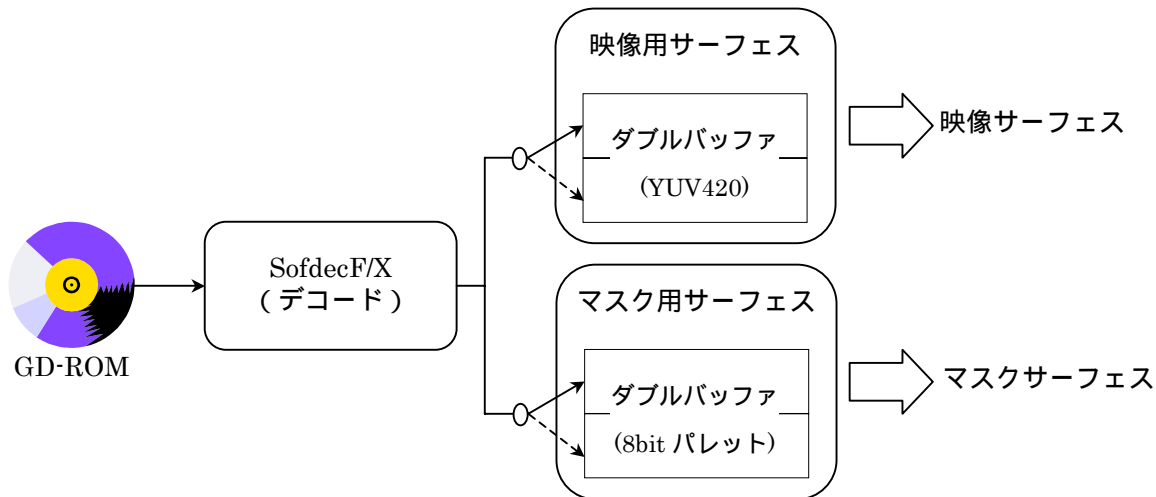


図 5 - 11 SofdecF/X のテクスチャサーフェス

#### (4) 合成モード

MPEG SofdecF/X は以下の合成モード (コンポジションモード) を持ちます。ハンドル生成パラメータの `cprm.compo_mode` メンバによって設定します。

```
cprm.compo_mode = MWD_PLY_COMPO_~; (合成モード)
ply = mwPlyCreateSofdec(&cprm);
```

- 不透明表示モード (MWD\_PLY\_COMPO\_OPEQ)  
不透明ポリゴンとしてムービーを表示します。デフォルトのモードです。
- 半透明合成モード (MWD\_PLY\_COMPO\_TRNSP)  
半透明ポリゴンとしてムービーを表示します。ピクセル単位の合成はできませんが、ウィンドウ全体としてのアルファ合成はできます。
- 加算合成モード (MWD\_PLY\_COMPO\_ADD)  
映像全体を加算します。従って、黒い部分が完全に抜けて見えます。光のエフェクトなどに向いています。
- ルミナンスキー合成モード (MWD\_PLY\_COMPO\_LUMI)  
輝度に応じてアルファ値が変化します。暗い部分ほど透過し、明るい部分は不透明になります。デフォルトでは、下記の設定になっています。mwPlySetAlphTbl 関数によって、輝度からアルファ値への変換テーブルを設定できます。  
0 ~ 15 : 完全に透明  
16 ~ 100 : アルファ値が 0.0 (透明) から 1.0 (不透明) まで徐々に増加  
100 ~ 255 : 完全に不透明
- 3 値アルファ合成モード (MWD\_PLY\_COMPO\_ALPH3)  
不透明・半透明 (アルファ値は 50%)・透明の 3 段階の合成ができます。量子化ビット数が約 7 ビット程度の画質になります。



- f. 5 値アルファ合成モード (MWD\_PLY\_COMPO\_ALPH5)  
透明 (0%)・25%・50%・75%・不透明 (100%) の 5 段階の合成ができます。しかしながら、映像の品質は劣化します。
- g. フルアルファ合成モード (MWD\_PLY\_COMPO\_ALPH256)  
256 段階のアルファ値によって合成できます。最高の品質で合成できますが、作業領域がほぼ 2 倍近く必要になります。小さなムービーに向いています。
- h. 混合合成モード (MWD\_PLY\_COMPO\_MIX)  
フルアルファ合成モード以外の合成モードを、mwPlySetCompoMode 関数によって再生途中で切り換えることができます。

各合成モードにおける使用リソースと CPU 負荷とレンダリング負荷を示します。これは、あくまで目安であり、ムービーデータにより変動します。

- 素材条件

解像度 : 256 × 256  
データレート : 300Kbyte/sec  
GOP パターン : N=6, M=1

表 5 - 7 各合成モードの使用リソースと負荷

合成モード	CPU 負荷	レンダリング負荷	テクスチャ	パレット
不透明表示	40%	不透明 1 枚	YUV422	未使用
半透明	40%	半透明 1 枚	YUV422	未使用
加算合成	40%	半透明 1 枚	YUV422	未使用
ルミナンス合成	50%	半透明 2 枚	YUV422PLT8	使用
3 値アルファ合成	50%	半透明 2 枚	YUV422PLT8	使用
5 値アルファ合成	50%	半透明 2 枚	YUV422PLT8	使用
フルアルファ合成	65%	半透明 2 枚	YUV422PLT8	使用
混合合成			YUV422PLT8	使用

- 使用リソース

YUV422 テクスチャ : 256Kbyte (256 × 256、2 枚)  
8 ビットパレットテクスチャ : 128Kbyte (256 × 256、2 枚)  
パレット : ARGB8888、768 番目から 256 エントリ



## (5) 加算合成

加算合成は、表示されている映像に対して、ムービーのイメージデータを単純に加算しています。光を利用したエフェクトに向いています。

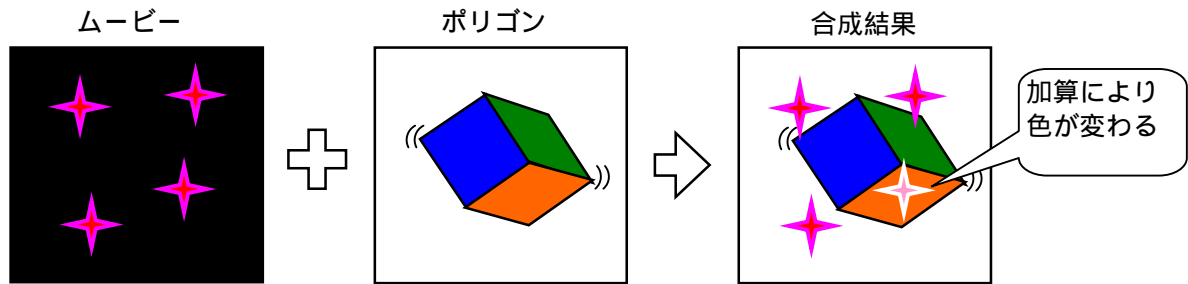


図 5 -12 加算合成

## ● サンプルプログラム

```
cprm.compo_mode = MWD_PLY_COMPO_ADD;          // 加算合成モード
:
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "effect.sfd");
:
```

## (6) ルミナンスキー合成

ルミナンスキー合成は、輝度に応じた混合比率により合成しています。黒い部分を抜く事ができます。炎・煙・稲妻などの黒い部分がないエフェクトに効果的です。

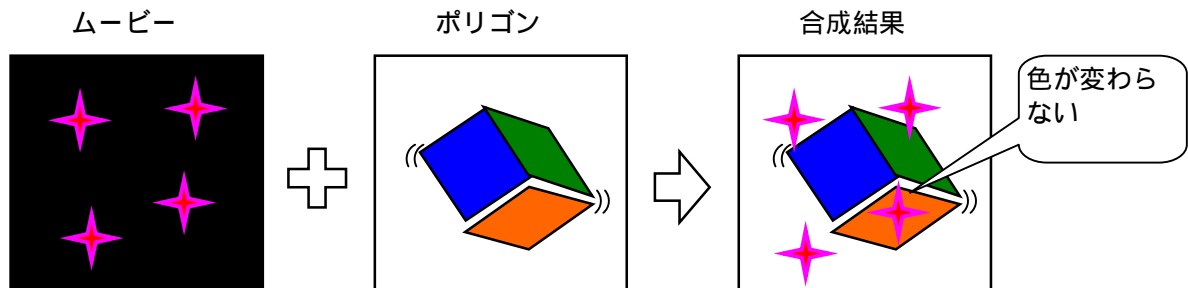


図 5 -13 ルミナンスキー合成

## ● サンプルプログラム

```
cprm.compo_mode = MWD_PLY_COMPO_LUMI;         // ルミナンスキー合成モード
:
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "effect.sfd");
:
```



## (7) 3 値アルファ合成

3 値アルファ合成は、不透明・半透明・透明の3段階のアルファ値を扱うことができます。アルファ情報を映像データ内に埋め込むため、映像の品質は7ビット相当の画質になります。セル画的な合成に向きます。また、ルミナンス合成では合成できない黒い映像を合成することができます。

## ● サンプルプログラム

```
cprm.compo_mode = MWD_PLY_COMPO_ALPH3;          // 3 値アルファ合成モード
:
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "effect.sfd");
:
```

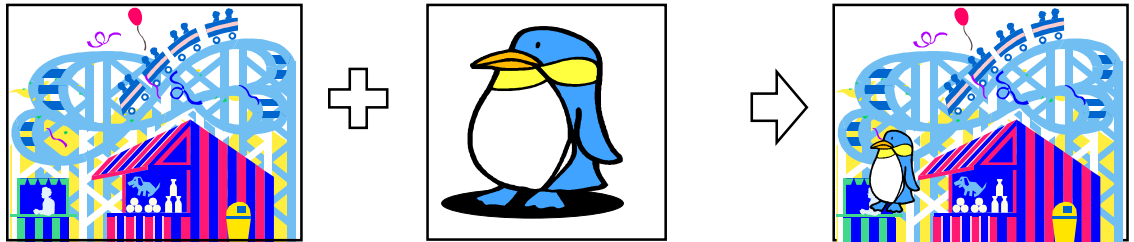


図 5 -14 3 値アルファ合成

## (8) 5 値アルファ合成

5 値アルファ合成は、0%(透明)、25%、50%、75%、100%(不透明)の5段階のアルファ値によって合成することができます。半透明を必要とするビジュアルエフェクト的な合成ができますが、アルファ情報を映像データ内に埋め込むため、映像の品質は5ビット相当の画質となってしまいます。

## ● サンプルプログラム

```
cprm.compo_mode = MWD_PLY_COMPO_ALPH5;          // 5 値アルファ合成モード
:
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "effect.sfd");
:
```

## (9) 256 値アルファ合成

256 段階のアルファ合成を行うことができます。最高の品質の合成を行うことができますが、CPU 負荷が増加します。また、メモリなどのリソースも他の合成方式よりも必要です。



## (10) 合成モードの切り替え

ムービー中のカット毎に合成モードを切り換えることができます。256 値アルファ合成以外の合成モードを切り換えることができます。ハンドル生成時に合成モードを MW\_PLY\_COMPO\_MIX に設定して下さい。

mwPlySetCompoMode 関数によって合成方式を変更できますが、エフェクトコールバック関数内で実行して下さい。このコールバック関数は、マスクを生成する直前に呼び出されます。引数としてフレーム番号を渡しますので、この番号に従って合成モードを変更して下さい。

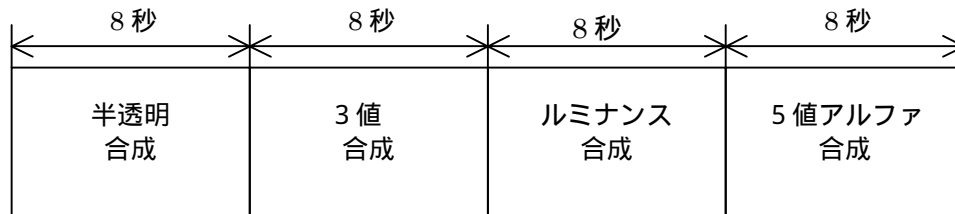


図 5 -15 合成モードの切り替え

## ● 合成モードの切り替えサンプル

```
void user_chg_compo(void *obj, Sint32 fno, Sint32 time, Sint32 tunit)
{
    MWPLY ply=(MWPLY)obj;

    if ( fno == 0 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_TRNSP);
    } else if ( fno == 1*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_LUMI);
    } else if ( fno == 2*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_ALPH3);
    } else if ( fno == 3*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_ALPH5);
    }
}

void main(void)
{
    :
    cprm.compo_mode = MWD_PLY_COMPO_MIX;
    ply = mwPlyCreateSofdec(&cprm);
    mwPlyEntryFxCb(ply, user_chg_compo, (void *)ply);
    mwPlyStartFname(ply, "vfx01.mlv");
    :
    :
}
```



## (11) フェードイン・アウト

表示ウィンドウの各頂点のアルファ値を操作することにより、動画のフェードイン・アウトを行うことができます。合成方式は不透明(MWD\_PLY\_COMPO\_OPEQ)以外を指定して下さい。また、表示タイプはサーフェス表示型でなければなりません。

## ● サンプルプログラム

```

cprm.dtype = MWD_PLY_COMPO_TRNSP;           // 半透明合成モード
cprm.dtype = MWD_PLY_DTYPE_SRF;             // サーフェス表示型
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "vfx01.mlv");
UsrInitSync(2);                             // フレーム同期システムの初期化
for (;;) {
    UsrSyncFrame();                          // フレーム同期
    mwExecTexSvr();                          // テクスチャ転送用サーバ関数
    UsrBeginDraw();                          // 描画開始
    mwExecDrawSvr();                         // 映像描画用サーバ関数

    if ( フェードイン ) {
        a += 1.0f / 60.0f;                  // 1秒でフェードイン
        a = ( a < 1.0f ) ? a : 1.0f;
        for (i=0; i<4; i++)
            mwPlySetSrfBright(ply, i, a, 1.0, 1.0, 1.0); // アルファ値の設定
    }
    if ( フェードアウト ) {
        a -= 1.0f / 60.0f;                  // 1秒でフェードアウト
        a = ( a > 0.0f ) ? a : 0.0f;
        for (i=0; i<4; i++)
            mwPlySetSrfBright(ply, i, a, 1.0, 1.0, 1.0); // アルファ値の設定
    }

    UsrEndDraw();                            // 描画終了
}

```

## 5.3.8 テクスチャムービー

SofdecF/X (MPEG、SAN) は、Kamui ドライバを使用して映像用テクスチャサーフェス、マスク用テクスチャサーフェスを確保しています。アプリケーションは、各ライブラリのハンドルからサーフェスを取得し、アプリケーションのオブジェクトサーフェスに映像とマスクを貼ることができます。

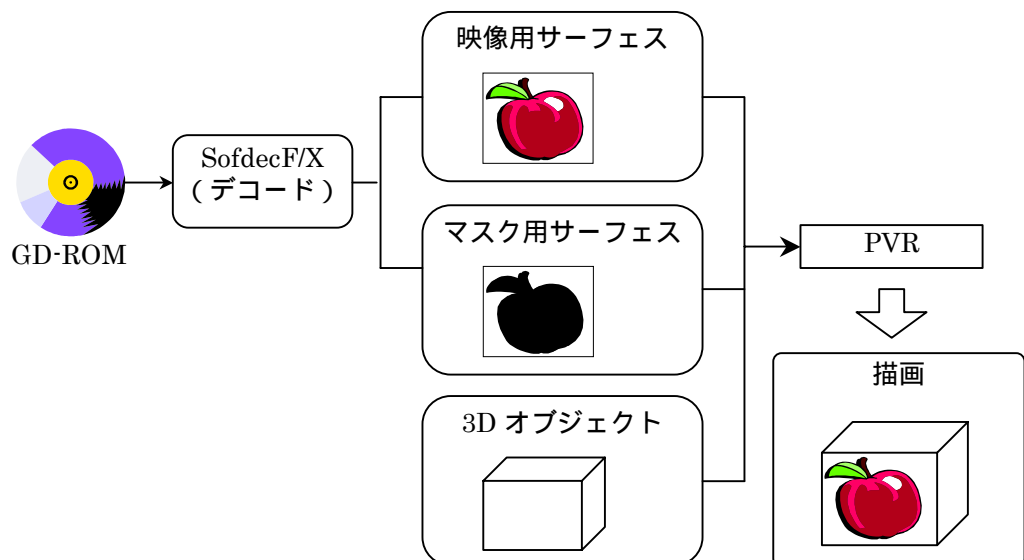


図 5 -16 テクスチャムービーの仕組み



マスクの付いていない通常のムービーは、ポリゴンに映像サーフェスを貼り付けるだけで、テクスチャムービーとして再生できます。マスク付きムービーは、あらかじめポリゴンに2枚のテクスチャを貼っておく必要があります。

- 内側（マスク面）： アルファ付き（FBS\_SA|FBD\_ISA）

SRGBBlendingMode=SRCALPHA, DSTBlendingMode=INVSRCALPHA

- 外側（映像面）： 加算（FBS\_SA|FBD\_ONE）

SRGBBlendingMode=SRCALPHA, DSTBlendingMode=ONE

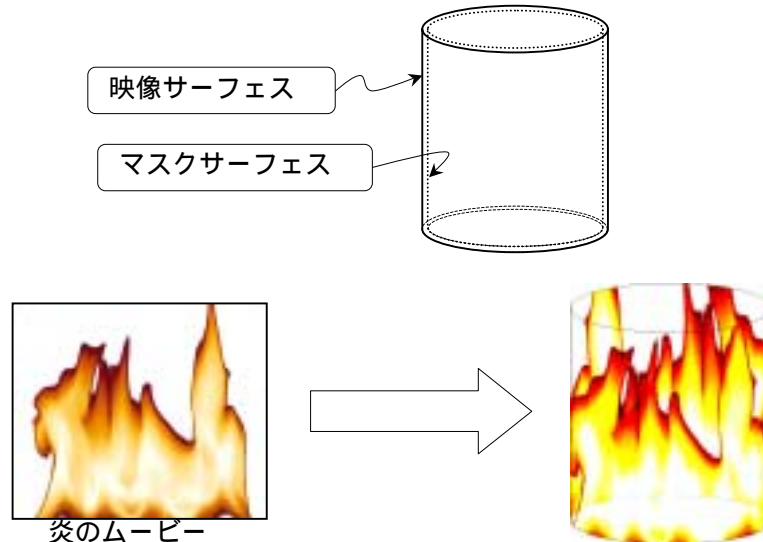


図 5-17 マスク付きテクスチャムービー

- MPEG SofdecF/X によるサーフェスの取得

```

    UserSyncFrame();           // フレーム同期
    mwExecTexSvr();           // テクスチャ転送用サーバ関数
    UserBeginDraw();          // 描画開始
    mwExecDrawSvr();          // 映像描画用サーバ関数

    mwPlyGetMvFrm(ply, &frm); // 映像サーフェスの取得
    ap_set_video_surface(frm.srf); // 映像サーフェスの貼り付け
    mwPlyGetMskFrm(ply, &frm); // マスクサーフェスの取得
    ap_set_msk_surface(frm.srf); // マスクサーフェスの貼り付け
    /* ポリゴンの描画 */

    UserEndDraw();            // 描画終了
    
```

frm.srf は、Kamui ドライバの割り当てたサーフェスです。



- Ninja ライブラリによるテクスチャムービー

njSetMvSurface 関数によって、ムービーサーフェスをテクスチャリスト内の指定された番号のテクスチャに登録できます。njSetMvSurface 関数は、ミドルウェアのサンプルプログラム内で定義しています。

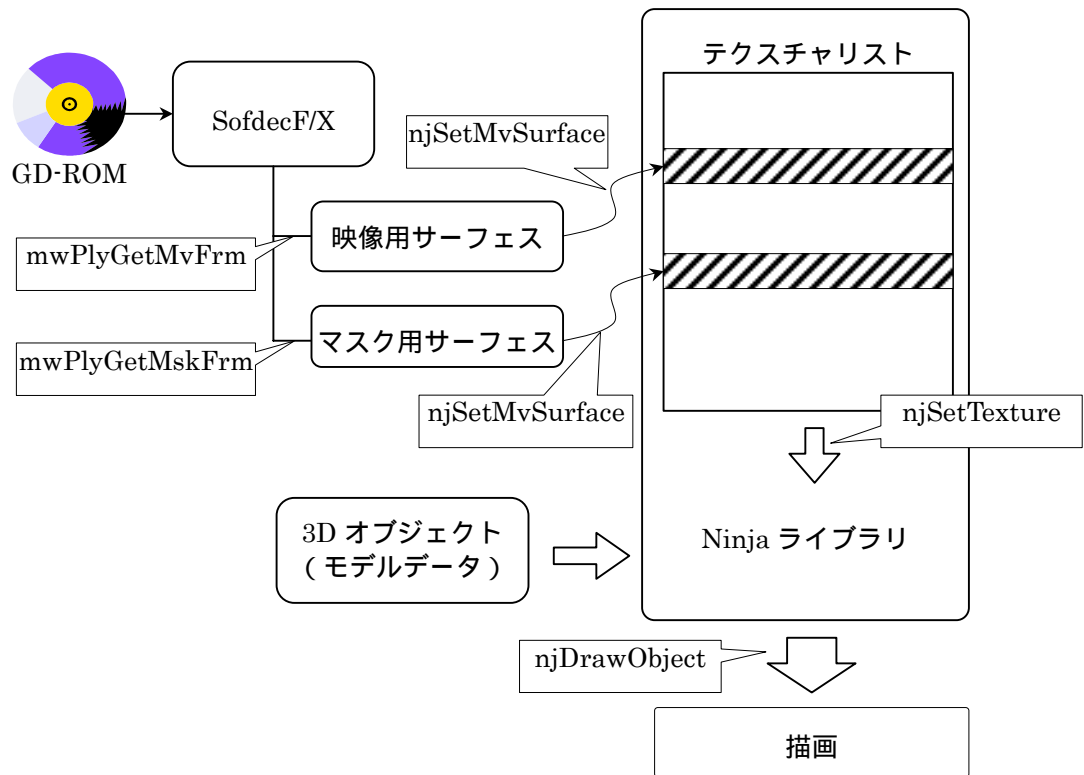


図 5 -18 Ninja ライブラリによるテクスチャムービー



### 5.3.9 マルチストリーム再生

#### (1) 仕組み

MPEG SofdecF/X では、GD-ROM 上に別々に格納された複数のムービーファイルを同時に再生できます。この機能は、ADX のマルチストリーミング機能を用いていますので、ADX をリンクする必要があります。以下のように複数のハンドルを生成し、各ハンドルに対して、非同期に再生開始することができます。

```
ply1 = mwPlyCreateSofdec(&cprm1);
ply2 = mwPlyCreateSofdec(&cprm2);
for (;;) {
    if ( Aボタンが押された )
        mwPlyStartFname(ply1, "movie1.sfd");
    if ( Bボタンが押された )
        mwPlyStartFname(ply2, "movie2.sfd");
}
```

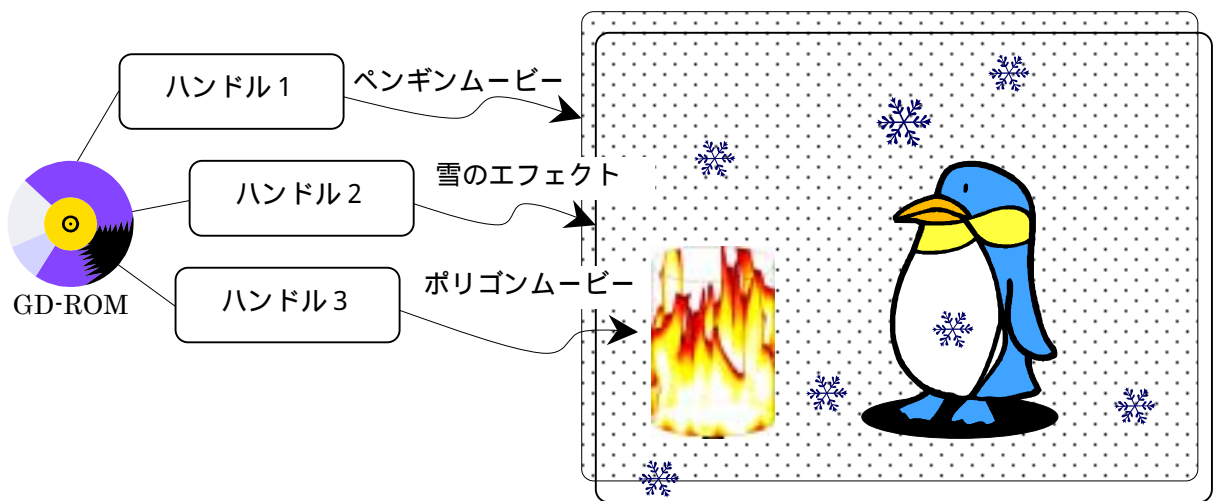


図 5 - 19 マルチストリーム再生

#### (2) バッファの設定

マルチストリーミングを行う場合は、入力バッファを大きくする必要があります。入力バッファの大きさは、生成パラメータの max\_bps によって決まります。例えば、2つのファイルを同時に再生する場合は、max\_bps を 2 倍にしなければなりません。

```
#define NSTM          (3)                                // 同時再生ストリーム数
#define MAX_BPS1      (150*1024*8)                       // 最大ビットレート 1 (150Kbyte/sec)
#define MAX_BPS2      (100*1024*8)                       // 最大ビットレート 2 (100Kbyte/sec)
#define MAX_BPS3      (75*1024*8)                       // 最大ビットレート 3 (75Kbyte/sec)

/* ハンドルの生成 1 (150Kbyte/sec 用) */
cprm1.max_bps = MAX_BPS1 * NSTM;
cprm1.wksize = mwPlyCalcWorkCprmsSfd(&cprm1);
cprm1.work = syMalloc(cprm1.wksize);
ply1 = mwPlyCreateSofdec(&cprm1);

/* ハンドルの生成 2 (100Kbyte/sec 用) */
cprm2.max_bps = MAX_BPS2 * NSTM;
cprm2.wksize = mwPlyCalcWorkCprmsSfd(&cprm2);
cprm2.work = syMalloc(cprm2.wksize);
ply2 = mwPlyCreateSofdec(&cprm2);

/* ハンドルの生成 3 (75Kbyte/sec 用) */
```



```
cprm3.max_bps = MAX_BPS3 * NSTM;
cprm3.wksize = mwPlyCalcWorkCprmSfd(&cprm3);
cprm3.work = syMalloc(cprm3.wksize);
ply3 = mwPlyCreateSofdec(&cprm3);
```

### 5.3.10 シームレス連続再生

#### (1) 複数ファイルのシームレス連続再生

MPEG SofdecF/X は、分割された複数のムービーファイルをシームレスに接続し再生できます。同じファイルを繰り返し再生することもできます。最大9時間まで連続して再生できます。現バージョンでは、映像の連続再生のみをサポートしています。CRI ADX ライブラリが必要です。

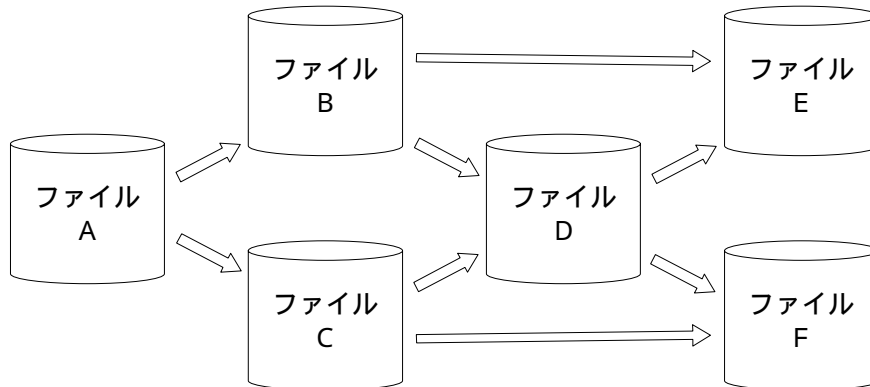


図 5-20 シームレス連続再生

下記の手順により、シームレス連続再生を行うことができます。

- 再生するファイルの登録 (mwPlyEntryFname 関数)
- シームレス連続再生の開始 (mwPlyStartSeamless 関数)
- 再生中に、更に接続するファイルを登録する事も可能
- シームレス連続再生モードの解除 (mwPlyReleaseSeamless 関数)
- 再生終了状態へ

最大8ファイルまで登録しておくことができます。最後のムービーファイルの再生が終了する1秒前までに、次のファイルを登録して下さい。

```
mwPlyEntryFname(ply, "movie1.mlv");
mwPlyEntryFname(ply, "movie2.mlv");
mwPlyStartSeamless(ply);
for (;;) {
    if ( Aボタンが押された )
        mwPlyEntryFname(ply, "movie3.mlv");
    if ( Bボタンが押された )
        mwPlyReleaseSeamless(ply);
}
```

#### (2) シームレスループ再生

mwPlyStartFnameLp 関数によって単一ファイルを簡単にシームレスループ再生することができます。

```
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFnameLp(ply, "movie1.mlv");
```



---

### 5.3.11 動画再生時の注意点

---

(1) ハンドルの生成・消去

ハンドル生成時に、以下のビデオリソースを確保します。

- YUV422 テクスチャサーフェス

サーフェスのサイズは、動画の画像サイズ以上の2のべき乗になります。

- 例

動画サイズ		サーフェスサイズ
320 × 240	→	512 × 256

- 8ビットパレットテクスチャサーフェス

サーフェスのサイズは、動画の画像サイズ以上の2のべき乗の正方形になります。

- 例

動画サイズ		サーフェスサイズ
320 × 240	→	512 × 512

また、ハンドル消去時にテクスチャサーフェスを解放します。従って、再生中にハンドル消去する場合は、再生を停止した後、1V または 2V 分待ってからハンドルを消去して下さい。また、mwPlyExecDrawSvr 関数は映像の描画を行っています。mwPlyDestroy 関数よりも後で、この関数を実行して下さい。

- サンプルプログラム

```
for (;;) {  
    if ( Bボタンがおされた ) {  
        mwPlyStop(ply);  
        UsrWaitVBlank();           // レイテンシが 2V の時は 1 回  
        UsrWaitVBlank();           // レイテンシが 3V の時は 2 回  
        mwPlyDestroy(ply);  
    }  
    mwPlyExecDrawSvr();             // mwPlyDestroy 関数よりもあとに実行  
    UsrWaitVBlank();  
}
```







## (4) キャッシュ

MPEG SofdecF/X は、オペランドキャッシュがインデックスキャッシュモードのときに最高のパフォーマンスで動作します。インデックスキャッシュモードでなくても動作しますが、最大で約 20% 程度のパフォーマンス低下が生じます。

mwPlyInitSofdec 関数、mwPlyInitSfdFx 関数によって、ライブラリを初期化する時に、キャッシュモードをインデックスキャッシュモードに切り換えます。また、mwPlyFinishSofdec 関数、mwPlyFinishSfdFx 関数は、Shinobi のデフォルトのキャッシュモードに戻します。アプリケーションを効率良く動作させるためには、初期化と終了処理を頻繁に行うか、キャッシュ操作関数によってキャッシュモードを切り換えて下さい。

また、ポリゴンとムービーが共存する場合は、ムービーの再生負荷よりキャッシュモードを決めて下さい。目安としては、ムービーの解像度が 160×120 以下の場合は、通常のキャッシュモード、それ以上の場合はインデックスキャッシュモードに設定して下さい。

## ● キャッシュのコントロール例 1

```
#define    CACHE_SHINOBI    ¥                // 通常のキャッシュモード
        (SYD_CACHE_FORM_IC_ENABLE | SYD_CACHE_FORM_OC_ENABLE | SYD_CACHE_FORM_P1_CB )

#define    CACHE_SOFDEC¥    // MPEG SofdecF/X 用キャッシュモード
        (SYD_CACHE_FORM_OC_INDEX | CACHE_SHINOBI)

mwPlyInitSofdec(...);                // インデックスキャッシュモード
syCacheInit(CACHE_SHINOBI);          // 通常モード
/* ポリゴンの描画 */
syCacheInit(CACHE_SOFDEC);           // インデックスキャッシュモード
mwPlyStartFname("sample.sfd");       // ムービー再生
```

## ● キャッシュのコントロール例 2

```
mwPlyInitSofdec(...);                // インデックスキャッシュモード
mwPlyFinishSofdec();                 // 通常モード
/* ポリゴンの描画 */
mwPlyInitSofdec(...);                // インデックスキャッシュモード
mwPlyStartFname("sample.sfd");       // ムービー再生
```

syCacheInit 関数は、SEGA Library Ver.2 では関数名が変更される可能性があります。

## (5) CPU 負荷の軽減

以下の方法によって、CPU 負荷を軽減し、コマ落ちを軽減できます。

## a. ハーフペル処理の簡略化

B ピクチャのハーフペル処理を簡略化し、3 ~ 4 % ぐらい高速化することができます。画質はわずかに劣化します。

```
mwPlySetFastHalfpel(ply, ON);
```

## b. B ピクチャの除去

エンコード時に B ピクチャを含まないストリームとしてエンコードします。例えば、N=6,M=1 のパラメータでデコードすることにより B ピクチャの含まないストリームを作成できます。B ピクチャのデコードは、他のピクチャよりも負荷が大きいため、B ピクチャが無くなることで 6 ~ 9 % 程度 CPU 負荷を軽減できます。

```
c:¥>sfvencd -in=sample.avi -out=sample.mlv -gop_n=6 -gop_m=1
```

## c. ビットレートの削減

CPU 負荷は、SFD データのビットレートに比例して大きくなります。データレートを落とすことにより CPU 負荷を軽減できます。



## d. 解像度の調整

解像度を低くすることにより CPU 負荷を軽減させることができます。例えば、NTSC の TV では  $640 \times 480$  の解像度の上下 16 ドットは表示されません。従って、 $320 \times 480$  の動画を  $320 \times 448$  の動画にクロップすることによって 6 %程度 CPU 負荷を減少させることができます。

## e. フレームプール数の増加

フレームプール数を増加させることにより、コマ落ちを減少させることができます。この方法では、CPU 負荷は変わりませんが、CPU 負荷の変動を吸収できます。MPEG では、ピクチャの種別、動画の複雑さにより CPU 負荷が大きく変動します。

```
cprm. nfrm_pool_wk = 6;           // この値を増やすことによりコマ落ちを軽減  
ply = mwPlyCreateSofdec(&cprm);
```



## 5.3.12 関数一覧

Sofdec ライブラリでは、以下の関数がサポートされています。  
 詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
mwPlyCalcSrfBufSize	サーフェスポインタ用バッファサイズの計算
mwPlyCalcWorkCprmSfd	作業領域サイズの計算
mwPlyEntryFname	シームレス連続再生ファイルの登録
mwPlyEntryFxCb	エフェクトコールバック関数の登録
mwPlyExecDrawSvr	映像描画用サーバ関数(Kamui2 用)
mwPlyExecTexSvr	テクスチャ転送用サーバ関数(Kamui2 用)
mwPlyFinishSfdFx	ライブラリの終了処理(Kamui、Kamui2 用)
mwPlyGetImgPos	イメージ位置の取得
mwPlyGetImgSize	イメージサイズの取得
mwPlyGetInputSj	入力ストリームジョイントの取得
mwPlyGetMskFrm	マスクフレームの取得
mwPlyGetMvFrm	ムービーフレームの取得
mwPlyGetNumDecPool	デコード済みフレーム数の取得
mwPlyGetNumDispWait	表示待ちフレーム数の取得
mwPlyGetNumIbufRoom	入力バッファの空き容量の取得
mwPlyGetNumLoadFrm	テクスチャメモリへ転送したフレーム数の取得
mwPlyGetNumRend	レンダリング中のフレーム数の取得
mwPlyGetNumTotalDec	デコードした全ピクチャ数の取得
mwPlyGetNumTotalSkip	スキップした全ピクチャ数の取得
mwPlyGetSrfBright	輝度の取得
mwPlyGetSrfBrightOfst	輝度オフセットの取得
mwPlyGetSrfPos	表示位置の取得
mwPlyInitSfdFx	ライブラリの初期化(Kamui、Kamui2 用)
mwPlyReleaseSeamless	シームレス連続再生の解除
mwPlySetAlphTbl	輝度から $\alpha$ 値への変換テーブルの設定
mwPlySetCompoMode	合成モードの設定
mwPlySetImgPos	イメージ位置の設定
mwPlySetLpFlg	シームレスループ再生の設定
mwPlySetSrfBright	輝度の設定
mwPlySetSrfBrightOfst	輝度オフセットの設定
mwPlySetSrfPntBuf	サーフェスポインタ用バッファの設定
mwPlySetSrfPos	表示位置の設定
mwPlySetVertexBuffer	頂点バッファの設定
mwPlyStartFnameLp	シームレスループ再生の開始
mwPlyStartSeamless	シームレス連続再生の開始



## 5.4 SAN ライブラリ

ここでは、SAN ライブラリについて説明します。

### 5.4.1 SAN ライブラリの概要

SAN ライブラリは、BMP ファイルを YUV420 フォーマットに変換して連結したデータを、容易に表示するためのライブラリです。

### 5.4.2 モジュール構成

簡易アニメーション (SAN) ライブラリのモジュール構成図を以下に示します。

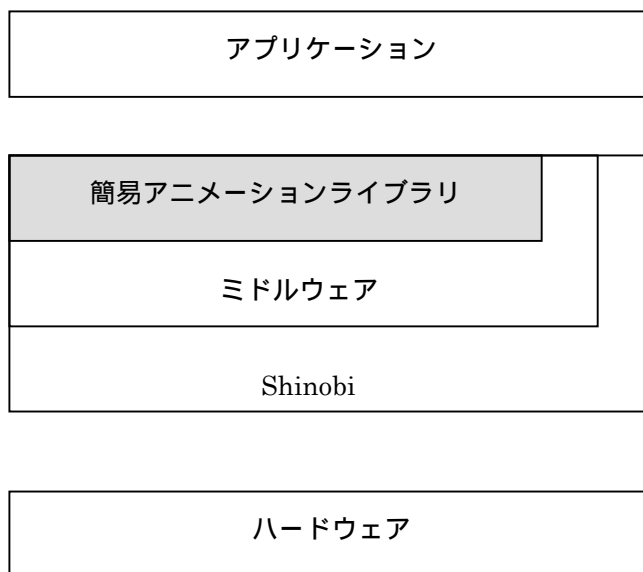


図 5 -21 SAN のモジュール構成図



### 5.4.3 SAN の仕組み

SAN\_LoadTex 関数は、SAN データ内の 1 つのピクチャをテクスチャバッファに転送する関数です。

SAN\_Draw 関数は、テクスチャバッファに転送したデータを、フレームバッファに転送し表示する関数です。

SAN データ内の 1 ピクチャのデータフォーマットは、YUV420 です。テクスチャバッファに転送されると、YUV422 フォーマットに変換されます。

SAN の仕組みを以下に示します。

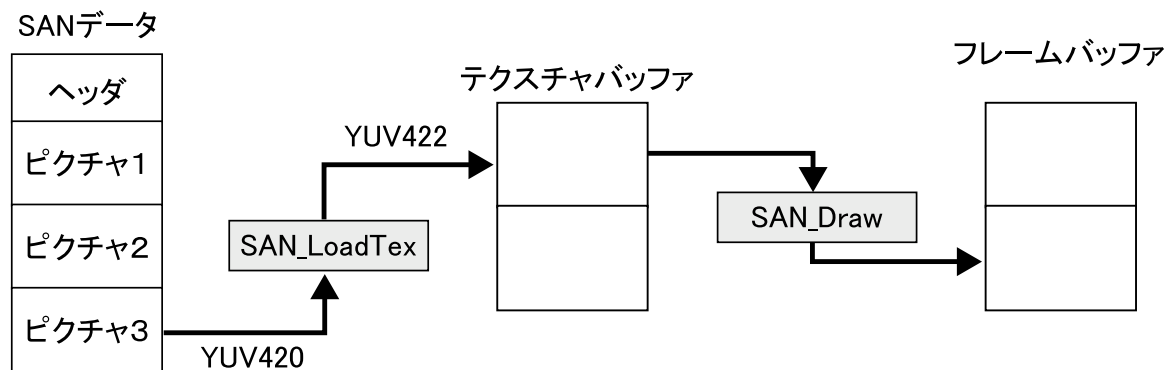


図 5 -22 SAN の仕組み



### 5.4.4 マルチウィンドウ

SAN は、映像を複数のウィンドウに表示することができます。

ウィンドウの各 4 頂点に対して、以下のパラメータを独立し設定することができます。

(1) 表示位置

X 座標、Y 座標、Z 座標を設定できます。X 座標や Y 座標はフレームバッファの座標です。左上が (0.0、0.0)、右下が (639.0、479.0) になります。Z 座標を指定しますが、透視変換は行いません。

(2) 輝度

アルファ、赤、緑、青の強さを指定します。指定する値は、1.0 から 0.0 までの値を設定します。映像データに対して、この設定値が乗じられます。赤、緑、青の値を 0.0 から 1.0 まで変化させると黒からフェードインし、1.0 から 0.0 まで変化させると黒へフェードアウトします。また、アルファは背景との混合比率になります。これを変化させることによりディゾルブ効果が得られます。

(3) 輝度オフセット

アルファ、赤、緑、青の加算成分を指定します。1.0 から 0.0 までの値を設定します。映像データに対して、この設定値と 255 を乗じた値が加算されます。赤、緑、青を 0.0 から 1.0 まで変化させると白へフェードアウトし、1.0 から 0.0 まで変化させると白からフェードインします。

(4) 映像切り出し位置

各頂点に対応する映像データの位置を UV 座標に相当する値を指定します。

その際に指定する値は、映像データに対するピクセル単位での位置になります。

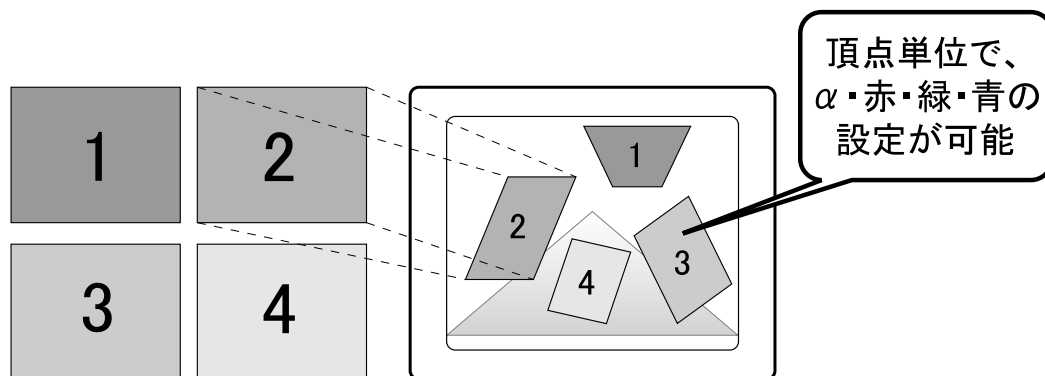


図 5 -23 マルチウィンドウ表示

a. マルチウィンドウ表示について

マルチウィンドウ表示を行うためには、サーフェスポイント用バッファの確保と設定が必要です。サーフェスポイントとは、ウィンドウの各頂点を示します。

ウィンドウは、4 つのサーフェスポイントから構成されます。例えば、25 個のウィンドウを表示するためには、100 個のサーフェスポイントが必要です。

以下に、サーフェスポイント用バッファの確保と設定の方法を示します。

● サーフェスポイント用バッファの確保と設定

```
size = SAN_CalcSrfBufSize(san, 4*25);
buf = syMalloc(size);
SAN_SetSrfPntBuf(san, 4*25, buf, size);
```



### 5.4.5 SAN テクスチャ

SAN は、Kamui ドライバを使用して映像用テクスチャサーフェスとマスク用テクスチャサーフェスを確保しています。

アプリケーションは SAN ハンドルからサーフェスを取得し、アプリケーション側のオブジェクトのサーフェスに映像とマスクを貼ることができます。

以下に、SAN テクスチャの仕組みを示します。

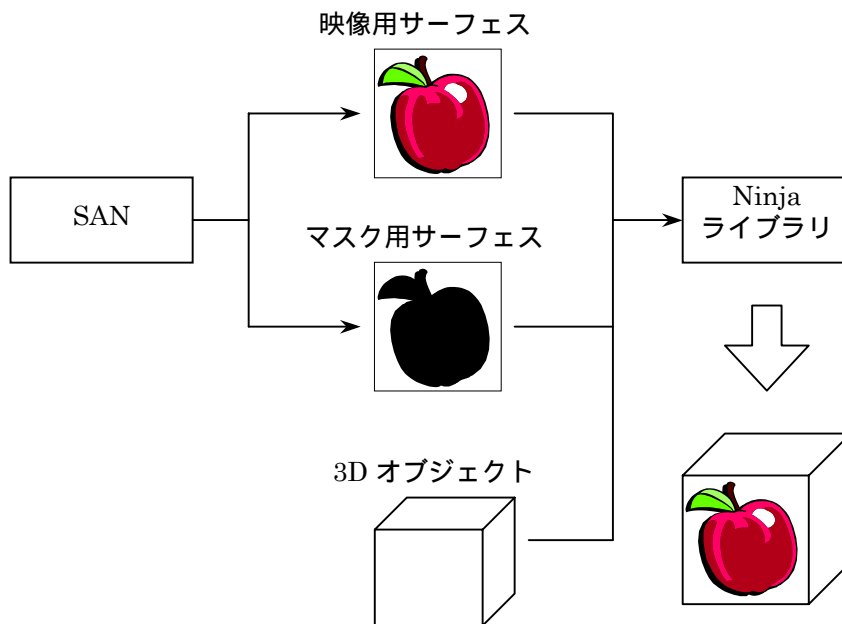


図 5-24 SAN テクスチャの仕組み

SAN では、ハンドルから以下の 2 枚のテクスチャサーフェスを取得できます。

- (1) 映像サーフェス  
矩形形式の YUV422 テクスチャです。
- (2) マスクサーフェス  
Twiddle 形式の 8bit/pixel のパレットテクスチャです。

#### ● サーフェスの取得方法

<code>SAN_LoadTex(san, 0);</code>	<code>/* テクスチャ RAM への転送 */</code>
<code>SAN_GetVideoPic(san, 0, &amp;pic);</code>	<code>/* 映像サーフェスの取得 */</code>
<code>user_set_video_srf(pic.srf);</code>	<code>/* 映像サーフェスの貼り付け */</code>
<code>SAN_GetMskPic(san, 0, &amp;pic);</code>	<code>/* マスクサーフェスの取得 */</code>
<code>user_set_msk_srf(pic.srf);</code>	<code>/* マスクサーフェスの貼り付け */</code>
<code>/* ポリゴンの描画 */</code>	

pic.srf は、Kamui ドライバの割り当てたサーフェスです。



- **Ninja ライブラリによる SAN テクスチャ**

njSetMvSurface 関数によって、SAN のサーフェスをテクスチャリスト内の指定された番号のテクスチャに登録することができます。njSetMvSurface 関数は、ミドルウェアのサンプルプログラム内に定義しています。

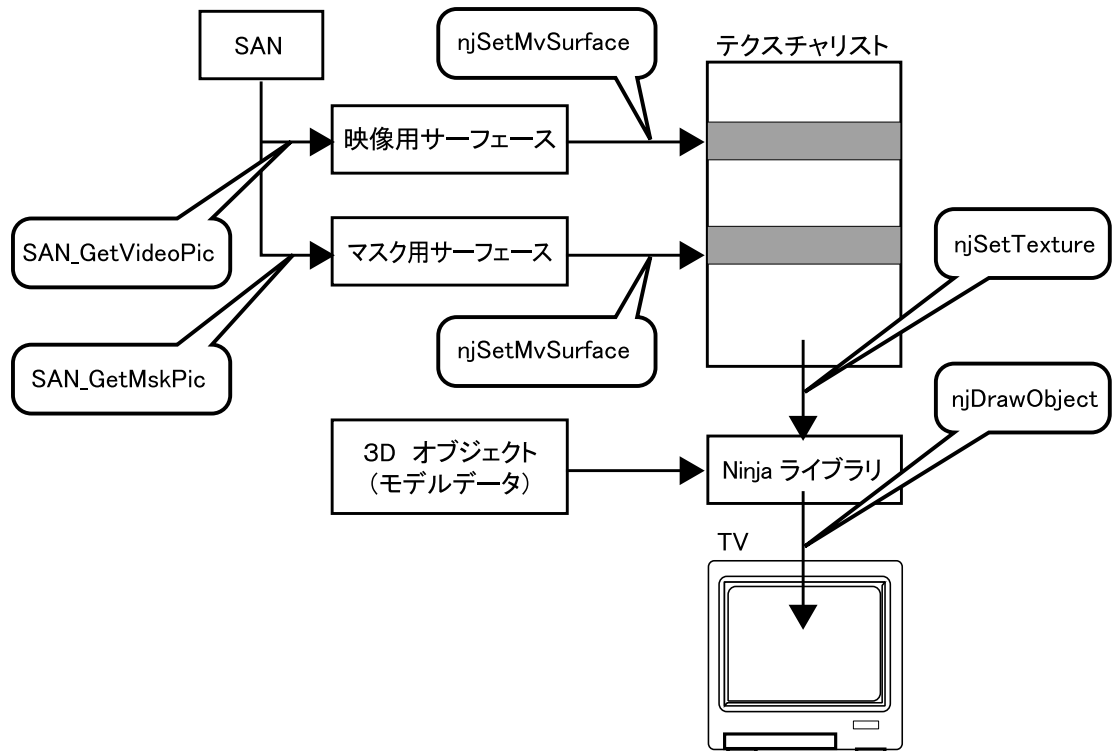


図 5 -25 Ninja による SAN テクスチャ



### 5.4.6 合成機能

SAN は、ビジュアルエフェクトのための合成機能を持っています。合成とは、複数の映像を重ね合わせ表示することを言います。

合成方法として、以下の方法があります。

- (1) 不透明表示  
映像を合成せずに描画します。
- (2) 半透明合成  
ウィンドウ単位でアルファブレンディングします。
- (3) 加算合成  
ポリゴンの RGB 値と SAN の RGB 値を単純に加算します。
- (4) アルファ合成  
ポリゴンの RGB 値と SAN の RGB 値をピクセル単位でアルファブレンディングします。現バージョンでは、フルアルファ合成、256 段階の 値を持つ合成のみ対応しています。

- アルファブレンディング

アルファブレンディングとは、混合値 により以下のように表現されます。

```
output =  x s + ( 1 -  ) x d
output : 合成後の RGB 値
        : 混合比率
s       : SAN の RGB 値
d       : ポリゴンの RGB 値
```

### 5.4.7 アルファ合成の仕組み

アルファ合成は、2 枚の半透明ポリゴンを描画することにより実現しています。マスクサーフェスを描画(乗算処理)後、映像サーフェスを加算しています。

以下に、合成の仕組みを示します。

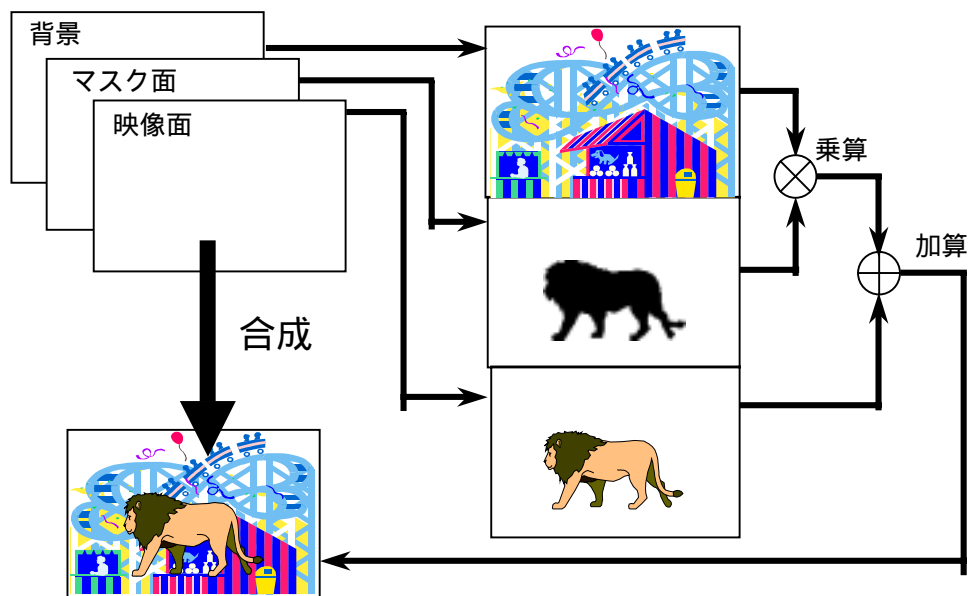


図 5 -26 合成の仕組み



### 5.4.8 データフォーマット

SAN データは、YUV420 データを連結したデータです。bmp2san.exe プログラムにより BMP ファイル（24bit・非圧縮）から作成します。

SAN データのフォーマットを以下に示します。

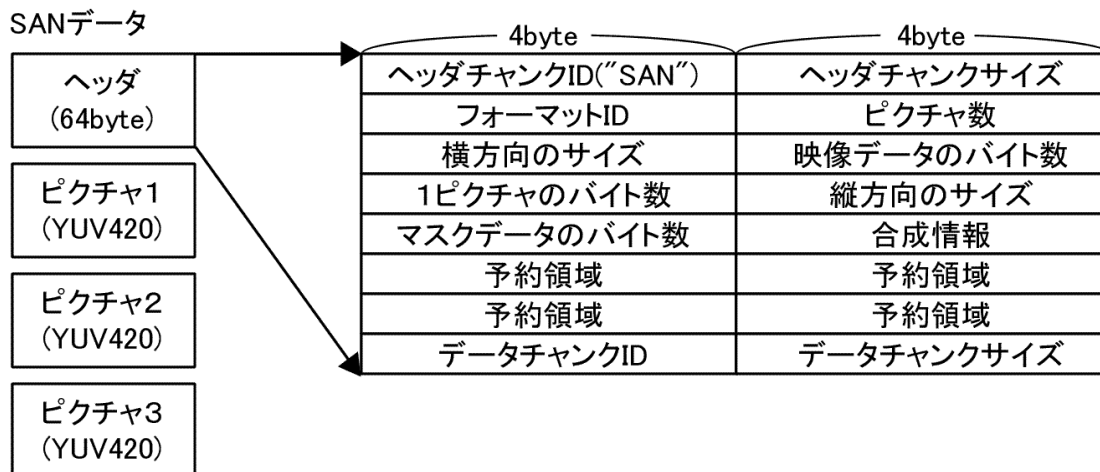


図 5 -27 SAN データのフォーマット

### 5.4.9 データの作成方法

(1) 不透明表示、半透明合成、加算合成

bmp2san.exe プログラムにより、複数の BMP ファイルを YUV420 フォーマットに変換し、連結します。

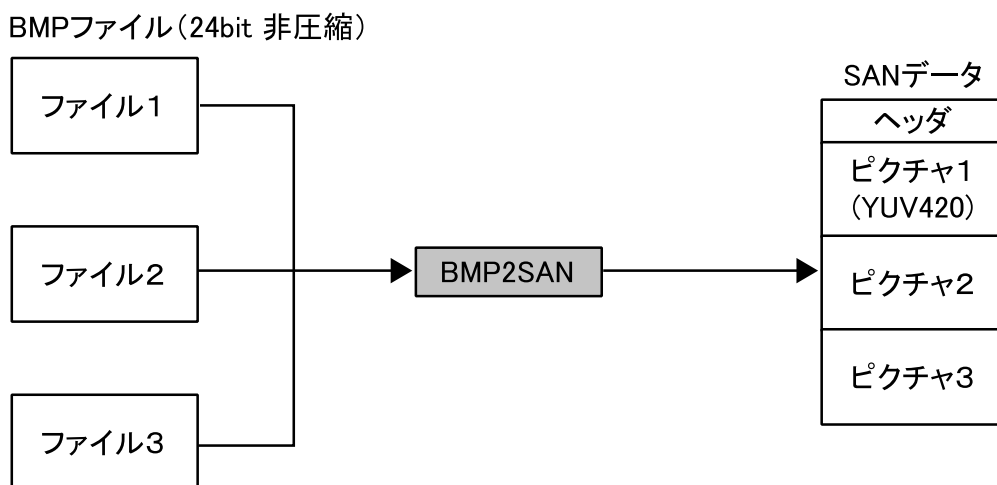
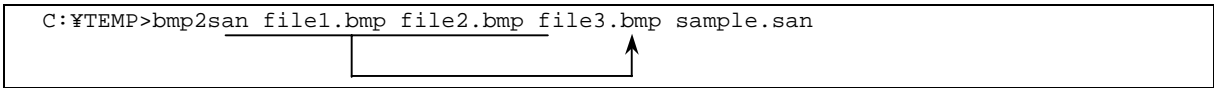


図 5 -28 SAN データの作成



コマンドプロンプトより、以下のように実行して下さい。最後の引数が出力ファイルになります。

```
C:¥TEMP>bmp2san file1.bmp file2.bmp file3.bmp sample.san
```



拡張子を省略することもできます。

```
C:¥TEMP>bmp2san file1 file2 file3 sample
```

サブコマンドファイルを使用する場合は以下のとおりです。

```
C:¥TEMP>bmp2san -sub=san.sub サブコマンドファイル(san.sub)
```

```
file1.bmp  
file2.bmp  
file3.bmp  
sample.san
```

以下の手順で、ディレクトリ内の全ての BMP ファイルを SAN データに変換することも可能です。

```
C:¥TEMP>dir /b *.bmp > tmp.sub  
C:¥TEMP>bmp2san -sub=tmp.sub sample1.san サブコマンドファイル(tmp.sub)
```

```
file1.bmp  
file2.bmp  
file3.bmp
```

- 半透明合成の場合

以下のように、オプションを追加して下さい。

```
C:¥TEMP>bmp2san -compo=trnsp -sub=tmp.sub sample1.san
```

- 加算合成の場合

以下のように、オプションを追加して下さい。

```
C:¥TEMP>bmp2san -compo=add -sub=tmp.sub sample1.san
```



## 5.4.10 アルファ合成

アルファ合成用 SAN データは、映像データとマスクデータをグラフィックツールによって作成してから、bmp2san.exe プログラムにより連結します。

マスクデータは、Twiddle 形式の 8bit パレットに変換されます。

BMPファイル(24bit 非圧縮)

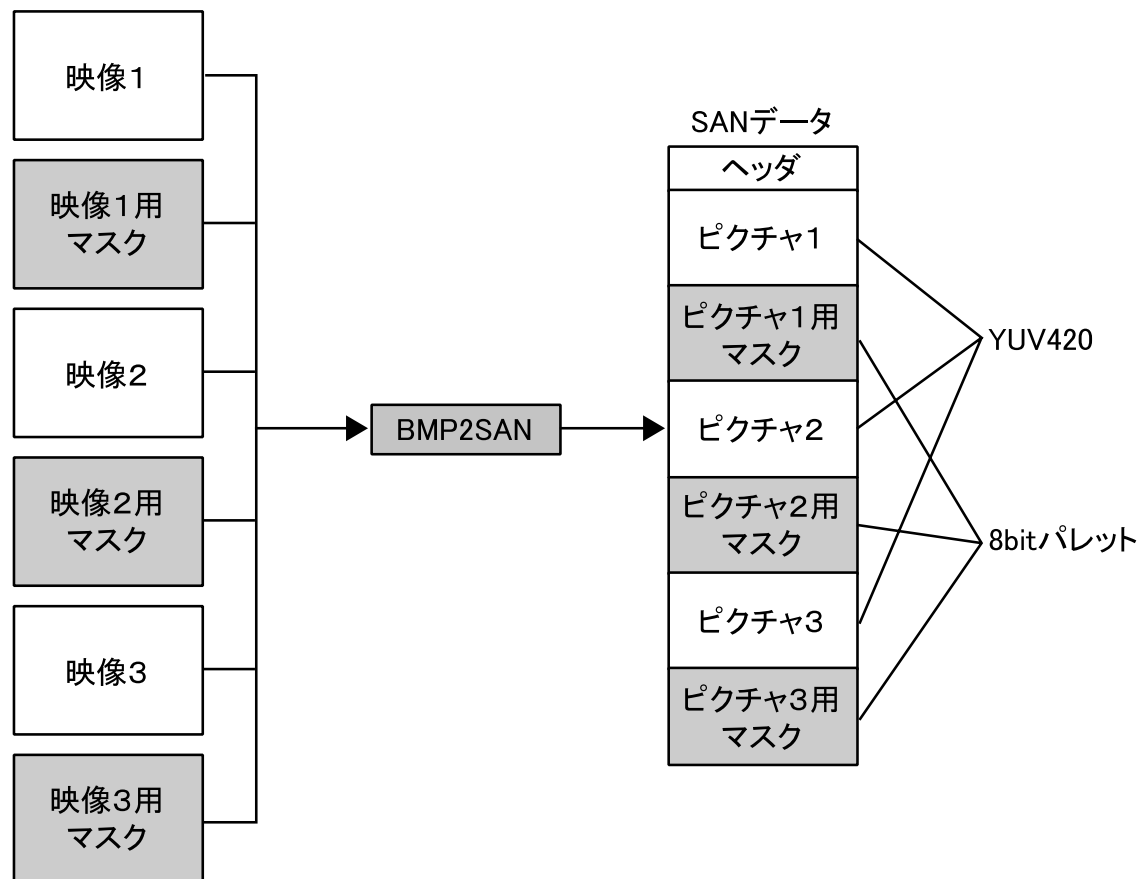


図 5 -29 アルファ合成用 SAN データの作成

コマンドプロンプトより以下のように実行して下さい。

合成モードにフルアルファ (“alph256”) を設定し、BMP ファイルは映像データ、マスクデータの順に入力します。

```
C:\¥TEMP>bmp2san -compo=alph256 file1v.bmp file1a.bmp file2v.bmp file2a.bmp sample.san
```

file1v.bmp, file2v.bmp ... 映像データ

file1a.bmp, file2a.bmp ... マスクデータ

不透明表示の場合と同様に、サブコマンドファイルを指定することもできます。



## 5.4.11 関数一覧

SAN ライブラリでは、以下の関数がサポートされています。  
 詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
SAN_CalcSrfBufSize	サーフェスポイント用バッファサイズの計算
SAN_CalcWorkSize	作業領域サイズの計算
SAN_CalcWorkSizeAlph	作業領域サイズの計算(合成用)
SAN_Create	ハンドルの生成
SAN_Destroy	ハンドルの消去
SAN_Draw	テクスチャの表示
SAN_EntryErrFunc	エラーコールバック関数の登録
SAN_Finish	終了処理
SAN_FinishKm	終了処理(Kamui 用)
SAN_GetDispBright	輝度の取得
SAN_GetDispBrightOfst	輝度オフセットの取得
SAN_GetDispZ	表示スクリーンの奥行き値の取得
SAN_GetImgPos	イメージ位置の取得
SAN_GetImgSize	イメージサイズの取得
SAN_GetMskPic	マスクピクチャの取得
SAN_GetNumPic	SAN データ内のピクチャ数の取得
SAN_GetSrfBright	輝度の取得
SAN_GetSrfBrightOfst	輝度オフセットの取得
SAN_GetSrfPos	表示位置の取得
SAN_GetVideoPic	ビデオピクチャの取得
SAN_Init	SAN ライブラリの初期化
SAN_InitKm	SAN ライブラリの初期化(Kamui 用)
SAN_LoadTex	YUV420 データのグラフィックメモリへの転送
SAN_SetDispBright	輝度の設定
SAN_SetDispBrightOfst	輝度オフセットの設定
SAN_SetDispPos	表示位置の設定
SAN_SetDispSize	表示サイズの設定
SAN_SetDispZ	表示スクリーンの奥行き値の設定
SAN_SetImgPos	イメージ位置の設定
SAN_SetSrfBright	輝度の設定
SAN_SetSrfBrightOfst	輝度オフセットの設定
SAN_SetSrfPntBuf	サーフェスポイント用バッファの設定
SAN_SetSrfPos	表示位置の設定
SAN_SetVertexBuffer	頂点バッファの設定(Kamui 用)



## 6 MPEG/Audio について

ここでは、Dreamcast 上で MPEG/Audio を使用する際の、「データの作成の方法」について説明します。

**参 照** ソフトウェア上で MPEG/Audio デコーダを使用するための関数等の仕様については、「Dreamcast ミドルウェアライブラリ基本仕様書」を参照して下さい。

### 6.1 MPEG/Audio の概要

MPEG/Audio デコーダは、ISO/IEC11172 準拠 MPEG1/Audio Layer II データの再生が可能です。また、サブサンプルと呼ぶ処理の後に MPEG エンコードされたデータも再生可能です。このサブサンプル処理を利用すると、音質の劣化を最小限に抑えながら処理量 (=CPU に対する負荷) を通常 MPEG 再生の約 70% (サブサンプル処理 1 回) ~ 40 % (同 2 回) に削減することが可能となり、その分を画像処理等にまわすことができます。

MPEG/Audio データの作成には、以下の 3 つが必要です。ただし、サブサンプル処理を必要としない場合は、(1) のみ必要となります。

- (1) ISO/IEC11172 準拠 MPEG1/Audio Encoder
- (2) サブサンプル処理ソフトウェア「sub\_smpl.exe」
- (3) サブサンプルフラグ付加ソフトウェア「set\_sflg.exe」

このうち(1)については、現在のところ弊社 (日立) からの提供は予定されていません。必要に応じて準備して下さい。また、(2)(3)はサブサンプル処理を利用する際に MPEG/Audio データに前処理を施すもので、「Dreamcast SDK」から提供されるものです。



## 6.2 MPEG/Audio 利用時の処理の流れ

ここでは、MPEG/Audio を利用時の処理の流れについて説明します。

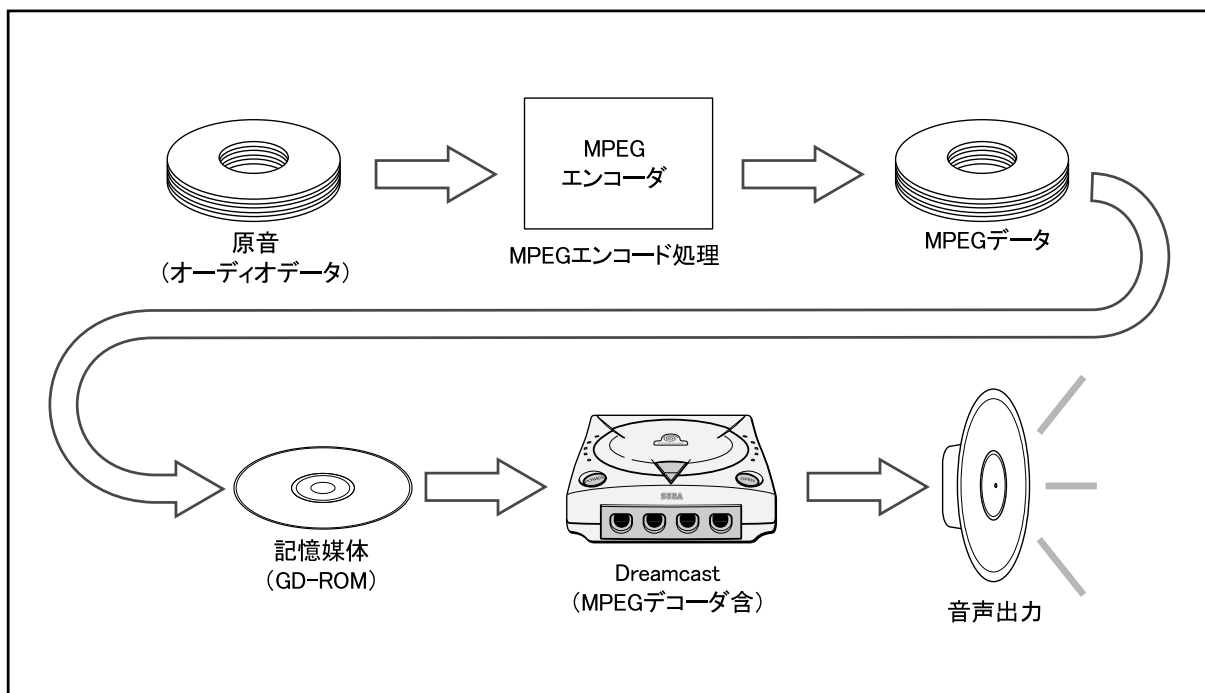


図 6 - 1 MPEG Audio 利用時の処理の流れ

上図は、MPEG 処理システム全体の構成を示しています。原音（オーディオデータ）は、MPEG/Audio エンコーダによって MPEG/Audio データにエンコードされます。このデータを記憶媒体（GD-ROM）に書き込み、Dreamcast 上の MPEG/Audio デコーダに処理させることで音声出力がなされます。本システムでは、一般的な MPEG/Audio 処理のほか、より低ビットレートで音質劣化の少ない音声出力を可能とするための独自の処理や、サブサンプル及びアップサンプル処理を行うことが可能となっています。

次に、MPEG/Audio エンコード処理とデコード処理とにわけて説明します。



## 6.3 MPEG/Audio エンコード処理の流れ

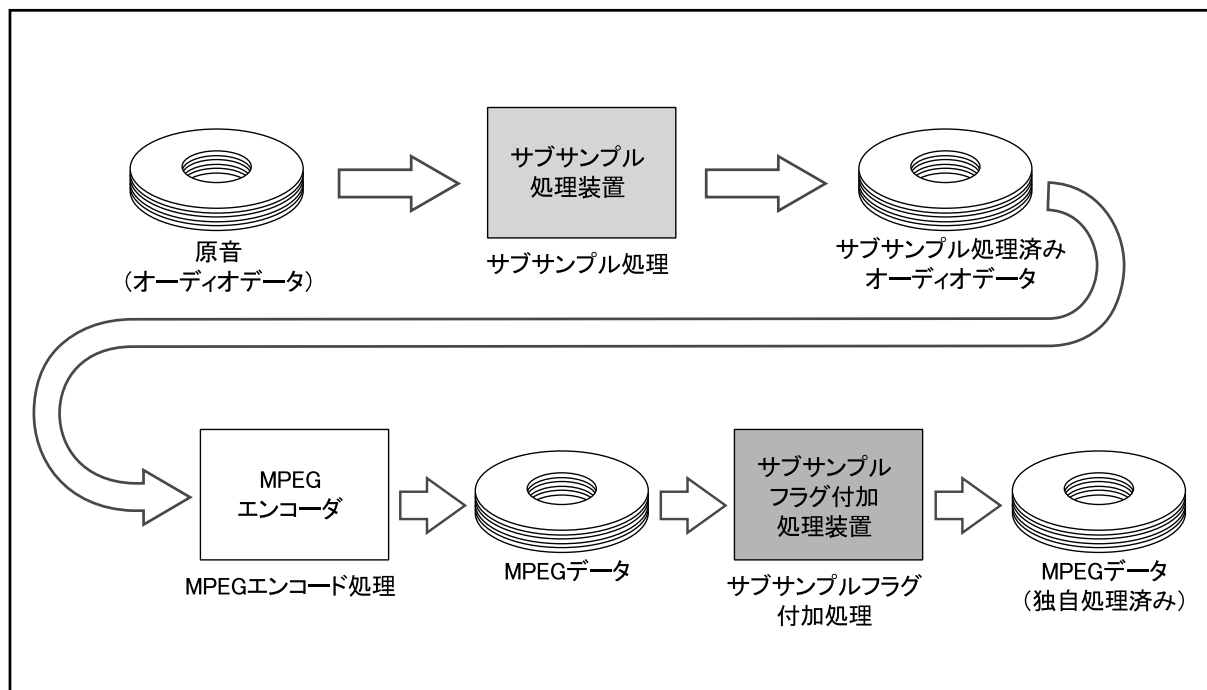


図 6 - 2 サブサンプル処理を含む MPEG エンコード処理の流れ

上図は、サブサンプル処理を含む MPEG エンコード処理の流れを示しています。

- サブサンプル処理を利用しない場合  
上図のような手順の必要は無く、原音を MPEG エンコードして MPEG データを得ることのみが必要となります。
- サブサンプル処理を利用する場合  
原音（オーディオデータ）は、初めにサブサンプル処理ソフトウェアによってデータ量を 1/2 または 1/4 に圧縮されます。このサブサンプル処理済みのデータを MPEG エンコーダに入力することで、MPEG データにエンコードされます。その後、サブサンプルフラグを付加するソフトウェアによって（サブサンプル処理が実行されたことを示す）フラグが付加され、データの作成が完了します。

上図に示した処理は、図 6 - 1 中の原音から MPEG データ作成までに相当します。



## 6.4 MPEG/Audio デコード処理の流れ

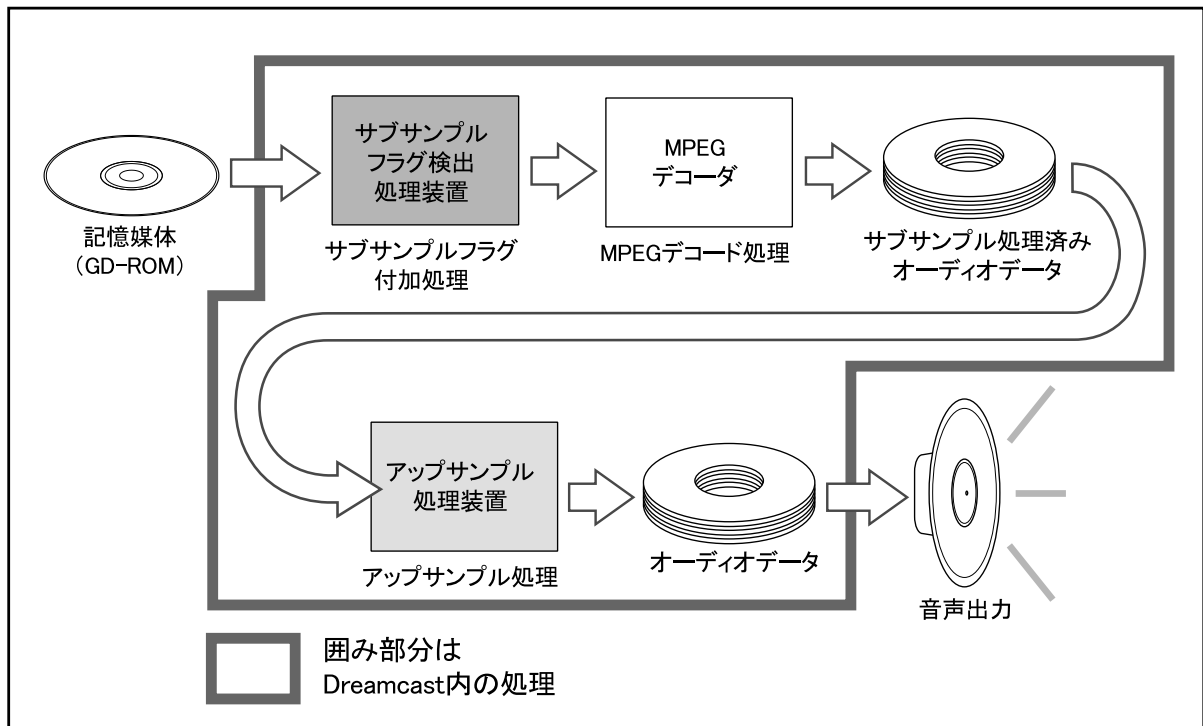


図 6 - 3 アップサンプル処理を含む MPEG デコード処理の流れ

上図は、アップサンプル処理を含む MPEG デコード処理の流れを示しています。

記憶媒体（GD-ROM 等）に記録された MPEG データは、Dreamcast によって読み出されます。このデータは、初めにサブサンプルフラグ検出処理装置によって、サブサンプルフラグの有無及びそのパラメータのチェックが実行されます。

MPEG デコーダからデコード処理がなされたデータは、サブサンプルフラグの有無及びその値に従ってアップサンプル処理装置によりアップサンプル処理が施されます。この時、データ量は 2 倍または 4 倍に伸長され、原音と同じデータ量になります。

以上の処理によって復元されたオーディオデータが、音声として実際に出力されます。

上図に示した処理は、図 6 - 1 中の記憶媒体から音声出力までに相当します。

上図のサブサンプルフラグ検出処理装置や MPEG デコーダ及びアップサンプル処理装置は、ソフトウェアで構成されています。これらは全て MPEG デコーダに含まれており、フラグの判別有無の判別やアップサンプル処理は自動的に行われます。このため、ソフトウェアで MPEG/Audio デコーダを使用する際には「再生開始」等の関数を使用するだけで良く、サブサンプルに関する情報は必要ありません。

**参 照** 関数等の仕様については、「Dreamcast ミドルウェアライブラリ基本仕様書」を参照して下さい。



## 6.5 関数一覧

MPEG/Audio では、以下の関数がサポートされています。  
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
mwPlyCreateMpa	MPEG/Audio 用ハンドルの生成
mwPlyFinishMpa	MPEG/Audio ライブラリの終了処理
mwPlyInitMpa	MPEG/Audio ライブラリの初期化



## 7 ADX 再生ライブラリ

ここでは ADX 再生ライブラリを、全般にわたり説明します。

### 7.1 概要

ADX 再生ライブラリは、ADX 圧縮された音声データを再生するためのライブラリです。本ライブラリ関数を使用することにより、圧縮された音声を簡単に再生することができます。本ライブラリは、Dreamcast 上に構成される「ADX 再生システム」を制御します。

### 7.2 ADX 再生ライブラリの特長

ADX 再生ライブラリの特長は、以下のとおりです。

- (1) メイン CPU にほとんど負荷をかけずに、8 声まで再生できます。

44.1kHz の音声データデコード処理の CPU 負荷

1 声	8 声
約 0.6%	約 5 %

- (2) BGM、セリフ、効果音等の様々な音声データを再生できます。

- 本ライブラリで扱うデータ

データ種別	内 容
ADX データ	音楽やセリフ等の単一の音声ファイルを圧縮したデータです。
ACX データ (ADX 効果音データ)	効果音やセリフ等、複数の ADX データを連結して 1 ファイルにしたデータです。メモリインデックス再生に使用します。
AFS データ (ADX ファイルシステムパーティションデータ)	ADX データやグラフィックデータ等複数のファイルを連結したデータです。GD インデックス再生に使用します。

- (3) 様々な再生方式をサポートしています。

- 本ライブラリによる再生方式

再生方式	内 容
メモリ再生	メモリ上の ADX データを再生します。
メモリインデックス再生	メモリ上の ACX データを再生します。
GD ストリーム再生	GD 上の ADX データを再生します。
GD インデックス再生	GD 上の AFS データを再生します。

- (4) サウンドドライバと同時に使用することが可能です。

- (5) 複数の GD ストリーム再生、GD インデックス再生を同時に行うことができます。

- (6) GD からシームレスループ再生を行うことができます。

- (7) ADX ファイルシステムライブラリ (AFS) により、GD ストリーム再生中に、さらに GD からゲームデータ等の別ファイルを読み込むことができます。

- (8) AFS により、大量のファイルを簡便に扱えます (1 ファイルあたり、約 2Byte で管理)。

- (9) 16 ビットの非圧縮 / 4 ビット ADPCM 圧縮形式の WAV データファイルを再生できます。



## 7.3 ADX 再生システム

ミドルウェアライブラリが制御する「ADX 再生システム」について説明します。

### 7.3.1 システム構成

ADX データは、メイン CPU でワーク RAM 上にデコードされた後、サウンド RAM へ転送されて音声として再生されます。

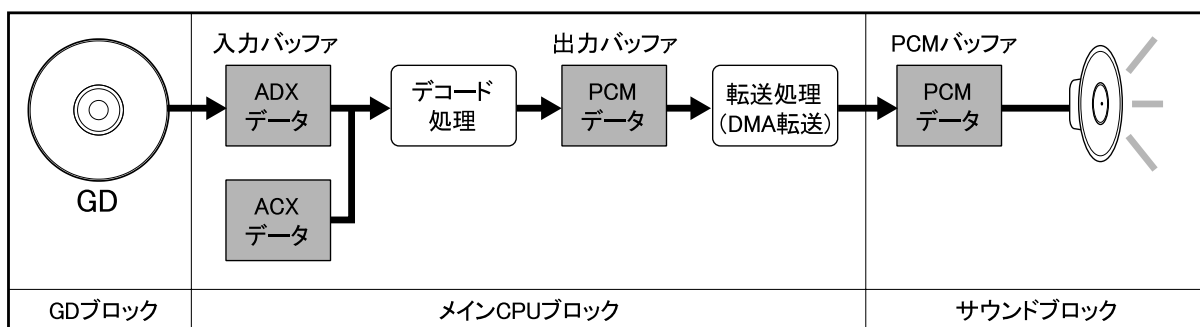


図 7 - 1 ADX 再生システムのシステム構成

入力バッファ	GD から読み込んだ ADX データを格納します。 ワーク RAM 上にサンプリング周波数と並行再生ストリーム数に応じたサイズ分の領域を必要とします。モノラル・44kHz を 3 ストリーム並行再生する場合、約 100KByte となります。 メモリから再生する場合は、必要としません。
出力バッファ	デコードされた 16 ビット PCM データを格納します。 ワーク RAM 上に 1 チャンネル当たり (4096+32) サンプル分 (4040HByte) の領域を必要とします。入力バッファと出力バッファの合計サイズを ADXT_CALC_WORK マクロによって計算できます。
PCM バッファ	サウンド RAM 上の PCM データを格納します。 サウンド RAM の最後から再生されるストリーム数分、サウンドライブラリによって確保されます。サウンドクリエイタは、この領域と重ならないようにマルチユニットファイルを作成する必要があります。 サイズは、4040H バイト / チャンネルです。
ADX データ	ADX 方式により圧縮されたデータです。1 つの音声ファイルが 1 つの ADX データとなります。
ACX データ (ADX 効果音データ)	複数の ADX データを 1 つにまとめたデータです。 結合時に指定したファイルリストの順番で、音声データ番号が 0 から割り当てられます。
AFS データ	複数の ADX データやゲームデータを 1 つにまとめたデータです。結合時に指定したファイルリストの順番で、ファイル (ADX ファイルシステム ID が 0 から順に割り当てられます。ACX データとの違いパーティションデータ) はセクタバウンダリに配置されていることです。



## 7.4 モジュール構成

ADX 再生システムは、サウンドライブラリ・サウンドドライバと共存することができます。従って、プログラマは、サウンドライブラリと ADX ライブラリを同時に使用することができます。例えば、ADX によりストリーム再生しながら、MIDI のシーケンスを再生することもできます。

本ライブラリのモジュール構成を、以下に示します。

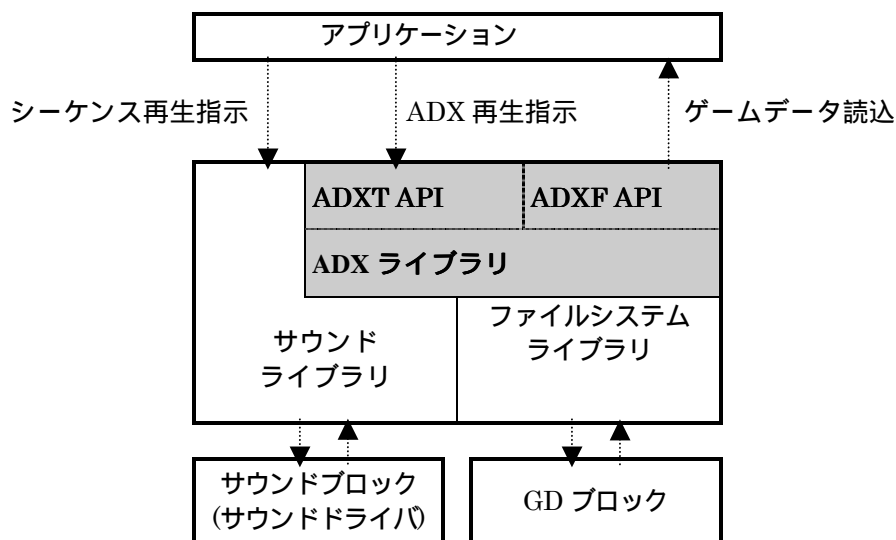


図 7 - 2 モジュール構成図



## 7.5 ADX データの流れ

(1) GD 上の ADX データの再生 (GD ストリーム再生、GD インデックス再生)

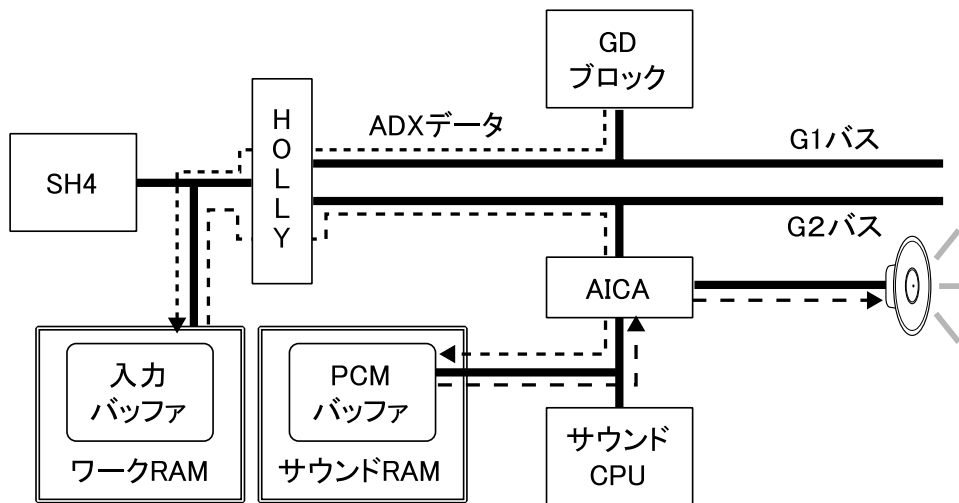


図 7 - 3 ADX データ再生

(2) メモリ上の ACX データの再生 (メモリ再生、メモリインデックス再生)

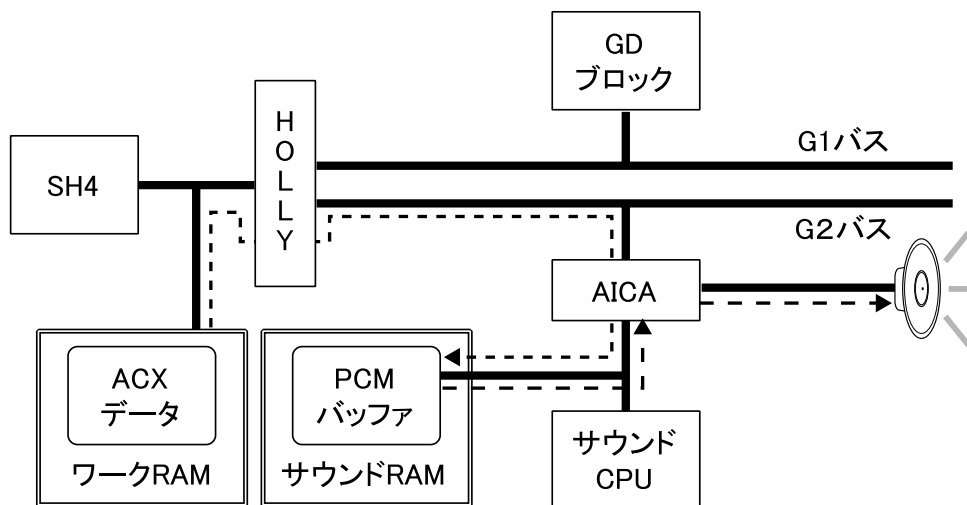


図 7 - 4 ACX データ再生



## 7.6 アプリケーションインタフェース (API)

ADX ライブラリの API は、オブジェクト指向的なインタフェースとなっています。まず、ADX を再生するためのハンドルである「ADX Talk(ADXT)」を生成します。そして、このハンドルに再生の指示を与えるだけで、GD やメモリ上の ADX データを簡単に再生することができます。ハンドル 1 個につき、1 つの ADX データを再生します。ADX データは、モノラルでもステレオでも再生することが可能です。複数のハンドルを生成することにより、同時に複数の ADX データを再生できます。

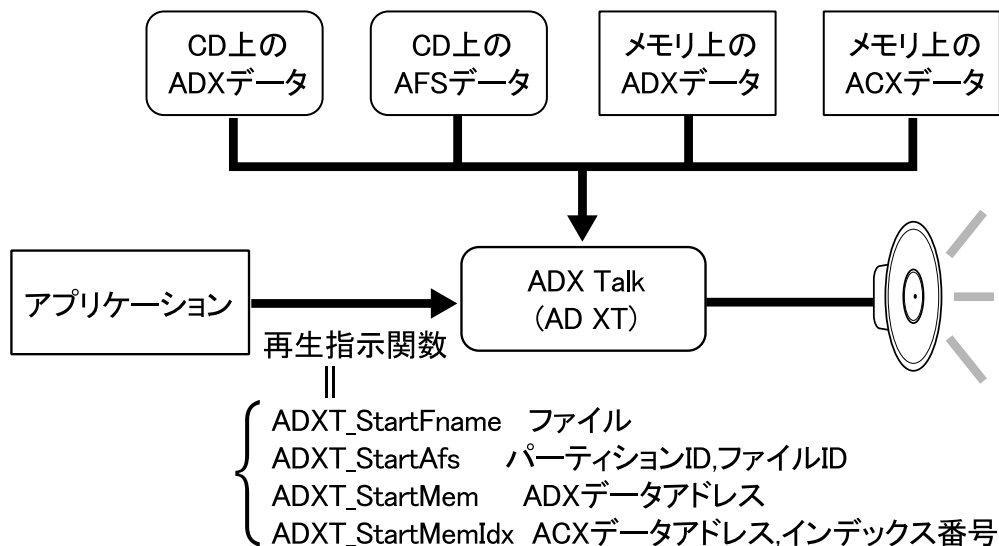


図 7 - 5 API 関連図

## 7.7 ADX データの作成方法

ADX データ、ACX データ及び AFS データの作成方法を示します。

### (1) ADX データ作成方法

ADX データは、PCM 音声データを ADX 圧縮したデータです。WAV または A I F F フォーマットの原音ファイルを、'adxencd.exe' プログラムにより圧縮します。操作方法は、以下のように引数に原音ファイル名を引数として、コマンドプロンプトより実行して下さい。カレントディレクトリに、拡張子が'ADX'になったファイルが生成されます。

```
C:\TEMP>adxencd sample.wav 'sample.wav' が 'sample.adx' に圧縮される。
```

原音と異なるサンプリング周波数の ADX データを作成する場合は、'sf' オプションを指定して下さい。指定できるサンプリング周波数は、原音の整数分の 1 となります。サンプリング周波数の変換を行う場合、変換後のサンプリング周波数の 0.45 倍をカットオフ周波数とする、バターワースフィルタで高域を減衰させています。高域の不足を感じる場合は、事前に市販のツールでサンプリング周波数を変換して下さい。

```
C:\TEMP>adxencd sample.wav -sf=22050 'sample.wav' をサンプリング周波数が 22050Hz の 'sample.adx' に圧縮する。
```



また、ループ再生するためには、以下のようにループスタートポイントとループエンドポイントをサンプル単位で指定します。

```
C:\TEMP>adxencd sample.wav -lps1500 -lpe200000
'sample.wav' が 'sample.adx' に圧縮される。
1500 サンプルから 199999 サンプルまでの間でループ再生する。
```

ループを指定する場合は、サンプリング周波数の指定は行わないで下さい。ループ時にノイズが発生する場合があります。サンプリング周波数の変換は、事前に別のツールで行って下さい。

さらに、lpa オプションによってファイル全体をループ再生することができます。

```
C:\TEMP>adxencd sample.wav -lpa
```

### ● 入力音声ファイルの形式

ファイルフォーマット	WAV または AIFF フォーマット
量子化ビット数	16 ビットまたは 8 ビット
サンプリング周波数	任意

### (2) ADX データの一括作成方法

ax.exe により大量の音声データを一括エンコードできます。dir/b コマンドによりファイルリスト作成し、そのリストファイルにより音声データファイルを一括エンコードします。

```
C:\TEMP>dir /b *.wav > wavlist.flis    WAV ファイルのリストファイルの作成
C:\TEMP>ax "adxencd %s" wavlist.flis    WAV ファイルを一括エンコード
```

### (3) ACX データ作成方法

ACX データは、上記の方法によって作成した複数の ADX データ（効果音やセリフ等）を連結したデータです。'adxcat.exe' プログラムにより結合します。以下のようにファイルリストを引数として与えます。

```
C:\TEMP>dir /b *.adx > se.flis          カレントディレクトリ内の ADX ファイルを結合
C:\TEMP>adxcat se.flis                  した、メモリインデックス再生用 'se.acx' が
                                         生成される。
```

### (4) AFS データ作成方法

AFS データは、ADX データやグラフィックデータ等を連結したデータです。'adxlnk.exe' プログラムにより結合します。以下のようにファイルリストを引数として与えます。

```
C:\TEMP>dir /b *.adx > pat01.als        カレントディレクトリ内のすべてのファイルを結合
C:\TEMP>afslnk pat01.als                した、'pat01.afs' が生成される。
```

カレントディレクトリ以外のディレクトリに格納されているデータを結合する場合は、以下のように指定します。

```
C:\TEMP>dir /b pat01\*.adx > pat01.als   pat01 ディレクトリ内のすべてのファイル
C:\TEMP>afslnk pat01.als -dir=pat01      を結合した、'pat01.afs' が生成される。
```



## 7.8 ライブラリ関数の使用方法

初期化、メモリ上の効果音の再生、GD からの ADX データファイルの再生等について、以下に説明します。

### 7.8.1 ライブラリの初期化と終了

ライブラリ全体の初期化を行うために ADXT\_Init 関数を、アプリケーションの起動時に実行して下さい。基本的にはゲームアプリケーションの開始時に 1 回だけ行って下さい。ADXT\_Finish 関数により ADX ライブラリを終了できます。

```
/* アプリケーション */
void main()
{
    SoundInit();
    /* 転送モードを DMA モードにする */
    sdMemBlkSetTransferMode(SDE_MEMBLK_TRANSFER_MODE_DMA);
    ADXT_Init();                /* ライブラリの初期化 */
    :
    ADXT_Finish();              /* ライブラリの終了 */
}
```

### 7.8.2 ADXT ハンドルの生成

初期化した後、作業用領域を確保し、以下のように ADXT ハンドルを生成して下さい。この関数によってサウンドライブラリの PCM ストリームポートのオープン等のリソースの確保を行います。従って、シーンの最初に必要なハンドルを確保し、シーンの途中での動的な生成と消去は行わない方が CPU 負荷が安定します。

```
/* 作業用領域の大きさを決定するためのパラメータ */
#define MAX_CH          (2)                /* ステレオ (モノラルも可能) */
#define GDSTM           (1)                /* GD ストリーム再生 */
#define MAX_SFREQ       (44100)            /* サンプリング周波数 44100 Hz */

/* 作業領域サイズ */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_STM, MAX_GDSTM, MAX_SFREQ))
/* 作業領域 */
static char work[WORKSIZE];

void adx_play_gdstm(void)
{
    ADXT adxt;                             /* ADXT ハンドル */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* ハンドルの生成 */
    :
    :
    (ゲームの 1 シーンの処理)
    :
    :
    ADXT_Destroy(adxt);                     /* ハンドルの消去 */
}
```



### 7.8.3 メモリ上の ADX データの再生 (メモリ再生)

メモリ上の ADX データは、ADXT\_StartMem 関数によって再生できます。ADXT ハンドルに対して ADX データのアドレスを指定して、ADXT\_StartMem 関数を実行します。メモリから再生する場合、作業領域は入力バッファを必要としないため、約 16Kbbyte×チャンネル数となります。

```
/* 作業用領域の大きさを決定するためのパラメータ */
#define MAX_CH (2) /* ステレオ (モノラルも可能) */
#define MAX_SFREQ (22050) /* サンプリング周波数 22050 Hz */

/* 作業領域サイズ */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_MEM, 0, MAX_SFREQ))
/* 作業領域 */
static char work[WORKSIZE];

void adx_play_mem(void)
{
    ADXT adxt; /* ADXT ハンドル */
    void *adx=(void *)0x8c100000; /* ADX データのアドレス */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* ハンドルの生成 */
    :
    ADXT_StartMem(adxt, adx); /* adx データを再生 */
    :
}
```

### 7.8.4 メモリ上の ACX データの再生 (メモリインデックス再生)

メモリ上の ACX データは、ADXT\_StartMemIdx 関数によって再生できます。ADXT ハンドルに対して ACX データのアドレスとインデックス番号を指定して、ADXT\_StartMemIdx 関数を実行します。

```
/* 作業用領域の大きさを決定するためのパラメータ */
#define MAX_CH (1) /* モノラルのみ */
#define MAX_SFREQ (22050) /* サンプリング周波数 22050 Hz */

/* 作業領域サイズ */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_MEM, 0, MAX_SFREQ))
/* 作業領域 */
static char work[WORKSIZE];

void adx_play_memidx(void)
{
    ADXT adxt; /* ADXT ハンドル */
    void *acx=(void *)0x8c100000; /* ACX データのアドレス */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* ハンドルの生成 */
    :
    ADXT_StartMemIdx(adxt, acx, no) /* acx データ内の no 番目を再生 */
    :
}
```



---

### 7.8.5 GD 上の ADX データの再生 (GD ストリーム再生)

---

GD 上の ADX データは、ADXT\_StartFname 関数によって再生できます。ADXT ハンドルに対してファイル名を指定して、ADXT\_StartFname 関数を実行します。

```
/* 作業用領域の大きさを決定するためのパラメータ */
#define MAX_CH      (2)          /* ステレオ (モノラルも可能) */
#define MAX_GDSTM   (3)          /* GD ストリーム並行再生数 (3) */
#define MAX_SFREQ   (44100)      /* サンプリング周波数 44100 Hz */

/* 作業領域サイズ */
#define WORKSIZE    (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_STM, MAX_GDSTM, MAX_SFREQ))
/* 作業領域 */
static char work[WORKSIZE];

void adx_play_gdstm(void)
{
    ADXT adxt;                  /* ADXT ハンドル */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* ハンドルの生成 */
    :
    ADXT_StartFname(adxt, "SE007.ADX");        /* "SE007.ADX" を再生 */
    :
    ADXT_StartFname(adxt, "SA019.WAV");        /* "SA019.WAV" を再生 */
    :
}
```



## 7.8.6 GD 上の AFS データの再生 (GD インデックス再生)

GD 上の ADX データは、以下の手順により再生できます。

- (1) 初期化時に ADXF\_LoadPartition 関数により、パーティション情報を読み込みます。この時、引数として関連づけるパーティション ID と AFS ファイル名、パーティション情報領域と最大ファイル数を指定します。
- (2) ADXT ハンドルに対してパーティション ID とファイル ID を指定して、ADXT\_StartAfs 関数を実行します。パーティション ID は (1) で定義した値、ファイル ID は結合したときのファイルリスト内の順番号となります。ファイル ID の指定に AFSLNK.EXE の出力するヘッダファイルを利用することもできます。

```
#include "pat01.h" /* ファイル ID の定義したヘッダファイル */
/* AFSLNK.EXE が出力 */

/* パーティション ID */
#define PAT01 (0)
/* パーティション内の最大ファイル数 */
#define MAX_PAT_NFILES (10000)

/* 作業用領域の大きさを決定するためのパラメータ */
#define MAX_CH (2) /* ステレオ (モノラルも可能) */
#define MAX_GDSTM (3) /* GD ストリーム並行再生数 (3) */
#define MAX_SFREQ (44100) /* サンプリング周波数 44100 Hz */
/* 作業領域サイズ */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_STM, MAX_GDSTM, MAX_SFREQ))

/* パーティション情報領域 */
static char ptinfo[ADXF_CALC_PTINFO_SIZE(MAX_PAT_NFILES)];
/* 作業領域 */
static char work[WORKSIZE];

void main(void)
{
    /* パーティション情報の読み込み */
    ADXF_LoadPartition(PAT01, "pat01.afs", ptinfo, MAX_PAT_NFILES);
    adx_play_afs();
}

void adx_play_afs(void)
{
    ADXT adxt; /* ADXT ハンドル */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* ハンドルの生成 */
    :
    ADXT_StartAfs(adxt, PAT01, SE007_ADX); /* "SE007.ADX" を再生 */
    /* SE007_ADX は "pat01.h" 内に定義 */
}
```



### 7.8.7 再生状態の取得

ADX ハンドルの状態を ADXT\_GetStat 関数によって取得することができます。この状態を監視することにより、ADX データの再生が終了しているか否かを確認することができます。

ADXT\_STAT\_PLAYING と ADXT\_STAT\_DECEND の時に実際に音声出力されます。

ADXT\_STAT\_PLAYEND になると再生が完了します。自動エラー回復モードが ADXT\_RMODE\_STOP の場合、GD-ROM のリードエラー等のエラーが発生すると、ADXT\_STAT\_STOP になり音声の再生が停止します。

以下に ADXT ハンドルの状態を示します。

定 数	値	状 態	音声出力
ADXT_STAT_STOP	0	停止中	停止
ADXT_STAT_DECINFO	1	ADX のヘッダ情報取得中	停止
ADXT_STAT_PREP	2	再生準備中	停止
ADXT_STAT_PLAYING	3	デコード & 再生中	発音
ADXT_STAT_DECEND	4	デコード終了	発音
ADXT_STAT_PLAYEND	5	再生終了	停止

#### ● 再生状態を取得する

```
long stat;

adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* ADXT ハンドルの生成 */
ADXT_StartFname(adxt, "SE007.ADX");          /* "SE007.ADX" を再生 */

for (;;) {
    stat = ADXT_GetStat(adxt);                /* 再生状態の取得 */
    if ( stat == ADXT_STAT_PLAYING || stat == ADXT_STAT_DECEND ) {
        /* 音声出力中の処理 */
        .
        .
    } else if ( stat == ADXT_STAT_PLAYEND ) {
        /* 再生終了時の処理 */
        .
        .
        break;
    } else if ( stat == ADXT_STAT_STOP ) {
        /* エラーにより停止 */
        .
        .
        break;
    }
}
```



## 7.9 サウンドデータ作成上の注意

ADX 再生ライブラリにより音声データを再生中に、サウンドブロック内の効果音や MIDI シーケンスを同時に再生できます。Shinobi 環境において ADX 再生ライブラリは、サウンドライブラリの PCM ストリーム機能を用いて実現されています。この時、PCM バッファは、サウンド RAM 最後から 4040H ずつ割り当てられます。従って、ADX ハンドルの使用するチャンネル数から PCM バッファのサイズを求め、その領域が重ならないようにマルチユニットファイルを作成する必要があります。

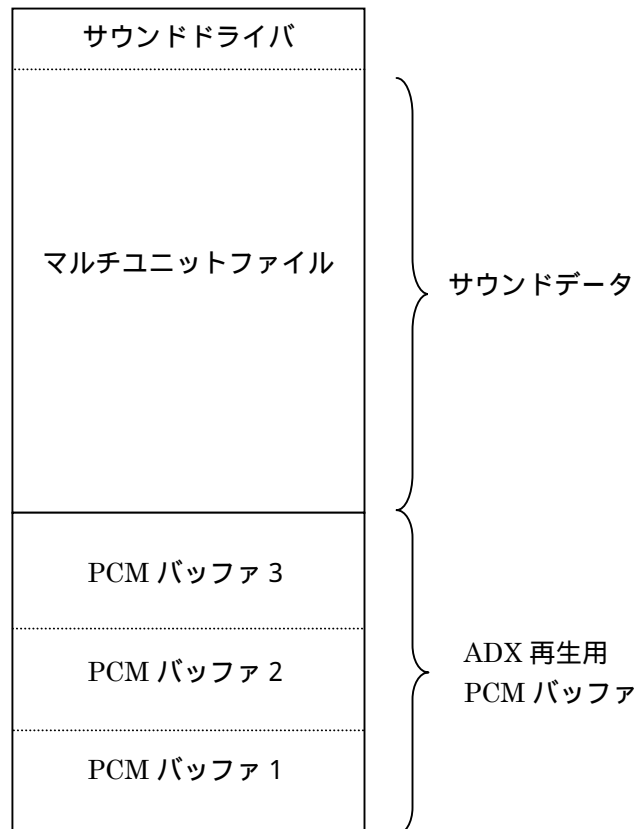


図 7 - 6 サウンド RAM の配置



## 7.10 データ読み込みの高速化について

ADX ファイルシステムにより、GD から音声を再生している最中に、データを読み込むことができます。しかしながら、デフォルトの設定では、入力バッファの 85% を割るとデータの再読み込みが始まるため、複数のゲームデータファイルを読み込む場合、ゲームデータと音声データを交互に読み込むことになります。ADXT\_SetReloadSct 関数を使用し音声データの再読み込み開始量を調整することにより、ゲームデータの読み込みを高速化できます。再読み込み開始量は、音声データの 1 秒分を指定します。例えば、44.1kHz のステレオ音声は約 50KByte/sec なので、25 セクタを指定することになります。また、入力バッファを大きくすることにより、再読み込みの間隔を大きくすることができるので、さらにデータ読み込みを高速化できます。暫定仕様なので、ADXT\_SetReloadSct 関数はリファレンスマニュアルには記載されていません。将来、このような指定をしなくても高速にデータが読み出せるようになる予定です。

```

/* 44kHz のステレオデータを再生する場合 */
#define WKSIZE      ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

char work[WKSIZE];          /* 作業領域 */
ADXT adxt;                  /* ADXT ハンドル */
ADXF adxf;                  /* ADXF ハンドル */

adxt = ADXT_Create(2, work, WKSIZE); /* ADXT ハンドルの生成 */
ADXT_SetReloadSct(adxt, 25); /* 再読み込み量の設定 */
.
ADXT_StartFname(adxt, "BGM.ADX"); /* 音声再生開始 */
.
adxf = ADXF_Open("GAMEDAT1.BIN", NULL); /* ファイルオープン */
ADXF_ReadNw32(adxf, nsct, buf); /* データ読み込み */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND )
;
ADXF_Close(adxf); /* ファイルクローズ */
.
(GAMEDAT1 が 2 秒以内に読み込めれば音声データの再読み込みは発生しない)
.
adxf = ADXF_Open("GAMEDAT2.BIN", NULL); /* ファイルオープン */
ADXF_ReadNw32(adxf, nsct, buf); /* データ読み込み */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND )
;
ADXF_Close(adxf); /* ファイルクローズ */

```

最大 GD ストリーム数 (ADXT\_CALC\_WORK の第 3 引数) を増やすことにより、入力バッファを増やすことができます。1 増やすごとに 1 秒分の余裕ができます。また、BGM とセリフを同時に再生するが、データを読み出すときにはセリフが再生されないことが保証されていれば、セリフのためのバッファ量をデータ読み込みに割り当てることができます。



## 7.11 シーク音の軽減方法

ADX 再生ライブラリは、デフォルトで反応速度を重視したバッファリング制御を行います。バッファの入力バッファの 85%を割るとデータ読み込みが発生します。従って、GD 上のデータの配置位置が離れていると頻繁にシーク音が発生します。このシーク音は、ADXT\_SetReloadSct 関数によって軽減することができます。

例えば、BGM とセリフを再生する場合、セリフの読み込み間隔を大きくすることによりシーク回数を減らすことができます。22kHz モノラルのセリフ再生する場合、ビットレートは、約 12.5KByte/sec になります。従って、2 ストリームの並行再生の場合、1 秒分バッファリングすれば並行再生できるので、セリフ側の再読み込みセクタ数を 7 セクタとすれば良いことになります。また、セリフ用の作業領域を計算する時に、並行再生ストリーム数を大きく指定することにより、入力バッファを大きくとることができます。これにより、よりシークの間隔を大きくすることができます。

```
/* 44kHz ステレオを 2 ストリーム並行再生する場合の作業領域 */
#define WKSIZ44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
/* 22kHz モノラルを 3 ストリーム並行再生する場合の作業領域 */
#define WKSIZ22M ADXT_CALC_WORK(1, ADXT_PLY_STM, 3, 22050)
/* 本来、2 ストリームしか再生しないが、3 ストリームとすることにより */
/* 入力バッファを増やすことができる */

char wk44[WKSIZ44S], wk22[WKSIZ22M]; /* 作業領域 */
ADXT adxt_bgm, adxt_srf; /* ADXT ハンドル */

adxt_bgm = ADXT_Create(2, wk44, WKSIZ44S); /* ADXT ハンドルの生成 */
adxt_srf = ADXT_Create(1, wk22, WKSIZ22M); /* ADXT ハンドルの生成 */
ADXT_SetReloadSct(adxt_srf, 7); /* 再読み込み量の設定 */
.
ADXT_StartFname(adxt_bgm, "BGM.ADX"); /* 音声再生開始 */
.
if ( イベント発生 )
    ADXT_StartFname(adxt_srf, "SRF001.ADX");
```



## 7.12 MPEG Sofdec との並行再生について

ADX 再生ライブラリにより MPEG Sofdec との並行再生ができます。BGM を GD ストリーム再生しながらムービーを非同期に並行再生することにより、ゲームシーンからムービーシーンへのシームレスな移り変わりを実現できます。しかしながら、ムービー用のバッファ容量を増やす必要があります。Sofdec ハンドルを生成する時に、最大ビットストリーム量を実際のストリーム量よりも大きい値を設定して下さい。目安としては、約 1.5 倍の数値を指定して下さい。これは、シークタイムを約 1 秒として見込んでいますので、データの配置の仕方によっては、減らせる可能性があります。また、ムービーが途切れる場合は、この値を増やして下さい。

```

/* 44kHz ステレオを 2 ストリーム並行再生する場合の作業領域 */
#define WKSIZ44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
/* 2 ストリームは、BGM とムービーという意味 */
char wk44[WKSIZ44S]; /* 作業領域 */

MWPLY ply; /* MWPLY ハンドル */
ADXT adxt_bgm; /* ADXT ハンドル */

adxt_bgm = ADXT_Create(2, wk44, WKSIZ44S); /* ADXT ハンドルの生成 */
ADXT_SetReloadSct(adxt_bgm, 25); /* シーク音の軽減のため */

/* MWPLY ハンドルの生成 */
memset(&cprm, 0, sizeof(cprm)); /* 予約メンバのゼロ設定のために必要 */
cprm.opt.first_hpel = ON;
cprm.ftype = MWD_PLY_FTYPE_MPV;
cprm.dtype = MWD_PLY_DTYPE_AUTO;
cprm.max_bps = 900*1024*8; /* 通常 600Kbyte/sec の 1.5 倍を指定 */
cprm.max_width = 320;
cprm.max_height = 240;
cprm.nfrm_pool_wk = 3;
cprm.wksize = mwPlyCalcWorkSofdec(
    cprm.ftype,
    cprm.max_bps,
    cprm.max_width,
    cprm.max_height,
    cprm.nfrm_pool_wk);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);

ADXT_StartFname(adxt_bgm, "BGM.ADX"); /* 音声再生開始 */
.
if ( イベント発生 )
    mwPlyStartFname(ply, "SCENE01.M1V"); /* ムービー再生 */

```



## 7.13 GDFS・Ninja ライブラリとの併用について

ADX ライブラリは、Shinobi ライブラリを使用しています。従って、基本的には GDFS との併用することができます。しかしながら、GDFS が GD-ROM のピックアップを占有してしまうと、音声データが読み込めずに音声途切れてしまうことがあります。基本的には、ADX ファイルシステムを使うことをお勧めしますが、njLoadTexture 関数等でどうしても GDFS を使用しなければならない場合は、ADXT\_IsIbufSafety 関数や ADXT\_GetNumSctIbuf 関数を用いて、入力バッファの状態を確認後、GDFS を使用して下さい。

ADXT\_IsIbufSafety 関数は、再読み込み開始セクタ数以上のデータが入力バッファにある時に 1 を返します。

ADXT\_GetNumSctIbuf 関数は、入力バッファ内のセクタ数を返します。音声ストリームのビットレートから音声途切れるまでの時間を求めて調節して下さい。

```
/* 44kHz のステレオデータを再生する場合 */
#define WKSIZE ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

char *work[WKSIZE]; /* 作業領域 */
ADXT adxt; /* ADXT ハンドル */
ADXF adxf; /* ADXF ハンドル */

adxt = ADXT_Create(2, work, WKSIZE); /* ADXT ハンドルの生成 */
.
ADXT_StartFname(adxt, "BGM.ADX"); /* 音声再生開始 */
.
while ( ! ADXT_IsIbufSafty(adxt) )
;
njLoadTexttrure(&tlist);
```



## 7.14 ADX マルチストリームシステムの仕組み

ADX マルチストリームシステムによって、GD-ROM 上の別々に格納された音声や動画ファイルを、並行再生することができます。

GD-ROM 上の音声ファイルであれば 4 ストリームまで同時に再生することができます。例えば、44kHz ステレオの BGM と効果音に、44kHz・モノラルのセリフ 2 つを同時に再生できます。また、SofdecF/X によって動画ファイルを再生しながら、別の音声ファイルを再生することもできます。

ADX ファイルシステムを併用することにより、GD-ROM 上の音声データを再生しながら、グラフィックデータ等を読み込むことができます。

以下に、ADX マルチストリームの仕組みを示します。

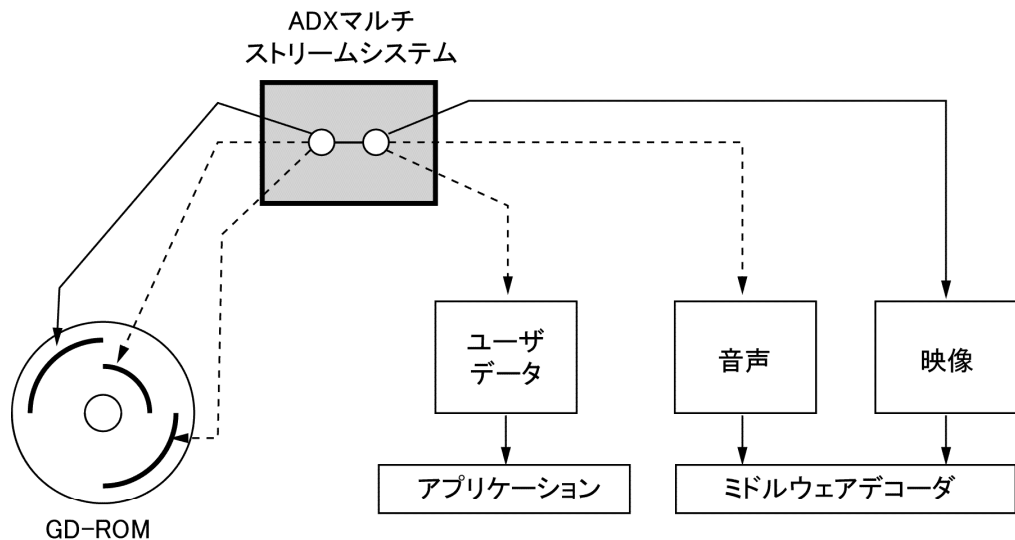


図 7 - 7 ADX マルチストリームの仕組み

ADX マルチストリームシステムは、データの流量管理を行います。各ストリームごとにデータバッファを持ち、データ量が再読込開始セクタ数を下回るとデータを読み込みます。デフォルトの再読込開始セクタ数は、バッファの 85%に設定されています。

以下に、ADX のデータ流量管理の仕組みを示します。

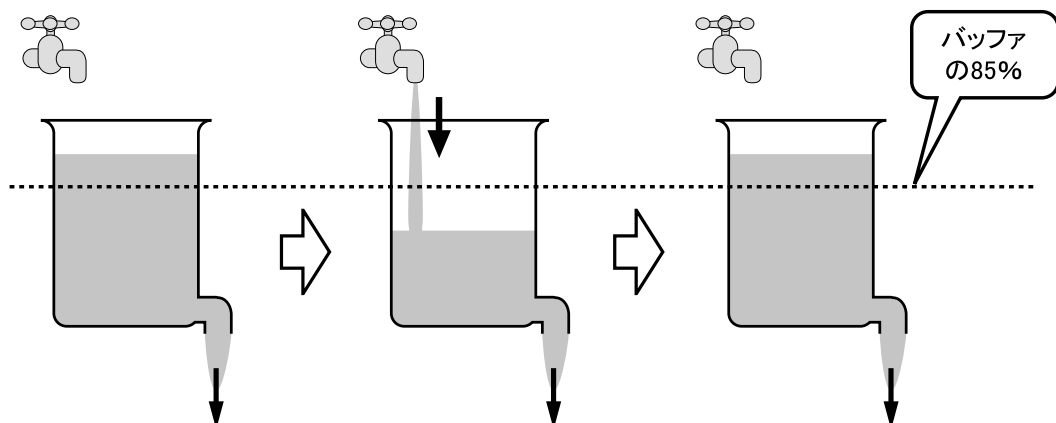


図 7 - 8 データ流量管理の仕組み



## 7.15 ループ再生

ADX は、音声データの一部をシームレスにループ再生できます。ADXT\_SetLpFlg 関数により、ループするか否かを設定できます。再生中にループを解除すると、その音声データの最後まで再生し終了します。再生中は、ループ解除のみ設定できます。

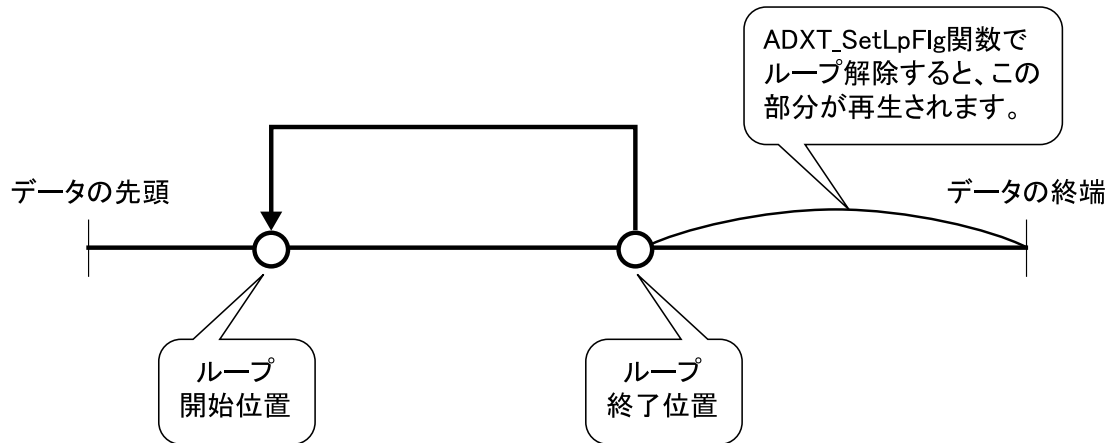


図 7 - 9 ループ再生



## 7.16 シームレス連続再生

ADX は、複数の音声データをシームレスに再生することができます。

下記の例では、音声データ smpsl1.adx、smpls1.adx、smpls2.adx、smpls2.adx、smpls3.adx の順に再生します。

```
// ADX ハンドルの生成
adxt = ADXT_Create(2, adxt_work, ADXT_WKSIZE);

// 再生する音声データの登録
ADXT_EntryFname(adxt, "smpls1.adx");
ADXT_EntryFname(adxt, "smpls1.adx");
ADXT_EntryFname(adxt, "smpls2.adx");
ADXT_EntryFname(adxt, "smpls2.adx");
ADXT_EntryFname(adxt, "smpls3.adx");

// シームレスループ再生の設定
ADXT_SetSeamlessLp(adxt, 1);

// シームレス連続再生の開始
ADXT_StartSeamless(adxt);
```

## 7.17 音声出力までのタイムラグをなくす方法

以下に、音声出力されるまでの流れを示します。

- (1) GD-ROM 上から音声データを読み込む。
- (2) 音声データのヘッダを解析する。
- (3) 音声データのデコード。
- (4) サウンド RAM 上の PCM ストリームバッファにデータをとどめる。
- (5) 音声出力開始。

従って、ADXT\_StartFname 関数を実行しても、すぐに音声は出力されません。  
そこで、以下の方法により、音声出力までのタイムラグをなくすることができます。

```

:
:
ADXT_Pause(adxt, ON);                /* 一時停止          */
ADXT_StartFname(adxt, "sample.adx"); /* 再生開始          */

for (;;) {
:
    if ( 音声を出力するタイミング ) {
        ADXT_Pause(adxt, OFF);        /* 一時停止の解除    */
    }
:
:
}
```



## 7.18 クロスフェード

クロスフェードすることにより、ゲーム中の音楽を途切れさせることなく、自然にシーンを切り替える事ができます。

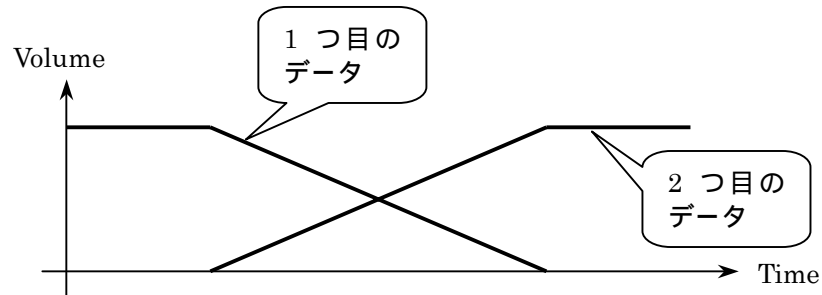


図 7-10 クロスフェードしているボリュームの時間変化

```

vol1 = 0;
vol2 = -1000;
for (;;) {
    if ( 1 回目のデータ再生を開始するタイミング ) {
        ADXT_SetOutVol(adxt1, vol1);
        ADXT_StartFname(adxt1, "sample1.adx");
    }
    if ( 2 回目のデータ再生を開始するタイミング ) {
        ADXT_SetOutVol(adxt2, vol2);
        ADXT_StartFname(adxt2, "sample2.adx");
    }
    if ( ADXT_GetStat(adxt2) == ADXT_STAT_PLAYING ) {
        ADXT_SetOutVol(adxt1, vol1-=10);
        ADXT_SetOutVol(adxt2, vol2+=10);
    }
}

```



## 7.19 サンプルプログラムについて

---

本ライブラリに添付しているサンプルプログラムの主なプログラムについて説明します。

- (1) adxgdply: 単純な GD リーム再生  
GD 上の音声データファイルを再生する簡単なサンプルプログラムです。'sample.adx'を再生します。
  
- (2) adxmem: メモリ再生&メモリインデックス再生  
GD 上の'sample.adx'と'smpl\_mem.acx'をワーク RAM 上に読み込み後、パッドによりメモリ再生とメモリインデックス再生を行うサンプルプログラムです。  
A ボタン : メモリ再生  
A ボタン以外 : メモリインデックス再生
  
- (3) adxmlt: マルチストリーム再生  
GD 上の4つのストリーム(ADX ファイル)を同時に再生するサンプルプログラムです。パッドにより、それぞれのストリームの再生を行うことができます。
  
- (4) adxrdsj : GD ストリーム再生 & GD データを読み込み (ADXF\_ReadSj32)  
GD 上の音声データを再生しながら、ADXF\_ReadSj32 関数を用いて、GD 上のデータファイル'sample.txt'を読み込むサンプルプログラムです。  
X ボタン : データファイルの読み込み開始
- (5) dxrdnw : GD ストリーム再生 & GD データを読み込み (ADXF\_ReadNw32)  
GD 上の音声データを再生しながら、ADXF\_ReadNw32 関数を用いて、GD 上の AFS ファイル'pat02.afs'を読み込むサンプルプログラムです。  
X、Y、左、右、下 : パーティション内の各データファイルの読み込み開始
  
- (6) adxslply: シームレス連続再生  
GD 上の複数の ADX ファイルを連続的に再生するサンプルプログラムです。
  
- (7) adxrec: メモリ録音  
マイクから入力された音声データをメモリ上に ADX として録音するサンプルプログラムです。  
X ボタン : メモリ上に ADX データとして録音開始  
Y ボタン : メモリ上に録音されたデータを再生開始



## 7.20 関数一覧

ADX 再生ライブラリでは、以下の関数がサポートされています。  
 詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
ADXT_Init	ADX ライブラリの初期化
ADXT_Finish	ADX ライブラリの終了処理
ADXT_CALC_WORK	作業領域サイズの計算
ADXT_Create	ADXT ハンドルの生成
ADXT_Destroy	ADXT ハンドルの消去
ADXT_StartFname	ファイル名指定による再生の開始
ADXT_StartMem	メモリ再生の開始
ADXT_StartMemIdx	メモリインデックス再生の開始
ADXT_StartAfs	GD インデックス再生の開始
ADXT_Stop	再生の停止
ADXT_GetStat	状態の取得
ADXT_GetTime	サンプル単位での再生時刻の取得
ADXT_GetTimeReal	実時間での再生時刻の取得
ADXT_GetNumSmpl	総サンプル数の取得
ADXT_GetSfreq	サンプリング周波数の取得
ADXT_SetOutVol	出力ボリュームの設定
ADXT_GetOutVol	出力ボリュームの取得
ADXT_SetOutPan	パンポットの設定
ADXT_GetOutPan	パンポットの取得
ADXT_SetSvrFreq	サーバ関数の呼び出し頻度の設定
ADXT_ExecServer	サーバ関数（内部状態の更新）
ADXT_GetErrCode	エラーコードの取得
ADXT_ClearErrCode	エラーコードのクリア
ADXT_SetAutoRcvr	自動エラー回復機能の設定
ADXT_EntryErrFunc	エラーコールバック関数の設定
ADXT_Pause	再生の一時停止・再開
ADXT_EntryFname	シームレス連続再生ファイルの登録
ADXT_StartSeamless	シームレス連続再生の開始
ADXT_SetSeamlessLp	シームレスループ再生の設定
ADXT_StartFnameLp	シームレスループ再生の開始
ADXT_ReleaseSeamless	シームレス連続再生の解除
ADXT_GetNumFiles	登録ファイル数の取得



## 8 ADX ファイルシステムライブラリ

この ADX ファイルシステムライブラリは、音声をストリーム再生させながらデータを読み込むためのライブラリです。

### 8.1.1 ADX ファイルシステムライブラリの概要

ADX ファイルシステムによって、大量のファイルを簡便に扱うことができます。複数のファイルをあらかじめ結合しておき、ADX ファイルシステムパーティションファイル（以下、AFS ファイル）として GD 上に格納します。ADX ファイルシステムは、結合された AFS ファイル内から、任意のファイルを読み出すことができます。この仕組みにより、大量のセリフデータ等を簡単に再生することができます。

#### (1) ADX ファイルシステムへのアクセス

ADX ファイルシステムのアクセスは、以下の 2 つのキーにより行います。

##### a. パーティション ID

ユーザの決定する任意の数値（0～255）。

ADXF\_LoadPartition 関数によって、パーティション情報を読み込み、パーティション ID と AFS ファイルと関連付けます。

##### b. ファイル ID

AFS ファイル内の順番号。

結合ツール（AFSLNK.EXE）は、ファイル ID をヘッダファイルとして出力します。

ADX ファイルシステムへのアクセスは、以下の手順により行います。

#### (2) パーティション情報の読み込み

ADXF\_LoadPartition 関数により、パーティション情報を読み込みます。

パーティション情報は、1 ファイルあたり約 2 Byte の領域が必要です。

#### (3) AFS ファイル内の ADX データの再生

ADXT\_StartAfs 関数により、パーティション ID とファイル ID を指定して再生します。

#### (4) AFS ファイル内のデータの読み出し

ADXF\_OpenAfs 関数により、ファイルハンドルをオープンし、ゲームデータ等読み出します。

ADX ファイルシステムは、GD ストリーム再生中でも、GD 上のデータを読み出すことが可能です。

### 8.1.2 ADX ファイルシステムライブラリの特徴

ADX ファイルシステムの特徴を、以下に示します。

- 音声をストリーム再生させながら、データを読み込むことができます。
- ISO9660 ファイル（普通のファイル）の他に、複数のファイルを結合したファイル（AFS ファイル）を読み込むことができます。
- 複数のファイルを同時に開き、並列に読み込むことができます。



8.1.3 モジュール構成

ADX ファイルシステムのモジュール構成図を以下に示します。

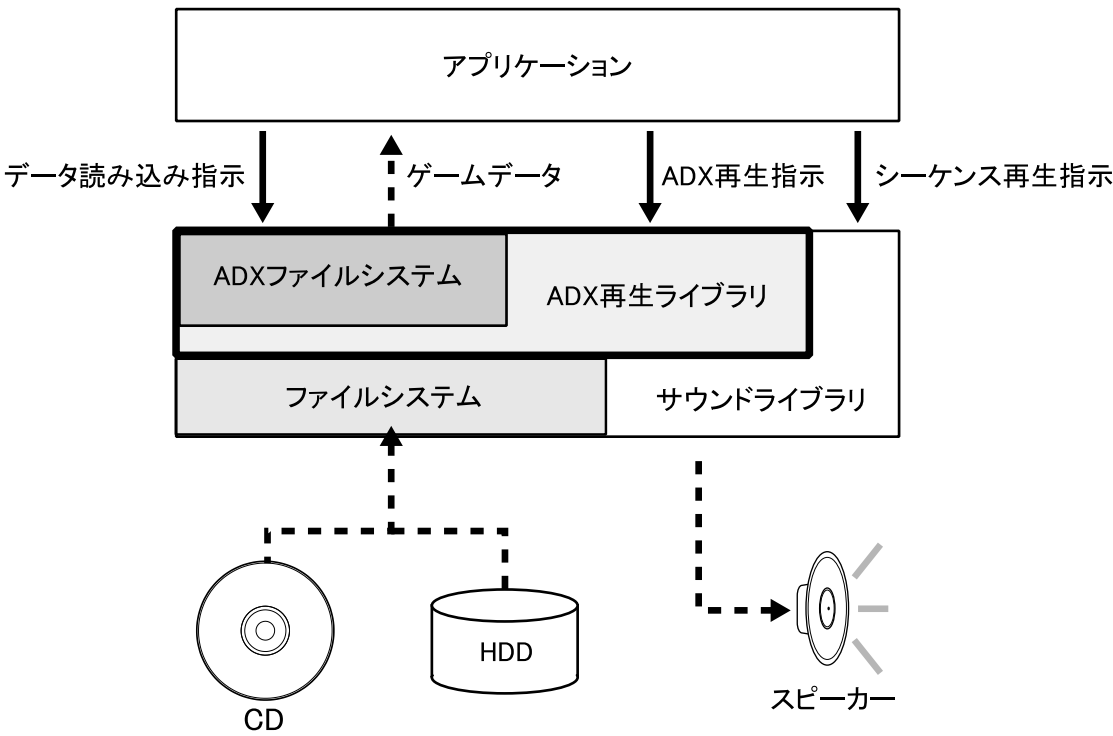


図 8 - 1 モジュール構成

8.1.4 ADX ファイルシステムの使用方法

以下の手順により、ディスクからメモリへのデータ転送を行います。

- a. ファイルをオープンします。 (ADXF\_Open/ADXF\_OpenAfs)
- b. 読み込みリクエストを発行します。 (ADXF\_ReadNw32)
- c. 読み込み終了状態になるまで、状態を管理します。 (ADXF\_GetStat)
- d. 終了状態になったら、ファイルをクローズします。 (ADXF\_Close)

ハンドルの状態を以下に示します。

表 8 - 1 ハンドルの状態

状 態	説 明
STOP	データ読み込みを停止している状態
READING	データ読み込み中の状態
READEND	データ読み込みが終了した状態
ERROR	エラー状態

ハンドルの状態は、ADXF\_ReadNw32 関数により STOP から READING に変わり、指定のセクタ数のデータを読み込み終わると、自動的に READEND に変わります。



状態遷移図を以下に示します。

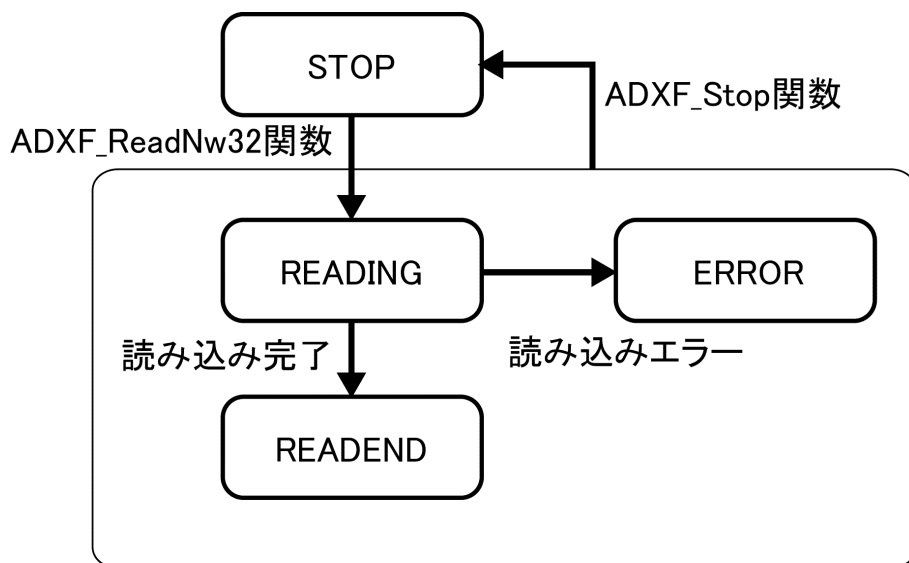


図 8 - 2 状態遷移図

### 8.1.5 ISO9660 ファイルの読み込み方

以下に、ISO9660 ファイルを読み込むサンプルプログラムを示します。

#### ● サンプルプログラム

```

/* アプリケーションメイン関数 */
void main(void)
{
    ADXF    adxf;                                /* ADXF ハンドル          */
    long    rd_nsct = 1000;                       /* 読み込みセクタ数      */
    char    *rd_buf = (char *)0x8C800000;         /* バッファアドレス      */

    /*
     * H / W等の初期化
     */
    ADXT_Init();                                /* ライブラリの初期化    */
    adxf = ADXF_Open("SAMPLE0.DAT");             /* ファイルオープン      */
    ADXF_ReadNw32(adxf, rd_nsct, rd_buf);         /* 読み込み開始          */
    for (;;) {
        if (ADXF_GetStat(adxf) == ADXF_STAT_READEND) /* 読み込み終了チェック */
            break;
    }
    /*
     * V-Sync 待ち
     */
}
ADXF_Close(adxf);                               /* ファイルクローズ      */
}

```



### 8.1.6 AFS ファイルへのアクセス方法

AFS ファイルを使用することによって、大量のファイルを簡便に扱うことができます。複数のファイルをあらかじめ結合しておき、AFS ファイルとしてディスク上に格納します。ADX ファイルシステムは、結合された AFS ファイル内から、任意のインサイドファイルを読み込むことができます。

### 8.1.7 AFS ファイルの作成方法

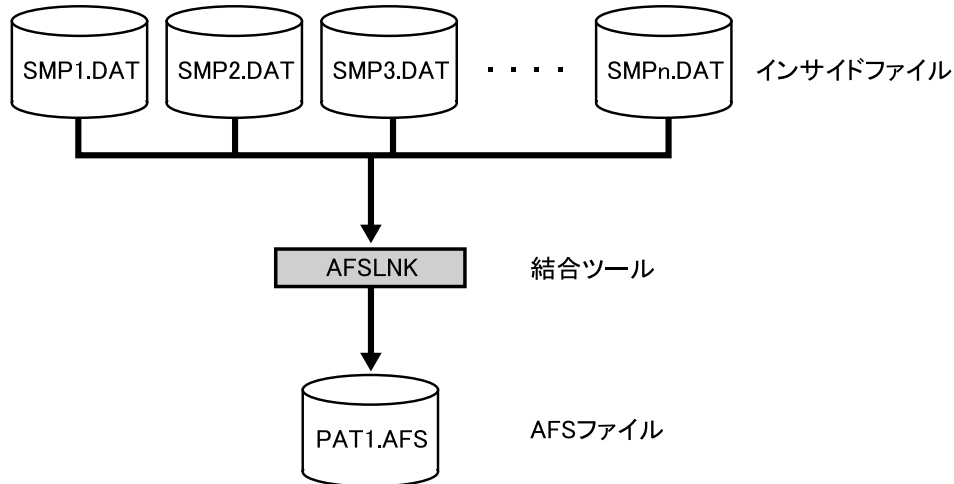


図 8 - 3 AFS ファイルの作成

AFS ファイルは、ADX データやグラフィックデータ等を連結したファイルです。afslnk.exe プログラムにより結合します。

以下のようにファイルリストを引数として与えます。

C:¥TEMP>dir /b *.adx > pat01.als	カレントディレクトリ内の全ての ADX ファイルを
C:¥TEMP>afslnk pat01.als	結合した、pat01.afs が生成されます。

カレントディレクトリ以外のディレクトリに格納されているデータを結合する場合は、以下のように指定します。

C:¥TEMP>dir /b pat01¥*.adx > pat01.als	pat01 ディレクトリ内の全ての ADX ファイル
C:¥TEMP>afslnk pat01.als -dir=pat01	を結合した、pat01.afs が生成されます。



### 8.1.8 インサイドファイルの読み込み方

以下では、インサイドファイルの読み込み方について説明するため、AFS ファイルをパーティションと呼びます。

インサイドファイルへのアクセスは、以下の手順により行います。

#### (1) パーティション情報の読み込み

ADXF\_LoadPartition 関数により、パーティション情報を読み込みます。パーティション情報は、1つのインサイドファイル当たり約 2 Byte の領域が必要です。

```
static char ptinfo[ADXF_CALC_PTINFO_SIZE(5)]; /* パーティション情報領域 */
long      pi_sz; /* パーティション情報サイズ */

ADXT_Init(); /* ライブラリの初期化 */
ADXF_LoadPartition(0, "PAT0.AFS", ptinfo, 5); /* パーティションのロード */
```

複数のパーティションを扱う場合は以下のように行います。1つのパーティション情報領域を複数のパーティションで使用することができます。

```
static char ptinfo[ADXF_CALC_PTINFO_SIZE(256)]; /* パーティション情報領域 */
long      pi_sz; /* パーティション情報サイズ */

ADXT_Init(); /* ライブラリの初期化 */
ADXF_LoadPartition(0, "PAT0.AFS", ptinfo, 5); /* パーティションのロード */
pi_sz = ADXF_GetPtinfoSize(0); /* パーティション情報サイズの取得 */
ADXF_LoadPartition(1, "PAT1.AFS", ptinfo+pi_sz, 8);
:
```

#### (2) パーティション内のデータの読み込み

ADXF\_OpenAfs 関数により、ファイルをオープンし、グラフィックのデータファイル等読み出します。

```
ADXF  adxf; /* ADXF ハンドル */
long  pati_id = 1; /* パーティション ID */
long  file_id = 3; /* ファイル ID */
long  rd_nsct = 1000; /* 読み込みセクタ数 */
char  *rd_buf = (char *)0x8C800000; /* バッファアドレス */

adxf = ADXF_OpenAfs(pati_id, file_id); /* ファイルオープン */
ADXF_ReadNw32(adxf, rd_nsct, rd_buf); /* 読み込み開始 */
for (;;) {
    if (ADXF_GetStat(adxf) == ADXF_STAT_READEND) /* 読み込み終了チェック */
        break;
    /* V-Sync 待ち */
}
ADXF_Close(adxf); /* ファイルクローズ */
```

### 8.1.9 追記したインサイドファイルの読み込み方

開発の途中段階で、インサイドファイルを追加・差し替えする場合、AFS ファイルを再結合すると作業効率が悪いので、追記機能を検討しています。現バージョンでは未実装のため説明を省きます。



### 8.1.10 関数一覧

ADX ファイルシステムライブラリでは、以下の関数がサポートされています。  
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
ADXF_AddPartition	追記パーティション情報の読み込み
ADXF_Close	ファイルのクローズ
ADXF_ExecServer	サーバ関数
ADXF_Finish	ライブラリの終了処理
ADXF_GetFsizeSct	ファイルサイズの取得
ADXF_GetNumReadSct	実際に読み込んだセクタ数の取得
ADXF_GetNumReqSct	読み込み要求情報の取得
ADXF_GetPtinfoSize	パーティション情報サイズの取得
ADXF_GetStat	ハンドルの状態の取得
ADXF_Init	ライブラリの初期化
ADXF_LoadPartition	パーティション情報の読み込み
ADXF_Open	ファイルオープン(ISO9660 フォーマット)
ADXF_OpenAfs	ファイルのオープン(AFS フォーマット)
ADXF_ReadNw32	データの読み込み開始
ADXF_ReadSj32	ストリームジョイントへの読み込み開始
ADXF_Seek	アクセスポインタの移動
ADXF_Stop	データの読み込み停止
ADXF_Tell	アクセスポインタの取得



## 9 DualSpeech

ここでは、DualSpeech についてや圧縮音声（DUS）データを作成する説明等を説明します。

### 9.1 概要

「DualSpeech 用音声データ作成ツール（以下、DUS データ作成ツール）」とは、素材データ（WAV フォーマット）から Dreamcast 用 DualSpeech がデコード可能な圧縮音声データ（DUS ファイル）を PC 上で作成したり、DUS ファイルから試聴確認用に WAV ファイルを再生成（伸長）するためのツールです。ツールには次の 2 つがあります。

- (1) DualSpeech エンコーダ：WAV フォーマットの音声データから、DualSpeech 用の DUS データに圧縮して作成します。（作成した DUS データは、Dreamcast 用 DualSpeech で利用可能です）
- (2) DualSpeech デコーダ：素材データとの比較や試聴用に、(1)で作成した DUS データから WAV フォーマットのファイルに伸長して生成します。

**注 意** 本ツールには、「素材データ」を新規に作成・編集する機能、及び「音声を出力」する機能を有しておりません。

そのため、「素材データ」の作成・編集を行うためのソフトウェアと「音声を出力」するためのソフトウェアを事前に用意して頂く必要があります。

### 9.2 動作環境

DUS データ作成ツールは、以下の環境で動作します。

表 9 - 1 動作環境

項 目	内 容
PC（CPU）	Pentium90MHz 以上を搭載の DOS / V 機
OS	MS - Windows95 / MS - WindowsNT4.0 以降
メモリ	1 6 M B 以上
ビデオ	640 × 480 256 色以上
サウンド	8 KHz サポートのサウンドカード
その他	MS-DOS プロンプトを使用



## 9.3 DUS データ作成の流れ

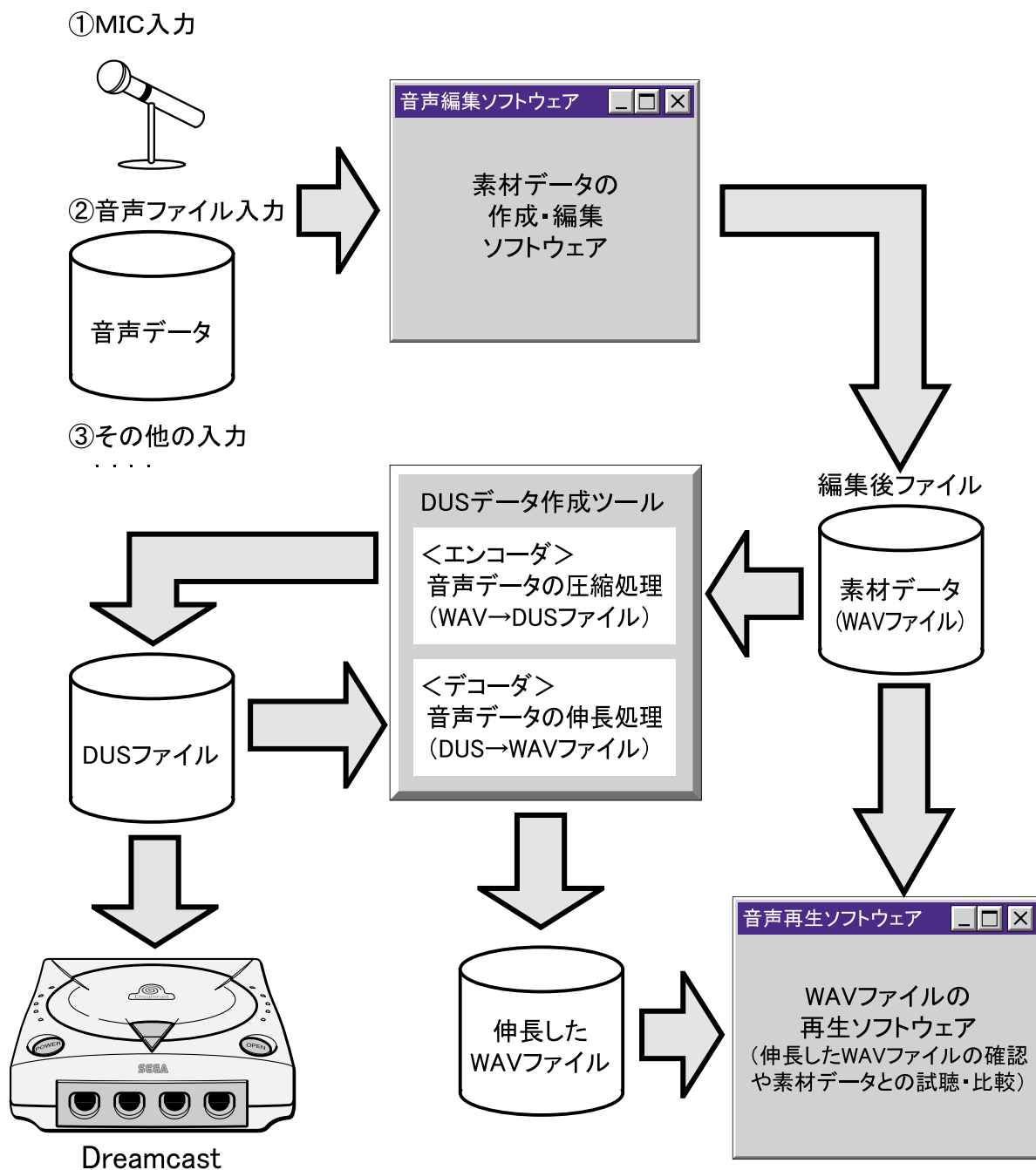


図 9 - 1 DUS データ作成の流れ



## 9.4 素材の準備

### 9.4.1 素材のフォーマットと準備方法

#### (1) 素材ファイルのフォーマットについて

素材ファイルは、WAV ファイルフォーマットを標準とします。本ツールの使用にあたっては、素材ファイルを次の条件で作成・編集する必要があります。

表 9 - 2 素材ファイル作成時の条件

項 目	内 容
ファイルフォーマット	WAV ファイルフォーマット
データタイプ	PCM データ
圧縮	非圧縮
サンプリング周波数	8000Hz
量子化ビット数	16 ビット
音声チャンネル	モノラル
データサイズ	特になし

#### (2) DUS ファイルのフォーマットについて

DUS ファイルは、PC 上の音声ファイル（WAV フォーマット）に対して、DualSpeech オリジナルの圧縮を行ったファイルフォーマットであり非公開です。その圧縮後のファイルサイズは、素材ファイルの内容にもよりますが通常、1 / 20 ~ 1 / 40 以下のファイルサイズ（無音部分が多い場合、特に圧縮率が高くなります）となります。

### 9.4.2 素材準備の注意点

本ツール（エンコーダ）が入力する素材ファイル（WAV ファイル）のフォーマットにおいて、当初の予定ではサンプリング周波数を 11025Hz にも対応させていましたが、処理速度 / データ品質（音質）等の兼ね合いから 8000Hz のみの対応としました。



## 9.5 DUS ファイルの作成方法

以下に、DUS ファイルの作成方法（エンコーダの使用法）について説明します。DUS ファイルの作成に当たっては、事前にエンコーダに入力可能な素材ファイルを用意して下さい。

- (1) 素材ファイル毎の個別に処理（エンコード）する場合
  - a. MS-DOS のコマンドプロンプトを実行し、制御を MS-DOS プロンプトウィンドウに移します。
  - b. エンコーダの実行パスが通っている（インストールした）ディレクトリに移動します。
  - c. エンコーダ名の指定の後に、実行（エンコード）したい素材ファイルのディレクトリ名とファイル名を第 1 パラメータとしてコマンド入力して実行させます。
- 例
 

```
> [インストールディレクトリ] wav2dus .bat [素材ファイルディレクトリ] sozai.wav
```
- d. 第 1 パラメータに指定した素材ファイルと同一ディレクトリ内に、「素材ファイル名.dus」というファイルが作成されていることを確認して下さい。（DUS ファイルのファイル拡張子は「.dus」です）
- (2) 複数の素材ファイルを一度に処理（エンコード）する場合
  - a. MS-DOS のコマンドプロンプトを実行し、制御を MS-DOS プロンプトウィンドウに移します。
  - b. エンコーダの実行パスが通っている（インストールした）ディレクトリに移動します。
  - c. エンコーダ名の指定の後に、実行（エンコード）したい素材ファイルのディレクトリ名とファイル名の代わりに「\*.wav」を第 1 パラメータとしてコマンド入力して実行させます。
- 例 1 ) : ディレクトリ内の全ファイルを実行
 

```
> [インストールディレクトリ] wav2dus .bat [素材ファイルディレクトリ] *.wav
```
- 例 2 ) ディレクトリ内で先頭から「onsei」をファイル名に持つファイルのみを実行
 

```
> [インストールディレクトリ] wav2dus .bat [素材ファイルディレクトリ] onsei*.Wav
```
- d. 第 1 パラメータに指定した素材ファイルと同一ディレクトリ内に、「素材ファイル名.dus」というファイルが実行したファイル対応数分作成されていることを確認して下さい。（DUS ファイルのファイル拡張子は「.dus」です）



## 9.6 DUS ファイルの確認方法

DUS ファイルの確認は、作成した DUS ファイルをもう一度 WAV フォーマット形式のファイルにする処理（伸長）を行った後に、音声ファイルの再生ソフトウェア等にて試聴確認します。以下に、DUS ファイルの確認方法（デコーダの使用方法）について説明します。

### (1) DUS ファイル毎の個別に処理（デコード）する場合

- a. MS-DOS のコマンドプロンプトを実行し、制御を MS-DOS プロンプトウィンドウに移します。
- b. デコーダの実行パスが通っている（インストールした）ディレクトリに移動します。
- c. デコーダ名の指定の後に、実行（デコード）したい DUS ファイルのディレクトリ名とファイル名を第 1 パラメータとしてコマンド入力して実行させます。

#### ● 例 1 ) ディレクトリ内の全ファイルを実行

```
> [インストールディレクトリ]dus2wav .bat [DUS ファイルディレクトリ]dualspeech .dus
```

- d. 第 1 パラメータに指定した DUS ファイルと同一ディレクトリ内に、「DUS ファイル名\_dec .wav」というファイルが作成されていることを確認して下さい（確認用ファイルのファイル拡張子は「.wav」です）。
- e. 「再生ソフトウェア」を起動して、作成した WAV ファイルを入力して試聴確認して下さい。

### (2) 複数の DUS ファイルを一度に処理（デコード）する場合

- a. MS-DOS コマンドプロンプトを実行し、制御を MS-DOS プロンプトウィンドウに移します。
- b. デコーダの実行パスが通っている（インストールした）ディレクトリに移動します。
- c. デコーダ名の指定の後に、実行（デコード）したい DUS ファイルのディレクトリ名とファイル名の代わりに「\*.wav」を第 1 パラメータとしてコマンド入力して実行させます。

#### ● 例 1 ) : ディレクトリ内の全ファイルを実行

```
> [インストールディレクトリ]dus2wav .bat [DUS ファイルディレクトリ]*.dus
```

#### ● 例 2 ) : ディレクトリ内で先頭から「dual」をファイル名に持つファイルのみを実行

```
> [インストールディレクトリ]dus2wav .bat [DUS ファイルディレクトリ]dual*.dus
```

- d. 第 1 パラメータに指定した素材ファイルと同一ディレクトリ内に、「DUS ファイル名\_dec .wav」というファイルが実行したファイル対応数分作成されていることを確認して下さい。（確認用ファイルのファイル拡張子は「.wav」です）
- e. 「再生ソフトウェア」を起動して、作成した WAV ファイルを入力して試聴確認して下さい。



## 9.7 関数一覧

Dualspeech ライブラリでは、以下の関数がサポートされています。  
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
mwExecMainServer	サーバ関数
mwPlyCreateDlsd	DualSpeech 用ハンドルの生成
mwPlyFinishDlsd	DualSpeech ライブラリの終了処理
mwPlyInitDlsd	DualSpeech ライブラリの初期化
mwRecCreateDlse	DualSpeech 用ハンドルの生成
mwRecDestroy	ハンドルの消去
mwRecEntryErrFunc	エラー発生時に呼ばれる関数の登録
mwRecFinishDlse	DualSpeech ライブラリの終了処理
mwRecGetStat	ハンドルの状態の取得
mwRecInitDlse	DualSpeech ライブラリの初期化
mwRecPause	ポーズの設定
mwRecStartMem	録音開始（メモリへの録音）
mwRecStartSj	録音開始（ストリームジョイントへの録音）
mwRecStop	録音停止



## 10 Sofdec 用動画データ作成ツール

ここでは、CRI MPEG Sofdec 用動画データ(以降 Sofdec データ)用の素材作成時の注意点、Sofdec データの作成方法、作成に必要なツールについて説明します。

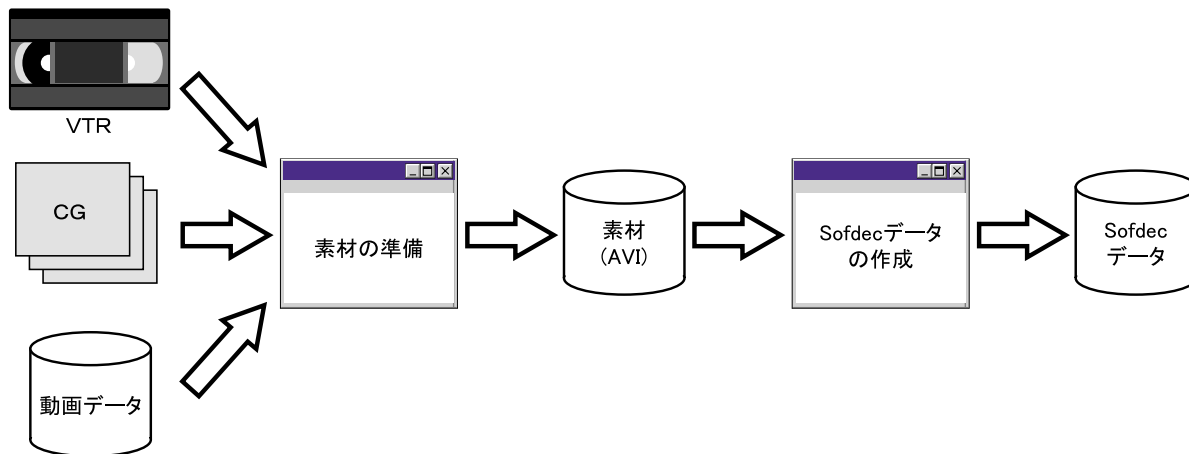


図 10-1 Sofdec データ作成の流れ

### 10.1 動作環境

各 Sofdec データ作成ツールは、以下の環境で動作します。

表 10-1 動作環境

項 目	説 明
ハードウェア	Pentium シリーズ搭載 DOS/V 機 (MMX 対応 CPU 搭載機を推奨)
OS	MS-Windows95 以降、MS-WindowsNT4.0 以降
メモリ	32MB 以上



## 10.2 素材の準備

素材の準備では、Sofdec データの元となる素材作成し、AVI ファイルにする作業を行います。

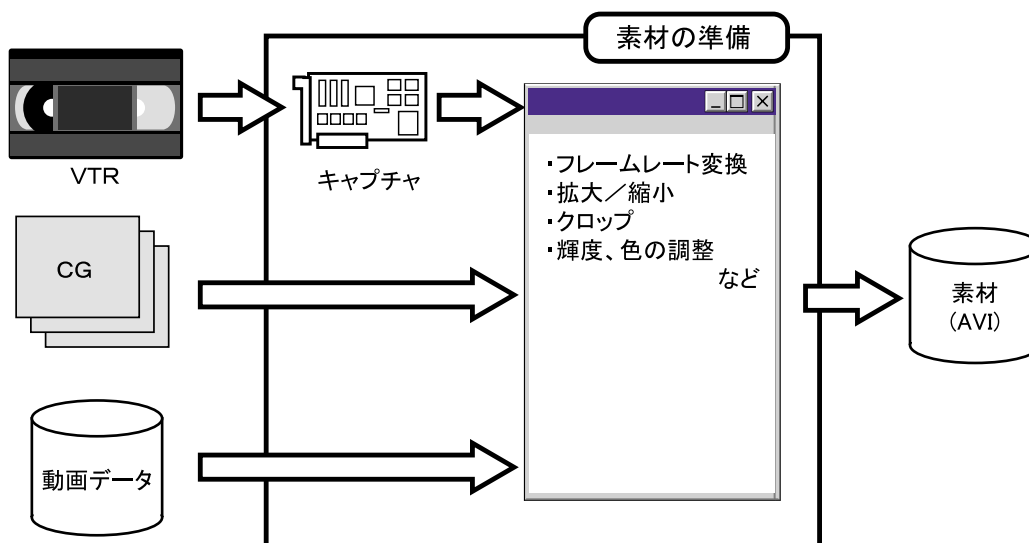


図 10-2 素材の準備

### 10.2.1 素材ファイルのフォーマットと方法

Sofdec データ作成ツールは AVI フォーマットを標準入力とします。

表 10-2 エンコード可能な映像ファイル

項 目	説 明
フォーマット	AVI フォーマット( 1 )
画像サイズ	横 32×縦 16 ピクセルの倍数
色数	24bit カラー ( 約 1600 万色 )
圧縮	非圧縮推奨
フレームレート(fps)	10, 15, 23.976, 24, 25, 29.97, 30 (10fps,15fps は、SEGA Dreamcast Movie Creator のみ対応)

1 : 動画フォーマットについて

SEGA Dreamcast Movie Creator プラグイン版は、Adobe Premiere のサポートするすべての動画フォーマットが使用可能です。Premiere は AVI 以外に、QuickTime、ビットマップシーケンス、TARGA シーケンス等に対応しています。プラグイン版はこれらのフォーマットから Sofdec データを作成することができます。

表 10-3 エンコード可能な音声ファイル

項 目	説 明
フォーマット	AVI フォーマット ( 映像とインタリーブ )、WAV フォーマット、AIFF フォーマット。
サンプリング周波数(Hz)	44100, 22050, 14700, 11025, 8820, 7350, 6300
圧縮	非圧縮推奨
量子化ビット数	8bit または 16bit
音声チャンネル	モノラルまたはステレオ



## 10.2.2 フレームレート

素材のフレームレートは次の点に注意し、設定や変換を行って下さい。

## (1) 新規に素材を作成する場合

Dreamcast が対応する映像信号方式には NTSC 方式、PAL 方式、VGA があります。素材のフレームレートを使用する映像信号方式にあわせて下さい。

表 10-4 映像信号方式

映像信号方式	フレームレート	ピクセル縦横比	使用地域	備 考
NTSC 方式	29.97fps	1.0950	日本、アメリカ	インタレース方式
PAL 方式	25fps	0.9157	ヨーロッパ	インタレース方式
VGA	59.94fps	1.0000	(PC 用モニター)	ノンインタレース方式 動画は 29.97fps で作成

## (2) 既存の素材を利用する場合

Sofdec データ作成ツールは、次表のフレームレートに対応しています。素材がすでに存在する場合はフレームレート変換を行わないで下さい。

表 10-5 Sofdec データ作成ツール対応フレームレート

項 目	対応値
フレームレート (fps)	10, 15, 23.976, 24, 25, 29.97, 30

## (3) 既存の素材を変換して使用する場合

既存の素材のうち、次の場合は変換が必要です。

- 素材のフレームレートが、Sofdec データ作成ツールの対応するフレームレートではない場合。  
動画編集ソフトで、フレームレートを変換して下さい。
- 24fps のフィルム素材（アニメ等）が VTR 等に録画されている場合。  
VTR 素材としてキャプチャした場合、フレームレートは 29.97fps になります。  
フィルム素材をキャプチャする場合、逆テレシネ変換を行い 24fps に変換して下さい



### 10.2.3 ピクセル縦横比

テレビのピクセル縦横比は映像信号方式によって異なります。素材を作成する場合、ピクセルの縦横比率を映像信号方式に合わせて下さい。

表示時のピクセル縦横比と素材が異なる場合、映像は正しい形になりません。たとえば画面に 100 × 100 ピクセルの四角形を書いた場合、NTSC 方式では縦長に、PAL 方式では横長に、VGA では正方形になります。素材と表示時のピクセル縦横比が同じになるよう注意して下さい。

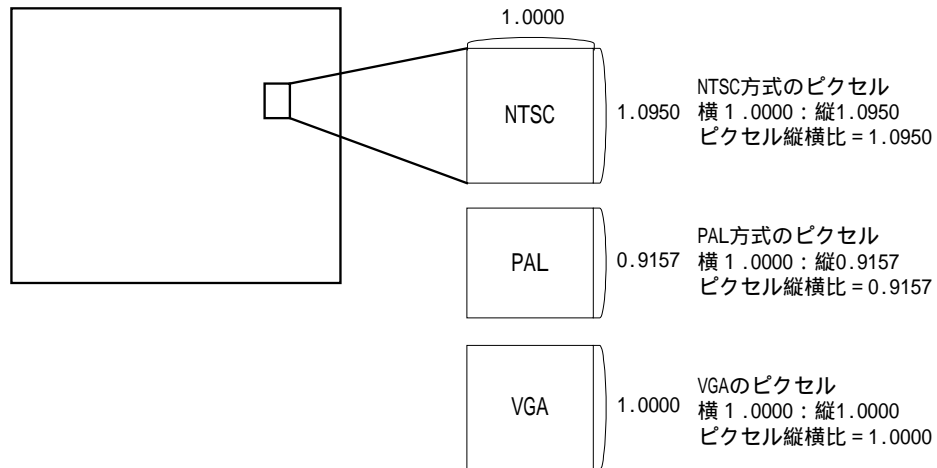


図 10-3 ピクセル縦横比

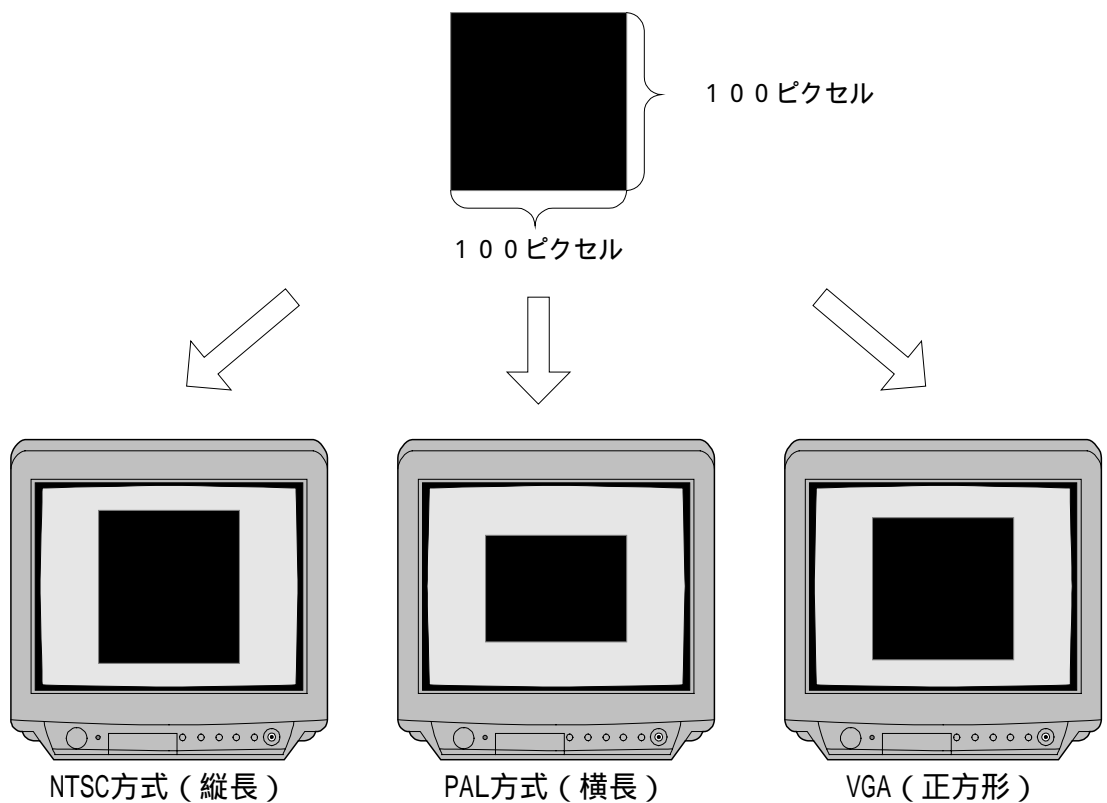


図 10-4 映像信号方式による表示の違い



## 10.2.4 輝度と色

Dreamcast の出力する映像は、PC モニタとは色の見え方が異なります。素材を作成する場合は、必ず Dreamcast のテレビ出力映像で色や輝度を確認して下さい。

特に、次の 2 点に注意して下さい。

### (1) Dreamcast の輝度信号の出力特性

Dreamcast は輝度信号を 10%程度増幅して出力します。再生時にブライトネス値を用いて輝度を調整することも可能です。その場合は色合いが変化しますので、あらかじめ動画編集ソフトで輝度を 10%程度下げるか、色を強調して下さい。

### (2) 映像信号方式の国別特性

テレビの映像信号は国によって差異があります。例えば日本と米国では同じ NTSC 規格でも、信号のオフセット値が異なり、米国のほうが若干暗く見えます。実際にターゲットとなる国のテレビで表示し、確認して下さい。

## 10.2.5 画像サイズ

デコード時の CPU 負荷を減らし、画質を向上させるため、素材の画像サイズを調整します。調整には、「クロップ」と「縮小」があります。

### (1) クロップ

VTR 等から素材をキャプチャした場合、実際に見えない領域を含みます。この部分をクロップし無駄を省きます。画像の不要部分をクロップします。例えば、Dreamcast の NTSC 方式での出力で、ユーザが実際に見ることのできる画像サイズは  $640 \times 448$  程度です。なお、ユーザが見ることのできる画像サイズは映像信号方式によっても異なりますのでご注意下さい。

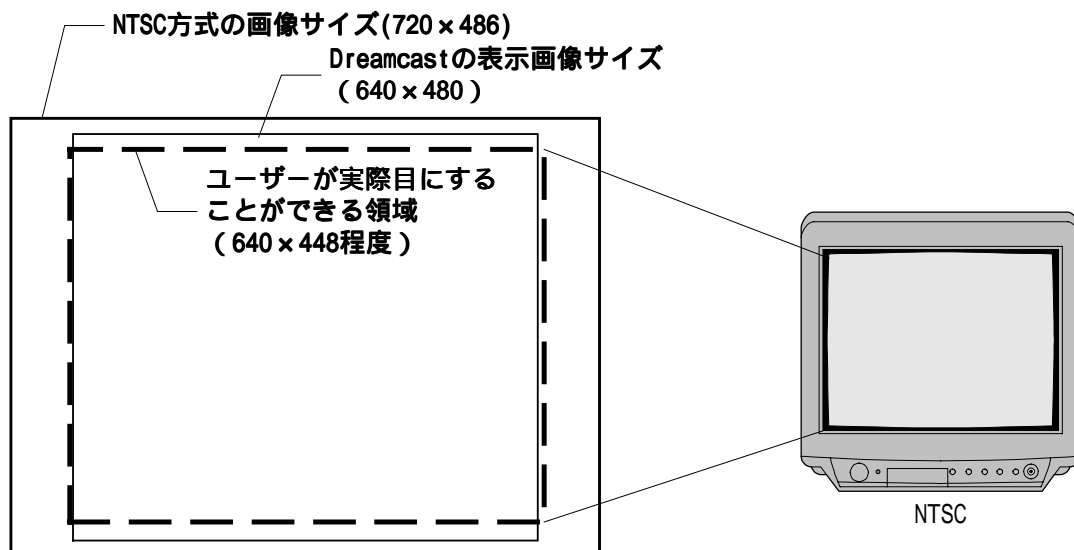


図 10-5 NTSC 方式テレビの表示サイズ

### (2) 縮小

デコード時の負荷を下げるため、画像を縮小します。縮小サイズは、再生時の CPU の使用状況、素材、用途等に応じて、適切なサイズを選択して下さい。



### 10.2.6 インタレース

NTSC 方式、PAL 方式は、1 フレームの映像を 1 ラインおきに分割し、交互に表示するインタレース方式です。交互に表示する映像をそれぞれ「第 1 フィールド」「第 2 フィールド」と呼びます。インタレース方式の映像を扱う場合、第 1・第 2 フィールドが逆転しないように注意が必要です。

以下に注意点を示します。

- キャプチャ  
キャプチャの際、第二フィールド優先で、第 1 フィールドが最上位ラインになるよう取り込みを行って下さい。取り込み開始フィールドと取り込み範囲の座標指定に注意して下さい。
- クロップ  
第 1 フィールドが最上位ラインになるようクロップして下さい。第 2 フィールドからクロップすると、再生時にフィールドが逆転し、正しく再生できない場合があります。
- 縮小  
インタレース方式の第 1 フィールドと第 2 フィールドの映像は時間的にずれています。インタレース方式の映像をそのまま縮小すると、二つのフィールドの映像が混ざり、像が二重になります。縮小を行う場合は、第 1 フィールド、または第 2 フィールドのみを抽出してから縮小して下さい。
- 再生  
再生位置によっては第 1 フィールドと第 2 フィールドが逆になる場合があります。その場合、縦方向に 1 ピクセルずらして下さい。



## 10.2.7 NTSC / PAL 両用素材

NTSC 方式と PAL 方式の両方で再生する Sofdec データを作成する場合、フレームレートの大きい NTSC 方式用に Sofdec データを作成して下さい。PAL 方式で再生する際の注意点を次に示します。

## (1) 同期処理

CRI MPEG Sofdec は、素材のフレームレートと出力するフレームレートが異なる場合、映像と音声はずれないよう、同期処理を行います。表示モードがノンインタレースに設定されている場合、同期は Vsync 単位 (フィールド単位) で行われます。(フレーム単位で補正を行う動画編集ソフトにくらべ、滑らかに再生されます。) PAL 方式で再生する際は、画面表示モードを PAL のノンインタレースモードに設定して下さい。

## a. CRI MPEG Sofdec による再生

NTSC 方式用の素材を PAL 方式で表示した場合、CRI MPEG Sofdec デコーダは Vsync 単位で補正をします。

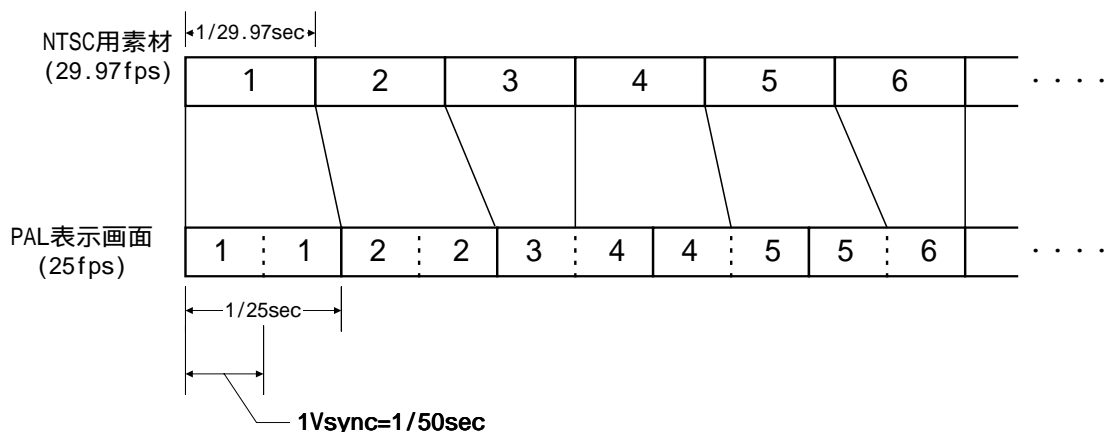


図 10-6 NTSC 方式用の素材を PAL 方式で再生した場合

## b. 動画編集ソフトで変換した場合 (参考)

NTSC 方式の素材を、動画編集ソフトで PAL 方式に変換すると、補正はフレーム単位で行われます。下図の場合では、補正のために第 6 フレームがなくなってしまっています。動画編集ソフトで素材のフレームレートを変換しないで下さい。

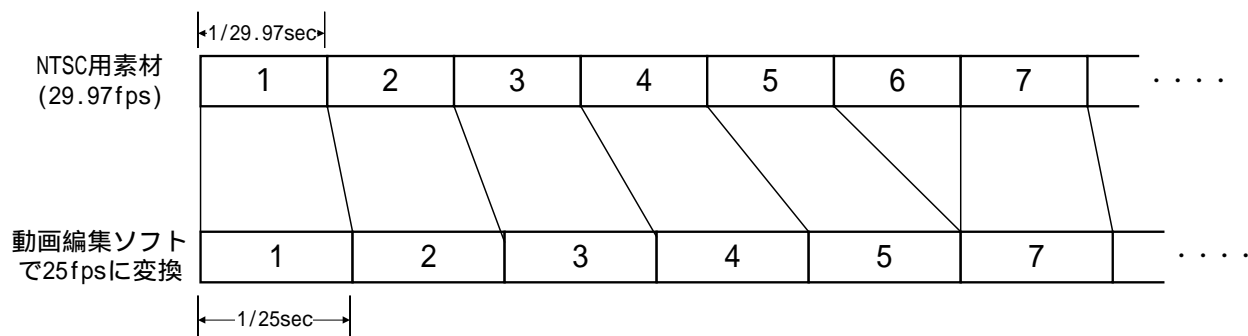


図 10-7 動画編集ソフトで変換した場合



## (2) ピクセル縦横比

NTSC方式とPAL方式ではピクセル縦横比が異なります。調整を行わないと、縦方向がつぶれて表示されます。縦方向に拡大（または横方向に縮小）して表示して下さい。拡大比率はピクセル縦横比より算出されます。画像の表示サイズが  $640 \times 480$  の場合の調整例を、次に示します。

## ● 縦方向を拡大する場合

$1.0950 \div 0.9157 = 1.1958$  倍に縦方向を拡大します。

$640 \times 480$  の画像は  $480 \times 1.1958 = 574$  ですので、再生領域の値を  $640 \times 574$  に設定します。

**注意** ただし、Dreamcast は  $640 \times 480$  までしか表示できないので、上下 47 ピクセルは表示されませんので、ご注意下さい。

## ● 横方向に縮小する場合

$(1 \div 1.0950) \div (1 \div 0.9157) = 0.8363$  倍に、横方向に縮小します。

$640 \times 480$  の画像は  $640 \times 0.8363 = 536$  となり、 $536 \times 480$  に設定、表示します。

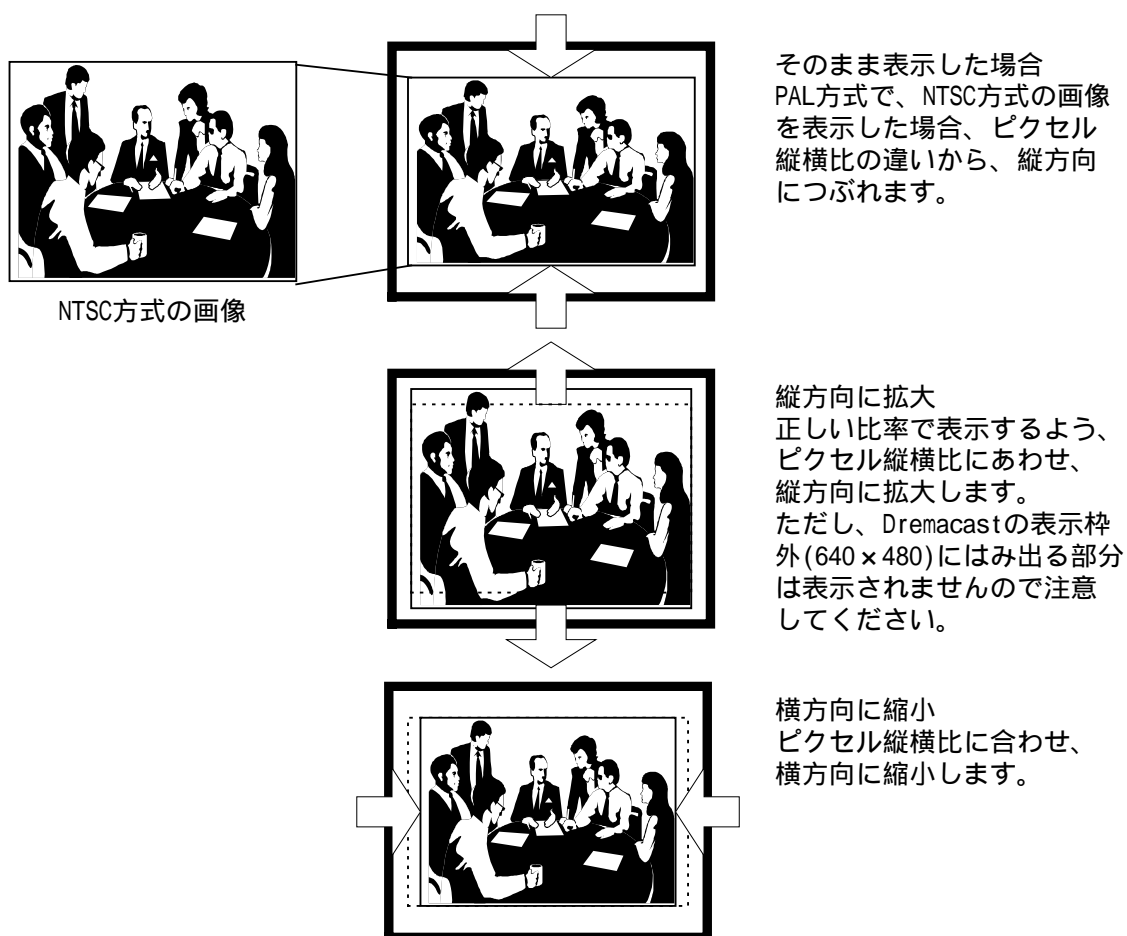


図 10-8 PAL方式に表示する場合の注意



## 10.3 Sofdec データの作成方法

### 10.3.1 エンコードパラメータ推奨値

エンコードパラメータの推奨値を次に示します。値は参考値です。必ず再生テストを行い、再生状態や映像の画質に応じてパラメータを調整して下さい。

表 10-6 一般的な素材の推奨パラメータ

目 的	画像サイズ	ビットレート	GOP シーケンス (N/M)
高精細映像	480 × 336	2.4Mbps ~ 3.6Mbps	N=12,M=2
インタレース映像	320 × 448	2.4Mbps ~ 3.6Mbps	N=12,M=2
フィルム素材 (24fps)	640 × 448	2.4Mbps ~ 3.2Mbps	N=6,M=1
動きが激しい映像	320 × 224	3.2Mbps ~ 4.8Mbps	N=15,M=3

映像によって、エンコード後の映像にノイズが発生したり、再生時にコマ落ちする場合があります。

その場合、下記の手順で対応して下さい。

- ノイズが発生した場合
  - ビットレートを上げる。
  - M 値を増やし、GOP シーケンスの B ピクチャの比率を上げる。
- コマ落ちする場合
  - M 値減らし、GOP シーケンスの B ピクチャの比率を下げる。
  - ビットレートを下げる。
  - 画像サイズを縮小する。



## 10.3.2 Sofdec データ作成例

推奨パラメータを使用して、CRI MPEG CRAFT で Sofdec データを作成する例を、以下に示します。

## (1) CG で作成した素材の場合

NTSC 方式のピクセル縦横比、フレームレートに合わせて作成された素材をエンコードします。画像サイズを高精細映像の推奨値である 480×336 に縮小し、エンコードします。

表 10-7 エンコードパラメータ

項 目	素 材	エンコード
画像サイズ	640×480	480×336
ピクセル縦横比	1.0950	1.0950
フレームレート	29.97fps	29.97fps
ビットレート	-	3.2Mbps
GOP シーケンス	-	N=12,M=2

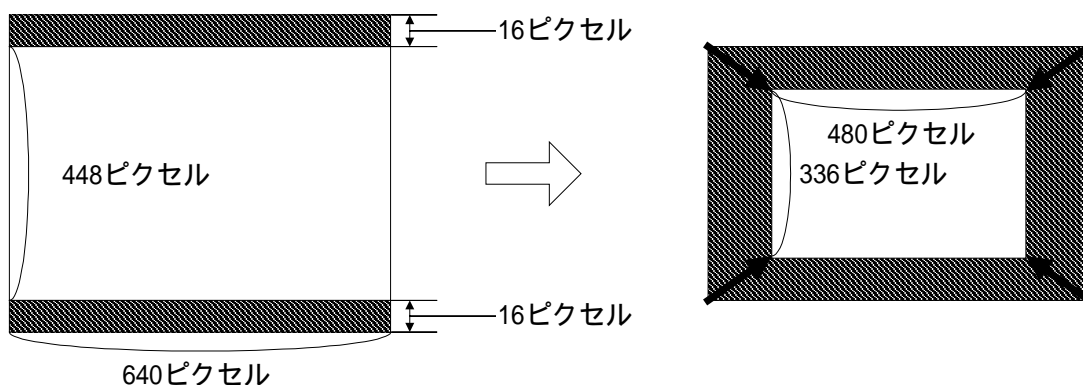


図 10-9 クロップ及び縮小

クロップ及び縮小は、動画編集ソフトまたはエンコーダの機能により行います。

素材ファイルから Sofdec データを作成するには、ビデオエンコード、オーディオエンコード、マルチプレクスの3つの処理を行う必要があります。以下にその手順と実際のパラメータ例を示します。

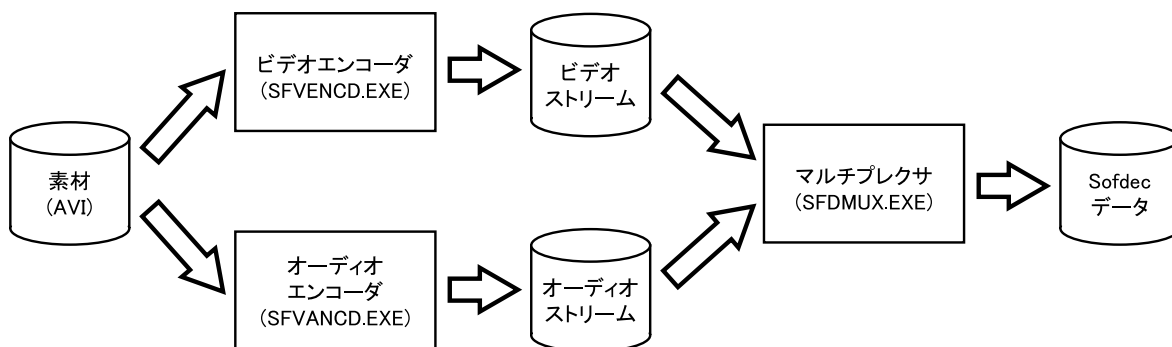


図 10-10 Sofdec データ作成の流れ



● ビデオエンコード

```
C:\¥>SFVENCOD -in=sample.avi -mbps=3.2 -gop_n=12 -gop_m=2 -crop=640,448,0,16 -scale=480,336
```

-in=sample.avi : 入力ファイル名の指定 (sample.avi)  
 -mbps=3.2 : ビットレートの指定 (3.2Mbps)  
 -gop\_n=12 -gop\_m=2 : GOP シーケンス指定 (N=12、M=2)  
 -crop=640,448,0,16 : クロップ指定 (640×448)  
 -scale=480,336 : 縮小 (480×336)

● オーディオエンコード

```
C:\¥>SFAENCOD sample.avi
```

sample.avi : 入力ファイル名の指定 (sample.avi)

● マルチプレクス

```
C:\¥>SFDMUX -V=sample.mlv -A=sample.sfa -S=sample.sfd
```

-V=sample.mlv : ビデオストリームのファイル名を指定 (sample.mlv)  
 -A=sample.sfa : オーディオストリームのファイル名を指定 (sample.sfa)  
 -S=sample.sfd : Sofdec データのファイル名を指定 (sample.sfd)

(2) VTR からキャプチャした自然画素材の場合

VTR からキャプチャする際、ピクセル縦横比、フレームレートの設定に注意して下さい。例として、720×486 でキャプチャした映像の Sofdec データ作成例を次に示します。画像サイズは、インタレース画像の推奨である 320×448 を使用します。

表 10-8 エンコードパラメータ

項 目	素 材	エンコード
画像サイズ	720×486	320×448
ピクセル縦横比	1.0950	1.0950
フレームレート	29.97fps	29.97fps
ビットレート	-	3.2Mbps
GOP シーケンス	-	N=12,M=2

なお、VTR にインタレース方式の映像が録画されている場合があります。クロップする場合、偶数ラインと奇数ラインがずれないようにご注意下さい。(例の場合、 $(486-448) \div 2=19$  ですが、偶数ラインからクロップするため、上 20 ピクセル、下 18 ピクセルを切り取ります。)

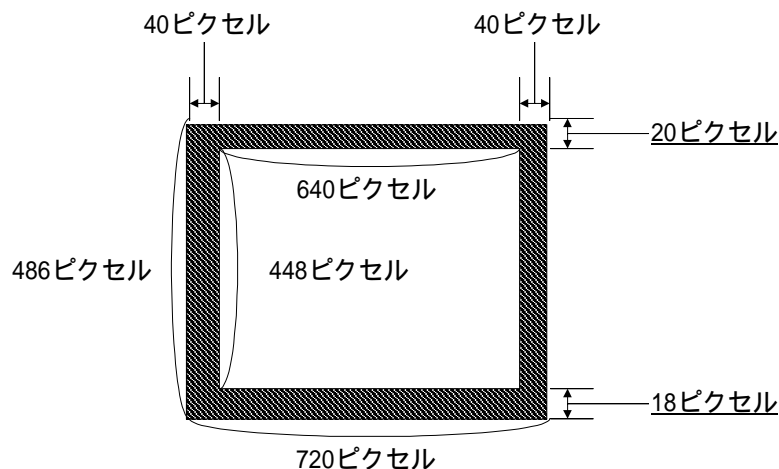


図 10-11 インタレース素材クリップ時の注意点



## ● ビデオエンコード

```
C:\>SFVENC -in=sample.avi -mbps=3.2 -gop_n=12 -gop_m=2 -crop=640,448,40,20 -scale=320,448
-crop=640,448,0,20 : クロップ指定 (640×448)
インタレースがずれないように、偶数行からクロップ。
```

## ● オーディオエンコード

```
C:\>SFAENC sample.avi
```

## ● マルチプレックス

```
C:\>SFDMUX -V=sample.mlv -A=sample.sfa -S=sample.sfd
```

## (3) アニメ素材の場合

アニメ素材等の 24fps の画像が VTR に録画されている場合、逆テレシネ変換をかけ 24fps にして下さい。

24fps の素材の場合、640×448 という画像サイズが選択可能です。ただし映像の動き等によってはノイズが見られたり、コマ落ちしたりする場合があります。その場合は画像サイズを縮小して下さい。

表 10-9 エンコードパラメータ

項 目	素 材	エンコード
画像サイズ	720×486	640×448
ピクセル縦横比	1.0950	1.0950
フレームレート	24fps	24fps
ビットレート	-	3.2Mbps
GOP シーケンス	-	N=6,M=1

## ● ビデオエンコード

```
C:\>SFVENC -in=sample.avi -mbps=3.2 -gop_n=6 -gop_m=1 -crop=640,448,40,19 -pr=24000
-pr=24000 : フレームレート指定 (24fps)
```

## ● オーディオエンコード

```
C:\>SFAENC sample.avi
```

## ● マルチプレックス

```
C:\>SFDMUX -V=sample.mlv -A=sample.sfa -S=sample.sfd
```



### 10.3.3 ビデオエンコード

ビデオエンコーダによって、素材の映像をエンコードします。エンコード時に設定する、重要なパラメータを以下に示します。

#### (1) ビットレート

1 秒当たりの映像にどれくらいのデータ量を割り当てるかを指定します。値が大きいほど画質が向上します。値が小さいほど圧縮され、CPU 負荷率、データ量は少なくなります。

CRI MPEG CRAFT の場合、デフォルト値は 3.6Mbps です。2.4Mbps ~ 4.8Mbps 程度の範囲で映像にあわせて調整して下さい。

#### (2) GOP シーケンス

GOP とは Group Of Picture の略です。キーフレーム (I ピクチャ) と 2 種類の差分フレーム (P ピクチャ、B ピクチャ) がどのような周期で並ぶかを示します。周期の設定は N、M という 2 つの値で行います。

N は I ピクチャが現れる周期です。M は P ピクチャが現れる周期です。

N は M の倍数でなければなりません。

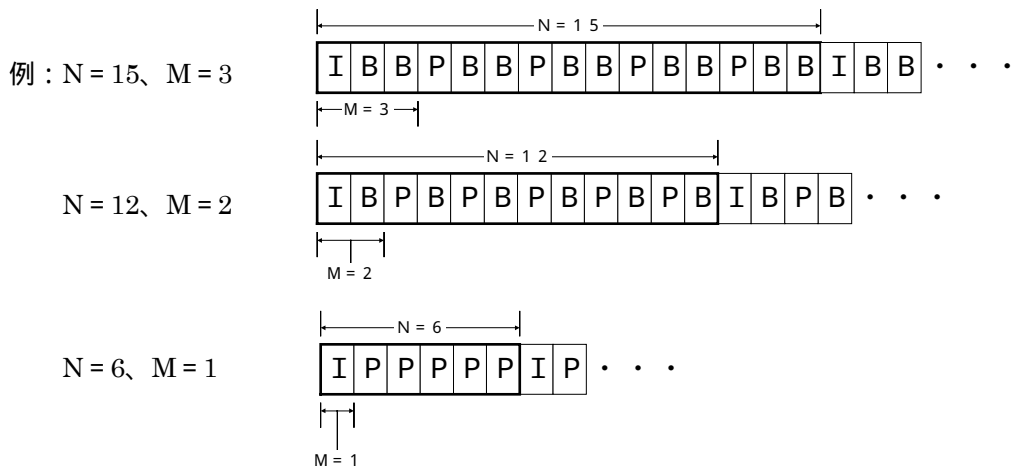


図 10-12 GOP シーケンスに含まれるピクチャ種別



表 10-10 GOP シーケンスに含まれるピクチャ種別

ピクチャ種別	内 容	データ量	再生 CPU 負荷
I ピクチャ	キーフレーム	大	小
P ピクチャ	I ピクチャからの差分	中	中
B ピクチャ	I,P ピクチャからの差分	小	大

データ量	エンコード後のデータ量です。I ピクチャは他のピクチャより、多くのデータ量を必要とします。
再生 CPU 負荷	各ピクチャをデコード、表示する際の CPU 負荷です。素材の映像により大きく変化しますが、平均的には I ピクチャが最も少なく、P,B ピクチャの順で負荷が大きくなります。

**注 意** I ピクチャのみの Sofdec データについては、I ピクチャは、B,P ピクチャと量子化の方法（エンコード時の量子化マトリクス）が異なるので、高ビットレートでエンコードしても I ピクチャの画質は一定以上向上しません。（VideoCD 程度の低ビットレートの場合では、I ピクチャのほうが高画質になる場合があります。）  
I ピクチャのみの Sofdec データは、低 CPU 負荷の動画を作成する場合のみに使用して下さい。

### (3) 色調補正

MPEG1 Video は、YUV 形式で画像情報を圧縮、保持します。一般の MPEG1 機器は、CCIR601 という映像規格に基づき、RGB と YUV の変換を行います。しかし、Dreamcast は再生時に CCIR601 と若干異なる変換を行います。Sofdec データ作成ツールは Dreamcast の出力にあわせてエンコードの段階で補正することが可能です。Dreamcast 用に色調補正するようパラメータを設定して下さい。

## 10.3.4 オーディオエンコード

オーディオエンコードによって、素材の音声をエンコードします。コーデックには Sofdec Audio を採用しています。エンコードのパラメータは特にありません。圧縮率は固定（約 1 / 4 に圧縮）ですので、エンコード後のデータサイズは、サンプリング周波数と、音声チャンネル数に比例します。エンコードの対応するサンプリング周波数と音声チャンネルを次に示します。

表 10-11 Sofdec Audio 対応値

内 容	対応値
サンプリング周波数 (Hz)	44100、22050、14700、11025、8820、7350、6300
音声チャンネル	モノラルまたはステレオ

## 10.3.5 マルチプレクス

マルチプレクサによって、ビデオストリームとオーディオストリームをインターリーブします。マルチプレクスは必ず Sofdec 専用のマルチプレクサで行わなければなりません。Sofdec 専用のマルチプレクサには次の 2 つがあります。

- CRI MPEG Sofdec 用マルチプレクサ
  - CRI MPEG CRAFT に同梱される SFDMUX.EXE
  - SEGA Dreamcast Movie Creator で使用されているマルチプレクサ



### 10.3.6 長尺素材

AVI ファイルのファイルサイズ上限は、一部のコーデックを除き 2GB です。素材がこれ以上の長尺（時間の長い素材）の場合、分割エンコードを行い、結合する必要があります。2GB で作成できる素材の時間を次に示します。下表より長尺の素材は分割エンコード、結合処理が必要です。

表 10-12 2GB の非圧縮素材の時間長

画像サイズ	1 秒あたりのバイト数	2GB 分の時間長
320 × 224 ( 29.97fps )	約 6.15MB/sec	約 5 分 30 秒
352 × 240 ( 29.97fps )	約 7.24MB/sec	約 4 分 40 秒
320 × 448 ( 29.97fps )	約 12.29MB/sec	約 2 分 45 秒
320 × 448 ( 24fps )	約 9.84MB/sec	約 3 分 20 秒
480 × 336 ( 29.97fps )	約 13.8MB/sec	約 2 分 25 秒
640 × 448 ( 29.97fps )	約 24.58MB/sec	約 1 分 20 秒
640 × 448 ( 24fps )	約 19.69MB/sec	約 1 分 40 秒
640 × 480 ( 29.97fps )	約 26.34MB/sec	約 1 分 15 秒
720 × 480 ( 29.97fps )	約 29.63MB/sec	約 1 分 00 秒
720 × 486 ( 29.97fps )	約 30.00MB/sec	約 1 分 00 秒

分割エンコードを行う場合、結合ツール（CRI より提供）が必要になります。

長尺ムービーの作成手順を次に示します。



### 10.3.7 長尺ムービー作成手順

- (1) 分割された素材の音声トラックのみを抜き出し、ひとつのファイルにする。
- (2) 映像を個々の素材ごとにエンコードする。
- (3) ビデオストリームを結合ツールで結合する。
- (4) 1 つにした音声ファイルをエンコードする。
- (5) 結合したビデオストリームとオーディオストリームをマルチプレクスし、Sofdec データを作る。

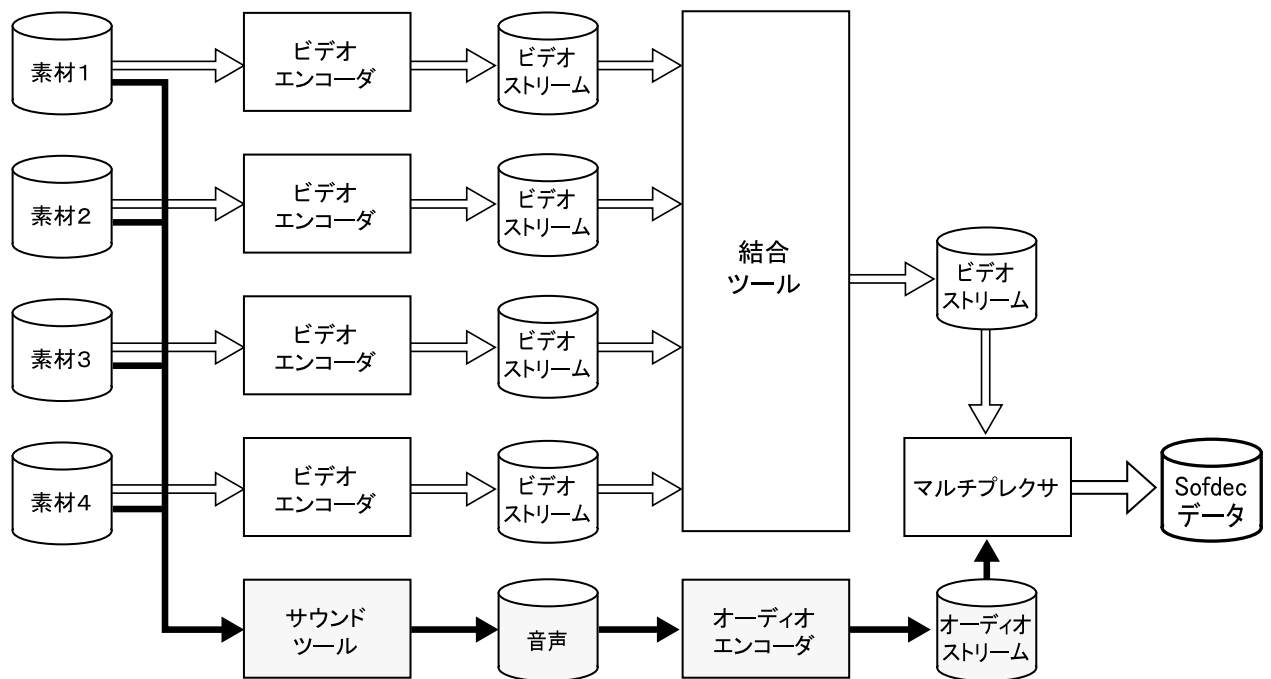


図 10-13 長尺素材作成手順



## 10.4 CRI MPEG CRAFT コマンド一覧

CRI MPEG CRAFT は、3 つのツールから構成される Sofdec データ作成ツールです。

### (1) Sofdec Video Encoder ( SFVENCD.EXE )

SFVENCD は、CRI が提供する Sofdec 専用ビデオエンコーダです。Dreamcast 用に最適化されています。

#### ● 特徴

- エンコード特性を Dreamcast に最適化。
- コマンドラインで制御できるので、より柔軟なバッチ処理が可能。
- アルファ合成用 Sofdec データが作成可能（再生には、CRI SofdecF/X ライブラリが必要）。

#### ● パラメータ

```
SFVENCD -in=infile [-out=outfile][-encode={ON|OFF}]
          [-mode={DEBUG|RELEASE}][-mbps=mbrate][-pr={23976|24000|25000|29970|30000}]
          [-start=startno][-end=endno][-gop_n=gopn][-gop_m=gopm]
          [-crop=crw,crh,crx,cry][-scope=spw,sph,spx,spy[,spr,spg,spb]]
          [-scale=scw,sch][-alpha={2|3|5|256|OFF}][-yuv={DC|CCIR}][sub=scfile]
SFVENCD -registry=keycode
```

表 10-13 パラメーター一覧

パラメータ	機 能
-in=infile	入力ファイル名の指定。
-out=outfile	出力ファイル名の指定。
-encode=value	エンコード実行フラグ。OFF=エンコードしない。
-mode=value	エンコードモード。DEBUG=高速エンコード（画質を無視しエンコード）
-mbps=mbrate	ビットレートの指定。Mbps 単位。
-pr=value	ピクチャレートの指定。映像のフレームレートを 1000 倍して記述。
-start=startno	エンコード開始フレームの指定。-1=ファイルの先頭から。
-end=endno	エンコード終了フレームの指定。-1=ファイルの終端まで。
-gop_n=gopn	GOP シーケンスの N 値を指定。N は M の倍数でなければならない。
-gop_m=gopm	GOP シーケンスの M 値を指定。
-crop=crw,crh,crx,cry	(crx,cry)の位置から、crw × crh ピクセル分の映像をクロップ。
-scope=spw,sph,spx,spy, spr,spg,spb	背景色(spr,spg,spb)、サイズ spw × sph の背景の(spx,spy)の位置に画像を置く。
-scale=scw,sch	scw × sch に画像を縮小。
-alpha=value	アルファマスク付きエンコード。3,5,256 から選択（詳細は別冊参照）
-yuv=value	yuv モードを指定。
-sub=scfile	サブファイル名を指定。
-registry=keycode	キーコードを登録。
-help	使用方法を表示。



## (2) CRI Sofdec Audio Encoder (SFAENCD.EXE)

SFAENCD は、CRI が提供する Sofdec 専用オーディオエンコーダです。AVI ファイル内の音声トラック、WAV 及び AIFF フォーマットの音声ファイルをエンコードする事が出来ます。

## ● 特徴

- Sofdec 専用のオーディオエンコーダ。
- 音声コーデックに CD レベルのクオリティーを持つ Sofdec Audio を使用。
- 周波数変換、AVI ファイル内の映像に合わせた任意区間のエンコードが可能。

## ● パラメータ

```
SFAENCD <Infile> [Outfile] [-sf=value][-ch=value][-vsno=value][-veno=value]
```

表 10-14 パラメーター一覧

パラメータ	機 能
Infile	入力ファイル名を指定。
Outfile	出力ファイル名、またはディレクトリを指定。
-sf=value	圧縮音声サンプリング周波数を指定 (Hz 単位)
-ch=value	音声データチャンネルを指定。 0=左 1=右 2=左右入れ換え 3=(左+右)/2(MONO)
vsno=value	エンコード開始ビデオフレーム番号
veno=value	エンコード終了ビデオフレーム番号

## (3) CRI Sofdec Multiplexer(SFDMUX.EXE)

SFDMUX は、CRI が提供する Sofdec 専用マルチプレクサです。ビデオストリームとオーディオストリームを読み込み、Sofdec データとして出力します。CRI MPEG Sofdec デコーダで再生するためには、本ツールでマルチプレクスしなければなりません。

## ● 特徴

- Sofdec データ専用マルチプレクサ。
- CRI MPEG Sofdec デコーダで再生するためには、本ツールでマルチプレクスしなければならない。(SEGA Dreamcast Movie Creator は、本マルチプレクサと同等の機能を内蔵)

## ● パラメータ

```
SFDMUX [-LANG={J|E}][SUB=filename] -V=video -A=audio -S=output
```

表 10-15 パラメーター一覧

パラメータ	機 能
-V=video	入力映像ファイル名を指定。
-A=audio	入力音声ファイル名を指定。
-S=output	出力 Sofdec データのファイル名を指定。
-LANG=value	出力言語の変更(J=日本語、E=英語)
-SUB=filename	パラメータサブファイルの指定。



## 11 ADPCM エンコーダ

ここでは、ADPCM エンコーダ（音声データ作成ツール）について説明します。

### 11.1 ツール概要

Dreamcast ADPCM エンコーダ（以下 ADPCM エンコーダ）とは、16BitWAVE ファイルフォーマットの音声素材を 4BitWAVE ファイルフォーマット（ADPCM）形式に変換するツールである。

### 11.2 動作環境

ADPCM エンコーダは、以下の環境で動作します。

表 11-1 動作環境

ハードウェア	MMX 機能付き Pentium（または Pentium ）搭載 DOS/V 機
OS	MS-Windows95 以降、MS-WindowsNT4.0 以降
メモリ	32MB 以上

### 11.3 用語説明

本項で使用する用語の説明をします。

表 11-2 本書で使用する用語

用 語	意 味
音声データ	ADPCM エンコーダで圧縮した音声データファイル。
素材ファイル	音声データの圧縮元のファイル。WAVE ファイル、AIFF ファイル

### 11.4 データ作成の流れ

音声データ作成の流れを、以下に示します。

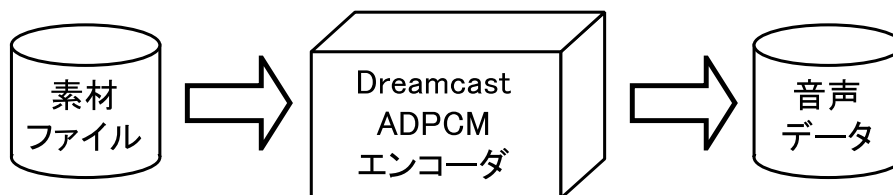


図 11-1 データ作成の流れ



## 11.5 素材の準備

### 11.5.1 素材のフォーマットと準備方法

(1) ファイルフォーマットについて

表 11-3 素材ファイル作成時の条件

項 目	内 容
ファイルフォーマット	WAVE フォーマット、AIFF フォーマット
サンプリング周波数	44100Hz、22050Hz、11025Hz
量子化ビット数	16Bit
音声チャンネル	モノラルまたはステレオ

## 11.6 音声データの作成方法

音声データの作成方法を示します。

### 11.6.1 音声データ作成方法

音声データは、素材となる WAVE ファイルまたは、AIFF ファイルを 4 BitADPCM (WAVE ファイルフォーマット) に圧縮したものです。

操作方法は、以下のように素材ファイル名を引数として、コマンドプロンプトより実行して下さい。素材ファイルと同一のディレクトリに音声データが生成されます。

また、圧縮する際は HDD 上に素材データの 25% 以上の空き領域が必要です。

(1) 圧縮方法 1

素材ファイル名のみを指定し圧縮する。

```
C:\TEMP>Dadenc infile.wav (.wav は省略可能)
```

この場合は、素材ファイルと同一ディレクトリに 'infile\_B4.wav' というファイル名で生成されます。

(2) 圧縮方法 2

素材ファイル名、音声データ名を指定して圧縮する。

```
C:\TEMP>Dadenc infile.wav outfile.wav (.wav は省略可能)
```

この場合は、素材ファイルと同一ディレクトリに 'outfile.wav' というファイル名で生成されます。



## 12 ミドルウェアの移行手順

ここでは、SEGA Library Ver.1 から Ver.2 へ、ミドルウェアの移行手順について説明します。  
SEGA Library Ver.2 でミドルウェアを再生するには、リンクするライブラリの変更とプログラムの一部変更が必要です。

変更の際の注意点を、以下に説明します。

**参 照** ここでは、ミドルウェアについてのみ記述します。Ninja2 及び Kamui2 の仕様については、それぞれのライブラリのドキュメントを参照して下さい。

### 12.1 ライブラリの変更

SEGA Library Ver.2 でミドルウェアを再生する場合、リンクするライブラリを以下のとおり変更して下さい。

- ミドルウェア基本ライブラリ：sg\_mw.lib      sg\_mwk2.lib
- SofdecF/X ライブラリ：sofdec.lib      sofdec2.lib

### 12.2 プログラムの変更

SEGA Library Ver.2 でミドルウェアを再生する場合、アプリケーションプログラムを一部変更する必要があります。変更箇所は以下の関数の部分です。

- mwPlyPreInitSofdec
- mwPlyInitSofdec
- mwPlyFinishSofdec
- mwExecMainServer

変更内容の詳細は以下のとおりです。

#### (1) mwPlyPreInitSofdec

mwPlyPreInitSofdec 関数は、割込みスタックの設定を行います。SEGA Library Ver.2 では、sbInitSystem 関数内で割込みスタックの設定が行われるため、mwPlyPreInitSofdec 関数を実行する必要はありません。

具体的には以下のようになります。

#### ● SEGA Library Ver.1

```
void main(void)
{
    mwPlyPreInitSofdec();           // 割込みスタックの設定
    sbInitSystem(...);
    :
}

<SEGA Library Ver.2>
void main(void)
{
    sbInitSystem();
    :
}
```



## (2) mwPlyInitSofdec

mwPlyInitSofdec 関数は、ミドルウェアの初期化と Ninja のレンダリング終了フラグの取得、Ninja の頂点バッファの取得を行います。SEGA Library Ver.2 用ミドルウェアでは、mwPlyInitSfdFx 関数と mwPlySetVertexBuffer 関数の 2 つの関数に mwPlyInitSofdec 関数の処理が別れます。

mwPlyInitSfdFx 関数の引数には、mwPlyInitSofdec 関数と同じものを入力して下さい。

mwPlySetVertexBuffer 関数には、Ninja2 を使用している場合は njGetVertexBuffDesc 関数で取得した頂点バッファへのポインタを設定して下さい。

Kamui2 を使用している場合は、KMVERTEXBUFFDESC で宣言している頂点バッファへのポインタを設定して下さい。

具体的には以下ようになります。

## ● SEGA Library Ver.1

```
sbInitSystem(...);
:
mwPlyInitSofdec(&iprm);           // ミドルウェアの初期化
:
```

## ● SEGA Library Ver.2

## a. Ninja2 の場合

```
sbInitSystem();
njInitDevice();
njInitSystem(...);
vtxbuf = njGetVertexBuffDesc();
:
mwPlyInitSfdFx(&iprm);           // ミドルウェアの初期化
mwPlySetVertexBuffer(vtxbuf);   // 頂点バッファの設定
:
```

## b. Kamui2 の場合

```
KMVERTEXBUFFDESC      VertexBufferDesc;
KMSYSTEMCONFIGSTRUCT  SystemConfig;

sbInitSystem();
kmInitDevice(KM_DREAMCAST);
kmSetDisplayMode(...);
kmSetWaitVsyncCount(1);
:
SystemConfig.pBufferDesc = &VertexBufferDesc;
:
kmSetSystemConfiguration(&SystemConfig);
:
mwPlyInitSfdFx(&iprm);           // ミドルウェアの初期化
mwPlySetVertexBuffer(&VertexBufferDesc); // 頂点バッファの設定
:
```



(3) MwPlyFinishSofdec

MwPlyInitSofdec 関数の変更に伴い、mwPlyFinishSofdec 関数を mwPlyFinishSfdFx 関数に変更する必要があります。

(4) mwExecMainServer

映像コーデックの場合、mwExecMainServer 関数は映像データのテクスチャメモリへの転送と描画処理を行います。SEGA Library Ver.2 用ミドルウェアでは、mwExecMainServer 関数の処理が mwPlyExecTexSvr 関数と mwPlyExecDrawSvr 関数の 2 つの関数に別れます。mwPlyExecTexSvr 関数は、映像データをテクスチャメモリへ転送します。mwPlyExecDrawSvr 関数は描画処理を行います。Ninja2 (Kamui2) では、njStartDraw(kmBeginPass)関数と njRender(kmEndPass)関数の間で、映像データのテクスチャメモリへの転送ができないので、njStartDraw (kmBeginPass)関数の前で mwPlyExecTexSvr 関数を実行する必要があります。

音声コーデックの場合は、従来どおり mwExecMainServer 関数を実行して下さい。

具体的には以下のようになります。

● **SEGA Library Ver.1**

```
for (;;) {  
    // ペリフェラル情報の取得等  
    mwExecMainServer();           // テクスチャメモリへの転送と描画処理  
    // ポリゴン表示等  
    njWaitVSync();               // Kamui:kmRender();kmFlipFrameBuffer();  
}
```

● **SEGA Library Ver.2**

a. Ninja2 の場合

```
for (;;) {  
    mwPlyExecTexSvr();           // テクスチャメモリへの転送  
    njStartDraw();  
    mwPlyExecDrawSvr();         // 描画処理  
    // ペリフェラル情報の取得等  
    // ポリゴン表示等  
    njRender();  
    njWaitVBlank();  
}
```

b. Kamui2 の場合

```
for (;;) {  
    mwPlyExecTexSvr();           // テクスチャメモリへの転送  
    kmBeginScene(&SystemConfig);  
    kmBeginPass(&VertexBufferDesc);  
    mwPlyExecDrawSvr();         // 描画処理  
    // ペリフェラル情報の取得等  
    // ポリゴン表示等  
    kmEndPass(&VertexBufferDesc);  
    kmRender(KM_RENDER_FLIP);  
    kmEndScene(&SystemConfig);  
    kmWaitVBlank();  
}
```



## 12.3 フレームワーク

Ninja2 (Kamui2) では、njWaitVBlank(kmWaitVBlank)関数は初期化時に設定するシステムカウントに関係なく 1 V しか待ちません。そのため、初期化時に設定したシステムカウントに応じて他の関数(njStartDraw など)で待ち (フリップのタイミングを待つ) が発生します。ムービー再生時は毎 V テクスチャをビデオメモリへ転送するため、待ちが発生する場所が一定していないと、ムービー再生は安定しません。そこで、ムービー再生時はアプリケーションで意図的に待ちを発生させる必要があります。

具体的には以下ようになります。

### a. Ninja2 の場合

```
Sint32 vcnt; // システムカウント

// V-Sync 割り込みに登録する関数
void vsync_func(void)
{
    vcnt--;
}

// MAIN 処理
vcnt = 2;
for (;;) {
    mwPlyExecTexSvr();
    njStartDraw();
    mwPlyExecDrawSvr();
    // アプリケーションプログラム

    msk = get_imask();
    set_imask(15); // 割り込み禁止
#ifdef 非同期モード
    while (vcnt > 0) { // vcnt が 0 になるまで待つ
        njWaitVBlank();
    }
#else 同期モード
    while (vcnt > 1) { // vcnt が 1 になるまで待つ
        njWaitVBlank();
    }
#endif
    set_imask(msk); // 割り込み禁止解除
    njRender();
    msk = get_imask();
    set_imask(15); // 割り込み禁止
    while (vcnt > 0) { // vcnt が 0 になるまで待つ
        njWaitVBlank();
    }
    for (;;) { // リソースロックの解除待ち
        if (njStartCheck() != NJD_START_OK) {
            njWaitVBlank();
        } else {
            break;
        }
    }
    vcnt = 2; // vcnt の再設定
    set_imask(msk); // 割り込み禁止解除
}
```



## b. Kamui2 の場合

```

Sint32  rndr_id[2] = {0,0};
Sint32  vcnt;                                // システムカウンタ

// V-Sync 割り込みに登録する関数
void vsync_func(void)
{
    vcnt--;
}

// MAIN 処理
vcnt = 2;
for (;;) {
    mwPlyExecTexSvr();
    kmBeginScene(&SystemConfig);
    kmBeginPass(&VertexBufferDesc);
    mwPlyExecDrawSvr();
    // アプリケーションプログラム

    msk = get_imask();
    set_imask(15);                            // 割り込み禁止
#ifdef 非同期モード
    while (vcnt > 0) {                        // vcnt が 0 になるまで待つ
        kmWaitVBlank();
    }
#else 同期モード
    while (vcnt > 1) {                        // vcnt が 1 になるまで待つ
        kmWaitVBlank();
    }
#endif
    set_imask(msk);                            // 割り込み禁止解除
    kmEndPass(&VertexBufferDesc);
    rndr_id[1] = rndr_id[0];
    rndr_id[0] = kmRender(KM_RENDER_FLIP);
    kmEndScene(&SystemConfig);
    msk = get_imask();
    set_imask(15);                            // 割り込み禁止
    while (vcnt > 0) {                        // vcnt が 0 になるまで待つ
        kmWaitVBlank();
    }
    for (;;) {                                // リソースロックの解除待ち
        stat = kmGetRenderStatus(usrg_rdr_id[1]);
        if ( (stat == KMSTATUS_UNDER_DMA)
            || (stat == KMSTATUS_FINISH_DMA)
            || (stat == KMSTATUS_UNDER_RENDER) ) {
            kmWaitVBlank();
        } else {
            break;
        }
    }
    set_imask(msk);                            // 割り込み禁止解除
}

```

割り込み禁止区間で V-sync 割り込みを待っていますが、正常に動作します。これは、`njWaitVBlank(kmWaitVBlank)`関数内でファイバが切り替わり、別のファイバにいる間は割り込み禁止が解除されているからです。



## 12.4 インターレースドムービー

SEGA Library Ver.2 からは、3ラインフィルタを解除してインターレースドムービーを再生することができます。インターレースドムービーを再生する場合は、表示のタイミングを偶数フレームにする必要があります。

具体的には、以下のようになります。

```
while ( kmyIsDisplayOddField() == TRUE ) {
    njWaitVBlank();           // Kamui : kmWaitVBlank();
}
mwPlyExecTexSvr();
njStartDraw();              // Kamui : kmBeginScene(...);
```

## 12.5 その他

SEGA Library Ver.1 から Ver.2 への移行手順とは直接関係ありませんが、Sofdec ( Ver.1.30 ) から SofdecF/X ( Ver.2.xx ) へ移行する時の注意点を以下に説明します。

- ハンドル生成パラメータ構造体  
 ハンドル生成パラメータ構造体は、各メンバにパラメータを設定する前に必ず 0 クリアしてから使用して下さい。  
 パラメータ構造体のメンバに compo\_mode が SofdecF/X ( Ver.2.xx ) から追加されています。単純再生の場合に設定する MWD\_PLY\_COMPO\_OPEQ は 0 なので、パラメータ構造体が 0 クリアしてあれば問題ありませんが、トラブル軽減のために明示的に MWD\_PLY\_COMPO\_OPEQ を設定して下さい。  
 パラメータ構造体のメンバ dtype に設定する MWD\_PLY\_DTYPE\_FULL は Sofdec ( Ver.1.xx ) との整合性のために残っていますが、MWD\_PLY\_DTYPE\_WND に変更して下さい。
- 作業領域サイズの計算  
 mwPlyCalcWorkSofdec 関数は引数が多く、エフェクトムービーに対応していないため、mwPlyCalcWorkCprmSfd 関数を使用して下さい。
- mwPlyStartFrame  
 mwPlyStartFrame 関数は、SofdecF/X ( Ver.2.xx ) から必要がなくなったので、使用しないで下さい。
- mwPlyExecServer  
 mwPlyExecServer 関数は、ミドルウェア再生ライブラリのサーバ関数です。ミドルウェア録音ライブラリのサーバ関数との整合性のために、mwExecMainServer 関数に統合しました。  
 mwPlyExecServer 関数は使用しないで下さい。

**注 意** ただし、mwExecMainServer 関数については別項で説明した通り、SEGA Library Ver.2 への移行の時に注意が必要です。



## 12.6 付録

### 12.6.1 ドアオープンチェック

以下に、ドアオープンチェックのサンプルプログラムを示します。

#### ● サンプルプログラム

```
/* GD ファイルシステムのエラー発生時に起動する関数 */
void UsrGdErrFunc(void *obj, Sint32 errcode)
{
    if (errcode == GDD_ERR_TRAYOPEND || errcode == GDD_ERR_UNITATTENT) {
        ADXF_Finish(); /* ADXF の終了処理 */
        sbExitSystem(); /* Shinobi ライブラリの終了処理 */
        syBtExit(); /* シンプルプレイヤヘジャンプ */
    }
}

/* ユーザが V-SYNC 割り込みに登録する関数 */
void UsrVsyncFunc(void)
{
    Sint32 dstat;

    /* ドアオープンチェック */
    dstat = gdFsGetDrvStat();
    if (dstat == GDD_DRVSTAT_OPEN || dstat == GDD_DRVSTAT_BUSY) {
        gdFsReqDrvStat();
    }
}

/* アプリケーションメイン */
void main(void)
{
    :
    :
    /* GD ファイルシステムエラーコールバック関数の登録 */
    gdFsEntryErrFuncAll((void *)UsrGdErrFunc, NULL);
    njSetVSyncFunction(UsrVsyncFunc);
    :
    :
}
```

### 12.6.2 リードエラー

ADX ファイルシステムでは、リードエラーが発生するとハンドルの状態がエラー状態へ遷移します。従って、ハンドル状態を監視して、リードエラー発生時はアプリケーション側で対処して下さい。

```
ADXF_ReadNw32(adxf, nsct, buf);
for (;;) {
    if (ADXF_GetStat(adxf) == ADXF_STAT_ERROR) {
        /* リードエラー処理 */
    }
}
```



## 12.6.3 シーク音の軽減

## (1) ディレクトリチェンジ

ディレクトリチェンジを行うと、シークが発生してデータ読み込みが1秒以上遅くなります。通常、ゲームデータは最外周から配置され、ディレクトリ情報は最内周に存在します。そのため、ディレクトリチェンジを行うと、ディレクトリ情報取得のために最内周までシークします。最外周から最内周までシークするのに約600[msec]程度かかるため、往復で1秒以上データ読み込みが遅くなります。

AFS ファイルを利用して、使用するデータを近くに配置することで、データをディレクトリと同様に管理しながら、シーク音を軽減することができます。

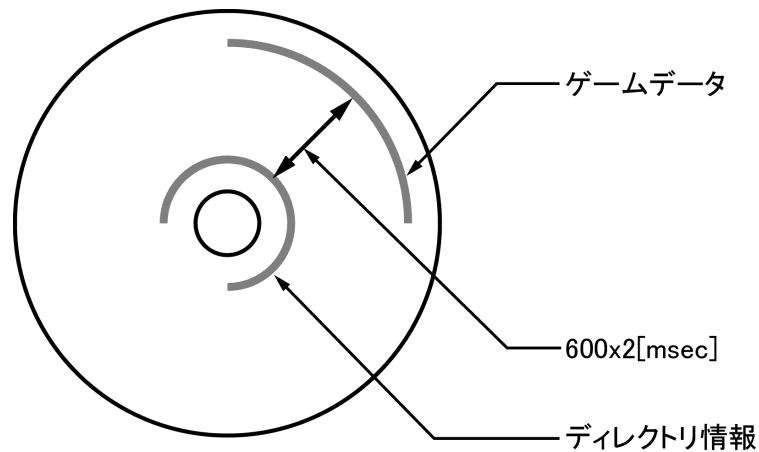


図 12- 1 ディレクトリチェンジ

## (2) AFS ファイルの有効利用

BGM データ、セリフデータ、...とそれぞれに AFS ファイルを作成せず、各シーン毎に AFS ファイルを作成して下さい。マルチストリームで BGM を鳴らしながら、セリフを再生したり、グラフィックデータを読み込むような場合、データの種類ごとに AFS ファイルを作成すると、シーク時間が長くなります。

各シーン毎に AFS ファイルを作成することにより、マルチストリーミング時のシークを軽減することができます。

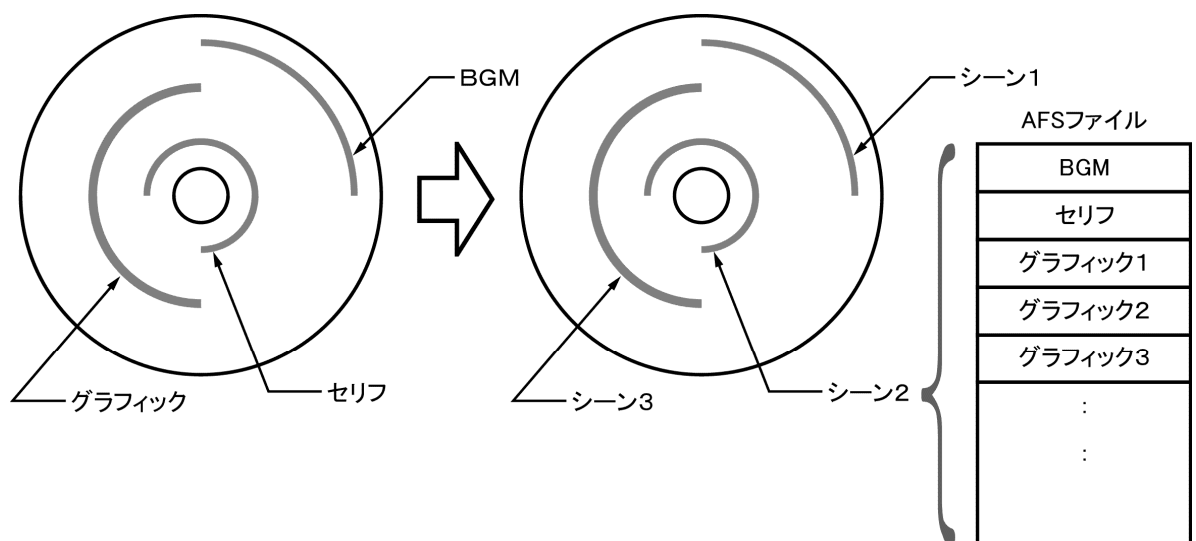


図 12- 2 AFS ファイルの有効利用



## 12.6.4 データ読み込みの高速化

ADX ファイルシステムによって、GD-ROM から音声を再生している最中にゲームデータを読み込むことが可能になります。

しかしながらデフォルトの設定では、ADX 再生の入力バッファの 85%を下回るとデータの再読み込みが始まるため、複数のゲームデータを読み込む場合、ゲームデータと音声データを交互に読み込むことになります。

ADXT\_SetReloadSct 関数を使用することで、音声データの再読み込みの開始量を調整することによって、ゲームデータの読み込みを高速化できます。

入力バッファに最低 1 秒分のデータを読み込んでおくことで、音声を滞りなく再生できるので、再読み込み開始量は音声データの 1 秒分を指定します。

例えば、44.1kHz のステレオ音声は約 50KByte/sec なので、25 セクタ (1 セクタ=2048 バイト) を指定することになります。また、入力バッファを大きくすることにより、再読み込みの間隔を大きくすることができるので、更にゲームデータ読み込みを高速化できます。暫定仕様なので、ADXT\_SetReloadSct 関数はリファレンスマニュアルには記載されていません。将来、このような指定をしなくても高速にゲームデータが読み込めるようになります。

```
/* 44kHz のステレオデータを再生する場合 */
#define WKSIZE      ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

char work[WKSIZE];          /* 作業領域 */
ADXT adxt;                  /* ADXT ハンドル */
ADXF adxf;                  /* ADXF ハンドル */

adxt = ADXT_Create(2, work, WKSIZE); /* ADXT ハンドルの生成 */
ADXT_SetReloadSct(adxt, 25);        /* 再読み込み量の設定 */
:
ADXT_StartFname(adxt, "BGM.ADX");   /* 音声再生開始 */
:
adxf = ADXF_Open("GAMEDAT1.BIN", NULL); /* ファイルオープン */
ADXF_ReadNw32(adxf, nsct, buf);        /* データ読み込み */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND );
ADXF_Close(adxf);                    /* ファイルクローズ */
:
/* (GAMEDAT1 が 2 秒以内に読み込めれば音声データの再読み込みは発生しない) */
:
adxf = ADXF_Open("GAMEDAT2.BIN", NULL); /* ファイルオープン */
ADXF_ReadNw32(adxf, nsct, buf);        /* データ読み込み */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND );
ADXF_Close(adxf);                    /* ファイルクローズ */
```

最大 GD-ROM ストリーム数 (ADXT\_CALC\_WORK の第 3 引数) を増やすことにより、入力バッファを増やすことができます。引数の値を 1 増やすごとに 1 秒分の余裕ができます。

また、BGM とセリフを同時に再生するが、データを読み込むときにはセリフが再生されないことが保証されていれば、セリフのためのバッファ量をデータ読み込みに割り当てることができます。



### 12.6.5 ストリームジョイントによる読み込み

ストリームジョイントによるデータ読み込みを行うことで、アプリケーションは容易にストリーミングすることができます。ADX ファイルシステムは、ストリームジョイントに空き領域があると、自動的にデータを読み込みます。

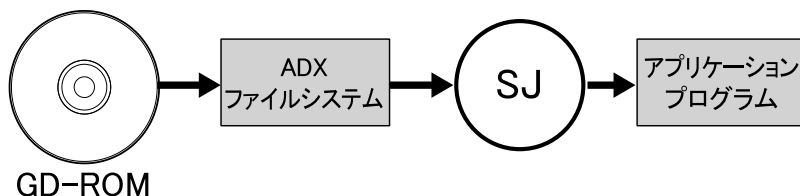


図 12-3 ストリームジョイントによる読み込み

この機能により、例えば Sofdec ファイルを AFS ファイルのインサイドファイルとして読み込むことができます。

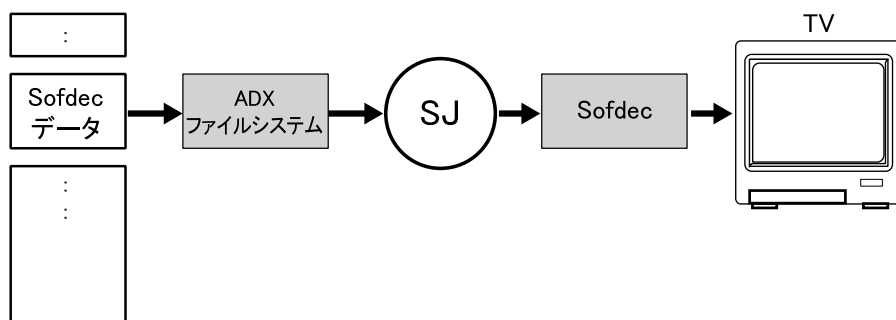


図 12-4 ストリームジョイントによる Sofdec の再生

#### ● サンプルプログラム

```

adx = ADXF_OpenAfs(PAT0, FID1);
sj = mwPlyGetInputSj(sj);
ADXF_ReadSj32(adxf, fsize, sj);
:

```

ストリームジョイントを使用して、可変ブロック長の圧縮データを読み込み展開するサンプルプログラムを以下に示します。

#### ● サンプルプログラム

```

char    buf[64*1024+2048]; // 最大ブロック長が 2048byte

sj = SJRBF_Create(buf, 64*1024, 2048); // 最大ブロック長分エキストラ領域として指定。
                                         // この指定により 2048 バイトの連続を保証。
ADXF_ReadSj32(adxf, 1000, sj);
for (;;) {
    SJ_GetChunk(sj, SJ_LIN_DATA, 2048, &ck);
    nused = user_decode(ck.data, ck.len); // ユーザデコーダ
    SJ_SplitChunk(&ck, nused, &ck, &ck2);
    SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    SJ_UngetChunk(sj, SJ_LIN_DATA, &ck2);
}

```







# 用語集



## Middleware ライブラリ編

日本語	英 語	内 容
AFS ファイル		複数の ISO9660 ファイルを PC 上で結合したファイル。最大 65536 個のファイルを結合することが可能
API	Application Programming Interface	アプリケーションが、OS 等のサービスを利用するために必要なインターフェース
DK3/DK4		TrueMotion 専用音声コーデックのこと。DK3 は 3bit、DK4 は 4bit に圧縮される
DUS ファイル (データ)		DualSpeech エンコーダが出力 / デコーダが入力するファイルフォーマットの総称
ISO11172		ビットレートが 96Kbps から 192Kbps の品質になる
ISO9660 ファイル	International Organization for Standardization	ISO9660 フォーマットのファイル (いわゆる標準のファイル)。ISO=国際標準化機構
MPEG	Moving Pictures Experts Group	ISO の下部組織として設置された標準化団体、およびそこで標準化された勧告の名称。 1、2、4 の 3 つの規格があり、それぞれに MPEG ビデオ (動画圧縮)、MPEG オーディオ (音声圧縮)、MPEG システム (オーディオ、ビデオのストリームフォーマット) が規定されている。
MPEG1/Audio Layer II		MPEG1 規格オーディオモード 2 を指す
SofdecAudio		Sofdec 専用音声コーデックのこと
Sofdec データ		CRI MPEG Sofdec で再生可能な映像と音声を含む動画ファイル。映像コーデックに MPEG1/Video、音声コーデックに Sofdec Audio を採用。
SIP	Sound Input Peripheral	Dreamcast に音を入力するためのペリフェラルのこと
V-sync		垂直同期信号。TV やモニタの映像信号の中で、走査線が画面の下から上に戻るタイミングを示す信号
WAV ファイル (データ)		DualSpeech ツールが扱える PCM タイプの非圧縮音声データフォーマットの総称
YUV420 フォーマット		輝 / 色差信号の色空間分解能力が、4 : 2 : 0 のものを指す
アルファ合成	Alpha Composition	アルファブレンディングにより、2 枚の絵を合成すること
アルファブレンディング	Alpha Blending	$\text{output} = \text{ } \times \text{ } s + (1 - \text{ } ) \times \text{ } d$ によって定義されるピクセル単位で混合すること
インサイドファイル	Inside File	AFS ファイル (パーティション) 内に結合されているファイル。サイズは最大 128Mbyte までで、2048byte (セクタ) 単位となる
インターリーブ	Interleave	交互に配置すること
オーディオストリーム	Audio Stream	素材ファイル内の音声データ
オーディオレンダラ	Audio renderer	音声データを再生するミドルウェア内部のモジュール
オーディオキャプチャ	Audio Capture	音声データを取り込むミドルウェア内部のモジュール
クロップ	crop	画像の不要部分を削除し、一部を切り出すこと
セクタバウンダリ	Sector boundary	フォーマットされたディスクでの最小単位どうしの境目
素材	Material	Sofdec データの元となるファイル。映像及び音声を含む。非圧縮の AVI フォーマットを標準として推奨。Sofdec に限った言葉ではない



日本語	英 語	内 容
素材ファイル (データ)	Material	DualSpeech ツールの DualSpeech エンコーダに入力させる音声データ(原音データ)
ディゾルブ効果	Dissolve Effect	画面が暗くなるのにオーバーラップして次の画面が現れる場面転換のような効果
パーティション	Partition	ハード・ディスク等の大容量補助記憶装置の領域を分割したもの
パーティション ID	Partition ID	パーティションを識別するための識別子。
ビデオレンダラ	Video Renderer	映像データを表示するミドルウェア内部のモジュール
ビデオストリーム	Video Stream	素材ファイル内の映像をエンコードしたデータ
ピクセル	Pixel	画面に表示されるひとつの点。「画素」「ドット」等とも呼ばれる
ピクセル縦横比	Pixel Aspect Ratio	ピクセルの縦横実寸比。通常「縦÷横」の値で示される。NTSC 方式の場合は縦1÷横0.91 1.0950となる。「ペルアスペクトレシオ」「ピクセルアスペクト比」等とも呼ばれる
フレーム	Frame	表示される 1 枚分の映像を示す
フレームレート	Frame Rate	素材の映像周波数、または出力画面の更新周波数を示す。単位は fps (Frame per sec)。NTSC 方式の場合は 29.97fps、PAL 方式の場合は 25fps
ファイル ID	File ID	インサイドファイルを識別するための識別子
マルチプレックス	multiplex	多重化すること