
SEGAライブラリユーザーズガイド

1999 年 3 月 16 日版



Dreamcast™

はじめに

弊社の Dreamcast 用アプリケーション開発にご協力いただき誠にありがとうございます。
本書は、Dreamcast のアプリケーション開発用「SEGA ライブラリ」の使いかたをカテゴリ別にまとめた技術資料です。

本書の内容

第 1 部 Ninja 編

SEGA ライブラリのグラフィックスライブラリ (Ninja) の描画能力、モデリングの手法等をご紹介します。

第 2 部 Shinobi 編

SEGA ライブラリのサウンドシステム関数、デバッグ支援関数 (Nindows) などの使いかたをご紹介します。

ご注意

本書に記載されている仕様、および本書に記載されている事項は、将来予告なしに変更することがあります。

商標

- ・ Dreamcast はセガ・エンタープライゼスの商標です。
- ・ MS、Microsoft、MS-DOS、Windows、Windows NT は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- ・ その他、記載されている会社名、製品名は、各社の商標および登録です。
- ・ 本文中には、TM、(R) マークは明記しておりません。

改版履歴

SEGA ライブラリユーザーズガイド

1999 年 2 月 8 日 初版発行

1999 年 3 月 16 日 第二版発行

Copyright (C) 1999 株式会社セガ・エンタープライゼス

編集・製作 株式会社アスキー AAP 書籍編集部

contents

目次

はじめに	2
------------	---

第 1 部

Ninja 編	15
---------------	----

第 1 章

Ninjaライブラリによる画像設定	17
-------------------------	----

1.1 画面解像度設定について	17
-----------------------	----

1.1.1 はじめに	17
1.1.2 注意点	17
1.1.3 表示周期	17
1.1.4 フリッカーフリー処理	17
1.1.5 表記の説明	19
1.1.6 VRAMの各画面モード時における占有率	19
1.1.7 画面設定モード	20
1.1.8 スクリーン座標系	22

1.2 各画面モードの詳細	23
---------------------	----

1.2.1 NJD_RESOLUTION_VGA	23
1.2.2 NJD_RESOLUTION_VGA_ANTI	23
1.2.3 NJD_RESOLUTION_320x240_NTSC(PAL)NI	24
1.2.4 NJD_RESOLUTION_320x240_NTSC(PAL)I	24
1.2.5 NJD_RESOLUTION_320x480_NTSC(PAL)NI	25
1.2.6 NJD_RESOLUTION_320x480_NTSC(PAL)I	25
1.2.7 NJD_RESOLUTION_640x240_NTSC(PAL)NI	26
1.2.8 NJD_RESOLUTION_640x240_NTSC(PAL)I	26
1.2.9 NJD_RESOLUTION_640x480_NTSC(PAL)NI	27
1.2.10 NJD_RESOLUTION_640x480_NTSC(PAL)I	28
1.2.11 NJD_RESOLUTION_640x480_NTSC(PAL)NI_FF	29

1.2.12	NJD_RESOLUTION_320x240_NTSC(PAL)NI_ANTI	30
1.2.13	NJD_RESOLUTION_320x240_NTSC(PAL)I_ANTI	31
1.2.14	NJD_RESOLUTION_320x480_NTSC(PAL)NI_ANTI	32
1.2.15	NJD_RESOLUTION_320x480_NTSC(PAL)I_ANTI	33
1.2.16	NJD_RESOLUTION_640x240_NTSC(PAL)NI_ANTI	34
1.2.17	NJD_RESOLUTION_640x240_NTSC(PAL)I_ANTI	35
1.2.18	NJD_RESOLUTION_640x480_NTSC(PAL)NI_ANTI	36
1.2.19	NJD_RESOLUTION_640x480_NTSC(PAL)I_ANTI	37
1.2.20	NJD_RESOLUTION_640x480_NTSC(PAL)I_ANTI_FF	38

第2章

Basic Model 仕様書

39

2.1	概要	39
2.2	モデル構造体	40
2.3	モデル構造	46
2.3.1	メッシュセット	46
2.3.2	テクスチャ構造	50
2.3.3	Ninjaアトリビュート	53
2.3.4	テクスチャフォーマット	56

第3章

Chunk Model 仕様書

59

3.1	概要	59
3.2	モデル構造体	61
3.3	Chunk 仕様	64
3.3.1	Chunkの種類	64
3.3.2	Chunkの構造	65
3.3.3	Chunk NULL	66
3.3.4	Chunk End	67
3.3.5	Chunk Bits	67
3.3.6	Chunk Tiny	72

3.3.7	Chunk Material	74
3.3.8	Chunk Vertex	82
3.3.9	Chunk Volume	102
3.3.10	Chunk Strip	106

3.4	アスキー出力の注意事項	118
-----	-------------------	-----

第4章

Motion 仕様書 119

4.1	概要	119
4.2	オブジェクト構造体	119
4.3	カメラ構造体	121
4.4	ライト構造体	121
4.5	モーション構造体	124
4.6	オブジェクトモーション	129
4.7	カメラモーション	132
4.8	ライトモーション	133
4.9	その他	135

第5章

Ninja モーションの注意事項 137

5.1	Ninjaにおけるモーションの考え方	137
5.2	Ninjaにおけるモーション作成時の手順	138

第6章

効率のよいストリップを実現する方法

139

6.1	ストリップとは？	139
6.2	ストリップ頂点のつなぎ方	141
6.3	マテリアルとテクスチャ	142
6.4	ストリップの表現方法の比較	142
6.5	インデックス	142
6.5.1	インデックスタイプ 構造	142
6.5.2	頂点直接表現構造	143
6.6	ストリップによるデータ削減率	144

第7章

View 関数について

145

7.1	初期化の方法	145
7.1.1	njInitView()を使用する。	145
7.1.2	VIEW 構造体のメンバーに直接値をセットする。	145
7.2	ビューの移動、回転	146
7.3	旧 View 関数使用時の注意	147
7.3.1	注意事項 1 の詳細	147
7.3.2	注意事項 1 の詳細	147
7.3.3	旧ビュー関数の使用例(正しい例 / 誤った例)	148

第8章

Light の設定方法

149

8.1	LIGHT の設定方法	149
8.1.1	マクロ	153

8.1.2	使用法	153
8.1.3	LIGHT 構造体の仕様	156

第9章

スクロールガイド

159

9.1	スクロールに関する画像単位	159
9.1.1	画像単位	159
9.2	スクロールの回転、拡大、移動	160
9.2.1	スクロールの回転、拡大、移動	160
9.3	スクロールの設定	164
9.3.1	スクロールの設定例	164
9.4	カラー	167
9.4.1	カラーモード	167
9.5	スクロール関数、構造体、定義説明	168
9.5.1	スクロール関連関数	168
9.5.2	スクロール関連構造体	169
9.5.3	スクロール関連定義	169
9.5.4	セル設定のためのテクスチャ構造体	169

第10章

テクスチャガイド

171

10.1	語句説明	171
10.1.1	語句説明	171
10.2	テクスチャ	173
10.2.1	PVRフォーマット	173
10.2.2	カテゴリコード	173
10.2.3	カラーフォーマット	175
10.3	メモリ	176
10.3.1	テクスチャメモリ	176

10.3.2	キャッシュ	176
10.4	テクスチャのロード	177
10.4.1	テクスチャロードの流れ	177
10.4.2	テクスチャバッファの設定	177
10.4.3	キャッシュバッファの設定	179
10.4.4	テクスチャリストの作成	180
10.4.5	テクスチャ番号	182
10.4.6	グローバルインデックス番号	183
10.4.7	グローバルインデックスの自動割付	183
10.4.8	テクスチャのロードエラー	183
10.4.9	メモリテクスチャ	185
10.4.10	レンダテクスチャ	186
10.5	サンプルプログラム	187
10.5.1	概要	187
10.5.2	サンプル	187
10.6	テクスチャ関数の注意点	191
10.6.1	Dev.BoxのSET2からSET4、SET5へ移行時の注意点	191
10.6.2	SET5での注意点	191

第11章

カメラについて

193

11.1	カメラの初期化について	193
11.1.1	カメラ構造体	193
11.2	カメラの位置について	194
11.3	カメラの視線について	195
11.4	カメラの画角について	196
11.5	カメラの深度について	197
11.6	実際のプログラムでの使用方法について	198

B.1	ファイル拡張子の説明	201
-----	------------------	-----

C.1	概要	203
C.2	バイナリ構造	204
C.3	Chunk 仕様	206
C.3.1	Chunkの種類	206
C.3.2	POF0のアルゴリズム	206

第2部

Shinobi 編

1.1	概要	211
1.1.1	Nindowsの特徴	211
1.1.2	最も簡単なNindowsアプリケーションの作成	211
1.1.3	Nindowsの組み込み	211
1.2	Nindows の操作とNindows ユーティリティ	213

1.2.1	Nindowsの操作	213
1.2.2	Nindows ユーティリティ	213
1.2.3	フォントの変更	217
1.3	ウィンドウ	218
1.3.1	概要	218
1.3.2	ウィンドウの作成	218
1.3.3	子ウィンドウの作成	219
1.3.4	ウィンドウに関連するパラメータ	220
1.3.5	ウィンドウサポート関数のサンプルと説明	220
1.4	スクロールウィンドウ	222
1.4.1	概要	222
1.4.2	スクロールウィンドウの作成	222
1.5	エディットウィンドウ	224
1.5.1	概要	224
1.5.2	エディットウィンドウの作成と使用方法	224
1.6	スクロールバーコントロール	226
1.6.1	概要	226
1.6.2	スクロールバーコントロールの作成	226
1.6.3	低レベルスクロールバー関数を利用したスクロールバーコントロールの作成	227
1.7	ボタンコントロール	229
1.7.1	概要	229
1.7.2	ボタンコントロールの作成	229
1.7.3	ボタンの有効化と無効化	230
1.8	メニュー	232
1.8.1	概要	232
1.8.2	メニューテーブルの作成と登録	232
1.8.3	メニューコールバック関数	234
1.8.4	チェックマーク	235
1.8.5	ポップアップメニューの作成	236
1.9	マウス	239
1.9.1	概要	239
1.9.2	マウス情報の取得	239
1.10	フォント	241

1.10.1	概要	241
1.10.2	フォントの変更にに関する問題	241

第2章

GDFSライブラリ

243

2.1	概要	243
2.1.1	基本機能	243
2.1.2	モジュール構成	243
2.2	機能説明	244
2.2.1	用語の説明	244
2.2.2	初期化	244
2.2.3	ディレクトリ操作	245
2.2.4	ファイル操作	246
2.2.5	ファイルへのアクセス	246
2.2.6	ドライブ情報の取得	249
2.2.7	コールバック登録	251
2.2.8	デジタルオーディオ再生(DA 再生)	251
2.2.9	サーバー関数	251
2.3	定数	252
2.3.1	データ型	254
2.3.2	関数一覧	255

第3章

Shinobiライブラリ補足資料

257

3.1	Shinobi環境のCPUモード	257
3.2	Shinobi環境でのバンクレジスタの使用等について	257
3.2.1	汎用レジスタ	257
3.2.2	浮動小数点レジスタ	258
3.2.3	GBRレジスタ	258
3.2.4	その他	258
3.3	キャッシュの操作について	258

3.3.1	ユーザのキャッシュ操作について	258
3.3.2	ライブラリのキャッシュ操作について	259
3.3.3	メモリマップ	259
3.3.4	8C00:0000 ~ 8C01:0000のメモリマップ	260
3.3.5	使用セクション	261



第 1 部

Ninja 編

第 1 章

Ninja ライブラリによる 画像設定

1.1 画面解像度設定について

1.1.1 はじめに

Dreamcast のグラフィックライブラリ Ninja で設定される画面モードについて解説します。
この設定は Dreamcast の機能である Image Super Sampling を使用する事で実現されており、様々なバリエーションのある画面設定が可能になります。

1.1.2 注意点

通常の描画モードに比べ、Image Super Sampling の機能を利用したモードでは、全画面アンチエイリアスの効果が得られますが、描画面積が 2~4 倍になります。

1.1.3 表示周期

- ・ NTSC 方式 : 1INT は 1/60 秒です (2INT は 1/30 秒です)
- ・ PAL 方式 : 1INT は 1/50 秒です (2INT は 1/25 秒です)
- ・ VGA 方式 : 1INT は 1/60 秒です (2INT は 1/30 秒です)

1.1.4 フリッカーフリー処理

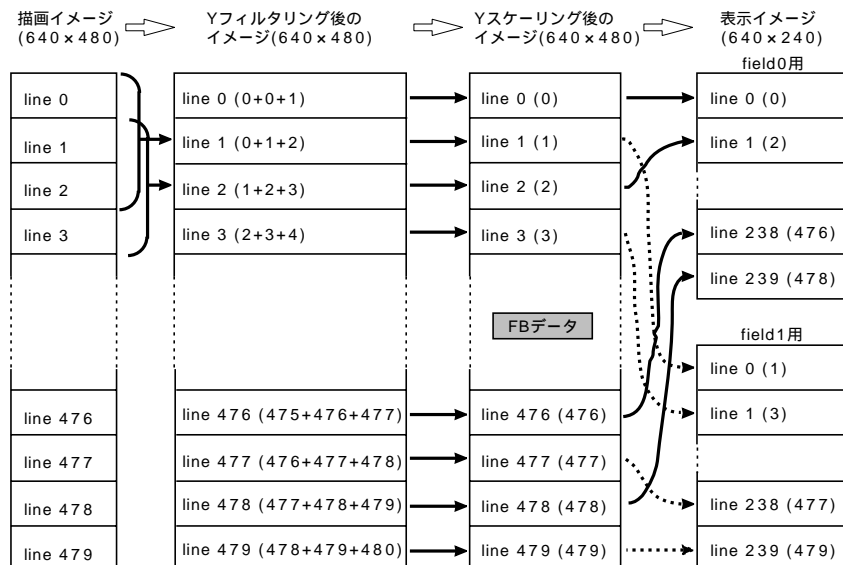
インターレス表示処理時に発生するフリッカー (垂直方向の揺れ、ちらつき) を低減する回路が Dreamcast には追加されています。この処理について簡単に解説いたします。

フリッカフリー機能は、Image Super Sampling 機能を利用して実現されます。PowerVR アーキテクチャでは、描画処理はすべて描画チップ内部で終了します。その結果をフレームバッファにコピーするのですが、その時に、3 ラインのフィルタ処理を行います。この 3 ラインのフィルタ処理に 2 のタイプがあります。

タイプ A

まず Image Super Sampling の Y スケーリング係数をできるだけ 1.0 倍に近い縮小係数にし、Y フィルタリングを有効にします。そして、3 ライン分のデータでフィルタリングを行った結果をフレームバッファへ格納します。つまり、480 ラインで行った描画結果をそのままのライン数でフレームバッファへ格納します。したがって、画面描画は 1 フレーム毎 (1/30 or 1/25 秒単位) に行えば良いことになります。

表示は、フレームバッファからのリード開始アドレスをフィールド毎に 1 ラインずらし、1 ラインずつ飛ばしてリードされます。

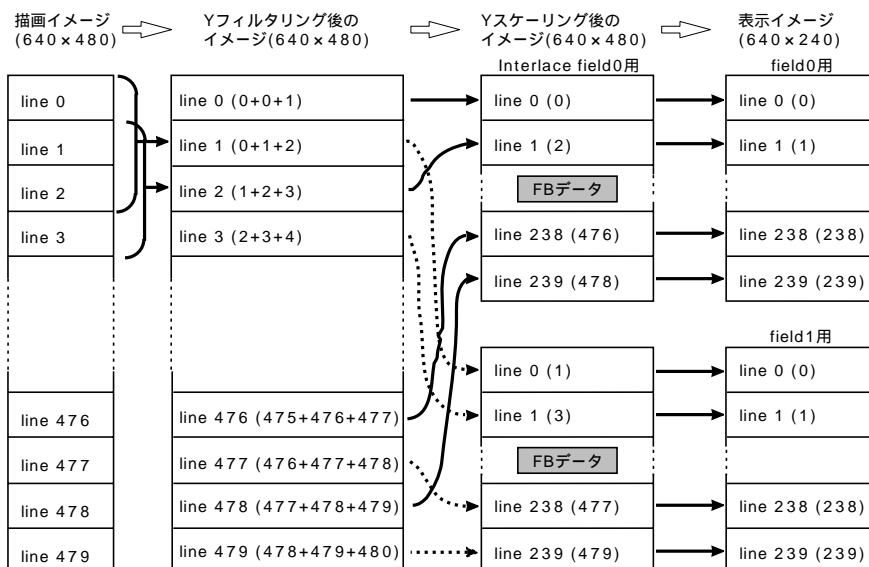


タイプ B

まず Image Super Sampling において 3 ライン分のデータでフィルタリングを行い、その結果を 1/2 スケーリングして指定された表示フィールドのために必要なラインデータだけをフレームバッファへ格納します。つまり、描画は 480 ラインで行う必要がありますが、フレームバッファへ格納されるピクセルデータは 240 ライン分で済みます。

ただし、これらの処理はタイル単位で描画されたピクセルデータをフレームバッファへ転送する際に行われるので、描画は毎フィールド (1/60 or 1/50 秒単位) 行わなければなりません。

表示は、フレームバッファからのそのままリードされます。



1.1.5 表記の説明

画面設定関数にセットする第一引数の値についての表示で、誤解が生じ易いので解説します。

(例1) NJD_RESOLUTION_640x480_PALNI_FF_ANTI

640x480 : 解像度。
PAL : PAL 方式。
NI : 表示コマ数。(ちなみにこれは秒間 60 コマを意味します。)
FF : タイプ B のフリッカーフリーインターレス。(タイプ A は単密以外のインターレスで自動設定。)
ANTI : アンチエイリアス付き。

(例2) NJD_RESOLUTION_640x480_NTSCI

NTSC : NTSC 方式。
I : 表示コマ数。(ちなみにこれは秒間 30 コマを意味します。)
倍密インターレスなので、タイプ A のフリッカーフリーを設定。

1.1.6 VRAM の各画面モード時における占有率

Holly2 の VRAM は以下の 3 つのバッファが存在します。

- (1) Native Buffer : 頂点バッファです。
- (2) Texture Buffer : テクスチャデータを格納します。
- (3) Frame Buffer : 描画結果を格納します。

Frame Buffer サイズの算出方法

Frame Buffer サイズを求める方法を提示します。

画面解像度	カラーサイズ	画面解像度 x カラーサイズ x バッファ数
320x240	16BPP	(320x240) x2 Byte x2 Buffer = 307200 Byte
	24BPP	(320x240) x3 Byte x2 Buffer = 460800 Byte
	32BPP	(320x240) x4 Byte x2 Buffer = 614400 Byte
320x480	16BPP	(320x480) x2 Byte x2 Buffer = 614400 Byte
	24BPP	(320x480) x3 Byte x2 Buffer = 921600 Byte
	32BPP	(320x480) x4 Byte x2 Buffer = 1228800 Byte
640x240	16BPP	(640x240) x2 Byte x2 Buffer = 614400 Byte
	24BPP	(640x240) x3 Byte x2 Buffer = 921600 Byte
	32BPP	(640x240) x4 Byte x2 Buffer = 1228800 Byte
640x480	16BPP	(640x480) x2 Byte x2 Buffer = 1228800 Byte
	24BPP	(640x480) x3 Byte x2 Buffer = 1843200 Byte
	32BPP	(640x480) x4 Byte x2 Buffer = 2457600 Byte

上記条件は、アンチエイリアスをかけても変わりませんが、タイプ B のフリッカーフリーを使用した場合にのみ半分になります。つまり、

NJD_RESOLUTION_640x480_NTSCNI_FF

NJD_RESOLUTION_640x480_NTSCNI_FF_ANTI

の場合、640x240 の解像度の時と同じになります。

Native Buffer の容量

Native Buffer の容量は

Native Buffer 容量 = VRAM 全体の容量 - Texture Buffer サイズ + 表示用 Frame Buffer サイズ

で算出されます。

ただし、Native Buffer (ダブルバッファ) の全容量は、Frame Buffer 全容量の半分を越えることはできません。

すなわちフレームバッファメモリ 8M バイトの量産システムでは、ネイティブデータバッファの容量は

最大 2M バイト × 2 個

が上限となります。上記の式で計算した結果がこの制限を越えた場合は、実際には制限いっぱい
の容量を Native Buffer (ダブルバッファ) として確保し、余った領域はテクスチャ領域
として使用されます。

1.1.7 画面設定モード

NinjaDef.h で定義される画面設定モードは次の種類になります。これを sbInitSystem() 関数、njInitSystem() 関数、njChangeSystem() 関数にて設定することで画面を切り替えます。

ノーマル画面設定モード (VGA)

```
NJD_RESOLUTION_VGA /* VGA モード */
```

アンチエイリアス画面設定モード (VGA)

```
NJD_RESOLUTION_VGA_ANTI /* VGA モード */
```

ノーマル画面設定モード (NTSC 方式)

NJD_RESOLUTION_320x240_NTSCNI	/* 320x240 ノンインターレス */
NJD_RESOLUTION_320x240_NTSCI	/* 320x240 単密インターレス */
NJD_RESOLUTION_320x480_NTSCNI	/* 320x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_320x480_NTSCI	/* 320x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x240_NTSCNI	/* 640x240 ノンインターレス */
NJD_RESOLUTION_640x240_NTSCI	/* 640x240 単密インターレス */
NJD_RESOLUTION_640x480_NTSCNI	/* 640x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_640x480_NTSCI	/* 640x480 倍密インターレス (Mode2) */

NJD_RESOLUTION_640x480_NTSCNI_FF

/* 640x480 Flicker Free(1INT Only) */

ノーマル画面設定モード (PAL 方式)

NJD_RESOLUTION_320x240_PALNI	/* 320x240 ノンインターレス */
NJD_RESOLUTION_320x240_PALI	/* 320x240 単密インターレス */
NJD_RESOLUTION_320x480_PALNI	/* 320x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_320x480_PALI	/* 320x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x240_PALNI	/* 640x240 ノンインターレス */
NJD_RESOLUTION_640x240_PALI	/* 640x240 単密インターレス */
NJD_RESOLUTION_640x480_PALNI	/* 640x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_640x480_PALI	/* 640x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x480_PALNI_FF	/* 640x480 Flicker Free(1INT Only) */

アンチエイリアス画面設定モード (NTSC 方式)

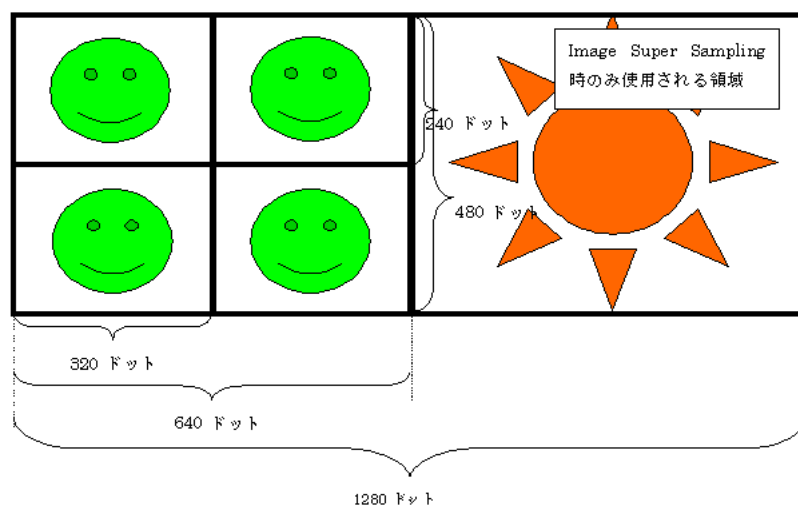
NJD_RESOLUTION_320x240_NTSCNI_ANTI	/* 320x240 ノンインターレス */
NJD_RESOLUTION_320x240_NTSCI_ANTI	/* 320x240 単密インターレス */
NJD_RESOLUTION_320x480_NTSCNI_ANTI	/* 320x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_320x480_NTSCI_ANTI	/* 320x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x240_NTSCNI_ANTI	/* 640x240 ノンインターレス */
NJD_RESOLUTION_640x240_NTSCI_ANTI	/* 640x240 単密インターレス */
NJD_RESOLUTION_640x480_NTSCNI_ANTI	/* 640x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_640x480_NTSCI_ANTI	/* 640x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x480_NTSCNI_FF_ANTI	/* 640x480 Flicker Free(1INT Only) */

アンチエイリアス画面設定モード (PAL 方式)

NJD_RESOLUTION_320x240_PALNI_ANTI	/* 320x240 ノンインターレス */
NJD_RESOLUTION_320x240_PALI_ANTI	/* 320x240 単密インターレス */
NJD_RESOLUTION_320x480_PALNI_ANTI	/* 320x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_320x480_PALI_ANTI	/* 320x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x240_PALNI_ANTI	/* 640x240 ノンインターレス */
NJD_RESOLUTION_640x240_PALI_ANTI	/* 640x240 単密インターレス */
NJD_RESOLUTION_640x480_PALI_ANTI	/* 640x480 倍密インターレス (Mode1) */
NJD_RESOLUTION_640x480_PALI_ANTI	/* 640x480 倍密インターレス (Mode2) */
NJD_RESOLUTION_640x480_PALNI_FF_ANTI	/* 640x480 Flicker Free(1INT Only) */

1.1.8 スクリーン座標系

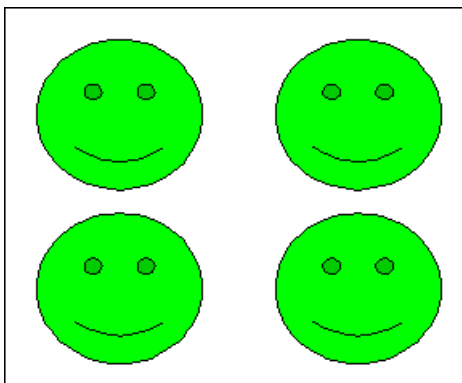
Ninja で定義されているスクリーン座標系を示します。



1.2 各画面モードの詳細

1.2.1 NJD_RESOLUTION_VGA

PC/AT のスタンダード VGA モードです。通常の TV では表示できません。スクリーン座標系のアスペクト比は 縦 : 横 = 1 : 1 で設定されます。



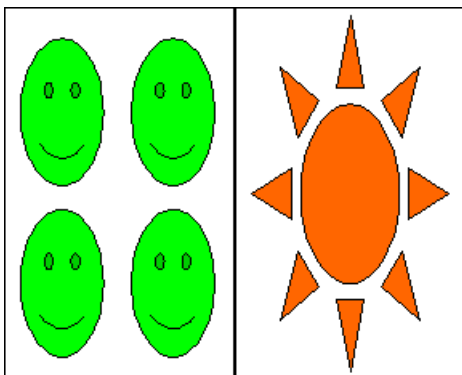
1.2.2 NJD_RESOLUTION_VGA_ANTI

PC/AT のスタンダード VGA モードです。通常の TV では表示できません。Image Super Sampling により、表示の際に水平方向のアンチエイリアスが掛かります。レンダリングサイズは 1280x480 で実際にフレームバッファに格納、表示されるサイズは 640x480 になります。スクリーン座標系のアスペクト比は

縦 : 横 = 1 : 0.5

につぶれて表示されます。

3D の場合は `njSetAspect()` 関数を利用して補正が可能です。2D の場合はあらかじめ縦に潰した画像を用意しなければなりません。



1.2.3 NJD_RESOLUTION_320x240_NTSC(PAL)NI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

ノンインターレス操作により EVEN 画像が表示されないため黒い横線が入ります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.4 NJD_RESOLUTION_320x240_NTSC(PAL)I

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

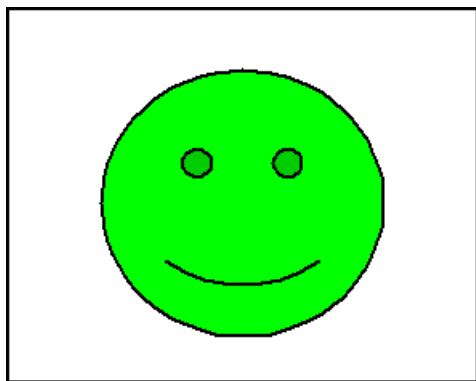
単密インターレス操作により ODD、EVEN とともに同じ画像が表示されます。このため、フリッカが発生し画像が縦に揺れます。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



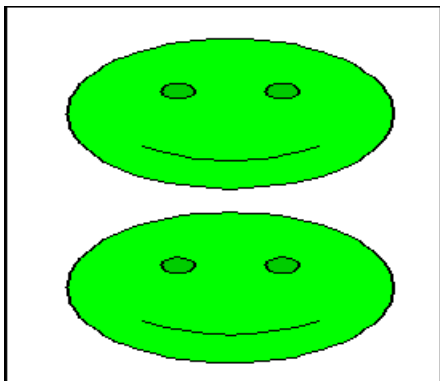
1.2.5 NJD_RESOLUTION_320x480_NTSC(PAL)NI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、1INT で描画を更新するため、フレームバッファから画像を表示する際に、ODDEVEN どちらかの画像が捨てられて表示されます。

スクリーン座標系のアスペクト比は縦：横 = 1：0.5 ですが、TV 表示の際に、垂直方向に 1/2 に潰されて表示されます。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



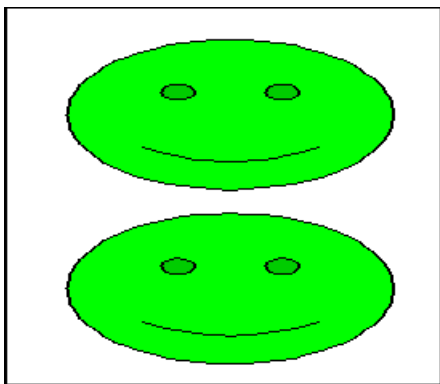
1.2.6 NJD_RESOLUTION_320x480_NTSC(PAL)I

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、2INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD EVEN の画像が捨てられることはありませんが、1INT 処理のゲームを作る事が出来ません。また、処理落ちした場合 4INT になるため注意が必要です。

スクリーン座標系のアスペクト比は縦：横 = 1：0.5 ですが、TV 表示の際に、垂直方向に 1/2 に潰されて表示されます。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.7 NJD_RESOLUTION_640x240_NTSC(PAL)NI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

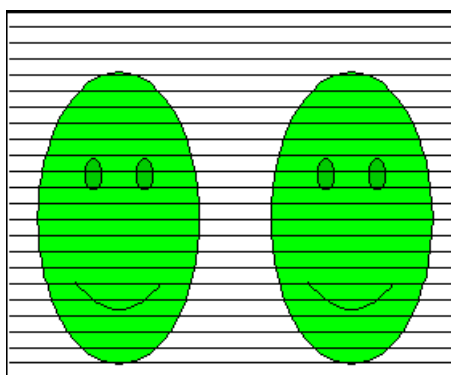
ノンインターレス操作により EVEN 画像が表示されないため黒い横線が入ります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.8 NJD_RESOLUTION_640x240_NTSC(PAL)I

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

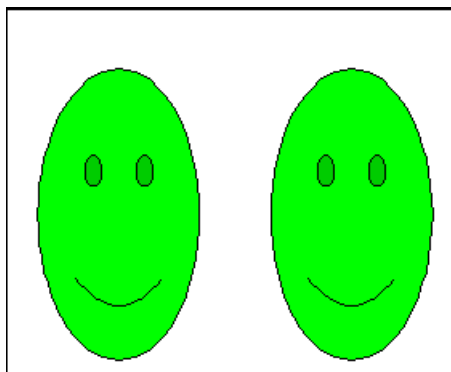
単密インターレス操作により ODD、EVEN とともに同じ画像が表示されます。このため、フリッカが発生し画像が縦に揺れます。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.9 NJD_RESOLUTION_640x480_NTSC(PAL)NI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

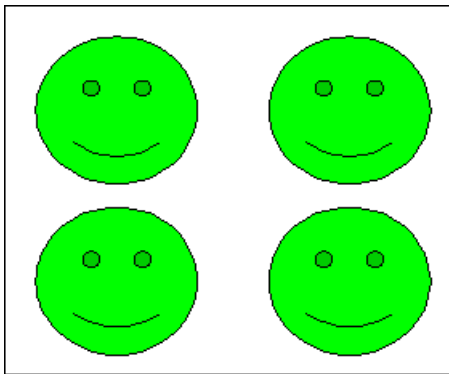
このモードでは、1INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD、EVEN どちらかの画像が捨てられて表示されます。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.10 NJD_RESOLUTION_640x480_NTSC(PAL)I

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレースにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、2INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD、EVEN の画像が捨てられることはありませんが、1INT 処理のゲームを作る事が出来ません。

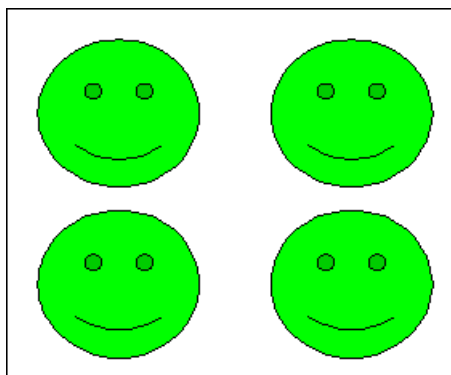
また、処理落ちした場合 4INT になるため注意が必要です。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.11 NJD_RESOLUTION_640x480_NTSC(PAL)NI_FF

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

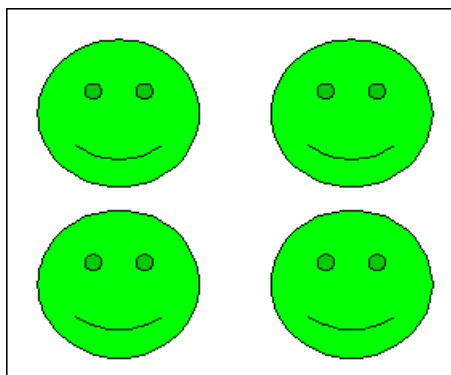
このモードでは、レンダリングされた画像をフレームバッファに置く際に、ODD、EVEN のどちらかを捨てます。フレームバッファのサイズは通常の半分で済みますが、1INT で確実に描画が終了しなければ、ならず、一回でも描画落ちした場合、表示の際に ODD、EVEN が逆転して絵が崩れます。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.12 NJD_RESOLUTION_320x240_NTSC(PAL)NI_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

ノンインターレス操作により EVEN 画像が表示されないため黒い横線が入ります。

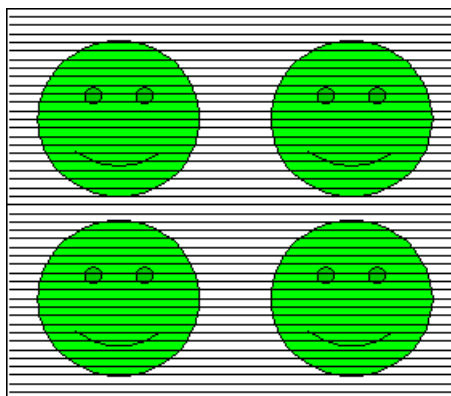
Image Super Sampling により、表示の際に水平、垂直方向のアンチエイリアスが掛かります。レンダリングサイズは 640x480 で実際にフレームバッファに格納、表示されるサイズは 320x240 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.13 NJD_RESOLUTION_320x240_NTSC(PAL)I_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

単密インターレス操作により ODD、EVEN とともに同じ画像が表示されます。このため、フリッカが発生し画像が縦に揺れます。

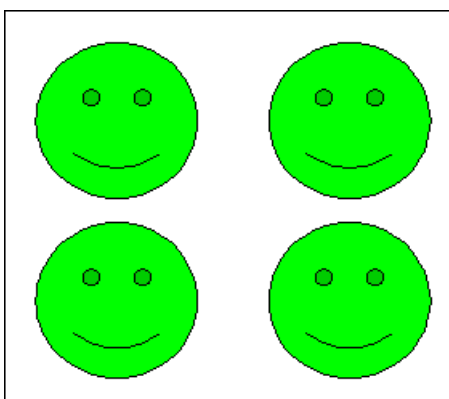
Image Super Sampling により、表示の際に水平、垂直方向のアンチエイリアスが掛かります。レンダリングサイズは 640x480 で実際にフレームバッファに格納、表示されるサイズは 320x240 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.14 NJD_RESOLUTION_320x480_NTSC(PAL)NI_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレースにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、1INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD、EVEN どちらかの画像が捨てられて表示されます。

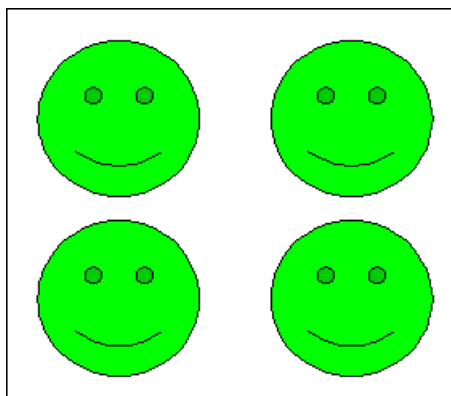
Image Super Sampling により、表示の際に水平方向のアンチエイリアスが掛かります。レンダリングサイズは 640x480 で実際にフレームバッファに格納、表示されるサイズは 320x480 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.15 NJD_RESOLUTION_320x480_NTSC(PAL)I_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、2INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD、EVEN の画像が捨てられることはありませんが、1INT 処理のゲームを作る事が出来ません。

また、処理落ちした場合 4INT になるため注意が必要です。

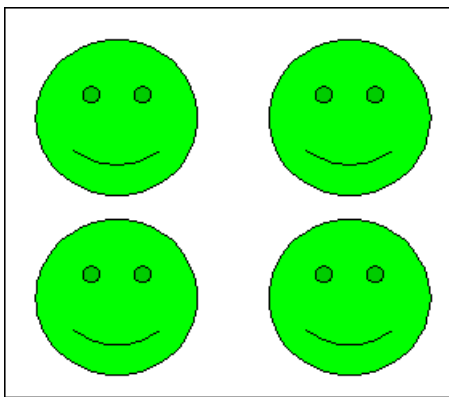
Image Super Sampling により、表示の際に水平方向のアンチエイリアスが掛かります。レンダリングサイズは 640x480 で実際にフレームバッファに格納、表示されるサイズは 320x480 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.91

PAL 方式では縦：横 = 1：1.27

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.16 NJD_RESOLUTION_640x240_NTSC(PAL)NI_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

ノンインターレス操作により EVEN 画像が表示されないため黒い横線が入ります。

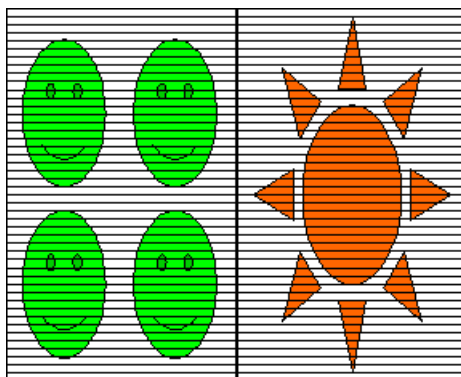
Image Super Sampling により、表示の際に水平、垂直方向のアンチエイリアスが掛かります。レンダリングサイズは 1280x480 で実際にフレームバッファに格納、表示されるサイズは 640x240 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.17 NJD_RESOLUTION_640x240_NTSC(PAL)I_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

単密インターレス操作により ODD、EVEN とともに同じ画像が表示されます。このため、フリッカが発生し画像が縦に揺れます。

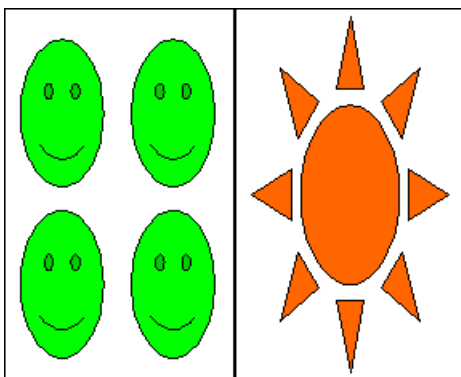
Image Super Sampling により、表示の際に水平、垂直方向のアンチエイリアスが掛かります。レンダリングサイズは 1280x480 で実際にフレームバッファに格納、表示されるサイズは 640x240 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.18 NJD_RESOLUTION_640x480_NTSC(PAL)NI_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレースにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、1INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD、EVEN どちらかの画像が捨てられて表示されます。

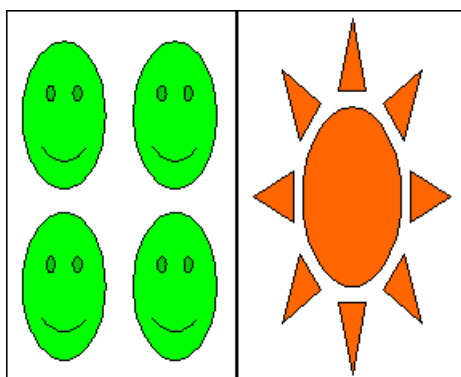
Image Super Sampling により、表示の際に水平方向のアンチエイリアスが掛かります。レンダリングサイズは 1280x480 で実際にフレームバッファに格納、表示されるサイズは 640x480 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ njSetAspect() 関数を利用して補正が可能です。



1.2.19 NJD_RESOLUTION_640x480_NTSC(PAL)I_ANTI

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、2INT で描画を更新するため、フレームバッファから画像を表示する際に、ODD

EVEN の画像が捨てられることはありませんが、1INT 処理のゲームを作る事が出来ません。

また、処理落ちした場合 4INT になるため注意が必要です。

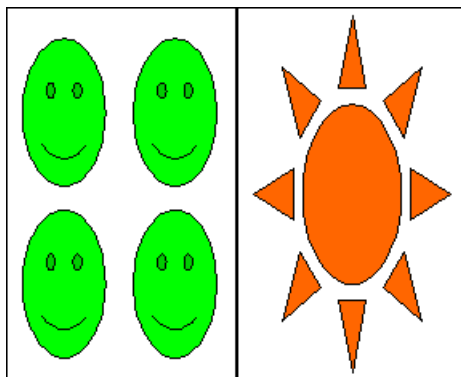
Image Super Sampling により、表示の際に水平方向のアンチエイリアスが掛かります。レンダリングサイズは 1280x480 で実際にフレームバッファに格納、表示されるサイズは 640x480 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



1.2.20 NJD_RESOLUTION_640x480_NTSC(PAL)I_ANTI_FF

家庭 TV の表示モードです。PC/AT 用のモニターでは表示できません。

倍密インターレスにより ODD、EVEN に異なった画像が表示されるため画像のクオリティは上がります。

このモードでは、レンダリングされた画像をフレームバッファに置く際に、ODD、EVEN のどちらかを捨てます。フレームバッファのサイズは通常の半分で済みますが、1INT で確実に描画が終了しなければ、ならず、一回でも描画落ちした場合、表示の際に ODD、EVEN が逆転して絵が崩れます。

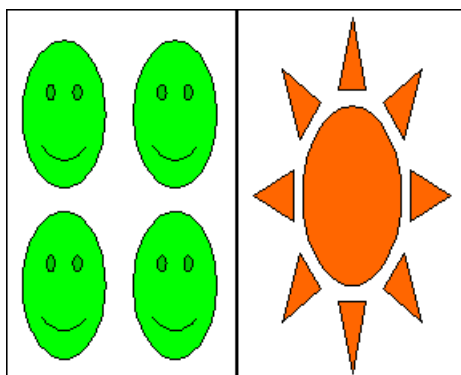
Image Super Sampling により、表示の際に水平方向のアンチエイリアスが掛かります。レンダリングサイズは 1280x480 で実際にフレームバッファに格納、表示されるサイズは 640x480 になります。

スクリーン座標系のアスペクト比は縦：横 = 1：1 ですが、TV 表示の際に、

NTSC 方式では縦：横 = 1：0.45

PAL 方式では縦：横 = 1：0.64

になります。これは 3D の場合にのみ `njSetAspect()` 関数を利用して補正が可能です。



2.1 概要

Ninja は本資料に説明されている Basic Model 形式以外に、Chunk Model 形式によるモデル構造をサポートしています。Chunk Model 形式は、描画実行中に SH4 のキャッシュを壊さないようにデータを連続するメモリ空間に配置しています。拡張性、柔軟性、データの表現効率に優れています。今後は Chunk Model 形式のモデルを中心としたチューニングを実施します。Basic Model はサポートされますが、新機能はサポートされません。

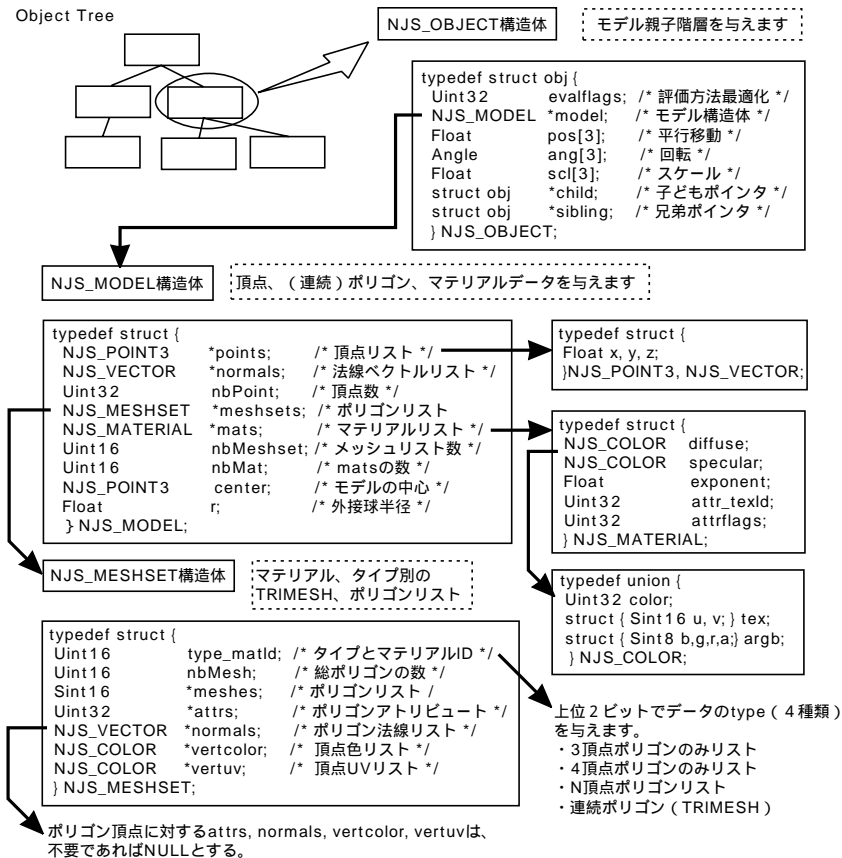
Chunk Model 形式は Model 構造体の中身を大幅に変更しますが、Object 構造体は Model 構造体のポインタから Chunk Model 構造体のポインタへの変更以外に変更はありません。

モデル以外のモーションおよびテクスチャについては、引き続き現行の形式が使用されます (ただしカメラ、ライト対応のための構造体メンバの型が変更されます)。

Chunk 形式モデルについては、『Chunk Model 仕様書』を参照してください。

2.2 モデル構造体

構造体図



構造体解説

Float, Angle

```
typedef float Float          /* 浮動小数点演算型 */
typedef Sint32 Angle        /* 回転角度 */
```

- ・アングルは 0x0000 ~ 0xFFFF が 0 ~ 360 度です。

カラー構造体

```
typedef union {
    Uint32 color;          /* ロングアクセス */
    struct {
        Sint16 u;          /* テクスチャu値 */
        Sint16 v;          /* テクスチャv値 */
    } tex;                 /* テクスチャアクセス */
}
```

```

    struct {
        Uint8 b;          /* b値          */
        Uint8 g;          /* g値          */
        Uint8 r;          /* r値          */
        Uint8 a;          /* アルファブレンド値 */
    } argb;               /* argbアクセス */
} NJS_COLOR;

```

- ・色、テクスチャUV を格納します。共用体を利用します。
- ・ツールは color からデータを設定し、ライブラリは tex, argb からアクセスします。

オブジェクト構造体

```

typedef struct obj {
    Uint32      evalflags; /* 評価方法の最適化フラグ */
    NJS_MODEL   *model;    /* モデル構造体ポインタ   */
    Float       pos[3];    /* 平行移動               */
    Angle       ang[3];    /* 回転                   */
    Float       scl[3];    /* スケール               */
    struct obj  *child;    /* 子どもobjectへのポインタ */
    struct obj  *sibling;  /* 兄弟objectへのポインタ  */
} NJS_OBJECT;

```

- ・モデルの親子階層を与えます。
- ・model にはポリゴン、TRIMESH(連続ポリゴン) がセットされます。

evalflags フラグ説明

```

#define NJD_EVAL_UNIT_POS      BIT_0 /* 移動が無視できる      */
#define NJD_EVAL_UNIT_ANG     BIT_1 /* 回転が無視できる      */
#define NJD_EVAL_UNIT_SCL     BIT_2 /* スケールが無視できる。 */
#define NJD_EVAL_HIDE        BIT_3 /* モデルを描画しない。  */
#define NJD_EVAL_BREAK       BIT_4 /* childのトレースの打ち切り */
#define NJD_EVAL_ZXY_ANG     BIT_5
/* LightWave3Dで期待される回転の評価の指定 */
#define NJD_EVAL_SKIP        BIT_6
/* モーションをスキップ */
#define NJD_EVAL_SHAPE_SKIP  BIT_7
/* シェイプモーションをスキップ */
#define NJD_EVAL_MASK        0xff
/* 上記ビットを抽出するためのマスク */

```

これらのフラグはコンバータで設定されます。

- ・NJD_EVAL_UNIT_POS は、平行移動量がゼロの場合に設定されます。平行移動のマトリクス計算が省略されます。
- ・NJD_EVAL_UNIT_ANG は、回転量がゼロの場合に設定されます。回転のマトリクス計算が省略されます。
- ・NJD_EVAL_UNIT_SCL は、スケールが xyz とともに 1 の場合に設定されます。スケールのマトリクス計算が省略されます。
- ・NJD_EVAL_UNIT_POS、NJD_EVAL_UNIT_ANG、NJD_EVAL_UNIT_SCL の3つが

セットされていると、すべてのマトリクス演算に加え、マトリクスの pushpop も省略されます。

- ・ NJD_EVAL_HIDE は、ユーザーによって立てられます。このフラグが立っている場合、モデルが描画されません。モデルに装着されたガンとブレードの切り替えなどに使用します。
- ・ NJD_EVAL_BREAK は、ユーザーによって立てられます。このフラグが立っている場合、child の探索をここで打ち切ります。例えば、ルートノードでこのフラグを立てると、モデル全体が消えます。NJD_EVAL_BREAK は、モーションと組み合わせるとデータがずれるので、ルートノードのみでを使用することをお勧めしますが、ユーザーの責任において途中のノードで使用することもできます。
- ・ LightWave3D では、回転の評価の順番は ZXY となります。Ninja は通常 XYZ の順番であるため、LightWave3D の場合の評価順をライブラリで実行するためのフラグとして、NJD_EVAL_ZXY_ANG が用意されます。このフラグが ON の場合、回転の計算順序が ZXY に変更されます。
- ・ NJD_EVAL_SKIP は、このノードがモーションデータに含まれないことを示します。モーション実行時はモーションのノードを次に進めずに、オブジェクト構造体の値でマトリクス計算を行い、次のノードに進みます。これにより、ポリゴンモデルの構成が異なるモデルでも、骨の構造が同じであればモーションが行えます。
- ・ NJD_EVAL_SHAPE_SKIP は、このノードがシェイプモーションデータに含まれないことを示します。

ポイント構造体

```
typedef struct {  
    Float    x;        /* X値    */  
    Float    y;        /* Y値    */  
    Float    z;        /* Z値    */  
} NJS_POINT3, NJS_VECTOR;
```

- ・ 頂点の XYZ 値を与えます。

テクスチャネーム構造体

```
typedef struct {  
    void      *filename; /* テクスチャファイルネーム */  
    Uint32     attr;     /* テクスチャアトリビュート */  
    void      *texaddr;  /* テクスチャメモリアドレス */  
} NJS_TEXNAME;
```

- ・ テクスチャはファイル名で指定します。
- ・ globalIndex は、Uint32 で一意につけられたテクスチャの番号です。ただし、0xffffffff ~ 0xffffffff はライブラリが内部フラグとして使用するので使わないでください。
- ・ globalIndex は、テクスチャファイルに格納されます。Ninja では、テクスチャファイルの先頭に必ず globalIndex のチャンクを置きます。
- ・ globalIndex は、ツールで割り振られ管理されます。Ninja では、この番号で同一テクス

チャを検出し、テクスチャメモリ上への二重登録を防ぎます。

- ・ attr は、テクスチャのタイプとキャッシュ指定に利用されます。

```
#define NJD_TEXATTR_TYPE_FILE      0          /*ファイルテクスチャ */
#define NJD_TEXATTR_CASHE          BIT_31
/*テクスチャをキャッシュに登録 */
#define NJD_TEXATTR_TYPE_MEMORY    BIT_30      /*メモリテクスチャ */
#define NJD_TEXATTR_BOTH           BIT_29
/* テクスチャをキャッシュとテクスチャメモリに登録 */
#define NJD_TEXATTR_MASK           0xE0000000
```

- ・ メモリタイプのテクスチャでは、attr にテクスチャのカラータイプとカテゴリコードを設定する必要があります。これは、.pvr ファイルのテクスチャタイプに設定されるビット列と同じです。

```
/* カラータイプ */
#define NJD_TEXFMT_ARGB_1555      (0x00)
#define NJD_TEXFMT_RGB_565        (0x01)
#define NJD_TEXFMT_ARGB_4444      (0x02)
#define NJD_TEXFMT_YUV_422        (0x03)
#define NJD_TEXFMT_BUMP           (0x04)
#define NJD_TEXFMT_RGB_555        (0x05)
#define NJD_TEXFMT_COLOR_MASK     (0xFF)

/* カテゴリコード */
#define NJD_TEXFMT_TWIDDLED        (0x0100)
#define NJD_TEXFMT_TWIDDLED_MM    (0x0200)
#define NJD_TEXFMT_VQ              (0x0300)
#define NJD_TEXFMT_VQ_MM          (0x0400)
#define NJD_TEXFMT_PALETTIZE4      (0x0500)
#define NJD_TEXFMT_PALETTIZE4_MM  (0x0600)
#define NJD_TEXFMT_PALETTIZE8      (0x0700)
#define NJD_TEXFMT_PALETTIZE8_MM  (0x0800)
#define NJD_TEXFMT_RECTANGLE       (0x0900)
#define NJD_TEXFMT_STRIDE          (0x0B00)
#define NJD_TEXFMT_SMALLVQ         (0x1000)
#define NJD_TEXFMT_SMALLVQ_MM     (0x1100)
#define NJD_TEXFMT_TYPE_MASK       (0xFF00)
```

- ・ カテゴリコード */texaddr は、ターゲット上で texlist をカレント texlist に設定した場合に割り振られるテクスチャメモリアドレスを格納します。このアドレスは、ライブラリ内部でカレントテクスチャを指定するときに利用されます。

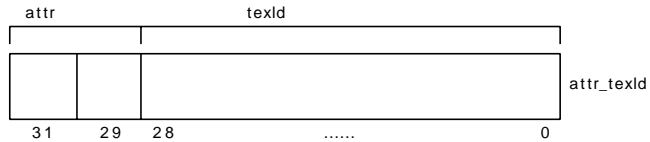
テクスチャリスト構造体

```
typedef struct {
    NJS_TEXNAME      *textures;          /* テクスチャネームリスト */
    Uint16           nbTexture;          /* テクスチャの数 */
} NJS_TEXLIST;
```

- ・ 複数のテクスチャを一括してテクスチャメモリに書き込むためのリストです。ライブラリのテクスチャ指定は、texlist 単位でされます。

マテリアル構造体

```
typedef struct {
    NJS_COLOR    diffuse;          /* 拡散反射( モデルの色 )0 ~ 255 */
    NJS_COLOR    specular;        /* 鏡面反射( ハイライト )0 ~ 255 */
    FLOAT        exponent;        /* ハイライトの広がり0 ~ 300 */
    Uint32       attr_texId;      /* アトリビュートとテクスチャID */
    Uint32       attrflags;       /* アトリビュートフラグ */
} NJS_MATERIAL;
```



- ・ attr_texId は、カレントテクスチャリスト texlist におけるテクスチャ番号を指定します。attr 領域は現在未使用です。
- ・ モデルツリーはテクスチャ情報として、texlist のエントリ番号に対応する texId しか持ちません。ユーザーは現在のモデルに対応する texlist をカレントに設定します。
- ・ attrflags に設定されるアトリビュートについては、『[Ninja アトリビュート](#)』を参照してください。

メッシュセット構造体

```
typedef struct {
    Uint16       type_matId;      /* タイプとマテリアルID(0-16383) */
    Uint16       nbMesh;         /* ポリゴン/連続ポリゴン数 */
    Sint16       *meshes;        /* ポリゴンリスト */
    Uint32       *attrs;         /* ポリゴンアトリビュート */
    NJS_VECTOR    *normals;      /* ポリゴンの法線ベクトルリスト */
    NJS_COLOR    *vertcolor;     /* ポリゴン頂点の色リスト */
    NJS_COLOR    *vertuv;        /* ポリゴン頂点のUVリスト */
} NJS_MESHSET;
```

- ・ attrs には、ポリゴン単位のアトリビュートが設定されます。attrs に設定されるアトリビュートは、NJS_MATERIAL の attrflags に設定されるものと同じです。

メッシュセットの詳細は、後述のモデル構造の『メッシュセット』で説明されています。

- ・ attrs に設定されるアトリビュートについては、『[Ninja アトリビュート](#)』を参照してください。

モデル構造体

```
typedef struct {
    NJS_POINT3    *points;        /* 頂点リスト */
    NJS_VECTOR    *normals;      /* 頂点の法線ベクトルリスト */
    Uint32       nbPoint;        /* 頂点の数。 */
    NJS_MESHSET   *meshsets;     /* ポリゴン、TRIMESHリスト */
    NJS_MATERIAL   *mats;        /* マテリアルのリスト */
    Uint16       nbMeshset;      /* meshsetの数。最大値65535 */
}
```

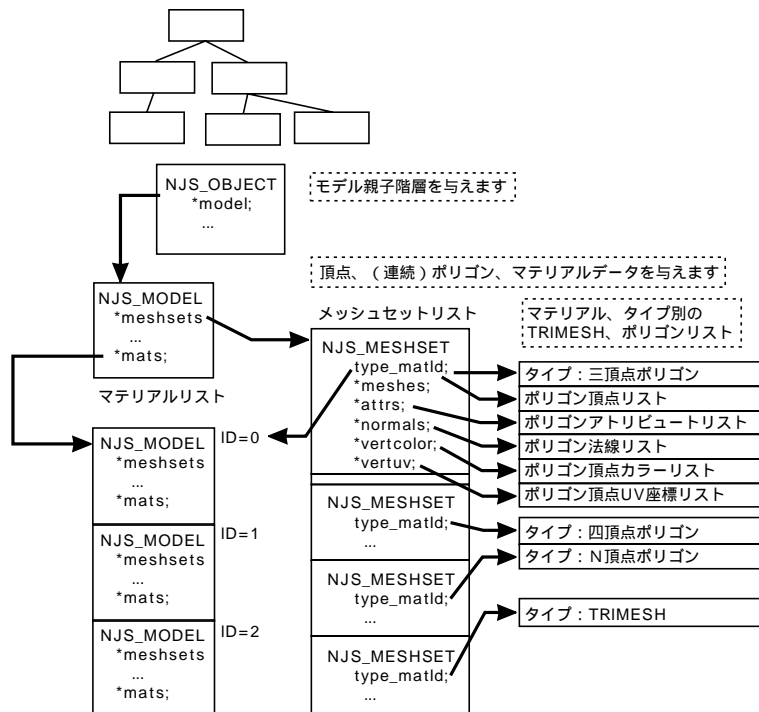


```
    Uint16          nbMat;          /* matsの数。最大値65535 */
    NJS_POINT3      center;        /* デルの中心 */
    Float           r;             /* モデル中心からの外接球半径 */
} NJS_MODEL;
```

- 頂点リストは、MODEL 構造体にセットされる複数の meshset で使われる頂点をすべて含みます。
- 頂点の法線が不要な場合は、normals に NULL を設定します。
- meshset は、単一のマテリアルを使用する単一のタイプ(三角ポリゴン、四角ポリゴン、N 角形ポリゴン、TRIMESH) のポリゴンの集合です。
- 各 meshset は使用するマテリアルの ID を持ち、mats 配列の何番目かを指示できます。
- center, r はモデルのコリジョン計算等に利用されます。

2.3 モデル構造

構造体関連図

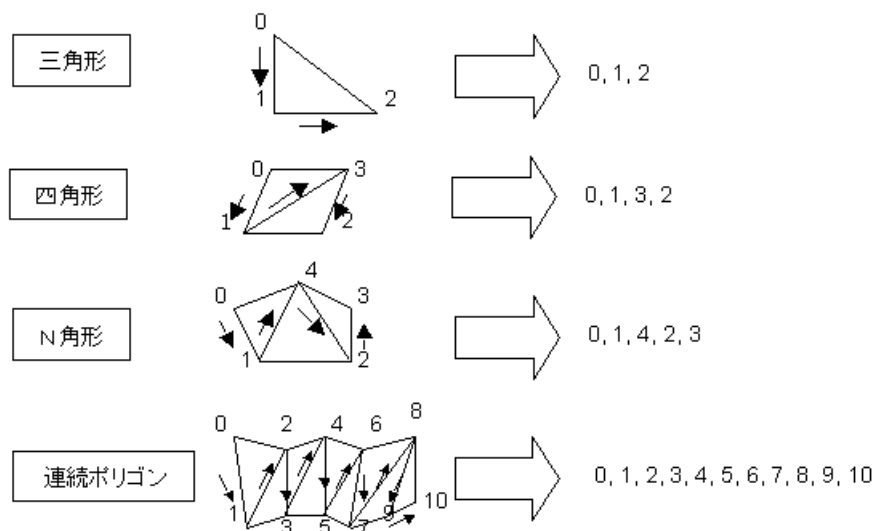


- ・メッシュセットリストは、使用されているもののみリストです。例えば、三角系ポリゴン
のみのリストだけの場合は、nbMeshset=1 です。
- ・attrs, normals, vertcolor, vertuv は、不要であれば NULL とします。

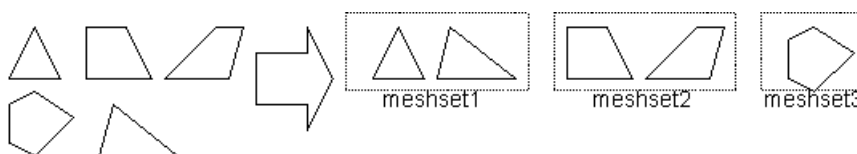
2.3.1 メッシュセット

```
typedef struct {
    Uint16      type_matId;      /* タイプとマテリアルID(0-16383) */
    Uint16      nbMesh;         /* ポリゴン/連続ポリゴン数 */
    Sint16      *meshes;        /* ポリゴンリスト */
    Uint32      *attrs;         /* ポリゴンアトリビュート */
    NJS_VECTOR  *normals;       /* ポリゴンの法線ベクトルリスト */
    NJS_COLOR   *vertcolor;     /* ポリゴン頂点の色リスト */
    NJS_COLOR   *vertuv;        /* ポリゴン頂点のUVリスト */
} NJS_MESHSET;
```

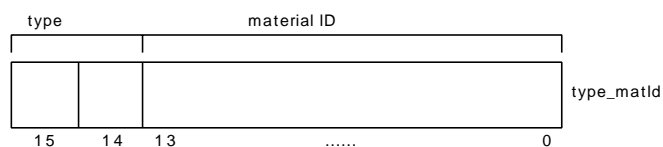
- ・ 三角形ポリゴンのみ、四角形ポリゴンのみ、N 角形ポリゴンのみ、TRIMESH (連続ポリゴン) のみのデータ列を格納します。
- ・ 頂点の順番は、すべて連続ポリゴンの描画順番 (ジグザグ) になります。



- ・ meshsets には、複数のタイプの meshset の配列がセットされます。
- ・ データに三角形ポリゴン、四角形ポリゴン、N 角形ポリゴンが含まれるデータの場合、Ninja コンバータにより頂点数で分割された meshset になります。



- ・ さらに、例えば三角形ポリゴンが複数のマテリアルを使用していた場合、meshset はマテリアル単位に分割されます。
- ・ type_matId は、上位 2 ビット (14-15 ビット) で meshset のタイプを、下位 14 ビット (0-13 ビット) でモデル構造体マテリアルリストの何番目のマテリアルが使用されているかを示します。



```
#define NJD_MESHSET_3      0x0000
#define NJD_MESHSET_4      0x4000
#define NJD_MESHSET_N      0x8000
#define NJD_MESHSET_TRIMESH 0xc000
```

次に各タイプ別のデータ構造について説明します。

三角形ポリゴンリスト (NJD_MESHSET_3) の場合

(例)

```

      Polygon1 Polygon2
      ┌───┐ ┌───┐
meshes[] = {3, 4, 5, 9, 8, 6, 2, 10, 7, 13, 14, 11, ...}
attrs[]   = NULL;
normals[] = {{1.0,0.0,0.0},{0.0, 1.0, 0.0}, ...}
vertcolor[] = {0xFFFF,0xEEEE,0xCCCC,...}
vertuv[]    = {0xEFAB,0xFF98,0x44FF,...}
nbMesh      = meshesの頂点数 ÷ 3
```

- ・ポリゴン単位のアトリビュート attrs は未対応です。
- ・法線はポリゴンに一つ割り当てられる。n 番目のポリゴンの法線は normals[n] です (n=0, 1, 2, ...)。
- ・meshes[i] の頂点に対応するカラーおよび UV はそれぞれ、vertcolor[i]、vertuv[i] です (i=0, 1, 2, ...)。
- ・不要な場合は、attrs、normals、vertcolor、vertuv には NULL ポインタを設定します。

四角形ポリゴンリスト (NJD_MESHSET_4) の場合

(例)

```

      Polygon1 Polygon2
      ┌───┐ ┌───┐
meshes[] = {3, 4, 5, 9, 8, 6, 2, 10, 7, 13, 14, 11, ...}
attrs[]   = NULL;
vertcolor[] = {0xFFFF,0xEEEE,0xCCCC,...}
vertuv[]    = {0xEFAB,0xFF98,0x44FF,...}
nbMesh      = meshesの頂点数 ÷ 4
```

- ・ポリゴン単位のアトリビュート attrs は未対応です。
- ・法線はポリゴンに一つ割り当てられます。n 番目のポリゴンの法線は normals[n] です (n=0, 1, 2, ...)。
- ・meshes[i] の頂点に対応するカラーおよび UV は、それぞれ vertcolor[i]、vertuv[i] です (i=0, 1, 2, ...)。
- ・不要な場合は、attrs、normals、vertcolor、vertuv には NULL ポインタを設定します。

連続ポリゴンリスト (NJD_MESHSET_TRIMESH) の場合

連続ポリゴンは、先頭に連続するポリゴンを構成する頂点数を書くことによって表現されます。

(例)

```

      trimesh1      trimesh2
      ┌──────────┐ ┌──────────┐
meshes[] = {6, 3, 4, 5, 9, 8, 6, 4, 2, 10, 7, 13, 11, ...}
attrs[]   = NULL;
normals[] = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, ...}
vertcolor[] = {0xFFFF, 0xEEEE, 0xCCCC, ...}
vertuv[]   = {0xEFAB, 0xFF98, 0x44FF, ...}
nbMesh     = meshesの数

```

- ・ポリゴン単位のアトリビュート attrs は未対応です。
- ・通常 trimesh の法線は外積により求められますが、この法線をデータとして normals に持つことができます。
- ＊ 法線は三角形ポリゴン換算で、各ポリゴンに一つ割り当てられます。n 番目の三角形ポリゴンの法線は normals[n] です (n=0, 1, 2, ...)
- ・ meshes[i] の頂点に対応するカラーおよび UV はそれぞれ、vertcolor[i - (k+1)]、vertuv[i - (k+1)] です (i=0, 1, 2, ...) (k=0, 1, 2, ...)。ここで k はカレントの trimesh 番号 (k 番目) です。
- ・ 不要な場合は、attrs、normals、vertcolor、vertuv に NULL ポインタを設定します。

特記事項

ninja では、trimesh(triangle strip) を効率よくつなぐために、trimesh の右回り、左回りからの開始をサポートしています。右回りの場合、長さを与える先頭の値の最上位ビットに 1 ビットのフラグが立ちます。

左回りから始まる場合 長さ、頂点 1、頂点 2、...

右回りから始まる場合 0x8000|長さ、頂点 1、頂点 2、...

N 角形ポリゴンリスト (NJD_MESHSET_N) の場合

- ・ここでの N は 5 以上を意味します。つまり五角形以上のポリゴンリストを生成します。コンバータオプションで 3 以上の場合のリストも作成可能にする予定です。
- ・N 角形ポリゴンの場合は、meshes の頂点番号と vertcolor、vertuv の番号がずれてくるので注意してください。

(例)

```

      Polygon1      Polygon2
      ┌──────────┐ ┌──────────┐
meshes[] = {5, 3, 4, 5, 9, 8, 7, 6, 2, 10, 7, 13, 14, 11, ...}

```

- ・アンダーラインの数値は N 頂点数を示し、その後ろに N 個の頂点が並びます。

```
attrs[] = NULL;
```

- ・ポリゴン単位のアトリビュートは未対応。

```
normals[] = {{1.0,0.0,0.0},{0.0, 1.0, 0.0}, ...}
```

- ・法線ベクトルは各ポリゴンに割り当てられます。k 番目のポリゴンの法線は normals[k] です (k=0, 1, 2, ...)。
- ・N 角形の各頂点は同一平面上にあるものとし、最初の三点から求められた法線をポリゴンの法線とします。

```
vertcolor[] = {0xFFFF,0xEEEE,0xCCCC,...}
vertuv[]    = {0xEFAB,0xFF98,0x44FF,...}
```

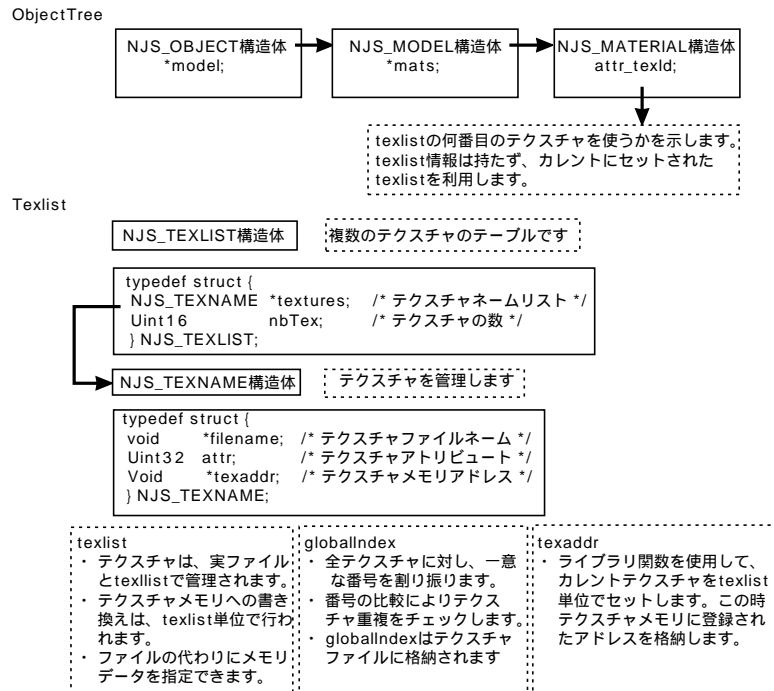
- ・meshes[i] に対応する頂点のカラーおよび UV は、それぞれ vertcolor[i-(k+1)]、vertuv[i-(k+1)] です (k=0, 1, 2, ...)、(i=0, 1, 2, ...)。ここでの k はカレントポリゴンの番号 (k 番目) です。これは、meshes は各ポリゴンごとに N 角形をあらわす N を持ちますが、vertcolor、vertuv は持たないためです。

```
nbMesh = N角形ポリゴン数
```

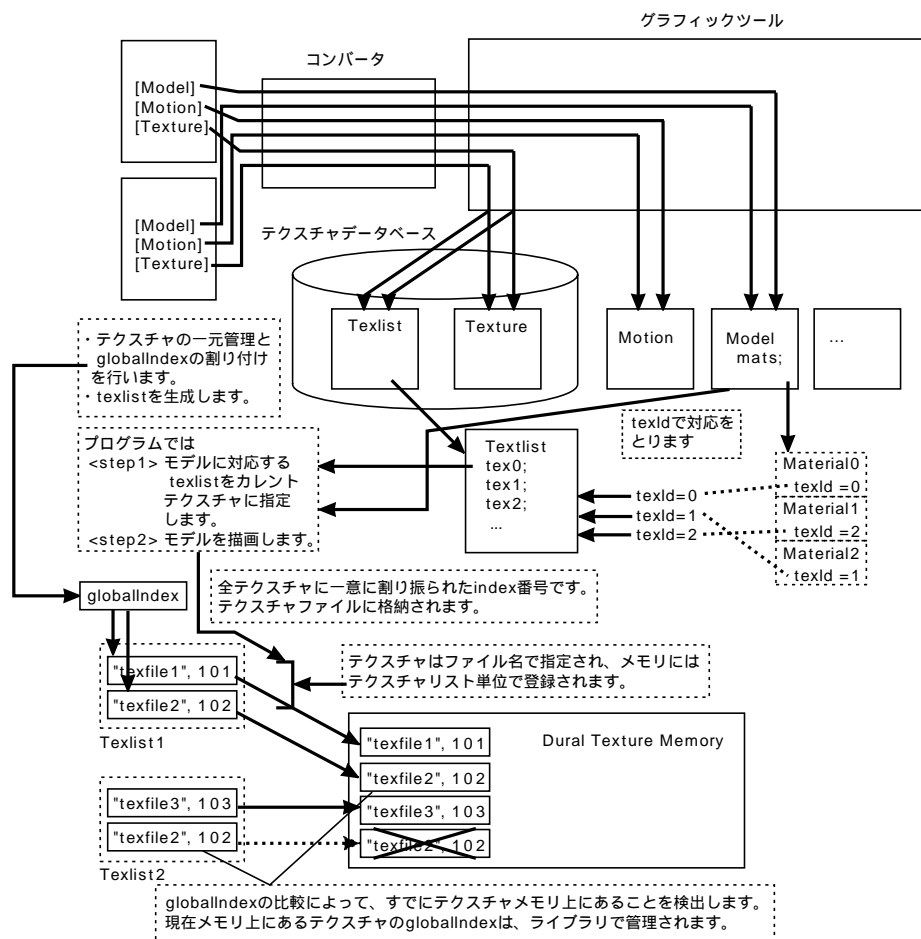
- ・不要な場合は、attrs、normals、vertcolor、vertuv に NULL ポインタを設定します。

2.3.2 テクスチャ構造

構造体図

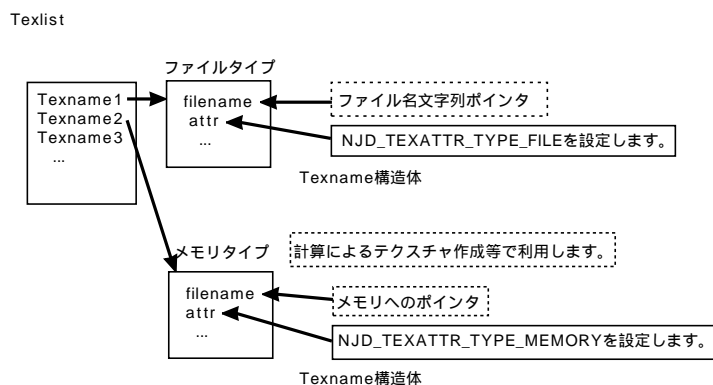


テクスチャ処理の概要



メモリ形式テクスチャ、テクスチャキャッシュ

詳細はライブラリの仕様書を参照してください。ここでは概要のみを示します。



attr に NJD_TEXATTR_CASHE フラグをセットすると、テクスチャキャッシュのみへのセットを指定します。NJD_TEXATTR_BOTH をセットすると、texlist 単位でのテクスチャメモリを登録する場合に、キャッシュにテクスチャがあれば、自動的にキャッシュからテクス

チャメモリへ登録します。

テキストチャ構造解説

- ・通常モデルには複数のテキストチャが貼られており、これらを一括して扱えるよう `texlist` 構造体を定義します。
- ・`texlist` 構造体は、複数のテキストチャファイル名の配列で構成されています。
- ・テキストチャファイルには `globalIndex` が格納されており、ロード時に `globalIndex` も取り込まれます。
- ・`globalIndex` のチャンクは、必ずテキストチャファイルの先頭に置きます。`globalIndex` はライブラリ性能向上に重要であるため必須です。
- ・`globalIndex` は Ninja グラフィックツールで管理され、プロジェクト全体を通して重複がない番号が割り振られます。
- ・同一テキストチャを含む複数の `texlist` のメモリ読み込み時の重複を `globalIndex` で検出します。
- ・モデルコンバート時は、オブジェクトツリー全体で使われるテキストチャ群が、各テキストチャファイルと一つの `texlist` として出力されます。
- ・モデルコンバートを繰り返す場合、出現したテキストチャがすでに `globalIndex` を割り振られた履歴があれば、その `globalIndex` が再度割り付けられます。
- ・ユーザーが独自に `globalIndex` を割り付けたい場合、使われる全テキストチャのテーブルを作成し、そのエントリ番号をすべてのテキストチャファイルの `globalIndex` チャンクに書き込みます。
- ・モデルのテキストチャ情報は `texlist` のエントリ番号である `texId` のみにより表現されるため、`globalIndex` は直接関係しません。
- ・テキストチャとモデルの対応は、モデル描画の前に使用する `texlist` をカレントテキストチャとしてセットすることにより行われます。`texlist` を変更することにより、簡単にテキストチャの差し替えができます。
- ・`texaddr` には、テキストチャメモリ上のアドレスがカレントテキストチャ登録時に格納されます。これはライブラリで利用されます。
- ・`texlist` のテキストチャファイル名の記述において、テキストチャファイルのパスは決められたフォルダに存在するものとして記述しません。
- ・テキストチャの拡張子 `".pvr"` は、データ量削減のため省略します。
- ・`texlist` と `model`(オブジェクトツリー) の整合性は、ユーザーの責任において管理されます。ライブラリは性能優先とし、`texlist` 内のテキストチャ数とオブジェクトツリーで使われているテキストチャ数の不整合を検出しません。

2.3.3 Ninja アトリビュート

ここで定義されるアトリビュートは、NJS_MATERIAL の attrflags に使用されます。ポリゴン単位のアトリビュート attr は、常に NULL です。ポリゴン単位のアトリビュートは、Chunk Model で実現されます。

31-29	28-26	25	24	23	22	21	20	19	18-17	16-15	14-13	12	11-8	7	6-0
-------	-------	----	----	----	----	----	----	----	-------	-------	-------	----	------	---	-----

31-29:	SRC Alpha Instruction	(ブレンディングパラメータ、後述)
28-26:	DST Alpha Instruction	(ブレンディングパラメータ、後述)
25:	Ignore Lights	(光源有効/無視。1 の時無視。)
24:	Flat Shading	(フラットシェーディング ON/OFF)
23:	Double Side	(両面ポリゴン ON/OFF)
22:	Environment Mapping	(環境マッピング ON/OFF)
21:	Use Texture	(テクスチャ有効/無効。1 の時有効。)
20:	Use Alpha	(の有効/無効。1 の時有効。)
19:	Ignore Specular	(スペキュラの無視。1 の时无効。)
18-17:	Flip UV	(フリップ制御)
16-15:	Clamp UV	(クランプ制御)
14-13:	Filter-Mode	0 ... Point Sampled(hard spec) 1 ... Bilinear Filter(hard spec) 2 ... Tri-linear Filter(hard spec)
12:	Super-Sample Texture	(Anisotropic Filter ON/OFF)
11-8:	Mip-Map 'D' adjust	(ミップマップレンジの調整 16 段階、後述)
7:	Pick Status	(ピックされている状態を保存。)
6-0:	User Flags	(未使用)

- ・ビット列 25-21,19,14-13,7-0 はハードウェアでは別の意味を持ちます。この部分はライブラリでマスクされ、ハードで期待される制御値をライブラリが設定し、利用します。
- ・ ブレンディングパラメータ

ブレンディング機能では、2 つの RGBA 値、SRC と DST が下記のように合成され、その結果は DST に書き戻されます。

$$\text{DST} := \text{SRC} * \text{BlendFunction}(\text{SRC Alpha Instruction}) + \text{DST} * \text{BlendFunction}(\text{DST Alpha Instruction})$$

ここで、BlendFunction(Instruction) には、SRC / DST カラーとともに 3 ビットの Instruction が入力されます。そして、それぞれの RGBA に対して、4 つの 値によって重み付けされた係数値を戻します。

Instruction	Field Value	Values Returned
Zero	0	(0, 0, 0, 0)
One	1	(1, 1, 1, 1)
'Other' Colour	2	(OR, OG, OB, OA)
Inverse 'Other' Colour	3	(1 - OR, 1 - OG, 1 - OB, 1 - OA)
SRC Alpha	4	(SA, SA, SA, SA)
Inverse SRC Alpha	5	(1 - SA, 1 - SA, 1 - SA, 1 - SA)
DST Alpha	6	(DA, DA, DA, DA)
Inverse DST Alpha	7	(1 - DA, 1 - DA, 1 - DA, 1 - DA)

'Other Colour' と 'Inverse Other Colour' は、SRC インストラクションに指定されたときには DST のカラーが使用され、DST インストラクションに指定されたときには SRC のカラーが使用されることを意味します。

係数が求められて、SRC/DST それぞれとの乗算が行われたあと、加算が行われます。その際、オーバーフローのチェックと得られた結果のクランプが適宜行われます。

Filter Mode

Field Values	Filter Mode
0	Point Sampled
1	Bilinear Filter
2	Tri-linear
3	Reserved

Mip-Map 'D' adjust

MIPMAP の 'D' 値は描画エンジン内で計算されますが、Aliasing と Blurring の妥協点を見出すために強制的に微調整する場合があります。この調整は、計算された 'D' 値に、指定した調整値 (4 ビットの符号なし固定小数点値、小数部は 2 ビット) を掛け合わせることで行われます。

Example 'D' Adjust bit pattern	Equivalent value
00.00	Illegal(設定不可)
00.01	0.25
01.00	1.0
11.11	3.75

Ninja 用フラグを以下のように定義します。2 ビット以上の field 以外を FLAG という名前で括ります。その他の部分はそれぞれ定義します。UV の Flip と Clamp は 2 ビット field ですが、2 ビットをそれぞれ U、V 用と考えることができるので FLAG として扱います。これらを抽出するための各種マスクを多く定義します。

```

/* SRC Alpha Instr(31-29) */
#define NJD_SA_ZERO      (BIT_0)                /* 0 zero */
#define NJD_SA_ONE       (BIT_29)               /* 1 one */
#define NJD_SA_OTHER     (BIT_30)               /* 2 Other Color */
#define NJD_SA_INV_OTHER (BIT_30|BIT_29)        /* 3 Inverse Other Color */

```

```

#define NJD_SA_SRC          (BIT_31)                /* 4 SRC Alpha */
#define NJD_SA_INV_SRC      (BIT_31|BIT_29)          /* 5 Inverse SRC Alpha */
#define NJD_SA_DST          (BIT_31|BIT_30)          /* 6 DST Alpha */
#define NJD_SA_INV_DST      (BIT_31|BIT_30|BIT_29)    /* 7 Inverse DST Alpha */
#define NJD_SA_MASK         (BIT_31|BIT_30|BIT_29)    /* MASK */

/* DST Alpha Instr(31-29) */
#define NJD_DA_ZERO         (0)                     /* 0 zero */
#define NJD_DA_ONE          (BIT_26)                 /* 1 one */
#define NJD_DA_OTHER        (BIT_27)                 /* 2 Other Color */
#define NJD_DA_INV_OTHER    (BIT_27|BIT_26)           /* 3 Inverse Other Color */
#define NJD_DA_SRC          (BIT_28)                 /* 4 SRC Alpha */
#define NJD_DA_INV_SRC      (BIT_28|BIT_26)           /* 5 Inverse SRC Alpha */
#define NJD_DA_DST          (BIT_28|BIT_27)           /* 6 DST Alpha */
#define NJD_DA_INV_DST      (BIT_28|BIT_27|BIT_26)    /* 7 Inverse DST Alpha */
#define NJD_DA_MASK         (BIT_28|BIT_27|BIT_26)    /* MASK */

/* filter mode */
#define NJD_FILTER_POINT    (0)
#define NJD_FILTER_BILINEAR (BIT_13)
#define NJD_FILTER_TRILINEAR (BIT_14)
#define NJD_FILTER_BLEND    (BIT_14|BIT_13)
#define NJD_FILTER_MASK     (BIT_14|BIT_13)

/* Mip-Map 'D' adjust */
#define NJD_D_025           (BIT_8)                 /* 0.25 */
#define NJD_D_050           (BIT_9)                 /* 0.50 */
#define NJD_D_075           (BIT_9|BIT_8)            /* 0.75 */
#define NJD_D_100           (BIT_10)                 /* 1.00 */
#define NJD_D_125           (BIT_10|BIT_8)            /* 1.25 */
#define NJD_D_150           (BIT_10|BIT_9)            /* 1.50 */
#define NJD_D_175           (BIT_10|BIT_9|BIT_8)       /* 1.75 */
#define NJD_D_200           (BIT_11)                 /* 2.00 */
#define NJD_D_225           (BIT_11|BIT_8)            /* 2.25 */
#define NJD_D_250           (BIT_11|BIT_9)            /* 2.50 */
#define NJD_D_275           (BIT_11|BIT_9|BIT_8)       /* 2.75 */
#define NJD_D_300           (BIT_11|BIT_10)            /* 3.00 */
#define NJD_D_325           (BIT_11|BIT_10|BIT_8)      /* 3.25 */
#define NJD_D_350           (BIT_11|BIT_10|BIT_9)      /* 3.50 */
#define NJD_D_375           (BIT_11|BIT_10|BIT_9|BIT_8) /* 3.75 */
#define NJD_D_MASK          (BIT_11|BIT_10|BIT_9|BIT_8) /* MASK */

/* flags */
#define NJD_FLAG_IGNORE_LIGHT (BIT_25)
#define NJD_FLAG_USE_FLAT     (BIT_24)
#define NJD_FLAG_DOUBLE_SIDE  (BIT_23)
#define NJD_FLAG_USE_ENV      (BIT_22)
#define NJD_FLAG_USE_TEXTURE  (BIT_21)
#define NJD_FLAG_USE_ALPHA    (BIT_20)
#define NJD_FLAG_IGNORE_SPECULAR (BIT_19)
#define NJD_FLAG_FLIP_U       (BIT_18)
#define NJD_FLAG_FLIP_V       (BIT_17)
#define NJD_FLAG_CLAMP_U      (BIT_16)
#define NJD_FLAG_CLAMP_V      (BIT_15)
#define NJD_FLAG_USE_ANISOTROPIC (BIT_12)
#define NJD_FLAG_PICK         (BIT_7)

/* flip と clamp のマスク */
#define NJD_FLAG_FLIP_MASK (NJD_FLAG_FLIP_U| NJD_FLAG_FLIP_V)
#define NJD_FLAG_CLAMP_MASK (NJD_FLAG_CLAMP_U| NJD_FLAG_CLAMP_V)
/* ハードウェアに直接送れるFLAGのマスク */

```

✕

```

#define NJD_FLAG_HARD_MASK (NJD_FLAG_USE_ALPHA
    | NJD_FLAG_FLIP_MASK | NJD_FLAG_CLAMP_MASK
    | NJD_FLAG_USE_ANISOTROPIC)
/* ライブラリで評価される(ハードウェアに直接送らない)FLAGのマスク */
#define NJD_FLAG_SOFT_MASK ( NJD_FLAG_IGNORE_LIGHT
    | NJD_FLAG_USE_FLAT| NJD_FLAG_DOUBLE_SIDE
    | NJD_FLAG_USE_ENV| NJD_FLAG_USE_TEXTURE
    | NJD_FLAG_IGNORE_SPECULAR|NJD_FLAG_PICK)
/* FLAG全体のマスク */
#define NJD_FLAG_MASK (NJD_FLAG_HARD_MASK
    | NJD_FLAG_SOFT_MASK)
/* デフォルトのユーザーマスク */
#define NJD_DEFAULT_USER_MASK
    (BIT_6|BIT_5|BIT_4|BIT_3|BIT_2|BIT_1|BIT_0)
/* デフォルトのシステムマスク */
#define NJD_DEFAULT_SYS_MASK ~NJD_DEFAULT_USER_MASK
/* ハードにそのまま送られるフィールドのマスク */
#define NJD_SYS_HARD_MASK (NJD_SA_MASK|NJD_SD_MASK
    |NJD_FLAG_HARD_MASK|NJD_D_MASK)

```

2.3.4 テクスチャフォーマット

"pvr"フォーマットを使用します。コンバータは pvrconv です。モデルコンバータに組み込まれ自動的にモデルに使われているテクスチャが全部変換されます。コンバータはもとの画像の 値をチェックし、自動的に次の三つのフォーマットを切り替えて出力します。

がない場合： RGB565 で出力。
 がある場合： ARGB4444 で出力。
 が 0, 255 の二値の場合： ARGB1555 で出力。

またテクスチャが正方形の場合 twiddled 形式が、長方形の場合 rectangle 形式がコンバータで自動選択されます。

twiddled 形式

テクスチャのピクセルを、高速にメモリから読み出せる順番に並べ替えたテクスチャです。Mipmap を利用できます。また表示が高速です。

rectangle 形式

ピクセルの順番をイメージそのままとしているテクスチャです。表示が twiddled に比べ低速です。Mipmap が使用できないので注意してください。

バンプマッピング

バンプマッピングテクスチャはグレー階調で用意します。RGB カラー画像のバンプは扱えません。コンバータでハードウェアが期待するデータに変換します。

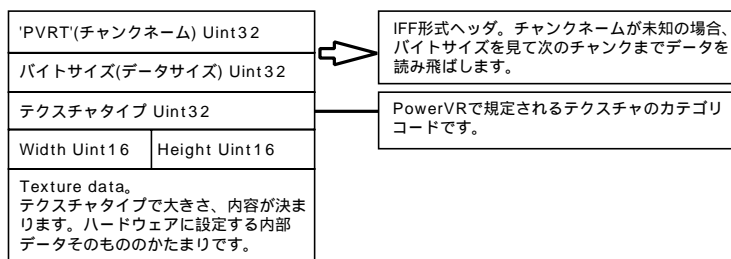
VQ

ベクトル量子化によりテクスチャを圧縮します。詳細は VQ の仕様書を参照してください。YUV422、パレットテクスチャは仕様検討中です。

テクスチャフォーマット

テクスチャフォーマット概要は次の通りです。IFF によるチャンク形式(ヘッダ+ サイズ+ データ)。データ部は、ハードウェアが期待する内部データ構造そのままになっています。詳細はここでは触れません。

チャンクの形式は次の通りです。



テクスチャタイプには、カラータイプとカテゴリコードの OR をとったビット列が設定されています。

```

/* カラータイプ */
#define NJD_TEXFMT_ARGB_1555      (0x00)
#define NJD_TEXFMT_RGB_565       (0x01)
#define NJD_TEXFMT_ARGB_4444     (0x02)
#define NJD_TEXFMT_YUV_422       (0x03)
#define NJD_TEXFMT_BUMP           (0x04)
#define NJD_TEXFMT_RGB_555       (0x05)
#define NJD_TEXFMT_COLOR_MASK     (0xFF)
/* カテゴリコード */
#define NJD_TEXFMT_TWIDDLED       (0x0100)
#define NJD_TEXFMT_TWIDDLED_MM   (0x0200)
#define NJD_TEXFMT_VQ             (0x0300)
#define NJD_TEXFMT_VQ_MM         (0x0400)
#define NJD_TEXFMT_PALETTIZE4     (0x0500)
#define NJD_TEXFMT_PALETTIZE4_MM  (0x0600)
#define NJD_TEXFMT_PALETTIZE8     (0x0700)
#define NJD_TEXFMT_PALETTIZE8_MM  (0x0800)
#define NJD_TEXFMT_RECTANGLE      (0x0900)
#define NJD_TEXFMT_STRIDE         (0x0B00)
#define NJD_TEXFMT_SMALLVQ        (0x1000)
#define NJD_TEXFMT_SMALLVQ_MM     (0x1100)
#define NJD_TEXFMT_TYPE_MASK      (0xFF00)

```

Ninja では、PVRT チャンク以外に GBIX, PVRI の 2 つのチャンクを定義します。

'GBIX'(チャンクネーム) Uint32
4 (byte) Uint32
globalIndex Uint32

テクスチャの globalIndex を記述する。Ninja で pvr ファイルを使う場合は、このチャンクがファイルの先頭にくるようにする。

'PVR!'(チャンクネーム) Uint32
バイトサイズ Uint32
Data。テクスチャ制御情報。

テクスチャ制御情報を格納します。詳細は検討中です。

3.1 概要

Ninja は Basic Model、Chunk Model の二つの形式のモデル構造をサポートしています。Chunk Model 形式は、描画実行中に SH4 のキャッシュを壊さないようにデータを連続するメモリ空間に配置します。拡張性、柔軟性、データの表現効率に優れています。今後は Chunk Model 形式のモデルを中心としたチューニングを実施します。Basic Model はサポートされますが、新機能はサポートされません。

Chunk Model 形式は Model 構造体の中身を大幅に変更しますが、Object 構造体は Model 構造体のポインタから Chunk Model 構造体のポインタへの変更以外に変更されません。モデル以外のモーションおよびテクスチャについては、引き続き現行の形式が使用されます(ただしカメラ、ライト対応のために構造体メンバの型が変更されます)。

Basic Model 及びテクスチャ構造については、『[Basic Model 仕様書](#)』を参照してください。Chunk Model の特徴を以下に示します。

Chunk Model の特徴

- ・トライアングルストリップ描画を基本とします。現在は三角形、四角形、N 角形を描画できません。性能優先で設計されています。
- ・データは、頂点リスト vlist とポリゴンリスト plist からなります。vlist、plist 上に IFF チャンク形式でデータを配置してメモリ領域を一本化し、描画実行中にキャッシュを壊さないようにしています。
- ・ポリゴン側と頂点側のどちらかに、頂点カラー情報を持つことができます。ポリゴン側では各ポリゴンごとに頂点に色が付けられます。
- ・ポリゴン側に頂点法線を持つことができます。頂点法線がポリゴン単位で持てるので、softimage の頂点法線をそのまま出力できます。Discontinuity データを出力できます。
- ・ポリゴン側 (最大 16bit × 3) と頂点側 (32bit × 1) にユーザーフラグ領域を持つことができます。現在は頂点カラーデータをユーザーフラグ領域へ出力するときに、この領域を使用します。今後この部分にユーザーデータを書き込めるツールを用意する予定です。
- ・マテリアルは plist に格納され、以前に設定したマテリアルの変更部分 (差分) のみの設定だけを更新します。これにより、Basic Model の場合よりもマテリアル設定回数が減少します。
- ・コンバータ出力時にマテリアルを削除できます。同じモデルの描画 (例えば木) などすべてのマテリアルをなくし、ユーザーが外で設定することでデータを最適化することができます。
- ・データ量削減のために 10 ビット法線がサポートされています。XYZ の法線が Uint32 の中

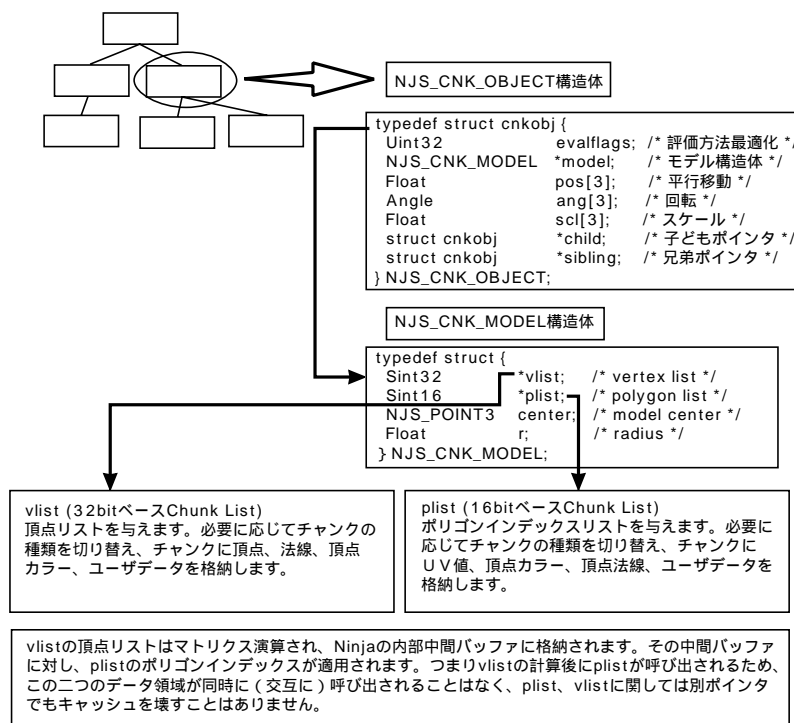
に各 10 ビットで格納されます。2 ビットはリザーブ領域として 0 を入れます。

- ・ コリジョン用データ Chunk Volume の出力がサポートされています。三角形、四角形、トライアングルストリップで出力できます。マテリアル情報を持ちません。ユーザーフラグ領域を持つことができます。現在この領域にマテリアルカラーを出力できます。
- ・ 独立三角形の Chunk Volume(volume3)は、モディファイアボリュームにも利用できます。
- ・ コンバータで volume34 を指定すると、面間角度が 0.1 度の三角形同士を接続し四角形を作ります。3D Studio MAX は三角形データしか出力できませんが、この場合でも四角形コリジョンが生成できます。
- ・ SH4 のハードウェアを効率良く利用し、高速処理を可能とする頂点形式がサポートされています(NJD_CV_SH、NJD_CV_VN_SH)。これは、SH4 のマトリックス演算命令を効果的に利用することによって、高速化を実現しています。データ量よりも性能優先の特殊な場合に利用します。
- ・ 二種類の UV 値表現を持ちます。0-255 による UVN、分解能を高めた 0-1023 による UVH です。UVN は、Basic Model でも利用していた従来の表現です。256 を超えるテクスチャでは、1 ピクセル単位で指定できません。UVH はハイレゾモード、1024x1024 のテクスチャで、1 ピクセル単位での指定が可能です。ただし UVH は UVN よりも分解能を上げた分だけ、テクスチャのリピートの表現回数が減少します(UVN が 128 回に対し UVH では 32 回)。コンバートオプションによりモデルツリー全体に対しまたマテリアルネームにより各モデル単位で UVN、UVH を切り替えることができます。マテリアルネームからの指定では単一のモデルに使われる複数のマテリアルのどれか一つに設定することでモデルに対する UV 表現が変更されます。デフォルトは UVN です。

3.2 モデル構造体

構造体図

Chunk Object Tree



構造体解説

Float, Angle

```

typedef float Float /* 浮動小数点演算型 */
typedef Sint32 Angle /* 回転角度 */
アングルは0x0000 ~ 0xFFFFが0 ~ 360度。
  
```

ポイント構造体

```

typedef struct {
    Float x; /* X値 */
    Float y; /* Y値 */
    Float z; /* Z値 */
} NJS_POINT3, NJS_VECTOR;
頂点のXYZ値を与えます。
  
```

チャンクモデル構造体

```
typedef struct {
    Sint32      *vlist;          /* 頂点チャンクリスト */
    Sint16      *plist;          /* ポリゴンチャンクリスト */
    NJS_POINT3   center;         /* モデルの中心 */
    Float        r;              /* モデルの半径 */
} NJS_CNK_MODEL;
```

- ・ vlist には頂点リストのデータを、Sint32 配列上に iff チャンク形式で並べます。plist にはポリゴンインデックスリストのデータを、Sint16 配列上に iff チャンク形式で並べます。center はモデルの外接球の中心位置であり、r はその半径です。

チャンクオブジェクト構造体

```
typedef struct cnkobj {
    Uint32      evalflags;       /* マトリクス演算評価フラグ */
    NJS_CNK_MODEL *model;        /* チャンクモデルポインタ */
    Float        pos[3];         /* 移動量 */
    Angle        ang[3];         /* 回転量 */
    Float        scl[3];         /* スケール */
    struct cnkobj *child;        /* 子供へのポインタ */
    struct cnkobj *sibling;      /* 兄弟ポインタ */
} NJS_CNK_OBJECT;
```

- ・ モデルの親子階層を与えます。evalflags はマトリクス演算最適化のためのフラグを与え、model にはチャンクモデル構造体ポインタがフックされます。ポリゴンを持たないノードでは、このポインタに NULL を設定します。pos は移動量、rot は回転量、scl はスケール、child、sibling は階層の子供、兄弟のオブジェクトポインタを与えます。

evalflags フラグ説明

```
#define NJD_EVAL_UNIT_POS    BIT_0 /* 移動が無視できる */
#define NJD_EVAL_UNIT_ANG    BIT_1 /* 回転が無視できる */
#define NJD_EVAL_UNIT_SCL    BIT_2 /* スケールが無視できる。 */
#define NJD_EVAL_HIDE        BIT_3 /* モデルを描画しない。 */
#define NJD_EVAL_BREAK       BIT_4 /* childのトレースの打ち切り */
#define NJD_EVAL_ZXY_ANG     BIT_5 /* LightWave3Dで期待される回転の評価の指定 */
#define NJD_EVAL_SKIP        BIT_6 /* モーションをスキップ */
#define NJD_EVAL_SHAPE_SKIP  BIT_7 /* シェイプモーションをスキップ */
#define NJD_EVAL_MASK        0xff /* 上記ビットを抽出するためのマスク */
```

これらのフラグはコンバータで設定されます。

- ・ NJD_EVAL_UNIT_POS は、平行移動量がゼロの場合に設定されます。平行移動のマトリクス計算が省略されます。
- ・ NJD_EVAL_UNIT_ANG は、回転量がゼロの場合に設定されます。回転のマトリクス計算が省略されます。

- ・ NJD_EVAL_UNIT_SCL は、スケールが xyz とともに 1 の場合に設定されます。スケールのマトリクス計算が省略されます。
- ・ NJD_EVAL_UNIT_POS, NJD_EVAL_UNIT_ANG, NJD_EVAL_UNIT_SCL の 3 つがセットされていると、すべてのマトリクス演算に加え、マトリクスの pushpop も省略されます。
- ・ NJD_EVAL_HIDE は、ユーザーによって立てられます。このフラグが立っている場合、モデルが描画されません。モデルに装着されたガンとブレードの切り替えなどに使用します。
- ・ NJD_EVAL_BREAK は、ユーザーによって立てられます。このフラグが立っている場合、child の探索をここで打ち切ります。例えば、ルートノードでこのフラグを立てると、モデル全体が消えます。NJD_EVAL_BREAK は、モーションと組み合わせるとデータがずれるので、ルートノードのみでを使用することをお勧めしますが、ユーザーの責任において途中のノードで使用することもできます。
- ・ LightWave3D では、回転の評価の順番は ZXY となります。Ninja は通常 XYZ の順番であるため、LightWave3D の場合の評価順をライブラリで実行するためのフラグとして、NJD_EVAL_ZXY_ANG が用意されます。このフラグが ON の場合、回転の計算順序が ZXY に変更されます。
- ・ NJD_EVAL_SKIP は、このノードがモーションデータに含まれないことを示します。モーション実行時はモーションのノードを次に進めずに、オブジェクト構造体の値でマトリクス計算を行い、次のノードに進みます。これにより、ポリゴンモデルの構成が異なるモデルでも、骨の構造が同じであればモーションが行えます。
- ・ NJD_EVAL_SHAPE_SKIP は、このノードがシェイプモーションデータに含まれないことを示します。
- ・ NJD_EVAL_SKIP、NJD_EVAL_SHAPE_SKIP はマテリアルネームからの指定が可能です。

3.3 Chunk 仕様

3.3.1 Chunk の種類

Chunk の名前	略号	サイズ	説明
Chunk NULL	CN	16bit	ロングワードアライメントあわせ。
Chunk End	CE	16bit	チャンクデータリストの終わりを示します。
Chunk Bits	CB	16bit	フラグを設定。Blend Alpha など。
Chunk Tiny	CT	32bit	フラグと単一の値を設定します。TexId など。
Chunk Material	CM	可変	Diffuse, Specular, Exponent, Ambient を設定します。
Chunk Vertex	CV	可変	頂点リストを与えます。
Chunk Volume	CO	可変	コリジョン、モディファイアボリューム用データを与えます。
Chunk Strip	CS	可変	ストリップデータを与えます。

基本構造をベースに、目的に合わせて構造を簡略化した Chunk(Bits, Tiny 等) を定義しています。頂点用の Chunk Vertex は Chunk Model 構造体の vlist に格納されます。それ以外のチャンクは plist に格納されます。これらは NinjaCnk.h に定義されています。

3.3.2 Chunk の構造

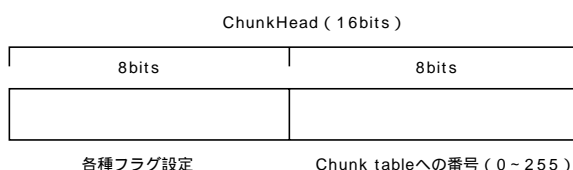
チャンクの基本構造は次の通りです。

Chunk Vertex の場合

[headbits(15-8)|ChunkHead(7-0)][longsize(15-0)][data]

Chunk Vertex 以外の場合

[headbits(15-8)|ChunkHead(7-0)][shortsize(15-0)][data]



ChunkHead は、そのチャンクを処理する関数 table のエントリ番号を与えます。ライブラリはこの番号から関数を選択し、実行します。チャンクごとに処理関数を分けることにより、描画ルーチンを単純化し、高速化ルーチンを実現することができます。テーブルサイズは最大 256 個とするため上位 8bit が余ります。この部分 (headbits と呼ぶ) 8bit へ、チャンクの種類により目的に合わせてアトリビュートフラグの一部を格納し、データサイズの効率を高めます。チャンクテーブルの関数エントリ番号を得るには、上位 8bit をマスクする必要があります。ことに注意してください。

shortsize、longsize は、次のチャンクの先頭までのオフセットを与えます。通常 iff 形式では次のチャンクまでのデータオフセットをバイトで与えますが対象とする plist が short の配列、vlist が long の配列であることからそれぞれ shortsize(2 バイト) 単位、longsize (4 バイト) 単位でオフセットを表現します。これは次のチャンクまでのオフセットの表現可能最大数を大きくする効果があります。

チャンク形式を利用しているため例えばユーザーがマテリアルだけを書き換えたい場合は、マテリアルのチャンクが見つかるまで shortsize を利用してデータをスキップし、マテリアルを発見したところで値を書き換えるようなことができます。

3.3.3 Chunk NULL

ChunkName : 'NJD_CN'

(Chunk NULL)

概要 :

plist においてロングワードアライメント調整のために、チャンクとチャンクの間に挿入されます。ChunkHead (16bit) のみのチャンク。

形式 :

```
[ChunkHead(15-0)]
  ChunkHead:
    NJD_CN
```

説明 :

```
#define NJD_CN (NJD_NULLOFF+0)
```

plist は Sint16 をベースとした一次元配列です。そのため、ストリップの終わりが必ずしも Sint32 バウンダリにならない場合があります。Chunk Material は Sint16 の配列上であっても、Sint32 のデータとして読み込んだ方が、効率が上がります。従って、Chunk Material の開始が Sint32 バウンダリになるように、Chunk Strip の後ろに付けられます。性能面からライブラリとの擦りあわせによりバウンダリあわせをするかどうかは、変更される可能性があります。バウンダリあわせが行われない場合、NJD_CN を使用しません。

3.3.4 Chunk End

ChunkName : 'NJD_CE'

(Chunk End)

概要 :

plist、vlist のチャンクリストの最後を与えます。ChunkHead (16bit) のみのチャンクです。vlist の場合は、ChunkHead (32bit) として扱われます。実際の値は最上位ビットに 1 が立つことにより検出されます。

形式 :

```
plistの場合  [ChunkHead(15-0)]( 16 bits chunk )
vlistの場合  [ChunkHead(31-0)]( 32 bits chunk )
ChunkHead:
    NJD_CE
```

説明 :

```
#define NJD_CE          (NJD_ENDOFF+0)
```

リストの完了を検出します。

3.3.5 Chunk Bits

Chunk Bits は、アトリビュートフラグのようなフラグを書き換えるために利用します。

```
[headbits(15-8)|ChunkHead(7-0)](16 bits chunk)
```

上位 8bit にフラグを格納し、下位 8bit でチャンクの番号を与えます。ChunkHead (16bit) のみのチャンクです。次に実際の Chunk Bits について説明します。

ChunkName : 'NJD_CB_BA'

(Chunk Bits Blend Alpha)

概要 :

plist においてアトリビュートフラグの Blend Alpha を設定します。

形式 :

```
[headbits(13-8) | ChunkHead(7-0)]
headbits:
    13-11 = SRC Alpha Instruction(3bit)
    10- 8 = DST Alpha Instruction(3bit)
ChunkHead:
    NJD_CB_BA
```

説明 :

```
#define NJD_CB_BA    (NJD_BITSOFF+0)
```

Blend Alpha は、2 種類の方法で設定されます。Chunk Material (後述) の headbits として、diffuse、specular、ambient の設定とともに Blend Alpha を設定できます。マテリアルを変更せず Blend Alpha のみ設定する場合に、NJD_CB_BA を利用します。

ブレンディング機能では、2 つの RBGA 値、SRC と DST が下記のように合成され、その結果は DST に書き戻されます。

```
DST := SRC * BlendFunction(SRC Alpha Instruction) +
      DST * BlendFunction(DST Alpha Instruction)
```

ここで、BlendFunction(Instruction) には、SRC/DST カラーとともに 3 ビットの Instruction が入力されます。そして、それぞれの RGBA に対して、4 つの 値によって重み付けされた係数値を戻します。

Instruction	Field Value	Values Returned
Zero	0	(0, 0, 0, 0)
One	1	(1, 1, 1, 1)
'Other' Colour	2	(OR, OG, OB, OA)
Inverse 'Other' Colour	3	(1 - OR, 1 - OG, 1 - OB, 1 - OA)
SRC Alpha	4	(SA, SA, SA, SA)
Inverse SRC Alpha	5	(1 - SA, 1 - SA, 1 - SA, 1 - SA)
DST Alpha	6	(DA, DA, DA, DA)
Inverse DST Alpha	7	(1 - DA, 1 - DA, 1 - DA, 1 - DA)

'Other Colour' と 'Inverse Other Colour' は、SRC インストラクションに指定されたときには DST のカラーが使用され、DST インストラクションに指定されたときには SRC のカラーが使用されることを意味します。

略号の意味は次の通りです。

ZER: Zero
ONE: One
OC: 'Other' Color
IOC: Inverse 'Other' Color
SA: Src Alpha
ISA: Inverse SRC Alpha
DA: DST Alpha
IDA: Inverse DST Alpha

Flag Blending Src :

```
#define NJD_FBS_SHIFT      11
#define NJD_FBS_ZER        (0<<NJD_FBS_SHIFT)
#define NJD_FBS_ONE        (1<<NJD_FBS_SHIFT)
#define NJD_FBS_OC         (2<<NJD_FBS_SHIFT)
#define NJD_FBS_IOC        (3<<NJD_FBS_SHIFT)
#define NJD_FBS_SA         (4<<NJD_FBS_SHIFT)
#define NJD_FBS_ISA        (5<<NJD_FBS_SHIFT)
#define NJD_FBS_DA         (6<<NJD_FBS_SHIFT)
#define NJD_FBS_IDA        (7<<NJD_FBS_SHIFT)

#define NJD_FBS_MASK        (0x7<<NJD_FBS_SHIFT)
```

Flag Blending Dst :

```
#define NJD_FBD_SHIFT      8
#define NJD_FBD_ZER        (0<<NJD_FBD_SHIFT)
#define NJD_FBD_ONE        (1<<NJD_FBD_SHIFT)
#define NJD_FBD_OC         (2<<NJD_FBD_SHIFT)
#define NJD_FBD_IOC        (3<<NJD_FBD_SHIFT)
#define NJD_FBD_SA         (4<<NJD_FBD_SHIFT)
#define NJD_FBD_ISA        (5<<NJD_FBD_SHIFT)
#define NJD_FBD_DA         (6<<NJD_FBD_SHIFT)
#define NJD_FBD_IDA        (7<<NJD_FBD_SHIFT)

#define NJD_FBD_MASK        (0x7<<NJD_FBD_SHIFT)
```

ChunkName : 'NJD_CB_DA'

(Chunk Bits 'D' Adjust)

概要 :

plist において Mipmap 'D' adjust 値を設定します。

形式 :

```
[headbits(11-8) | ChunkHead(7-0)]
headbits:
    11- 8 = Mipmap 'D' adjust(4)
ChunkHead:
    NJD_CB_DA
```

説明 :

```
#define NJD_CB_DA                (NJD_BITSOFF+1)
```

Mipmap の切り替えの深さを調整します。デフォルトは 1.00 です。これは頻繁に切り替えるものではなく、mipmap の切り替えが目立つ場合に、これを抑制する目的で利用します。

Flag 'D' Adjust :

```
#define NJD_FDA_SHIFT            8
#define NJD_FDA_025              (1<<NJD_FDA_SHIFT)    /* 0.25    */
#define NJD_FDA_050              (2<<NJD_FDA_SHIFT)    /* 0.50    */
#define NJD_FDA_075              (3<<NJD_FDA_SHIFT)    /* 0.75    */
#define NJD_FDA_100              (4<<NJD_FDA_SHIFT)    /* 1.00    */
#define NJD_FDA_125              (5<<NJD_FDA_SHIFT)    /* 1.25    */
#define NJD_FDA_150              (6<<NJD_FDA_SHIFT)    /* 1.50    */
#define NJD_FDA_175              (7<<NJD_FDA_SHIFT)    /* 1.75    */
#define NJD_FDA_200              (8<<NJD_FDA_SHIFT)    /* 2.00    */
#define NJD_FDA_225              (9<<NJD_FDA_SHIFT)    /* 2.25    */
#define NJD_FDA_250              (10<<NJD_FDA_SHIFT)   /* 2.25    */
#define NJD_FDA_275              (11<<NJD_FDA_SHIFT)   /* 2.25    */
#define NJD_FDA_300              (12<<NJD_FDA_SHIFT)   /* 3.00    */
#define NJD_FDA_325              (13<<NJD_FDA_SHIFT)   /* 3.25    */
#define NJD_FDA_350              (14<<NJD_FDA_SHIFT)   /* 3.50    */
#define NJD_FDA_375              (15<<NJD_FDA_SHIFT)   /* 3.75    */

#define NJD_FDA_MASK              (0xf<<NJD_FDA_SHIFT)
```

ChunkName : 'NJD_CB_EXP'

(Chunk Bits Exponent)

概要 :

plist においてスペキュラの Exponent を設定します。0~16 の間が有効です。

形式 :

```
[headbits(12-8) | ChunkHead(7-0)]
headbits:
    12- 8 = Exponent(5) range:0-16
ChunkHead:
    NJD_CB_EXP
```

説明 :

```
#define NJD_CB_EXP                (NJD_BITSOFF+2)
```

exponent は、2 種類の方法で設定されます。Chunk Material (後述) の中で specular を設定する場合は、Chunk Material の specular 成分の上位 8bit で exponent を設定します (ERGB8888 となる)。その前に設定した specular 値が変化せず exponent のみが変わった場合に限り、NJD_CB_EXP が使用されます。

Flag EXPonent(range 0-16) :

```
#define NJD_FEXP_SHIFT            8
#define NJD_FEXP_00                (0<<NJD_FEXP_SHIFT)    /* 0.0    */
#define NJD_FEXP_01                (1<<NJD_FEXP_SHIFT)    /* 1.0    */
#define NJD_FEXP_02                (2<<NJD_FEXP_SHIFT)    /* 2.0    */
#define NJD_FEXP_03                (3<<NJD_FEXP_SHIFT)    /* 3.0    */
#define NJD_FEXP_04                (4<<NJD_FEXP_SHIFT)    /* 4.0    */
#define NJD_FEXP_05                (5<<NJD_FEXP_SHIFT)    /* 5.0    */
#define NJD_FEXP_06                (6<<NJD_FEXP_SHIFT)    /* 6.0    */
#define NJD_FEXP_07                (7<<NJD_FEXP_SHIFT)    /* 7.0    */
#define NJD_FEXP_08                (8<<NJD_FEXP_SHIFT)    /* 8.0    */
#define NJD_FEXP_09                (9<<NJD_FEXP_SHIFT)    /* 9.0    */
#define NJD_FEXP_10                (10<<NJD_FEXP_SHIFT)   /* 10.0   */
#define NJD_FEXP_11                (11<<NJD_FEXP_SHIFT)   /* 11.0   */
#define NJD_FEXP_12                (12<<NJD_FEXP_SHIFT)   /* 12.0   */
#define NJD_FEXP_13                (13<<NJD_FEXP_SHIFT)   /* 13.0   */
#define NJD_FEXP_14                (14<<NJD_FEXP_SHIFT)   /* 14.0   */
#define NJD_FEXP_15                (15<<NJD_FEXP_SHIFT)   /* 15.0   */
#define NJD_FEXP_16                (16<<NJD_FEXP_SHIFT)   /* 16.0   */

#define NJD_FEXP_MASK              (0x1f<<NJD_FEXP_SHIFT)
```

3.3.6 Chunk Tiny

Chunk Tiny は、フラグと一つの値を設定するために利用します。これに対応するものとして TexId があります。形式は ChunkHead16bit と 16bit の定数の 32bit サイズ固定となります。

[headbits(15-8)|ChunkHead(7-0)][value(15-0)] (32 bits chunk)

現在定義されている Chunk Tiny には、TexId 及びテクスチャ関係のアトリビュートフラグを設定する NJD_CT_TID だけが定義されています。

ChunkName : 'NJD_CT_TID'

(Chunk Tiny TexId)

概要 :

plist において、TexList のエントリ番号である TexId を設定します。

形式 :

```
[headbits(15-8)|ChunkHead(7-0)][texbits(15-13)|TexId(12-0)]
headbits:
    15-14 = FlipUV(2)
    13-12 = ClampUV(2)
    11- 8 = Mipmap 'D' adjust(4)
ChunkHead:
    NJD_CT_TID
texbits:
    15-14 = Filter Mode(2)
    13    = Super Sample(1)
TexId:
    0 ~ 8191 (TexId Max = 8191)
```

説明 :

```
#define NJD_CT_TID (NJD_TINYOFF+0)
```

テクスチャ関連のアトリビュートフラグと TexId を設定します。テクスチャの切り替えタイミングでそれに関係するフラグを設定することにより、効率よくテクスチャの切り替えを行います。TexId に割り振られたビット数は 13bit のため、TexId の最大値は 8191 となる。

Flag FLip (headbits) :

```
#define NJD_FFL_U (BIT_15)
#define NJD_FFL_V (BIT_14)
```

UV 値のフリップを制御します。

Flag CLamp (headbits) :

```
#define NJD_FCL_U          (BIT_13)
#define NJD_FCL_V          (BIT_12)
```

UV 値のクランプを制御します。

Flag Filter Mode (texbits) :

PS : Point Sampled
BF : Bilinear Filter (default)
TF : Tri-liner Filter

```
#define NJD_FFM_SHIFT      14
#define NJD_FFM_PS         (0<<NJD_FFM_SHIFT)
#define NJD_FFM_BF         1<<NJD_FFM_SHIFT)
#define NJD_FFM_TF         (2<<NJD_FFM_SHIFT)

#define NJD_FFM_MASK        (0x3<<NJD_FFM_SHIFT)
```

フィルタリングモードを制御します。

Flag Super Sample (texbits) :

```
#define NJD_FSS             (BIT_13)
```

スーパーサンプリングを制御します。

```
#define NJD_TID_MASK        (~ (NJD_FSS | NJD_FFM_MASK))
```

3.3.7 Chunk Material

Chunk Material は、diffuse、specular、ambient を設定します。その前に設定されている値から変化があった差分だけ設定されます。Chunk Strip(後述) の headbits で specular 無視のフラグ (NJD_FST_IL) が設定されている場合、specular の設定は省略されます。ambient 無視のフラグ (NJD_FST_IA) が設定されていると ambient の設定が省略されます。headbits (上位 8bit) に Blend Alpha を設定します。この部分は NJD_CB_BA と等価です。Blend Alpha の詳細については、[NJD_CB_BA](#) を参照してください。specular の最上位 8bit は、exponent 設定に使用されます。この部分は NJD_CB_EXP と機能的には等価です。NJD_CB_EXP では、0~16 までのフラグビット (NJD_FEXP_*) を用意しこれに設定していたが、Chunk Material の specular では 0~16 の値を直接設定します。

```
[headbits(13-8)|ChunkHead(7-0)][shortsize(15-0)][Data]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_D、NJD_CM_A、NJD_CM_DA、NJD_CM_S、
    NJD_CM_DS、NJD_CM_AS、NJD_CM_DAS
Data:
    Diffulse = ARGB8888
    Specular = ERGB8888( E : exponent 0 ~ 16 )
    Ambient = NRGB8888( N : NOOP )
```

ChunkName : 'NJD_CM_D'

(Chunk Material Diffuse)

概要 :

plist において diffuse の値を設定します。diffuse の上位 8bit には、値が格納されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][2(shortsize)][ARGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_D
ARGB:
    ARGB8888
```

説明 :

```
#define NJD_CM_D (NJD_MATOFF+1)
```

Blend Alpha、diffuse のみを設定します。specular、ambient については、現在の設定値のままにします。

ChunkName : 'NJD_CM_A'

(Chunk Material Ambient)

概要 :

plist において ambient の値を設定します。ambient の上位 8bit は、ダミー (NOOP) として 255 が格納されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][2(shortsize)][NRGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_A
NRGB:
    NRGB8888( N : NOOP 255 )
```

説明 :

```
#define NJD_CM_A                (NJD_MATOFF+2)
```

Blend Alpha、ambient のみを設定します。diffuse、specular については現在の設定値のままにします。

ChunkName : 'NJD_CM_DA'

(Chunk Material Diffuse and Ambient)

概要 :

plist において diffuse、ambient の値を設定します。diffuse の上位 8bit には、値が設定されます。ambient の上位 8bit は、ダミー (NOOP) として 255 が格納されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][4(shortsize)][ARGB][NRGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_DA
ARGB:
    ARGB8888
NRGB:
    NRGB8888( N : NOOP 255 )
```

説明 :

```
#define NJD_CM_DA                (NJD_MATOFF+3)
```

Blend Alpha、diffuse、ambient を設定します。specular については、現在の設定値のままにします。

ChunkName : 'NJD_CM_S'

(Chunk Material Specular)

概要 :

plist において specular の値を設定します。specular の上位 8bit には、exponent (0 ~ 16) が設定されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][2(shortsize)][ERGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_S
ERGB:
    ERGB8888(E : Exponent 0 ~ 16)
```

説明 :

```
#define NJD_CM_S                (NJD_MATOFF+4)
```

Blend Alpha、specular のみを設定します。diffuse、ambient については、現在の設定値のままにします。

ChunkName : 'NJD_CM_DS'

(Chunk Material Diffuse and Specular)

概要 :

plist において diffuse、specular の値を設定します。diffuse の上位 8bit には、値が設定されます。specular の上位 8bit には、exponent (0 ~ 16) が設定されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][4(shortsize)][ARGB][ERGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_DS
ARGB:
    ARGB8888
ERGB:
    ERGB8888(E : Exponent 0 ~ 16)
```

説明 :

```
#define NJD_CM_DS                (NJD_MATOFF+5)
```

Blend Alpha、diffuse、specular を設定します。ambient については、現在の設定値のままにします。

ChunkName : 'NJD_CM_AS'

(Chunk Material Ambient and Specular)

概要 :

plist において diffuse、specular の値を設定します。diffuse の上位 8bit には、値が設定されます。specular の上位 8bit には、exponent (0 ~ 16) が設定されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][4(shortsize)][NRGB][ERGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_AS
NRGB:
    NRGB8888(N : NOOP 255)
ERGB:
    ERGB8888(E : Exponent 0 ~ 16)
```

説明 :

```
#define NJD_CM_AS                (NJD_MATOFF+6)
```

Blend Alpha、ambient、specular を設定します。diffuse については、現在の設定値のままにします。

ChunkName : 'NJD_CM_DAS'

(Chunk Material Diffuse, Ambient and Specular)

概要 :

plist において diffuse、specular の値を設定します。diffuse の上位 8bit には、値が設定されます。specular の上位 8bit には、exponent (0 ~ 16) が設定されます。

形式 :

```
[headbits(13-8)|ChunkHead(7-0)][6(shortsize)][ARGB][NRGB][ERGB]
headbits:
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_DAS
ARGB:
    ARGB8888
NRGB:
    NRGB8888(N : NOOP 255)
ERGB:
    ERGB8888(E : Exponent 0 ~ 16)
```

説明 :

```
#define NJD_CM_DAS                (NJD_MATOFF+7)
```

Blend Alpha、diffuse、ambient、specular を設定します。

3.3.8 Chunk Vertex

Chunk Vertex はモデルの頂点リストを与えます。頂点リストにユーザーの目的とするデータを格納するために、頂点リストもチャンク形式を利用します。チャンクの種類を切り替えることで、頂点、頂点法線、頂点カラー、ユーザーフラグ等のデータをケースバイケースで頂点リストに持たせることができます。ChunkHead の上位 8bit の部分 (headbits) は、Chunk Vertex に関して未使用です。モデルに複数のチャンクタイプを同時に使用することは、ライブラリ側の処理を複雑にするため現在のところ対応していません。一つのモデルに対しては、一種類の Chunk Vertex を利用します。Sint16 ベースでヘッダは構成されますが、格納する配列が

[ChunkHead(31-16)|longsize(15-0)]

[IndexOffset(31-16)|nbIndices(15-0)][Data]

ChunkHead:

< SH4 で高速な計算が可能な形式 >

NJD_CV_SH、NJD_CV_VN_SH

< 標準的な形式 (頂点法線なし) >

NJD_CV、NJD_CV_D8、NJD_CV_UF、

NJD_CV_NF、NJD_CV_S5、NJD_CV_S4、NJD_CV_IN

< 標準的な形式 (頂点法線あり) >

NJD_CV_VN、NJD_CV_VN_D8、NJD_CV_VN_UF、

NJD_CV_VN_NF、NJD_CV_VN_S5、NJD_CV_VN_S4、

NJD_CV_VN_IN

< 32bit 頂点法線 (x,y,z に各 10bit) 形式 >

NJD_CV_VNX、NJD_CV_VNX_D8、NJD_CV_VNX_UF

略号の意味は以下のとおりです。

VN : use vertex normal

VNX : 32bits vertex normal reserved(2)|x(10)|y(10)|z(10)

SH : SH4 optimize

D8 : Diffuse ARGB8888 only

S5 : Diffuse RGB565 and Specular RGB565

S4 : Diffuse RGB4444 and Specular RGB565

IN : Diffuse(16)|Specular(16)

NF : NinjaFlags32 for extention

UF : UserFlags32

IndexOffset は、ライブラリの頂点中間バッファの使用開始位置オフセットを与えます。例えば、オフセット 0 である親ノードの頂点を計算し、その子供が親の頂点数と等しいオフセットを指定して、その位置から頂点計算結果を格納したとすると、親の頂点計算結果は上書きされずに中間バッファに残ります。この状態で親の頂点インデックスまでさかのぼる番号を指定することにより、親と子の頂点を結んだポリゴンを表現することができます。

nbIndices は、チャンクに格納される頂点数を与えます。

ChunkName : 'NJD_CV_SH'

(Chunk Vertex for SH4 Optimize)

概要 :

vlist において頂点リストを定義します。頂点法線なし。SH4 のマトリクス演算命令の特性を考慮したデータ配置により、計算処理を高速化します。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]  
[IndexOffset(31-16)|nbIndices(15-0)] [Data]  
  ChunkHead:  
    NJD_CV_SH  
  longsize:  
    次のチャンクまでのオフセットです。  
  IndexOffset:  
    頂点中間バッファの開始位置を与えます。  
  nbIndices:  
    頂点の数を与えます。  
  Data:  
    x,y,z,1.0F, ...
```

説明 :

```
#define NJD_CV_SH (NJD_VERTOFF+0)
```

マトリックス演算命令に、128 ビット単位でデータを読み込むためのダミー 1.0F を x,y,z の後ろに挿入します。そのままマトリクス演算が可能のため、高速処理を実現できます（その効果はこれから検証する予定です）。頂点法線なし。光計算を行わず頂点カラーだけで描画するモデルなどでは、頂点法線は不要です。このような場合に使用します。

ChunkName : 'NJD_CV_VN_SH'

(Chunk Vertex VertexNormal for SH4 Optimize)

概要 :

vlist において頂点リストを定義します。頂点法線あり。SH4 のマトリクス演算命令の特性を考慮したデータ配置により、計算処理を高速化します。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_VN_SH
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,1.0F,nx,ny,nz,0.0F,...
```

説明 :

```
#define NJD_CV_VN_SH (NJD_VERTOFF+1)
```

マトリックス演算命令に、128 ビット単位でデータを読み込むためのダミー 1.0F を x,y,z の後ろに、0.0F を法線 nx,ny,nz の後ろに挿入します。そのままマトリクス演算が可能のため、高速処理が実現できます (その効果はこれから検証する予定です)

ChunkName : 'NJD_CV'

(Chunk Vertex)

概要 :

vlist において頂点リストを定義します。頂点法線なし。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z, ...
```

説明 :

```
#define NJD_CV (NJD_VERTOFF+2)
```

頂点リストを与えます。頂点法線なし。

ChunkName : 'NJD_CV_D8'

(Chunk Vertex Diffuse ARGB8888)

概要 :

vlist において頂点リストを定義します。頂点法線なし。頂点カラーあり。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_D8
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,D8888,...
```

説明 :

```
#define NJD_CV_D8                (NJD_VERTOFF+3)
```

頂点リストを与えます。頂点法線なし。頂点カラーあり。頂点カラーは Sint32 にパックされます。

ChunkName : 'NJD_CV_UF'

(Chunk Vertex UserFlag)

概要 :

vlist において頂点リストを定義します。頂点法線なし。ユーザーフラグ領域を 32bit 持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
ChunkHead:
    NJD_CV_UF
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,UserFlags32, ...
```

説明 :

```
#define NJD_CV_UF                (NJD_VERTOFF+4)
```

頂点リストを与えます。頂点法線なし。ユーザーフラグ領域を 32bit 持ちます。現在この領域に頂点カラーを出力できます。将来ツールからここにユーザーデータを書き込めるようにする予定です。

ChunkName : 'NJD_CV_NF'

(Chunk Vertex NinjaFlags32)

概要 :

vlist において頂点リストを定義します。頂点法線なし。Ninja 拡張フラグ領域を 32bit 持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_NF
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,NinjaFlags32, ...
```

説明 :

```
#define NJD_CV_NF                (NJD_VERTOFF+5)
```

頂点リストを与えます。頂点法線なし。Ninja 拡張フラグ領域を 32bit 持ちます。Ninja 機能拡張のためにリザーブされます。

ChunkName : 'NJD_CV_S5'

(Chunk Vertex Diffuse RGB565 and Specular RGB565)

概要 :

vlist において頂点リストを定義します。頂点法線なし。頂点カラーとして diffuse、specular を持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_S5
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,D565(31-16)|S565(15-0),...
```

説明 :

```
#define NJD_CV_S5 (NJD_VERTOFF+6)
```

頂点リストを与えます。頂点法線なし。頂点カラーあり。diffuse と specular を設定できます。表現効果向上の目的で、頂点カラーには specular が用意されています。現在、specular 頂点カラーの設定方法は用意されていません。コンバート時にシーンの光源位置から各頂点カラーの diffuse、specular を計算し、設定できるようにする (pre light) 予定です。

ChunkName : 'NJD_CV_S4'

(Chunk Vertex Specular RGB565 and Diffuse ARGB4444)

概要 :

vlist において頂点リストを定義します。頂点法線なし。頂点カラーとして diffuse、specular を持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_S4
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,D4444(31-16)|S565(15-0),...
```

説明 :

```
#define NJD_CV_S4                (NJD_VERTOFF+7)
```

頂点リストを与えます。頂点法線なし。頂点カラーあり。diffuse と specular を設定できます。表現効果向上の目的で、頂点カラーには specular が用意されています。diffuse を ARGB4444 とし 値を設定できます。現在、specular 頂点カラーの設定方法は用意されていません。コンバート時にシーンの光源位置から各頂点カラーの diffuse、specular を計算し、設定できるようにする (pre light) 予定です。

ChunkName : 'NJD_CV_IN'

(Chunk Vertex INtensity Diffuse and Specular)

概要 :

vlist において頂点リストを定義します。頂点法線なし。処理の高速な Intensity モードにおいて頂点カラーを与えます。頂点カラーとして diffuse、specular を持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_IN
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,D16|S16,...
```

説明 :

```
#define NJD_CV_IN          (NJD_VERTOFF+8)
```

頂点リストを与えます。頂点法線なし。Intensity モードにおいて頂点カラーを与えます。表現効果向上の目的で、頂点カラーには specular が用意されています。Intensity モードでは、diffuse と specular は輝度値だけで与えられます。その値をそれぞれ 16bit レンジで D16、S16 に設定します。現在、specular 頂点カラーの設定方法は用意されていません。コンバート時にシーンの光源位置から各頂点カラーの diffuse、specular を計算し、設定できるようにする (pre light) 予定です。

ChunkName : 'NJD_CV_VN'

(Chunk Vertex VertexNormal)

概要 :

vlist において頂点リストを定義します。頂点法線あり。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_VN
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,nx,ny,nz, ...
```

説明 :

```
#define NJD_CV_VN (NJD_VERTOFF+9)
```

頂点リストを与えます。頂点法線あり。もっとも標準的な頂点リストであり頻繁に使用されます。

ChunkName : 'NJD_CV_VN_D8'

(Chunk Vertex VertexNormal and Diffuse ARGB8888)

概要 :

vlist において頂点リストを定義します。頂点法線あり。頂点カラーあり。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_VN_D8
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,nx,ny,nz,D8888,...
```

説明 :

```
#define NJD_CV_VN_D          (NJD_VERTOFF+10)
```

頂点リストを与えます。頂点法線あり。頂点カラーあり。頂点カラーは Sint32 にパックされます。

ChunkName : 'NJD_CV_VN_UF'

(Chunk Vertex VertexNormal and UserFlags32)

概要 :

vlist において頂点リストを定義します。頂点法線あり。ユーザーフラグ領域を 32bit 持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
  NJD_CV_VN_UF
longsize:
  次のチャンクまでのオフセットです。
IndexOffset:
  頂点中間バッファの開始位置を与えます。
nbIndices:
  頂点の数を与えます。
Data:
  x,y,z,nx,ny,nz,UserFlags32,...
```

説明 :

```
#define NJD_CV_VN_UF          (NJD_VERTOFF+11)
```

頂点リストを与えます。頂点法線あり。ユーザーフラグ領域を 32bit 持ちます。現在この領域に頂点カラーを出力できます。将来的には、ツールからここにユーザーデータを書き込めるようにする予定です。

ChunkName : 'NJD_CV_VN_NF'

(Chunk Vertex VertexNormal and NinjaFlags32)

概要 :

vlist において頂点リストを定義します。頂点法線あり。Ninja 拡張フラグ領域を 32bit 持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]  
[IndexOffset(31-16)|nbIndices(15-0)] [Data]  
  ChunkHead:  
    NJD_CV_VN_NF  
  longsize:  
    次のチャンクまでのオフセットです。  
  IndexOffset:  
    頂点中間バッファの開始位置を与えます。  
  nbIndices:  
    頂点の数を与えます。  
  Data:  
    x,y,z,nx,ny,nz,NinjaFlags32,...
```

説明 :

```
#define NJD_CV_VN_NF          (NJD_VERTOFF+12)
```

頂点リストを与えます。頂点法線あり。Ninja 拡張フラグ領域を 32bit 持ちます。Ninja 機能拡張のためにリザーブされます。

ChunkName : 'NJD_CV_VN_S5'

(Chunk Vertex VertexNormal, Diffuse RGB565 and Specular RGB565)

概要 :

vlist において頂点リストを定義します。頂点法線あり。頂点カラーとして diffuse、specular を持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_VN_S5
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,nx,ny,nz,D565(31-16)|S565(15-0),...
```

説明 :

```
#define NJD_CV_VN_S5          (NJD_VERTOFF+13)
```

頂点リストを与えます。頂点法線あり。頂点カラーあり。頂点カラーは Sint32 にパックされます。diffuse と specular を設定できます。表現効果向上の目的で、頂点カラーには specular が用意されています。現在、specular 頂点カラーの設定方法は用意されていません。コンバート時にシーンの光源位置から各頂点カラーの diffuse、specular を計算し、設定できるようにする (pre light) 予定です。

ChunkName : 'NJD_CV_VN_S4'

(Chunk Vertex VertexNormal, Specular RGB565 and Diffuse ARGB4444)

概要 :

vlist において頂点リストを定義します。頂点法線あり。頂点カラーとして diffuse、specular を持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_VN_S4
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,nx,ny,nz,D4444(31-16)|S565(15-0),...
```

説明 :

```
#define NJD_CV_VN_S4 (NJD_VERTOFF+14)
```

頂点リストを与えます。頂点法線あり。頂点カラーあり。頂点カラーは Sint32 にパックされます。diffuse と specular を設定できます。表現効果向上の目的で、頂点カラーには specular が用意されています。diffuse を ARGB4444 とし、値を設定できます。現在、specular 頂点カラーの設定方法は用意されていません。コンバート時にシーンの光源位置から各頂点カラーの diffuse、specular を計算し、設定できるようにする (pre light) 予定です。

ChunkName : 'NJD_CV_VN_IN'

(Chunk Vertex VertexNormal, INtensity Diffuse and Specular)

概要 :

vlist において頂点リストを定義します。頂点法線あり。処理の高速な Intensity モードにおいて頂点カラーを与えます。頂点カラーとして diffuse、specular を持ちます。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_VN_IN
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,nx,ny,nz,D16|S16,...
```

説明 :

```
#define NJD_CV_VN_IN (NJD_VERTOFF+15)
```

頂点リストを与えます。頂点カラーあり。頂点カラーは Sint32 にパックされます。Intensity モードにおいて頂点カラーを与えます。表現効果向上の目的で、頂点カラーには specular が用意されています。Intensity モードでは、diffuse と specular は輝度値だけで与えられます。その値をそれぞれ 16bit レンジで D16、S16 に設定します。現在、specular 頂点カラーの設定方法は用意されていません。コンバート時にシーンの光源位置から各頂点カラーの diffuse、specular を計算し、設定できるようにする (pre light) 予定です。

ChunkName : 'NJD_CV_VNX'

(Chunk Vertex VertexNormal 32bits(X))

概要 :

vlist において頂点リストを定義します。32bit 頂点法線あり。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_VNX
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,nxyz32, ...
```

説明 :

```
#define NJD_CV_VNX (NJD_VERTOFF+16)
```

頂点リストを与えます。32bit 頂点法線あり。頂点法線データを削減し、データ量を少なくします。x,y,z にそれぞれ 10bit を割り振ります。残り 2bit はリザーブされます。分解能は 1024 です。この分解能で求められた頂点法線で、グロー計算が行われます。

ChunkName : 'NJD_CV_VNX_D8'

(Chunk Vertex VertexNormal 32bits(X) and Diffuse ARGB8888)

概要 :

vlist において頂点リストを定義します。32bit 頂点法線あり。頂点カラーあり。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
ChunkHead:
    NJD_CV_VNX_D8
longsize:
    次のチャンクまでのオフセットです。
IndexOffset:
    頂点中間バッファの開始位置を与えます。
nbIndices:
    頂点の数を与えます。
Data:
    x,y,z,nxyz32,D8888,...
```

説明 :

```
#define NJD_CV_VNX_D8          (NJD_VERTOFF+17)
```

頂点リストを与えます。32bit 頂点法線あり。頂点法線データを削減し、データ量を少なくします。x,y,z にそれぞれ 10bit を割り振ります。残り 2bit はリザーブされます。分解能は 1024 です。この分解能で求められた頂点法線で、グロー計算が行われます。頂点カラーあり。頂点カラーは Sint32 にパックされます。

ChunkName : 'NJD_CV_VNX_UF'

(Chunk Vertex VertexNormal 32bits(X) and UserFlags32)

概要 :

vlist において頂点リストを定義します。32bit 頂点法線あり。ユーザーフラグ領域あり。

形式 :

```
[ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)] [Data]
  ChunkHead:
    NJD_CV_VNX_UF
  longsize:
    次のチャンクまでのオフセットです。
  IndexOffset:
    頂点中間バッファの開始位置を与えます。
  nbIndices:
    頂点の数を与えます。
  Data:
    x,y,z,nxyz32,UserFlags32,...
```

説明 :

```
#define NJD_CV_VNX_UF          (NJD_VERTOFF+18)
```

頂点リストを与えます。32bit 頂点法線あり。頂点法線データを削減し、データ量を少なくします。x,y,z にそれぞれ 10bit を割り振ります。残り 2bit はリザーブされます。分解能は 1024 です。この分解能で求められた頂点法線で、グロー計算が行われます。ユーザーフラグ領域を 32bit 持ちます。

3.3.9 Chunk Volume

Chunk Volume は、コリジョン、モディファイアボリューム用に用意されます。直接ライブラリで描画に使用されません。マテリアルデータは持ちません。現在 Chunk Volume は三種類用意されます。NJD_CO_P3 は独立三角形データから、また NJD_CO_P4 は独立四角形から構成されています。3D Studio MAX では、すべてデータが三角形データとして出力されますが NinjaExport では、オプションで面間角度が 0.1 度以下の面同士を接続して四角形を復元し、独立四角形として出力できます。これにより、3D Studio MAX においても四角形によるコリジョンを生成できます。また元データが三角形/四角形/N 角形混在データの場合に、コンパタオプションですべてを三角形に分割し、再び独立三角形、四角形に再構成することができます。この場合、データには N 角形がなくなり、plist には NJD_CO_P3 と NJD_CO_P4 が並びます。NJD_CO_ST は、トライアングルストリップによる Chunk Volume です。Chunk Volume は Chunk Strip(後述) のものと、等価なポリゴンに対するユーザーフラグ領域 (16, 32, 48bit) を持つことができます。またモデラーで設定されたマテリアルカラーを、各ポリゴンのユーザーフラグ領域に出力することができます。

NJD_CO_P3 の独立三角形 Chunk Volume は、モディファイアボリュームにそのまま利用できます (ただしモディファイアボリュームに使用するには、3D 空間的に閉じたボリュームにする必要があります。ハード仕様。モディファイアボリュームは独立三角形しか使用できないことに注意してください)。ChunkHead の上位 8bit の部分 (headbits) は、Chunk Volume に関して未使用です。

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14)|nbPolygon(13-0)][Data]
```

ユーザーフラグ領域は 16bit 単位 (plist が Sint16 の一次元配列のため) で扱われ、16、32、48bit の三つが選べます。この領域サイズの指定は、ポリゴン数を与える nbPolygon の最上位 2bit に割り振られた UserOffset で行われます。

UserFlags Offset:

```
#define NJD_UFO_SHIFT      14
#define NJD_UFO_0          (0<<NJD_UFO_SHIFT)
#define NJD_UFO_1          (1<<NJD_UFO_SHIFT)
#define NJD_UFO_2          (2<<NJD_UFO_SHIFT)
#define NJD_UFO_3          (3<<NJD_UFO_SHIFT)
#define NJD_UFO_MASK       (0x3<<NJD_UFO_SHIFT) /* 0 - 3 */
```

UserOffset=NJD_UFO_0 の時 UserFlags サイズは 0bit。
UserOffset=NJD_UFO_1 の時 UserFlags サイズは 16bit。
UserOffset=NJD_UFO_2 の時 UserFlags サイズは 32bit。
UserOffset=NJD_UFO_3 の時 UserFlags サイズは 48bit。

ChunkName : 'NJD_CO_P3'

(Chunk vOluMe Polygon3)

概要 :

plist において volume 用ポリゴンリストを定義します。直接描画には使用されず、コリジョン、モディファイアポリュームに使用されます。マテリアルは持ちませんが、ユーザーフラグ領域にポリゴンカラーを出力できます。

形式 :

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14)|nbPolygon(13-0)][Data]
ChunkHead:
  NJD_CO_P3
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbPolygon:
  ポリゴン数を与えます。
Data:
  index0, index1, index2, UserflagPoly0(*N),
  index3, index4, index5, UserflagPoly1(*N), ...
```

説明 :

```
#define NJD_CO_P3 (NJD_VOLOFF+0)
```

コリジョン、モディファイアポリューム用の独立三角形データです。マテリアルは持ちませんが、ユーザーフラグ領域にポリゴンカラーを出力できます。このポリゴンカラーは、マテリアルで設定されたものです。ポリゴンインデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。元データが三角形/四角形/N 角形の混在する場合でも、コンバータオプションですべてを独立三角形に分割し、NJD_CO_P3 に出力することができます。

ChunkName : 'NJD_CO_P4'

(Chunk vOluMe Polygon4)

概要 :

plist において volume 用ポリゴンリストを定義します。直接描画には使われずコリジョンに使用します。マテリアルは持ちませんが、ユーザーフラグ領域にポリゴンカラーを出力できます。

形式 :

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14)|nbPolygon(13-0)][Data]
ChunkHead:
  NJD_CO_P4
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbPolygon:
  ポリゴン数を与えます。
Data:
  index0, index1, index2, index3, UserflagPoly0(*N),
  index4, index5, index6, index7, UserflagPoly1(*N), ...
```

説明 :

```
#define NJD_CO_P4 (NJD_VOLOFF+1)
```

コリジョン用の独立四角形データです。マテリアルは持ちませんが、ユーザーフラグ領域にポリゴンカラーを出力できます。このポリゴンカラーはマテリアルで設定されたものです。3D Studio MAX ではオブジェクトカラーが利用されます。ポリゴンインデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。3D Studio MAX の場合すべてのデータが独立三角形で出力されますが、コンバータオプションで面間角度が 0.1 度以下の場合二つの三角形を連結して四角形を再現し、これにより独立四角形データを生成できます。

ChunkName : 'NJD_CO_ST'

(Chunk vOluMe Triangle STrip)

概要 :

plist において volume 用ポリゴンリストを定義します。直接描画には使用されず、コリジョンに使用されます。マテリアルは持ちませんが、ユーザーフラグ領域にポリゴンカラーを出力できます。

形式 :

```
[ChunkHead(15-0)][shortsize(15-0)]
  [UserOffset(15-14)|nbPolygon(13-0)][Data]
ChunkHead:
  NJD_CO_ST
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbPolygon:
  ポリゴン数を与えます。
Data:
  [flag(15)|len(14-0), i0, i1, i2, Userflag2(*N), i3, Userflag3(*N), ...]
```

説明 :

```
#define NJD_CO_ST (NJD_VOLOFF+2)
```

コリジョンとして使用されます。NJD_CO_P3 及び NJD_CO_P4 よりも、コリジョンデータサイズを小さくすることができます。ただしストリップでは、三角形が連結される方向はより効率良い方向に進むため、必ずしもユーザーの意図した方向へは進まないのに注意してください。flag は、ストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は、符号により右回り/左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。i? はポリゴン頂点インデックスを意味します。マテリアルは持ちませんが、ユーザーフラグ領域にポリゴンカラーを出力できます。このポリゴンカラーは、マテリアルで設定されたものです。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。元データが三角形/四角形/N 角形の混在する場合でも、コンバータは自動的にすべてを独立三角形に分割し、NJD_CO_ST に出力します。

3.3.10 Chunk Strip

Chunk Strip は、Chunk Vertex の頂点リストからライブラリ内に展開された頂点中間バッファの頂点エントリ番号指定により、トライアングルストリップを生成します。頂点カラー、頂点法線、ポリゴン単位のユーザーフラグ領域を持つことができます。ポリゴン側に頂点カラーを持つことにより、同一頂点でもポリゴン単位に頂点カラーを設定できます。ポリゴン側に頂点法線を持つことで、ポリゴン間のエッジを立たせることができます。softimage の Discontinuity に対応でき、softimage で設定された頂点法線をそのまま出力することができます。ユーザーフラグ領域には、マテリアルから設定されたポリゴンカラーを出力できます。Chunk Vertex と Chunk Strip の両方に頂点カラーを持つことはできません。Chunk Strip 側に頂点カラー出力が指定された場合、Chunk Vertex 側の頂点カラーは出力されません。また Chunk Vertex と Chunk Strip の両方に頂点法線を持つことはできません。Chunk Strip 側に頂点法線出力が指定された場合、Chunk Vertex 側の頂点法線は出力されません。

ChunkHead の上位 8bit の部分 (headbits) には、マテリアルから設定されるアトリビュートフラグ (ChunkFlags) が設定されます。

略号の意味は次のとおりです。

IL : Ignore light
IS : Ignore specular
IA : Ignore ambient
UA : Use alpha
DB : Double side
FL : Flat shading
ENV : Environment mapping

Flag STRip :

```
#define NJD_FST_SHIFT      8
#define NJD_FST_IL        (0x01<<NJD_FST_SHIFT)
#define NJD_FST_IS        (0x02<<NJD_FST_SHIFT)
#define NJD_FST_IA        (0x04<<NJD_FST_SHIFT)
#define NJD_FST_UA        (0x08<<NJD_FST_SHIFT)
#define NJD_FST_DB        (0x10<<NJD_FST_SHIFT)
#define NJD_FST_FL        (0x20<<NJD_FST_SHIFT)
#define NJD_FST_ENV        (0x40<<NJD_FST_SHIFT)
#define NJD_FST_MASK      (0xFF<<NJD_FST_SHIFT)
```

次に Chunk Strip のフォーマットを示します。ポリゴン単位の UserFlags が index2 以降に一つずつ挿入されていることに注意してください。これは三点目で始めて一つ目の三角形が生成されることによります。四点目以降は一点ごとに三角形が一つ生成されるため、一頂点ごとに頂点の後ろにユーザーフラグが配置されています。

ポリゴン頂点法線なし、頂点カラーなし、テクスチャなしの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),    index0(15-0),
 index1(15-0),
 index2, UserFlag2(*N), ...]
```

ポリゴン頂点法線なし、頂点カラーなし、テクスチャありの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),    index0(15-0), U0(15-0), V0(15-0),
 index1, U1, V1,
 index2, U2, V2, UserFlag2(*N), ... ]
```

ポリゴン頂点法線あり、頂点カラーなし、テクスチャなしの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),    index0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
 index1, vnx1, vny1, vny1,
 index2, vnx2, vny2, vnz2, UserFlag2(*N),
 index3, vnx2, vny2, vnz2, UserFlag3(*N), ... ]
```

ポリゴン頂点法線あり、頂点カラーなし、テクスチャありの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),
 index0(15-0), U0(15-0), V0(15-0),vnx0(15-0), vny0(15-0), vnz0(15-0),
 index1, U1, V1, vnx1, vny1, vny1,
 index2, U2, V2, vnx2, vny2, vnz2, UserFlag2(*N),
 index3, U3, V3, vnx3, vny3, vnz3, UserFlag3(*N), ... ]
```

ポリゴン頂点法線なし、頂点カラーあり、テクスチャなしの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),
 index0(15-0), AR0(15-0), GB0(15-0),
 index1, AR1, GB1,
 index2, AR2, GB2, UserFlag2(*N),
 index3, AR3, GB3, UserFlag3(*N), ... ]
```

ポリゴン頂点法線なし、頂点カラーあり、テクスチャありの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),
 index0(16), U0(16), V0(16), AR0(16), GB0(16),
 index1, U1, V1, AR1, GB1,
 index2, U2, V2, AR2, GB2, UserFlag2(*N), ... ]
```

二種類の UV 値表現を持ちます。0-255 による UVN、分解能を高めた 0-1023 による UVH です。UVN は、Basic Model でも利用していた従来の表現です。256 を超えるテクスチャでは、1 ピクセル単位で指定できません。UVH はハイレゾモード、1024x1024 のテクスチャで、1 ピクセル単位での指定が可能です。ただし UVH は UVN よりも分解能を上げた分だけ、テクスチャのリピートの表現回数が減少します (UVN が 128 回に対し UVH では 32 回)。コンバートオプションによりモデルツリー全体に対しまたマテリアルネームにより各モデル単位で UVN、UVH を切り替えることができます。マテリアルネームからの指定では単一のモデルに使われる複数のマテリアルのどれか一つに設定することでモデルに対する UV 表現が変更されます。デフォルトは UVN。

UVN : Normal type Uv (0-255)

UVH : Hiresolution type Uv (0-1023)

Chunk Volume と同様に、Chunk Strip でもユーザーフラグ領域は nbStrip の最上位 2bit に割り付けられた UserOffset で与えられます。16bit 単位 (plist が Sint16 の一次元配列のため) で扱われ、16、32、48bit の三つが選べます。

UserFlags Offset:

```
#define NJD_UFO_SHIFT          14
#define NJD_UFO_0              (0<<NJD_UFO_SHIFT)
#define NJD_UFO_1              (1<<NJD_UFO_SHIFT)
#define NJD_UFO_2              (2<<NJD_UFO_SHIFT)
#define NJD_UFO_3              (3<<NJD_UFO_SHIFT)
#define NJD_UFO_MASK           (0x3<<NJD_UFO_SHIFT) /* 0 - 3 */
```

UserOffset=NJD_UFO_0の時UserFlagsサイズは0bitです。
UserOffset=NJD_UFO_1の時UserFlagsサイズは16bitです。
UserOffset=NJD_UFO_2の時UserFlagsサイズは32bitです。
UserOffset=NJD_UFO_3の時UserFlagsサイズは48bitです。

ChunkName : 'NJD_CS'

(Chunk Strip)

概要 :

plist においてポリゴンリストを定義します。ポリゴン頂点法線なし、頂点カラーなし、テクスチャなし。

形式 :

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
  NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
  NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
  NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
  NJD_FST_ENV( 環境マッピング )。
ChunkHead:
  NJD_CS
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbStrip:
  ストリップの頂点数です。
Data:
  [flag(15)|len(14-0), index0(15-0),
                                index1(15-0),
                                index2, UserFlag2(*N), ...]
```

説明 :

```
#define NJD_CS                (NJD_STRIPOFF+0)
```

flag は、ストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は、符号により右回り/左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index?はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形/四角形/N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_UVN'

(Chunk Strip UVN)

概要 :

plist においてポリゴンリストを定義します。ポリゴン頂点法線なし。頂点カラーなし。テクスチャあり。UV 値は UVN(0-255) で与えられます。

形式 :

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
    NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
    NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
    NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
    NJD_FST_ENV( 環境マッピング )。
ChunkHead:
    NJD_CS_UVN
shortsize:
    次のチャンクまでのオフセットです。
UserOffset:
    ユーザーフラグ領域のサイズを与えます。
nbStrip:
    ストリップの頂点数です。
Data:
    [flag(15)|len(14-0), index0(15-0), U0(15-0), V0(15-0),
      index1, U1, V1,
      index2, U2, V2, UserFlag2(*N), ... ]
```

説明 :

```
#define NJD_CS_UVN                (NJD_STRIPOFF+1)
```

UV 値には UVN (0-255) が使用されます。flag は、ストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は、符号により右回り/左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数。index? はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形/四角形/N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_UVH'

(Chunk Strip UVH)

概要:

plist においてポリゴンリストを定義します。ポリゴン頂点法線なし。頂点カラーなし。テクスチャあり。UV 値は UVH(0-1023) で与えられます。

形式:

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
    NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
    NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
    NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
    NJD_FST_ENV( 環境マッピング )。
ChunkHead:
    NJD_CS_UVH
shortsize:
    次のチャンクまでのオフセットです。
UserOffset:
    ユーザーフラグ領域のサイズを与えます。
nbStrip:
    ストリップの頂点数です。
Data:
    [flag(15)|len(14-0), index0(15-0), U0(15-0), V0(15-0),
      index1, U1, V1,
      index2, U2, V2, UserFlag2(*N), ... ]
```

説明:

```
#define NJD_CS_UVH (NJD_STRIPOFF+2)
```

UV 値には UVH (0-1023) が使用されます。UVN に比べリピートの上限値が低くなっている (32 回) ことに注意してください。flag は、ストリップ先頭の三角形の回転方向 (右回り / 左回り) を与えます。ChunkModel は、符号により右回り / 左回りを切り替える。マイナス値の場合は右回りです。len は、ストリップに含まれる頂点数の数です。index? はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形 / 四角形 / N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_VN'

(Chunk Strip VertexNormal)

概要:

plist においてポリゴンリストを定義します。ポリゴン頂点法線あり。頂点カラーなし。テクスチャなし。

形式:

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
  NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
  NJD_FST_IA( アンビエント無視 )、 NJD_FST_UA( Use Alpha )、
  NJD_FST_DB( 両面 )、 NJD_FST_FL( フラットシェーディング )、
  NJD_FST_ENV( 環境マッピング )。
ChunkHead:
  NJD_CS_VN
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbStrip:
  ストリップの頂点数です。
Data:
  [flag(15)|len(14-0), index0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
    index1, vnx1, vny1, vny1,
    index2, vnx2, vny2, vnz2, UserFlag2(*N),
    index3, vnx2, vny2, vnz2, UserFlag3(*N), ... ]
```

説明:

```
#define NJD_CS_VN (NJD_STRIPOFF+3)
```

flag はストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は符号により右回り/左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index?はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形/四角形/N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_UVN_VN'

(Chunk Strip UVN VertexNormal)

概要:

plist においてポリゴンリストを定義します。ポリゴン頂点法線あり。頂点カラーなし。テクスチャあり。UV 値は UVN(0-255) で与えられます。

形式:

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
  NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
  NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
  NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
  NJD_FST_ENV( 環境マッピング )。
ChunkHead:
  NJD_CS_UVN_VN
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbStrip:
  ストリップの頂点数です。
Data:
  [flag(15)|len(14-0),
  index0(15-0), U0(15-0), V0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
  index1, U1, V1, vnx1, vny1, vny1,
  index2, U2, V2, vnx2, vny2, vnz2, UserFlag2(*N),
  index3, U3, V3, vnx3, vny3, vnz3, UserFlag3(*N), ... ]
```

説明:

```
#define NJD_CS_UVN_VN (NJD_STRIPOFF+4)
```

UV 値には UVN(0-255) が使用されます。flag はストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は符号により右回り/左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index?はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形/四角形/N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_UVH_VN'

(Chunk Strip UVH VertexNormal)

概要:

plist においてポリゴンリストを定義します。ポリゴン頂点法線あり。頂点カラーなし。テクスチャあり。UV 値は UVH(0-1023) で与えられます。

形式:

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
  NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
  NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
  NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
  NJD_FST_ENV( 環境マッピング )。
ChunkHead:
  NJD_CS_UVH_VN
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbStrip:
  ストリップの頂点数です。
Data:
  [flag(15)|len(14-0),
  index0(15-0), U0(15-0), V0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
  index1, U1, V1, vnx1, vny1, vnz1,
  index2, U2, V2, vnx2, vny2, vnz2, UserFlag2(*N),
  index3, U3, V3, vnx3, vny3, vnz3, UserFlag3(*N), ... ]
```

説明:

```
#define NJD_CS_UVH_VN (NJD_STRIPOFF+5)
```

UV 値には UVH (0-1023) が使用されます。UVN に比べリピートの上限値が低くなっている (32 回) ことに注意してください。flag は、ストリップ先頭の三角形の回転方向 (右回り / 左回り) を与えます。ChunkModel は符号により右回り / 左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index? はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形 / 四角形 / N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_D8'

(Chunk Strip Diffuse ARGB8888)

概要:

plist においてポリゴンリストを定義します。ポリゴン頂点法線なし。頂点カラーあり。テクスチャなし。

形式:

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
  NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
  NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
  NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
  NJD_FST_ENV( 環境マッピング )。
ChunkHead:
  NJD_CS_D8
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbStrip:
  ストリップの頂点数です。
Data:
  [flag(15)|len(14-0),
  index0(15-0), AR0(15-0), GB0(15-0),
  index1, AR1, GB1,
  index2, AR2, GB2, UserFlag2(*N),
  index3, AR3, GB3, UserFlag3(*N), ... ]
```

説明:

```
#define NJD_CS_D8 (NJD_STRIPOFF+6)
```

flag は、ストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は、符号により右回り/左回りを切り替える。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index?はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形/四角形/N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_UVN_D8'

(Chunk Strip UVN Diffuse ARGB8888)

概要:

plist においてポリゴンリストを定義します。ポリゴン頂点法線なし。頂点カラーあり。テクスチャあり。UV 値は UVN(0-255) で与えられます。

形式:

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
    NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
    NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
    NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
    NJD_FST_ENV( 環境マッピング )。
ChunkHead:
    NJD_CS_UVN_D8
shortsize:
    次のチャンクまでのオフセットです。
UserOffset:
    ユーザーフラグ領域のサイズを与えます。
nbStrip:
    ストリップの頂点数です。
Data:
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
    [flag(15)|len(14-0),
    index0(16), U0(16), V0(16), AR0(16), GB0(16),
    index1, U1, V1, AR1, GB1,
    index2, U2, V2, AR2, GB2, UserFlag2(*N), ... ]
```

説明:

```
#define NJD_CS_UVN_D8          (NJD_STRIPOFF+7)
```

UV 値には UVN (0-255) が使用されます。flag は、ストリップ先頭の三角形の回転方向 (右回り/左回り) を与えます。ChunkModel は、符号により右回り/左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index? はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリはユーザーフラグ領域に関して何も関与せずスキップします。三角形/四角形/N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

ChunkName : 'NJD_CS_UVH_D8'

(Chunk Strip UVH Diffuse ARGB8888)

概要 :

plist においてポリゴンリストを定義します。ポリゴン頂点法線なし。頂点カラーあり。テクスチャあり。UV 値は UVH(0-1023) で与えられます。

形式 :

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
ChunkFlags:
  NJD_FST_IL(光源無視)、NJD_FST_IS( スペキュラ無視 )、
  NJD_FST_IA( アンビエント無視 )、NJD_FST_UA( Use Alpha )、
  NJD_FST_DB( 両面 )、NJD_FST_FL( フラットシェーディング )、
  NJD_FST_ENV( 環境マッピング )。
ChunkHead:
  NJD_CS_UVH_D8
shortsize:
  次のチャンクまでのオフセットです。
UserOffset:
  ユーザーフラグ領域のサイズを与えます。
nbStrip:
  ストリップの頂点数です。
Data:
  [ChunkFlags(15-8)|ChunkHead(7-0)]
  [shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
  [flag(15)|len(14-0),
    index0(16), U0(16), V0(16), AR0(16), GB0(16),
    index1, U1, V1, AR1, GB1,
    index2, U2, V2, AR2, GB2, UserFlag2(*N), ... ]
```

説明 :

```
#define NJD_CS_UVH_D8          (NJD_STRIPOFF+8)
```

UV 値には UVH (0-1023) が使用されます。UVN に比べリピートの上限值が低くなっている (32 回) ことに注意してください。flag は、ストリップ先頭の三角形の回転方向 (右回り / 左回り) を与えます。ChunkModel は、符号により右回り / 左回りを切り替えます。マイナス値の場合は右回りです。len はストリップに含まれる頂点数の数です。index? はポリゴン頂点インデックスを意味します。ポリゴン頂点インデックスの後ろにユーザーフラグ領域を持ちます。このサイズは、UserOffset の値 (NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit) で決まります。ライブラリは、ユーザーフラグ領域に関して何も関与せずスキップします。三角形 / 四角形 / N 角形を自動的に独立三角形に分割し、ストリップに変換して出力します。

3.4 アスキー出力の注意事項

Chunk Model .nja ファイル出力に対する注意事項をまとめます。

vlist は Sint32 の配列のため、この中に float 値の頂点、頂点法線をそのまま入力できません。そのためアスキー出力では、ヘキサ表現になっています。もし float として値を確認したい場合は、コンバータオプションによりコメントとして float 値を出力できます。

Chunk Model では、フラグもすべて文字列で出力します。なるべく短くかつ一般に使われない文字列を心がけました。NJD_をはずした文字列がそのまま nja ファイルに利用されています。

詳細は NjDef.h を参照してください。

4.1 概要

Ninja はモデル、カメラ、ライトのモーションを同一構造体で定義します。

キーフレームを設定する単位にデータを配列化し、そのポインタテーブルでモーション全体を定義します。この方法により、必要箇所のみのモーションを持たせたり、パラメータごとにキーフレーム補完を行うことができます。また、カメラとライトのモーション共通部分を使い回すこともできます。

4.2 オブジェクト構造体

オブジェクト構造体は、child、sibling ポインタで他のオブジェクトを連結することにより、親子階層モデルを表現します。親子階層を child、sibling の順にトレースし、その順番でノードを一行に並べたモーションデータとオブジェクト構造体の pos、ang、scl の値を組み合わせて、階層モデルのモーションを実現します。evalflags はマトリクス計算を省略したり、モーション時のオプション動作を制御します。モデルのデータ構造に関しては、『[Basic Model 仕様書](#)』または『[Chunk Model 仕様書](#)』を参照してください。

Basic Object 構造体

```
typedef struct obj {
    Uint32          evalflags;      /* 評価方法の最適化フラグ */
    NJS_MODEL       *model;         /* モデル構造体ポインタ */
    Float           pos[3];         /* 平行移動 */
    Angle           ang[3];         /* 回転 */
    Float           scl[3];         /* スケール */
    struct obj       *child;         /* 子どもobjectへのポインタ */
    struct obj       *sibling;      /* 兄弟objectへのポインタ */
} NJS_OBJECT;
```

Chunk Object 構造体

```
typedef struct cnkobj {
    Uint32          evalflags;      /* 評価方法の最適化フラグ */
    NJS_CNK_MODEL   *model;         /* モデル構造体ポインタ */
    Float           pos[3];         /* 平行移動 */
    Angle           ang[3];         /* 回転 */
    Float           scl[3];         /* スケール */
    struct obj       *child;         /* 子どもobjectへのポインタ */
    struct obj       *sibling;      /* 兄弟objectへのポインタ */
} NJS_CNK_OBJECT;
```

- ・モデルの親子階層を与えます。
- ・model にはポリゴン、TRIMESH(連続ポリゴン) がセットされます。

evalflags フラグ説明

```
#define NJD_EVAL_UNIT_POS    BIT_0    /* 移動が無視できる */
#define NJD_EVAL_UNIT_ANG    BIT_1    /* 回転が無視できる */
#define NJD_EVAL_UNIT_SCL    BIT_2    /* スケールが無視できる。 */
#define NJD_EVAL_HIDE        BIT_3    /* モデルを描画しない。 */
#define NJD_EVAL_BREAK        BIT_4    /* childのトレースの打ち切り */
#define NJD_EVAL_ZXY_ANG      BIT_5    /* LightWave3Dで期待される回転の評価の指定 */
#define NJD_EVAL_SKIP         BIT_6    /* モーションをスキップ */
#define NJD_EVAL_SHAPE_SKIP   BIT_7    /* シェイプモーションをスキップ */
#define NJD_EVAL_MASK         0xff     /* 上記ビットを抽出するためのマスク */
```

これらのフラグはコンバータで設定されます。

- ・NJD_EVAL_UNIT_POS は、平行移動量がゼロの場合に設定されます。平行移動のマトリクス計算が省略されます。
- ・NJD_EVAL_UNIT_ANG は、回転量がゼロの場合に設定されます。回転のマトリクス計算が省略されます。
- ・NJD_EVAL_UNIT_SCL は、スケールが xyz とともに 1 の場合に設定されます。スケールのマトリクス計算が省略されます。
- ・NJD_EVAL_UNIT_POS, NJD_EVAL_UNIT_ANG, NJD_EVAL_UNIT_SCL の三つがセットされていると、すべてのマトリクス演算に加え、マトリクスの pushpop も省略されます。
- ・NJD_EVAL_HIDE フラグは、ユーザーによって立てられます。このフラグが立っている場合、モデルが描画されません。モデルに装着されたガンとブレードの切り替えなどに使用します。
- ・NJD_EVAL_BREAK フラグは、ユーザーによって立てられます。このフラグが立っている場合、child の探索をここで打ち切ります。例えば、ルートノードでこのフラグを立てることにより、モデル全体が消えます。NJD_EVAL_BREAK は、モーションと組み合わせるとデータがずれるので、ルートノードのみで使用することをお勧めしますが、ユーザーの責任において途中のノードで使用することもできます。
- ・LightWave3D では、回転の評価の順番が ZXY となります。Ninja は通常 XYZ の順番であるため、LightWave3D の場合の評価順をライブラリで実行するためのフラグとして、NJD_EVAL_ZXY_ANG が用意されます。このフラグが ON の場合、回転の計算順序が ZXY に変更されます。
- ・NJD_EVAL_SKIP は、このノードがモーションデータに含まれないことを示します。モーション実行時はモーションのノードを次に進めずに、オブジェクト構造体の値でマトリクス計算を行い、次のノードに進みます。これにより、ポリゴンモデルの構成が異なるモデルでも、骨の構造が同じであればモーションが行えます。

- ・NJD_EVAL_SHAPE_SKIP は、このノードがシェイプモーションデータに含まれないことを示します。

4.3 カメラ構造体

カメラの構造体は以下のとおりです。

モーションとして使用するパラメータは、ポジション、ベクトル、ロール、画角の4つです。

NJS_CAMERA 構造体

```
typedef struct{
    Float px, py, pz;           (カメラのポジション)
    Float vx, vy, vz;           (カメラの単位方向ベクトル[ローカルZ軸])
    Angle roll;                 (カメラのロール)
    Angle ang;                  (カメラの画角)
    Float n_clip;               (ニアクリップ)
    Float f_clip;               (ファークリップ)
    NJS_VECTOR lx ,ly ;         (カメラのローカルX,Y軸)
} NJS_CAMERA
```

4.4 ライト構造体

ライトの構造体は以下のとおりです。

モーションできるパラメータとして、スポットライト以外の場合はポジション、ベクトル、カラーの4つが、スポットライトの場合はこれに前方の限界値、後方の限界値、内側の限界角、外側の限界角が加わります。ライト構造体に関する詳しい説明は、『[ライトの設定方法](#)』を参照してください。

NJS_LIGHT 構造体

```
struct {
    NJS_MATRIX      mtrx;       ( 光源マトリクス )
    NJS_POINT3      pnt;        ( 光源のポジション )
    NJS_VECTOR      vctr;       ( 光源の単位方向ベクトル )
    BOOL            stat;       ( ステータス：光源使用・不使用 )
    Int              reserve;    ( リザーブ )
    NJS_LIGHT_CAL    ltcal;     ( ライト計算用構造体 )
    NJS_LIGHT_ATTR   attr;      ( アトリビュート構造体 )
} NJS_ LIGHT;
```

```
<stat>
#define NJD_LIGHT_ON      光を反映させる。
#define NJD_LIGHT_OFF     光を反映させない。
```

NJS_LIGHT_ATTR 構造体

```
struct {
    Int          lsrc;      ( 光源の種類 )
    Float        ispc;      ( スペキュラライトの強さ : 0から1 )
    Float        idif;      ( デフューズの強さ : 0から1 )
    Float        iamb;      ( アンビエントの強さ : 0から1 )
    Float        nrang;     ( 光の強度が最大である距離 : 前方の限界値 )
    Float        frang;     ( 光の強度をカットオフする距離 : 後方の限界値 )
    void*        func;      ( コールバック関数のポインタ )
    Angle        iang;      ( 光の強度が最大である角度 : 内側の限界角 )
    Angle        oang;      ( 光の強度をカットオフする角度 : 外側の限界角 )
    NJS_ARGB     argb;      ( 光の色 )
} NJS_LIGHT_ATTR
```

<lsrc>

光源の種類。

#define NJD_SPOT_LIGHT	スポットライト
#define NJD_DIR_LIGHT	平行光源
#define NJD_POINT_LIGHT	点光源
#define NJD_AMBIENT	アンビエント
#define NJD_SPEC_DIR	平行光源ハイライト
#define NJD_SPEC_POINT	点光源ハイライト
#define NJD_LAMBERT_DIR	平行光源ランバート
#define NJD_LAMBERT_POINT	点光源ランバート
#define NJD_PHONG_DIR	平行光源フォン
#define NJD_PHONG_POINT	点光源フォン
#define NJD_USER_LIGHT	ユーザー設定ライト
#define NJD_BLOCK_LIGHT	ブロックライト
等	

<ispc, idif, iamb>

光のバランス (以下の式による)

$\text{SPECULAR}(R,G,B) \times \text{ispc} + \text{DIFFUSE}(R,G,B) \times \text{idif} + \text{AMBIENT}(R,G,B) \times \text{iamb}$

ただし、上限 (下限) はクランプします。

<near, far>

NJD_POINT_LIGHT (点光源) NJD_SPOT_LIGHT (スポットライト)

等で指定される光の有効範囲 (距離)

nrang 光が上限値である距離の限界値。デフォルト値:1.f

frang 光の計算を行う距離の限界値。デフォルト値:65535.f

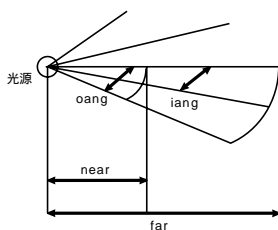
<iang, oang>

NJD_SPOT_LIGHT (スポットライト) 等で指定される光の有効範囲 (距離)

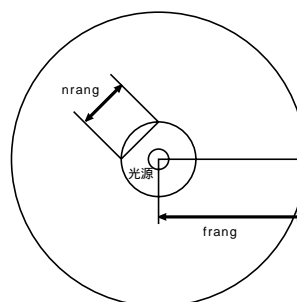
iang 光が上限値である角度の限界値。デフォルト値:(DEG)10.f

oang 光の計算を行う角度の限界値。デフォルト値:(DEG)30.f

(例:スポットライト)

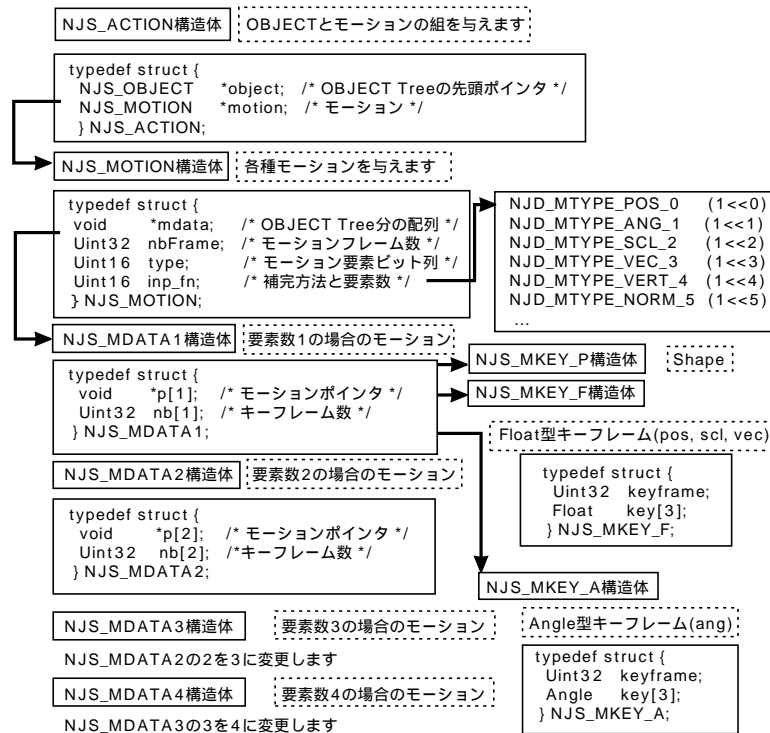


(例:点光源)



4.5 モーション構造体

構造体図



モデル用アクション構造体

```
typedef struct {
    NJS_OBJECT      *object;          /* OBJECT Treeの先頭ポインタ */
    NJS_MOTION      *motion;          /* モーションリスト          */
} NJS_ACTION;
```

- ・ object は親子階層を持つ tree 構造です。
- ・ motion には、object にしたい motion をセットします。

カメラ用アクション構造体

```
typedef struct {
    NJS_CAMERA      *camera;          /*カメラ構造体へのポインタ */
    NJS_MOTION      *motion;          /*モーションリスト          */
}NJS_CACTION
```

ライト用アクション構造体

```
typedef struct {
    NJS_LIGHT       *light;           /*ライト構造体へのポインタ */
    NJS_MOTION      *motion;          /*モーションリスト          */
}NJS_LACTION
```

モーション構造体

```
typedef struct {
    void            *mdata;           /* OBJECT Tree分の配列      */
    Uint32          nbFrame;          /* モーションフレーム数      */
    Uint16          type;             /* モーション要素ビット列    */
    Uint16          inp_fn;           /* 補完方法と要素数。        */
}NJS_MOTION;
```

- ・ mdata には、OBJECT Tree に含まれる全 NJS_OBJECT に対応する数の NJS_MDATA が配列として入ります。
- ・ NJS_MDATA では、モーション構成要素数により NJS_MDATA1～5 構造体をそれぞれ使用します。

```
#define NJD_MTYPE_POS_0      (1<<0)  /* NJS_MKEY_Fを利用          */
#define NJD_MTYPE_ANG_1      (1<<1)  /* NJS_MKEY_Aを利用          */
#define NJD_MTYPE_SCL_2      (1<<2)  /* NJS_MKEY_Fを利用          */
#define NJD_MTYPE_VEC_3      (1<<3)  /* NJS_MKEY_Fを利用          */
#define NJD_MTYPE_VERT_4     (1<<4)  /* NJS_MKEY_Pを利用          */
#define NJD_MTYPE_NORM_5     (1<<5)  /* NJS_MKEY_Pを利用          */
#define NJD_MTYPE_TARGET_3   (1<<6)  /* NJS_MKEY_Fを利用          */
#define NJD_MTYPE_ROLL_6     (1<<7)  /* NJS_MKEY_A1を利用         */
```



```

void          *p[4];          /* モーションポインタ          */
Uint32        nb[4];          /* キーフレーム数              */
} NJS_MDATA4;

typedef struct {
void          *p[5];          /* モーションポインタ          */
Uint32        nb[5];          /* キーフレーム数              */
} NJS_MDATA5;

```

キー構造体

```

typedef struct {
Uint32        keyframe;       /* キーフレーム番号            */
Float         key[3];         /* Float型キー値(配列3)        */
} NJS_MKEY_F;

```

- ・平行移動 (POS) スケール (SCL) ベクトル (VEC) に利用します。

```

typedef struct {
Uint32        keyframe;       /* キーフレーム番号            */
Angle         key[3];         /* Angle型キー値(配列)         */
} NJS_MKEY_A;

```

- ・回転 (ANG) に利用します。

```

typedef struct {
Uint32        keyframe;       /* キーフレーム番号            */
Void          *key;           /* ポインタ                     */
} NJS_MKEY_P;

```

- ・シェイプ (SHAPE) に利用します。

```

typedef struct {
Uint32        keyframe;       /* キーフレーム番号            */
Uint32        key;           /* 符号無しint32型キー値       */
} NJS_MKEY_UI32;

```

- ・ライト色 (RGB) に利用します。

```

typedef struct {
Uint 32       keyframe;       /* キーフレーム番号            */
Sint32        key;           /* 符号有りint32型キー値       */
} NJS_MKEY_A1;

```

- ・カメラロール (ROLL)、角度 (ANGLE) に利用します。

```

typedef struct {
Uint32        keyframe;       /* キーフレーム番号            */
Float         key;           /* Float型キー値                */
} NJS_MKEY_F1;

```

- ・ ライト強度 (INTENSITY) に利用します。

```
typedef struct {
    Uint32      keyframe;      /* キーフレーム番号          */
    Float       key[2];        /* Float型キー値(配列3)      */
} NJS_MKEY_F2;
```

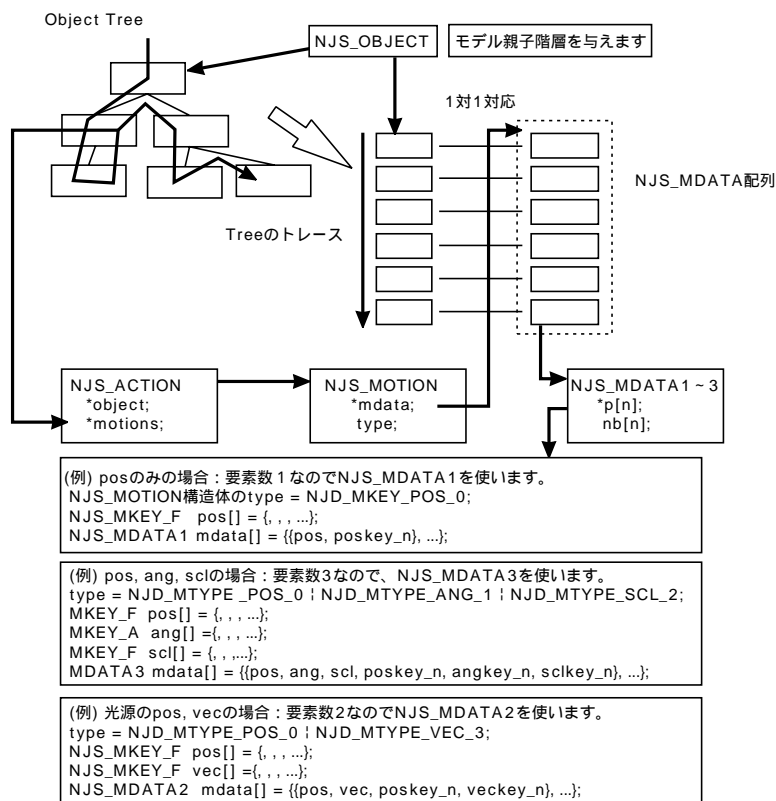
- ・ 点光源 (POINT) に利用。

```
typedef struct {
    Uint32      keyframe;      /* キーフレーム番号          */
    Float       nrang;         /* float型・前方の限界値のキー値 */
    Float       frang;         /* float型・後方の限界値キー値  */
    Angle       iang;          /* Angle型・内側の限界角キー値  */
    Angle       oang;          /* Angle型・外側の限界角キー値  */
} NJS_MKEY_SPOT;
```

- ・ スポットライト (SPOT) に利用します。

4.6 オブジェクトモーション

構造体関連図



構造解説

- ・ モーションはすべてキーフレームデータで与えられます。
- ・ ユーザーはキーフレームデータを線形補完、スプライン補完を使用することにより、モーションを実行します。
- ・ Ninja ライブラリでは、コールバック関数により補完方法をユーザー定義できます (現在未対応)。
- ・ キーフレーム番号はゼロから始まります。マイナス値は使用できません。

Position	平行移動
Angle	回転
Scale	拡大縮小
Vertex	ポリゴン頂点移動によるアニメーション (shape)。
Normal	ポリゴン頂点移動アニメーション (shape) の際の法線。

- ・ オブジェクトモーションの要素として、上記の五つを考えます。
- ・ ライブラリ実装面の問題から、shape データである Vertex、Normal について、Position、Angle、Scale (.nam) とは別データ (.nas) として出力します。このため最大要素数は三要素です。オブジェクトモーション用として、NJS_MDATA1～3 構造体が利用されます。光源、カメラ用に NJS_MDATA4、NJS_MDATA5 が定義されています。
- ・ NJS_MDATA の各要素を格納するポインタは void であり、各場合におけるデータの格納順番を規定する必要があります。

```
#define NJD_MTYPE_POS_0      (1<<0)      /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_ANG_1      (1<<1)      /* NJS_MKEY_Aを利用 */
#define NJD_MTYPE_SCL_2      (1<<2)      /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_VEC_3      (1<<3)      /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_VERT_4      (1<<4)      /*NJS_MKEY_Pを利用 */
#define NJD_MTYPE_NORM_5      (1<<5)      /*NJS_MKEY_Pを利用 */
```

- ・ define 文字列の最後に示す番号の若いデータが先に並びます。上記フラグはモーション構造体のメンバ type に設定されます。

```
( 例 )pos と ang の場合
type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1;
mdata[] = {pos, ang, ...}
```

- ・ モーション補完方法は、type の上位 2 ビットで指定されます。

```
#define NJD_MTYPE_LINER      0x0000
#define NJD_MTYPE_SPLINE     0x0040
#define NJD_MTYPE_USER       0x0080
#define NJD_MTYPE_MASK       0x00c0
```

- ・ NJD_MTYPE_LINER は線形補完です。
- ・ NJD_MTYPE_SPLINE はスプライン補完です。
- ・ NJD_MTYPE_USER はユーザー定義のルーチンによる補完です。
- ・ ルートが pos, ang で他は ang のみのモーションモデルでは、NJS_MDATA2 構造体を使用し、ルート以外の pos には NULL ポインタを使用することにより対応します。

```
type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1;  
NJS_MDATA2 mdata[] = {  
    { *pos1, *ang1 },  
    { NULL, *ang2 },  
    { NULL, *ang3 },  
    .... }
```

- ・ この場合、ang2, ang3 を左につめてはならないことに注意してください。

4.7 カメラモーション

カメラは親子階層を構成しないため、1 オブジェクトに対するモーション構造と基本的に同じです。

アクション構造体には NJS_CACTION を使用します。

カメラには

- ・ 位置 (POS)
- ・ 方向 (VEC) またはターゲット (TARGET)
- ・ ロール (ROLL)
- ・ 画角 (ANGLE)

の 4 要素があり、必要に応じて NJS_MTYPE_1 から NJS_MTYPE_4 を使用します。

```
#define NJD_MTYPE_POS_0      (1<<0)          /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_VEC_3      (1<<3)          /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_TARGET_3   (1<<6)          /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_ROLL_6     (1<<7)          /* NJS_MKEY_A1を利用 */
#define NJD_MTYPE_ANGLE_7    (1<<8)          /* NJS_MKEY_A1を利用 */
```

Vec はベクトル、Target はターゲットのポジションを意味します。

NJD_MTYPE_VEC_3 と NJD_MTYPE_TARGET_3 が同じ 3 なのは、同時に使えないことを示します。

- ・ フリーカメラ (向きをベクトルで持つカメラ) の場合

```
type=NJD_MTYPE_POS_0|NJD_MTYPE_VEC_3|NJD_MTYPE_ROLL_6|
NJD_MTYPE_ANGLE_7;
```

- ・ ターゲットカメラ (向きをターゲット位置で持ち画角アニメーションを持つカメラ) の場合

```
type=NJD_MTYPE_POS_0|NJD_MTYPE_TARGET_3|NJD_MTYPE_ROLL_6|
NJD_MTYPE_ANGLE_7;
```

- ・ 補完計算方法を inp_fn の上位 2 ビットで指定します。オブジェクトモーションと同じです。

4.8 ライトモーション

ライトは親子階層を構成しないため、1 オブジェクトに対するモーション構造と基本的に同じです。

アクション構造体は NJS_LACTION を使用します。

ライトのモーションの対象としては、点光源、平行光源、スポットライトを想定しています。

点光源には、

- ・ 位置 (POS)
- ・ 範囲 (POINT)
- ・ 色 (RGB)
- ・ 光の強度 (INTENSITY)

の4要素があります。

範囲の要素は、前方の限界値 (NearRange) 後方の限界値 (FarRange) をひとまとめにします。

平行光源には、

- ・ 位置 (POS)
- ・ 方向 (VEC) またはターゲット (TARGET)
- ・ 色 (RGB)
- ・ 光の強度 (INTENSITY)

の4要素があります。

スポットライトには、平行光源の4要素に前方の限界値 (near)、後方の限界値 (far)、内側の限界角 (iang)、外側の (oang) 限界角をひとまとめにした、

- ・ スポット (SPOT)

という要素を加えて、要素数は5つとなります。

よって、点光源、平行光源では NJS_MDATA_1 から NJS_MDATA_4 を使用し、スポットライトでは NJS_MDATA_1 から NJS_MDATA_5 を使用します。

```
#define NJD_MTYPE_POS_0          (1<<0)      /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_VEC_3          (1<<3)      /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_TARGET_3       (1<<6)      /* NJS_MKEY_Fを利用 */
#define NJD_MTYPE_RGB_8          (1<<9)      /* NJS_MKEY_UI32を利用 */
#define NJD_MTYPE_INTENSITY_9    (1<<10)     /* NJS_MKEY_F1を利用 */
#define NJD_MTYPE_SPOT_10        (1<<11)     /* NJS_MKEY_SPOTを利用 */
#define NJD_MTYPE_POINT_10       (1<<12)     /* NJS_MKEY_F2を利用 */
```

Vec はベクトル、Target はターゲットのポジションを意味します。

NJD_MTYPE_VEC_3 と NJD_MTYPE_TARGET_3 が同じ 3、NJD_MTYPE_SPOT_10 と NJD_MTYPE_POINT_10 が同じ 10なのは、同時に使用できないことを示します。

Type の設定例

- ・点光源

```
type=NJD_MTYPE_POS_0|NJD_MTYPE_RGB_8|NJD_MTYPE_INTENSITY_9|  
NJD_MTYPE_POINT_11 ;
```

- ・平行光源 (向きをベクトルで持つ平行光源) の場合

```
type=NJD_MTYPE_POS_0|NJD_MTYPE_VEC_3 |NJD_MTYPE_RGB_8|  
NJD_MTYPE_INTENSITY_9;
```

- ・スポットライト (向きをターゲットで持つスポットライト) の場合

```
type=NJD_MTYPE_POS_0|NJD_MTYPE_TARGET_3 |NJD_MTYPE_RGB_8|  
NJD_MTYPE_INTENSITY_9|NJD_MTYPE_SPOT_10 ;
```

補完計算方法を inp_fn の上位 2 ビットで指定します。オブジェクトモーションと同じです。

4.9 その他

ライトファイルの先頭には、

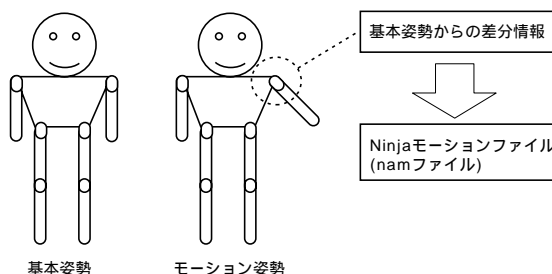
```
# if USE_LIGHT_ALIGN
# pragma USE_ALIGNDATA(LightName)
# endif
```

という記述がありますが、これは SH 用の記述で、アライメント調整を設定するための記述です。

Ninja モーションの 注意事項

5.1 Ninjaにおけるモーションの考え方

モーションは通常 1 つのモデルに対し複数用意されます。つまり 1 つのツリー階層モデルにモーションに必要なトランス、ローテーション、スケールを別に与えることによりモデルを動かします。Ninja ではモーションは基本姿勢からの差分として表現します。つまり動かす必要のある部分だけの情報でモーションを構成しています。基本姿勢は各モーションに含まれる不動の関節部分のパラメータを多く含んでいるとその差分情報が小さくなりますのでその点に注意して決定する必要があります。



Ninja には次の 3 つのモーション形式があります。データ量を少なくするためにこれらは必要に応じて使い分けます。

タイプ A(TRRR)	ルートノードのみトランスレーションとローテーションを持ち、他のノードはローテーションのみ。
タイプ B(TRTR)	各ノードがトランスレーションとローテーションを持つ。
タイプ C(TRS)	各ノードがトランスレーションとローテーションとスケールを持つ。

この他にシェイプモーション、また補完方法 (線形、スプライン等) の種類などの説明が必要であるがここでは省略します。またフォーマットの詳細などは別の資料を参照のこと。

それぞれタイプ A、B において省略されたパラメータの代わりに基本姿勢のモデルツリーデータの各ノードのトランスレーション、ローテーション、スケールが利用されます。したがって省略されたデータは一定であり基本姿勢のモデルツリーと値が一致する必要があります。また例えばタイプ A においてローテーションが省略される場合があります。つまり基本姿勢におけるノードのローテーションがあるモーションの間まったく変化せずその値が基本姿勢のままの場合これが省略されます。

以上の方法により Ninja はモーションデータを小さく押さえることを実現しています。

5.2 Ninjaにおけるモーション作成時の手順

- <step 1> 基本姿勢を決めます。
- <step 2> モデルをコンバートし nja ファイルと mrs ファイルを生成します。mrs ファイルとはモーションリソースを意味し、基本姿勢における親子階層、各ノードのトランスレーション、ローテーション、スケールの値を含む情報ファイルです。
- <step 3> 基本姿勢情報を含む mrs ファイルを利用しモーション nam ファイルに変換します。このモーションデータは単一の基本姿勢からの差分情報となります。

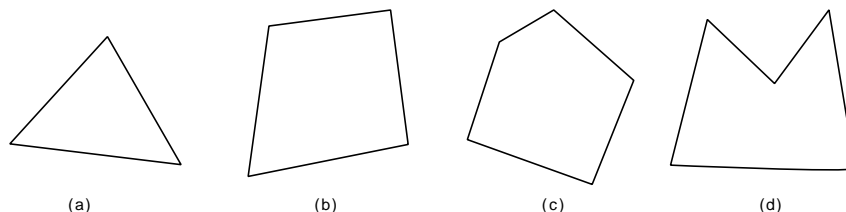
基本的にモデルとモーションはモデラー上では1シーンの中に格納されており同時に変換が可能です。これをするとそのシーンに含まれる姿勢を基本姿勢として差分が作られるため各モーション毎に基本姿勢が代わるため複数のモーションを1つの基本姿勢で動かすことができなくなりますので注意願います。モーションを必要とするモデルではまず基本姿勢のモデルを作りモーション出力のベースとなる mrs ファイルを作ること、またこれを使ってモーションを作ることを徹底してください。

注意

本ドキュメントは作成途中です。
現在のストリップ仕様およびフォーマットは暫定的なものです。

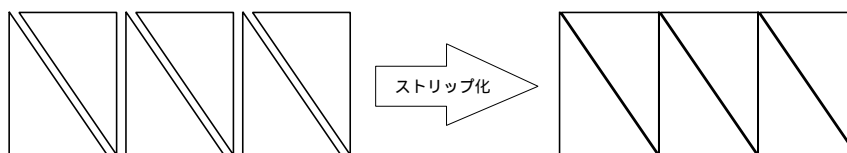
6.1 ストリップとは？

ストリップとは連続ポリゴンを意味します。従来ポリゴンとは 3 点以上の点で構成される面を意味します。



(d) の凹多角形ポリゴンはハードウェアの都合上使いません。

ストリップは隣接する複数のポリゴンをつなぎあわせ 1 つのデータとして扱う (三角形ポリゴンでこれをする場合トライアングルストリップと呼ぶ) ことでデータ/計算量の削減をし性能向上を実現します。また描画ハードウェアへのデータ転送量削減によるバス転送ネックの改善によりハードウェアのピーク性能を高める効果があります (ただしこれはハードウェアがストリップデータを処理する機能があることが前提です、今回のハードウェアはトライアングルストリップをサポートしています)。



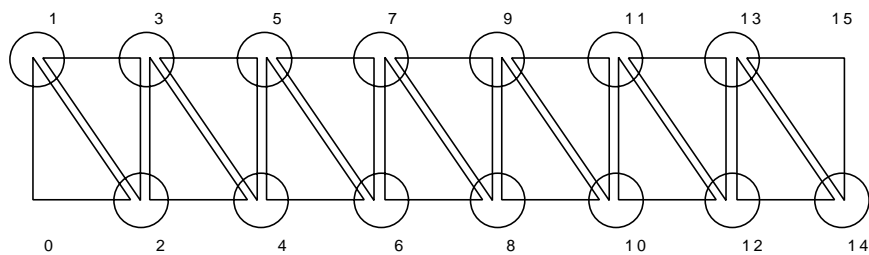
次になぜ独立三角形ポリゴンよりもデータ量が減るかを説明します。

四角形ポリゴンは 2 つの三角形ポリゴンからなると考えられます。また 2 つの三角形ポリゴンをつなげたストリップと考えることができます。三角形ポリゴン 1 つを表現するのに必要な頂点数は 3 つですので 2 つの三角形ポリゴンには 6 頂点が必要になります。一方四角形ポリゴンは 4 頂点で済みますので同じ四角形を表現するのに 2 頂点少なくすむことがわかり

ます。



これは2つの三角形ポリゴンの接続面にある2つの頂点が共有可能なために起きる現象です。ゲームで利用するポリゴンモデルも三角形ポリゴンを三次元的に敷き詰めてモデルを構成しており多くの共有可能な頂点を含みます。これを共有化しデータの表現効率を高めようというのがストリップです。ストリップ化によるデータ量の削減量は次のようになります。



頂点番号0、15以外は共有可能であることがわかります。また頂点番号2～13までは隣接する3つの三角形ポリゴンで共有されていることがわかります。つまりストリップでは最初と最後の頂点以外は1頂点の色計算結果を2～3回使いまわせることがわかります。その結果として計算量が減らせます。また2つ目の三角形以降は前の三角形の後の2つの頂点にもう1つ頂点を指定するだけで表現できることがわかります。N個の三角形ポリゴンをつないだストリップに必要な頂点数は次の式で表現されます。

$$\text{ストリップ頂点数} = 3(\text{最初の三角形頂点数}) + (N-1)$$

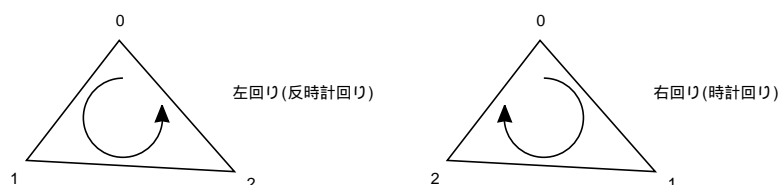
図4の例では14個の三角形ポリゴンをつないでますので

$$3 + (14 - 1) = 16(0 \text{ から } 15 \text{ 頂点まで } 16 \text{ 個})$$

となります。ここでわかることは最初の三角形以外はストリップでは三角形当たり1頂点で表現できている点です。つまりストリップが十分長ければほぼ三角形ポリゴン当たりのデータ量は1頂点と考えることができます。モデルを構成するすべての三角形ポリゴンが1列につながり十分長ければ独立三角形ポリゴン3頂点に対しほぼ1頂点になりますのでデータ量の削減率は1/3(33.333...)になります。これがトライアングルストリップのデータ量削減の理論限界値になります。

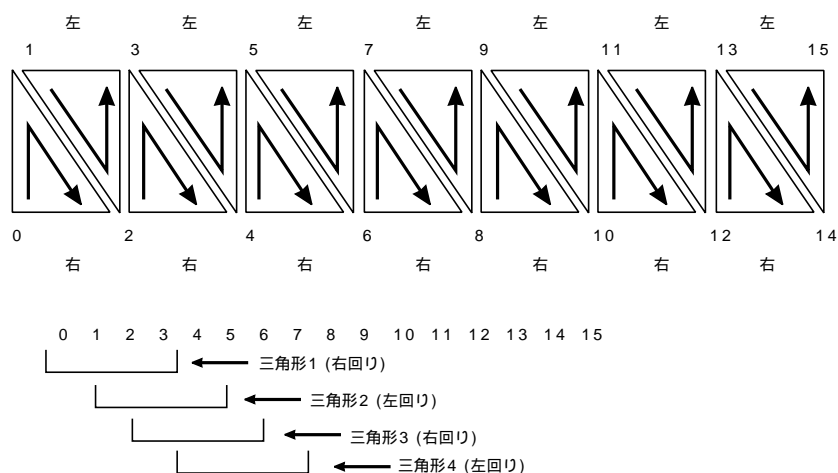
6.2 ストリップ頂点のつなぎ方

ポリゴンを表現する方法はポリゴンを構成する頂点を列記する方法で表現されます。表現の方法は右回り、左回りの2つが考えられます。

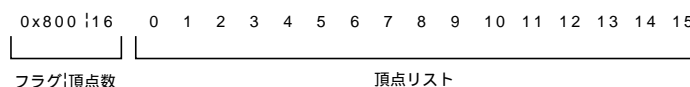


左回り、右回りは文化であり約束事として決めればどちらでも問題ありません。ここでは通常の独立ポリゴンは左回りに表現されるものとします。またこの回転の方向は三次元空間にポリゴンを配置した場合にこれの表面がどちらかを定めるために重要な役割を果たしています。つまりポリゴンの頂点をたどり左回りする面を表としそうでない側を裏としています。通常ポリゴンを裏から見ても見えません。視点から見てポリゴンが裏を向いている判定がされた場合カリング(描画リストからの削除)の対象となります。

ストリップでは1つ前のポリゴンの頂点を2つ使う必要から左回り、右回りが交互に発生します。ストリップでは交互に左回り、右回り側を表とみなして処理することになります。



Ninja ではストリップを少しでも長くするために開始ポリゴンの回り方向を左回り、右回り両方にしています。右回りの場合はストリップの長さを与えるパラメータの最上部にフラグを立てて表現しています。



6.3 マテリアルとテクスチャ

図 4 での説明の通り、ストリップの頂点は隣接する三角形列により 3 回使われます。そのためには三角形で使われる頂点のすべての情報が同一である必要があります。頂点はグロー計算で使われる法線、マテリアル、テクスチャ、UV 値の情報を持ちますが隣接する三角形でこれらの情報が一致している共有可能頂点を含む三角形どおしのみストリップデータとして連結できます。

6.4 ストリップの表現方法の比較

ストリップリスト (ポリゴンリストも同様) には次の 2 つの形式が考えられます。

- ・ 頂点リストの対するインデックスタイプ
- ・ 頂点の直接表現構造

6.5 インデックス

6.5.1 インデックスタイプ構造

インデックスタイプは頂点リストとそのエントリ番号から構成されるストリップリストからなります。

インデックス 番号	頂点リスト	ストリップリスト
0	xyz nx ny nz u v ...	0, 1, 2, 3, ...
1	xyz nx ny nz u v ...	インデックス番号列記によるストリップ表現
2	xyz nx ny nz u v ...	
3	xyz nx ny nz u v ...	
	...	

メリット

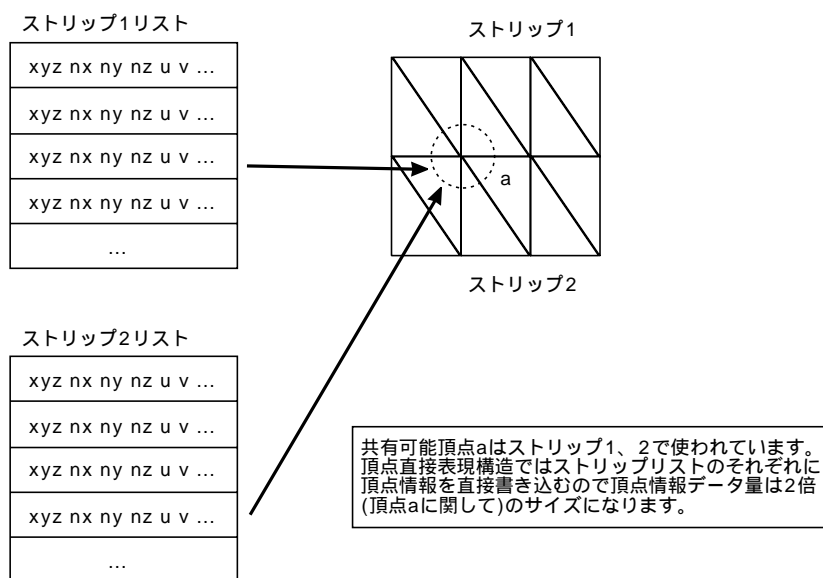
- ・ モデルに使われている頂点は重複なくリスト化されているので一度計算した結果をインデックスで引くことで重複計算なくストリップが表現できる。
- ・ 複数使われるデータはインデックスで表現されるため頂点を直接指定する場合よりもデータ量を押さえることができます。

デメリット

- ・ インデックスと頂点リストが別アドレスとなるので CPU キャッシュエラーを発生しやすい (ただしキャッシュエラーを極力押さえるように作ればこの問題は解決できます)。
Ninja ではインデックス形式のストリップを採用しています。

6.5.2 頂点直接表現構造

直接頂点データを列記してストリップリストを構成します。



直接頂点データを列記してストリップリストを構成します。頂点は複数のストリップに使用される場合それぞれのリストに利用回数だけ繰り返し登録されます。これはインデックスで同じ頂点を指定する場合よりもデータ量が大きくなります。1つの頂点の利用頻度は独立ポリゴンの場合1～10回程度(平均4回(推定))、ストリップで良くなつたデータモデルで1～3回程度(平均2～2.5回(推定))は使われると考えられますのでインデックスタイプのストリップの場合よりも(単純に直接表現にすると)データ量は大きくなります。このデータ量の大きさをカバーするためには頂点、法線データの表現ビット数を減らすなどの方法である程度データ量の増加を抑えるアプローチが必要となります。またインデックスの場合と異なり頂点は使われる度に再度3D計算をしなければなりません。4回使われる頂点はインデックスタイプの場合に比べ4倍の計算が発生します。

メリット

- ・データアドレスが連続するのでキャッシュエラーを回避できる。
- ・計算した後の頂点リストを格納するバッファがいらない。
- ・処理単位がポリゴン単位にできるので小さなプログラムを高速に実行できる。

デメリット

- ・データ量が大きくなる(単純に考えれば数倍)。ただしストリップ性能の向上、精度を下げる手法等によりある程度抑えられます。
- ・頂点アニメーションが表現しづらい。
- ・同じ頂点に対して3D計算を利用回数分だけする必要がある。

6.6 ストリップによるデータ削減率

実際ストリップ化によりどれくらいデータ量削減が期待できるかまとめます。それに対する注意点として以下の項目があります。

- ・ストリップ (ここではトライアングルストリップを意味する) は本来共有可能な頂点から構成される独立三角形ポリゴン群

View 関数について

7.1 初期化の方法

ビューの初期化は `njSetView()` 関数を実行する前に必ず済ませておかねばなりません。
初期化には次の 2 通りの方法があります。

7.1.1 `njInitView()` を使用する。

これによりビューは

現在の視点の位置	$(px, py, pz) = (0, 0, 0)$
現在の視点の向き	$(vx, vy, vz) = (0, 0, -1)$
現在の視点の傾き	(roll、視線の Z 軸に対する傾き) 0 度
基の視点の位置	$(apx, apy, apz) = (0, 0, 0)$
基の視点の向き	$(avx, avy, avz) = (0, 0, -1)$
基の視点の傾き	(aroll、視線の Z 軸に対する傾き) 0 度
ビューマトリクス	= 単位マトリクス

に設定されます。

7.1.2 VIEW 構造体のメンバーに直接値をセットする。

セットしなければならないのは、

相対的な操作を行う場合は、

- `px, py, pz`
- `vx, vy, vz`
- `roll`

絶対的な操作を行う場合は、

- `px, py, pz`
- `vx, vy, vz`
- `roll`

以上に直接値をセットした後、`void njSetBaseView(NJS_VIEW *v)` を実行。

もしくは、

- `apx, apy, apz`
- `avx, avy, avz`
- `aroll`

にもそれぞれ同じ値をセットします。
ビューマトリクスの設定は必要ありません。

注意 視線ベクトルは必ず単位ベクトル化してください。

7.2 ビューの移動、回転

すべての `nj*Relative()` 及び `nj*Absolute()` 関数は `njSetView()` 関数の前に実行されなければなりません。

例)

ビューの初期化。

```
.
.
while(1){
    njSetView();
    .
    .
    描画。
    .
    .<-----ここでnj*Relative() / nj*Absolute()を実行、次の視点へ。
    .
}
```

注意 プログラムの流れとして、必ず `nj*Relative()`、もしくは `nj*Absolute()` 関数が `njSetView()` 関数より先に実行されます。

7.3 旧 View 関数使用時の注意

```
njMultiViewMatrix
njRotateViewX
njRotateViewY
njRotateViewZ
njRotateViewXYZ
njTranslateView
njTranslateViewV
njUnitViewMatrix
```

以上の 8 関数はこれまでの Ninja との互換性を保つ為に残してある関数です。

将来ライブラリから削除される可能性がある為、新しくプログラムを組む場合には使用しないようお願いします。

また、やむなく使用する際には次の注意事項を守るようお願いします。

注意事項 1)

`njSetView()` 関数の後で実行する。

注意事項 2)

上記関数の実行後、必ず `njClearMatrix()` 関数を実行する。

7.3.1 注意事項 1) の詳細

旧 View 関数は VIEW 構造体のメンバー、`NJS_MATRIX m` を直接操作しますが、現在の View では

```
Float   px,py,pz;      // 現在の視点の位置
Float   vx,vy,vz;      // 現在の視点の向き(ベクトル)
Angle   roll;          // 現在の視線のZ軸に対する傾き
Float   apx,apy,apz;    // 基の視点の位置
Float   avx,avy,avz;    // 基の視点の向き(ベクトル)
Angle   aroll;         // 基の視線のZ軸に対する傾き
```

の各メンバーを操作し、メンバー、`NJS_MATRIX m` は `njSetView()` 関数によって更新されます。

従って、`njSetView()` 関数を実行する以前に旧 View 関数を用いて VIEW の操作を行ってもマトリクスは `njSetView()` 関数によって上書きされてしまいます。

7.3.2 注意事項 1) の詳細

VIEW は必ずマトリクススタックに反映されなければなりません。

この機能は現在 `njSetView()` 関数に組み込まれていますが、上記の理由から `njSetView()` 関数を用いて VIEW をマトリクススタックに反映させることは出来ません。

そのため旧 View 関数を使用する場合、旧 MATRIX 関数、`njClearMatrix()` の実行を省略することは出来ません。

7.3.3 旧ビュー関数の使用例（正しい例 / 誤った例）

正しい旧 View 関数の使用例

```
NJS_VIEW _view_;

njInitView(&_view_);
njSetView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
njClearMatrix();
```

誤った旧 View 関数の使用例 1

```
NJS_VIEW _view_;

njInitView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
njSetView(&_view_);
njClearMatrix();
```

`njSetView()` 関数によって VIEW のマトリクスが上書きされてしまう為、VIEW は初期化状態のままになってしまいます。

誤った旧 View 関数の使用例 2

```
NJS_VIEW _view_;

njInitView(&_view_);
njSetView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
```

`njTranslateView()` 関数を実行した結果がマトリクススタックに反映されていない為 VIEW は初期化状態のままになってしまいます。

```
/* VIEW 構造体 */
typedef struct {
    Float    px,py,pz;           // 現在の視点の位置
    Float    vx,vy,vz;           // 現在の視点の向き(ベクトル)
    Angle    roll;               // 現在の視線のZ軸に対する傾き
    Float    apx,apy,apz;        // 基の視点の位置
    Float    avx,avy,avz;        // 基の視点の向き(ベクトル)
    Angle    aroll;              // 基の視線のZ軸に対する傾き
    NJS_MATRIX m;               // ビューマトリクス
} NJS_VIEW;
```


8.1 LIGHT の設定方法

```
void njCreateLight(NJS_LIGHT*, Int)
```

パラメータ: NJS_LIGHT *ptr
 Int lsrc

説明: 光源の種類 lsrc を定め、ライト ptr を新たに登録します。

戻り値: なし

備考: なし

```
void njDeleteLight(NJS_LIGHT*)
```

パラメータ: NJS_LIGHT *ptr

説明: 設定されたライト ptr を削除します。

戻り値: なし

備考: なし

```
void njLightOff(NJS_LIGHT*)
```

パラメータ: NJS_LIGHT *ptr

説明: 設定されたライト ptr を反映させません。

戻り値: なし

備考: マクロで OK。

```
void njLightOn(NJS_LIGHT*)
```

パラメータ: NJS_LIGHT *ptr

説明: 設定されたライト ptr をモデルに反映させます。

戻り値: なし

備考: マクロで OK。

```
void njMultiLightMatrix(NJS_LIGHT*, NJS_MATRIX*)
```

パラメータ: NJS_LIGHT *ptr
 NJS_MATRIX*m

説明: njCreateLight によって登録されたライト行列に行列 m を掛けます。

戻り値: なし

備考: 行列 m にはスケールファクタが入ってはいらない。

void njSetLight(NJS_LIGHT*)

パラメータ: NJS_LIGHT *ptr

説明: すでにツール等で定められたライト ptr を新たに登録します。

戻り値: なし

備考: なし

void njSetLightAlpha(NJS_LIGHT*, Float)

パラメータ: NJS_LIGHT *ptr
Float alpha

説明: njCreateLight によって登録されたライトへアルファ値を設定します。

戻り値: なし

備考: 現在考慮中。

void njSetLightAngle(NJS_LIGHT*, NJS_Angle, NJS_Angle)

パラメータ: NJS_LIGHT *ptr
NJS_Angle iang
NJS_Angle oang

説明: njCreateLight によって登録されたライトへ角度限界値を設定します。

戻り値: なし

備考: スポットライトのみ使用。(現在)

void njSetLightColor(NJS_LIGHT*, Float, Float, Float)

パラメータ: NJS_LIGHT *ptr
Float red
Float green
Float blue

説明: njCreateLight によって登録されたライトへ RGB 値を設定します。

戻り値: なし

備考: なし

void njSetLightDirection(NJS_LIGHT*, Float, Float, Float)

パラメータ: NJS_LIGHT *ptr
Float dx
Float dy
Float dz

説明: njCreateLight によって登録されたライトの光源方向を設定します。

戻り値: なし

備考: 平行光源、スポットライトのみ使用。(現在)

void njSetLightIntensity(NJS_LIGHT*, Float, Float, Float)

パラメータ: NJS_LIGHT *ptr
Float spc
Float dif
Float amb

説明: njCreateLight によって登録されたライトの光の強度を設定します。

戻り値: なし

備考: なし

void njSetLightLocation(NJS_LIGHT*, Float, Float, Float)

パラメータ: NJS_LIGHT *ptr
Float px
Float py
Float pz

説明: njCreateLight によって登録されたライトの光源位置を設定します。

戻り値: なし

備考: 点光源、スポットライトのみ使用。(現在)

void njSetLightRange(NJS_LIGHT*, Float, Float)

パラメータ: NJS_LIGHT *ptr
Float nrang
Float frang

説明: njCreateLight によって登録されたライトへ距離限界値を設定します。

戻り値: なし

備考: スポットライト、点光源のみ使用。(現在)

void njSetUserLight(NJS_LIGHT*, NJF_LIGHT_FUNC*)

パラメータ: NJS_LIGHT *ptr
NJF_LIGHT_FUNC func

説明: ライト ptr にユーザー設定ライト関数 func を設定します。

戻り値: なし

備考: なし

void njUnitLightMatrix(NJS_LIGHT*)

パラメータ: NJS_LIGHT *ptr

説明: njCreateLight によって登録されたライト行列を単位行列に設定します。

戻り値: なし

備考: なし

void njTranslateLightV(NJS_LIGHT*, NJS_VECTOR*)
.....

パラメータ: NJS_LIGHT *ptr
NJS_VECOTR*vctr

説明: njCreateLight によって登録されたライト行列をベクトル vctr で平行移動します。

戻り値: なし

備考: なし

void njTranslateLight(NJS_LIGHT*, Float, Float, Float)
.....

パラメータ: NJS_LIGHT *ptr
Float tx
Float ty
Float tz

説明: njCreateLight によって登録されたライト行列を (tx, ty, tz) で平行移動します。

戻り値: なし

備考: なし

void njRotateLightX(NJS_LIGHT*, NJS_Angle)
.....

パラメータ: NJS_LIGHT *ptr
NJS_Angle ang

説明: njCreateLight によって登録されたライト行列を X 軸で角度 ang 回転します。

戻り値: なし

備考: なし

void njRotateLightXYZ(NJS_LIGHT*, NJS_Angle, NJS_Angle, NJS_Angle)
.....

パラメータ: NJS_LIGHT *ptr
NJS_Angle xang
NJS_Angle yang
NJS_Angle zang

説明: njCreateLight によって登録されたライト行列を XYZ 軸で回転します。

戻り値: なし

備考: なし

void njRotateLightY(NJS_LIGHT*, NJS_Angle)
.....

パラメータ: NJS_LIGHT *ptr
NJS_Angle ang

説明: njCreateLight によって登録されたライト行列を Y 軸で角度 ang 回転します。

戻り値: なし

備考: なし

```
void njRotateLightZ(NJS_LIGHT*, NJS_Angle)
```

パラメータ: NJS_LIGHT *ptr
NJS_Angle ang

説明: njCreateLight によって登録されたライトの行列を Z 軸で角度 ang 回転します。

戻り値: なし

備考: なし

8.1.1 マクロ

NJS_LIGHT *l としています。

```
#define NJM_LIGHT_INIT_VECTOR(l) l->vctr ( ライトの行列計算前の方向 )
#define NJM_LIGHT_INIT_POINT(l) l->pnt ( ライトの行列計算前の位置 )
#define NJM_LIGHT_MATRIX(l) l->mtrx ( ライトの行列 )
#define NJM_LIGHT_VECTOR(l) (l->ltcal).lvctr ( ライトの現在の方向 )
#define NJM_LIGHT_POINT(l) (l->ltcal).lpnt ( ライトの現在の位置 )
#define NJM_LIGHT_AMB(l) (l->ltcal).amb ( 環境光強度 )
#define NJM_LIGHT_DIF(l) (l->ltcal).dif ( デフューズ光強度 )
#define NJM_LIGHT_SPC(l) (l->ltcal).spc ( スペキュラ光強度 )
#define NJM_LIGHT_EXP(l) (l->ltcal).exp ( 指数 : スペキュラの広がり )
#define NJM_LIGHT_COLOR(l) (l->attr).argb ( ライトの色 )
```

8.1.2 使用法

光源計算は、位置・方向・カラー・種類等の必要な光源情報が記載されてあるライト構造体に基づいて行われます。ライト構造体は 2 種類のライト関数により設定されます。ライト関数 njCreateLight は一般的な使用法で、これは引数で決められた光源種類でライト構造体を新規に造ります。より詳しい情報は、njSetLight.... を使用することでつけ加えることができます。他方のライト構造体の設定法は、njSetLight を用いるもので、予め情報が設定されている構造体に対して行います。これは、光源 (= ライト構造体) の登録を行う他は何も行わないことに注意してください。

登録した光源は、デフォルトでモデルへ反映されます。そこで、もしあるモデルへ光源の計算を止めたい時には、モデル描画の前に njLightOff を設定しなければなりません。njLightOff、njLightOn はカレントの状態を保ち続けることに注意してください。

また、関数・引数・構造体等の詳しい情報はリファレンスを参照してください。

例 1) ライト lt1 をスポットライト, lt2 を環境光 + 点光源 (ランバートモデル) に設定します。

```
#include <ninja.h>
.....
// ライトを宣言します。
NJS_LIGHT lt1, lt2;
.....
// 初期化ルーチンです。
/*ライトを初期化・登録します。*/
njCreateLight(&lt1, NJD_SPOT_LIGHT);
njSetLightAngle(&lt1, DegToAngle(30.f), DegToAngle(60.f));
njSetLightRange(&lt1, 1000.f, 1500.f);
```

```

njSetLightLocation(&lt1, 0.f, 10.f, 15.f);
njSetLightDirection(&lt1, 0.f, 1.f, 0.f);
njSetLightColor(&lt1, 1.f, 0.f, 0.f);

njCreateLight(&lt2, NJD_LAMBERTIAN_POINT);
/* ライト属性の各種設定します。*/
njSetLightColor(&lt2, 0.5f, 0.5f, 0.5f);
njSetLightIntensity(&lt2, 0.f, 1.f, 1.f); // デフォルト強度は( 1.f,
1.f, 1.f )です。
njSetLightRange(&lt2, 100.f, 1500.f);

.....

// 描画ルーチンは以下の通りです。
while(-1)
{
.....

    /* ライトlt1, lt2をモデルに反映させます。*/
    njDrawModel(...);

.....

    /* ライトlt2を消します。*/
    njLightOff(lt2);

    /* ライトlt1をモデルに反映させます。*/
    njDrawModel(...);

    /* ライトlt2を反映させます。*/
    njLightOn(lt2);

.....
}

```

例2) 分岐処理等でライト lt1 のスポットライト色を変更し、平行光源 lt3 を新たに追加設定します。

```

.....
/* ライトlt1のカラー変更します。*/
njSetLightColor(&lt1, 0.f, 1.f, 1.f);

/* ライトを初期化・登録します。*/
njCreateLight(&lt3, NJD _ DIRECTIONAL_LIGHT);

/* ライト属性の各種設定します。*/
njSetLightDirection(&lt3, 1.f, 0.f, 0.f);
njSetLightIntensity(&lt3, 0.f, 1.f, 1.f);

.....

// 描画ルーチンは以下の通り。
while(-1)
{
.....

    /* ライトlt1, lt2, lt3をモデルに反映させます。*/
    njDrawModel(...);

.....
}

```

例 3) ライト lt にユーザー関数を設定します。

```
//ユーザー関数を設定します。(関数の引数は以下の通りです)
void
userfunc(NJS_ARGB* argb, NJS_POINT3* pnt, NJS_VECTOR* nml, NJS_LIGHT_PTR
light)
{
    .....

    // ポリゴン法線と光線方向の内積をとります。
    deg = - nml->x * NJM_LIGHT_VECTOR(light).x
          - nml->y * NJM_LIGHT_VECTOR(light).y
          - nml->z * NJM_LIGHT_VECTOR(light).z;

    .....

    argb->a = deg * NJM_LIGHT_DIF(light).a;
    argb->r = deg * NJM_LIGHT_DIF(light).r;
    argb->g = deg * NJM_LIGHT_DIF(light).g;
    argb->b = deg * NJM_LIGHT_DIF(light).b;
}

//メインルーチン中です(一部省略)。
.....
njCreateLight(&lt, NJD_USER_LIGHT);

.....
/*ライトltにユーザー関数userfuncを設定*/
njSetUserLight (&lt, userfunc);
/*ライトltのカラーの設定*/
njSetLightColor(&lt, 0.f, 1.f, 1.f);
.....

// 描画ルーチンは以下の通りです。
while(-1)
{
    .....

    /* ライトltをモデルに反映させます。*/
    DrawModel(...);

    .....
}
```

8.1.3 LIGHT 構造体の仕様

ユーザーはライト構造体を原則として直接触る必要はありませんが、参考までにライト構造体の仕様を下記に示します。

Softimage \ Ninja	[NJS_MATERIAL 構造体]	[NJS_LIGHT_ATTR 構造体]
[specular]	argb or rgb	intensity_spec
[diffuse]	argb or rgb	intensity_diff
[ambient]	(argb or rgb :pending)	intensity_amb
[exponent]	exp	なし

specular : ハイライト。Softimage では、通常 RGB 値に対しそれぞれ 0~1 を設定。

diffuse : 通常光。Softimage では、通常 RGB 値に対しそれぞれ 0~1 を設定。

ambient : 環境光。Softimage では、通常 RGB 値に対しそれぞれ 0~1 を設定。

exponent : ハイライトの広がり。Softimage では通常 0~300 を設定。

(HSV はライブラリでサポートしない予定です)

- NJS_LIGHT 構造体のメンバ -

```
struct {  
    BOOL          stat;          ( ステータス : 光源使用 ・ 不使用 )  
    NJS_POINT3    pnt;          ( 光源の位置 )  
    NJS_VECTOR    vctr;        ( 光源の単位方向ベクトル )  
    NJS_MATRIX    mtrx;        ( 光源マトリクス )  
    NJS_LIGHT_ATTR attr;        ( アトリビュート構造体 )  
    NJS_LIGHT_CAL ltcal;        ( ライト計算用構造体 )  
} NJS_LIGHT;
```

<stat>

#define NJD_LIGHT_ON 光を反映させます。

#define NJD_LIGHT_OFF 光を反映させません。

- NJS_LIGHT_ATTR 構造体のメンバ -

NJS_LIGHT_CAL ltcal; (ライト計算用構造体)

```
struct {  
    Int          lsrc;          ( 光源の種類 )  
    Float        ispc;          ( スペキュラライトの強さ : 0から1 )  
    Float        idif;          ( デフューズの強さ : 0から1 )  
    Float        iamb;          ( アンビエントの強さ : 0から1 )  
    Float        nrang;         ( 光の強度が最大である距離 : 前方の限界値 )  
    Float        frang;         ( 光の強度をカットオフする距離 : 後方の限界値 )  
    void*        func;          ( コールバック関数のポインタ )  
    Angle        iang;          ( 光の強度が最大である角度 : 内側の限界角 )  
    Angle        oang;          ( 光の強度をカットオフする角度 : 外側の限界角 )  
    NJS_ARGB     argb;          ( 光の色 )  
} NJS_LIGHT_ATTR
```

<lsrc>

光源の種類。

#define NJD_SPOT_LIGHT スポットライト

#define NJD_DIR_LIGHT 平行光源

#define	NJD_POINT_LIGHT	点光源
#define	NJD_AMBIENT	アンビエント
#define	NJD_SPEC_DIR	平行光源ハイライト
#define	NJD_SPEC_POINT	点光源ハイライト
#define	NJD_LAMBERTIAN_DIR	平行光源ランバート
#define	NJD_LAMBERTIAN_POINT	点光源ランバート
#define	NJD_PHONG_DIR	平行光源フォン
#define	NJD_PHONG_POINT	点光源フォン
#define	NJD_USER_LIGHT	ユーザー設定ライト
#define	NJD_BLOCK_LIGHT	ブロックライト
等		

<ispc, idif, iamb>

光のバランス (以下の式によります)

$\text{SPECULAR}(R,G,B) \times \text{ispc} + \text{DIFFUSE}(R,G,B) \times \text{idif} + \text{AMBIENT}(R,G,B) \times \text{iamb}$

ただし、上限 (下限) はクランプします。

<near, far>

NJD_POINT_LIGHT (点光源) NJD_SPOT_LIGHT (スポットライト)

等で指定される光の有効範囲 (距離)

nrang 光が上限値である距離の限界値。デフォルト値:1.f

frang 光の計算を行う距離の限界値。デフォルト値:65535.f

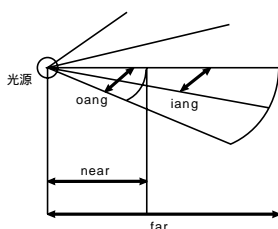
<iang, oang>

NJD_SPOT_LIGHT (スポットライト) 等で指定される光の有効範囲 (距離)

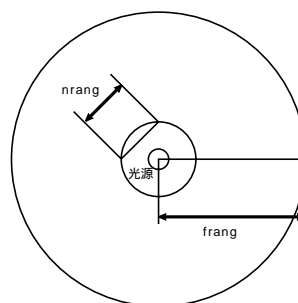
iang 光が上限値である角度の限界値。デフォルト値:(DEG)10.f

oang 光の計算を行う角度の限界値。デフォルト値:(DEG)30.f

(例:スポットライト)



(例:点光源)



- NJS_LIGHT_CAL 構造体のメンバ -

```
struct
{
    Float      ratten;    ( 減衰率 : ブロックライトが使用 )
    Float      ipd;       ( 内積 : ブロックライトが使用 )
    Float      nrr;       ( 光源の限界判定値、近 : nrang * nrang )
    Float      frr;       ( 光源の限界判定値、遠 : frang * frang )
    Float      cosi;      ( 光源の限界判定値、内 : cos * cos )
    Float      cose;      ( 光源の限界判定値、外 : cos * cos )
    Float      idev;      ( 光源の分割判定値、内 )
    Float      odev;      ( 光源の分割判定値、外 )
    Float      rate;      ( 光源の減衰比率 - スポットライト用 )
    Float      intns;     ( 光源の強さ、0 ~ 1 )
    Int        exp;       ( 光源の拡散具合 )
    Int        reserve;   ( リザーブ )
    NJS_POINT3 lpnt;      ( 光源の位置 )
    NJS_VECTOR lvctr;     ( 光源の方向ベクトル )
    NJS_VECTOR lmvctr;    ( 光源の方向ベクトル : ブロックライトが使用 )
    NJS_ARGB   atten;     ( intns * argb( 光源色 ) )
    NJS_ARGB   amb;       ( iamb*atten )
    NJS_ARGB   dif;       ( idif*atten )
    NJS_ARGB   spc;       ( ispc*atten )
} NJS_LIGHT_CAL;
```

<exp>

拡散幅をもたします。material 構造体で使します。

9.1 スクロールに関する画像単位

NINJA でのスクロールで使用する画像単位について説明します。

9.1.1 画像単位

ピクセル (pixel)

画像構成の最小分割単位。

セル (cell)

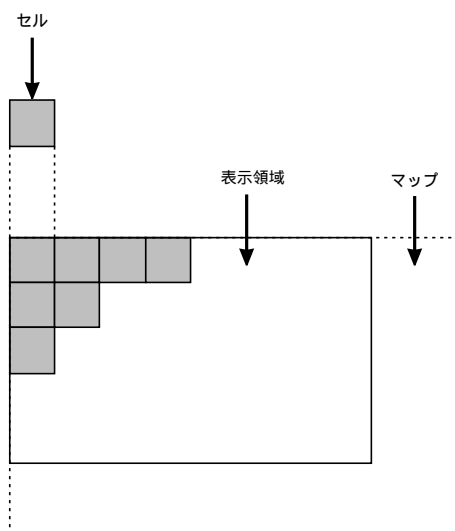
スクロール面を構成する画像の最小単位です。

NINJA では 8 ~ 1024 ピクセルの正方形とします。

セルの最大登録数は NJD_CELL_NUM_MAX で定義します。

マップ (map)

セルの集合で構成されます。マップの最大登録数は NJD_MAP_MAX で定義します。



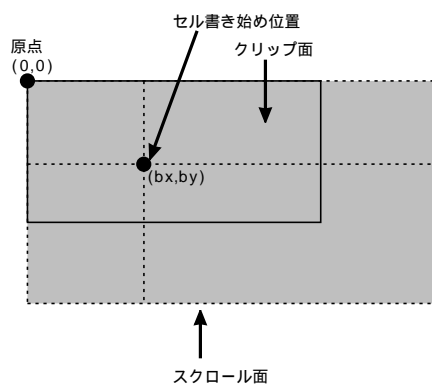
9.2 スクロールの回転、拡大、移動

ここでは、スクロール画面とスクロール構造体に設定する各値がどのような意味を持ち計算されるかを説明します。

9.2.1 スクロールの回転、拡大、移動

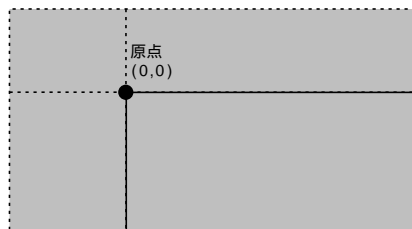
スクロール面の回転、拡大、移動は次のようになっています。

- (1) スクロール面、クリップ面とも左上を原点とする。スクロールセル原点の書出し位置の x 座標、 y 座標をそれぞれ bx, by とする。



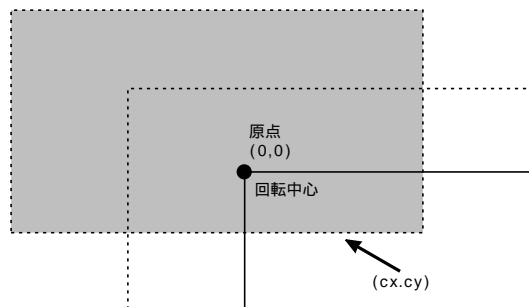
- (2) スクロール面上の点は bx, by によって原点から $-bx, -by$ だけ移動する。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x - bx \\ y - by \end{pmatrix}$$



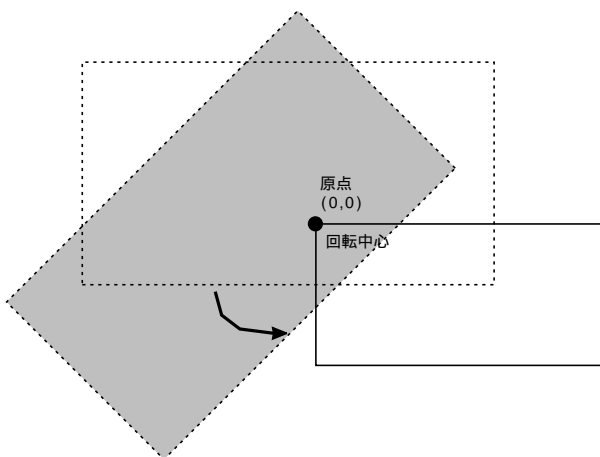
(3) 次に回転操作を行うために、回転中心 (cx,cy) の分だけ原点方向に移動する。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix}$$



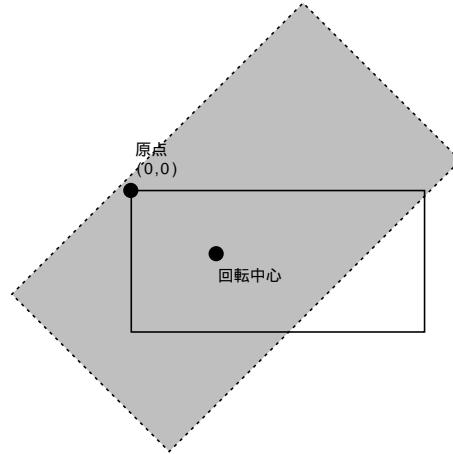
(4) 回転中心を原点とし、マトリックス m で回転させる。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = m \begin{pmatrix} x - bx - cy \\ y - by - cy \end{pmatrix}$$



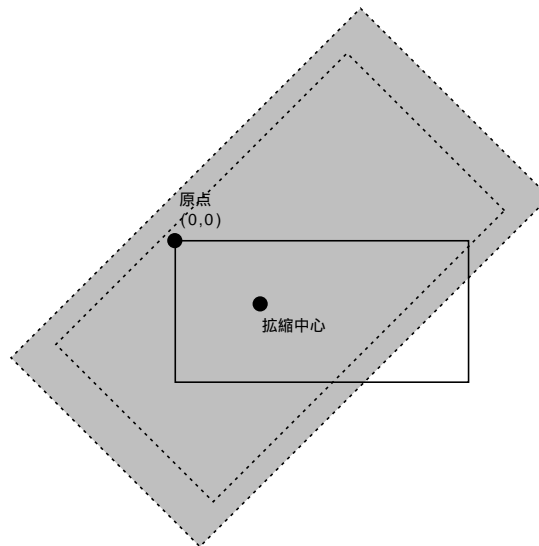
(5) 回転後スクロール面を移動した分、元に戻す。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = m \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix}$$



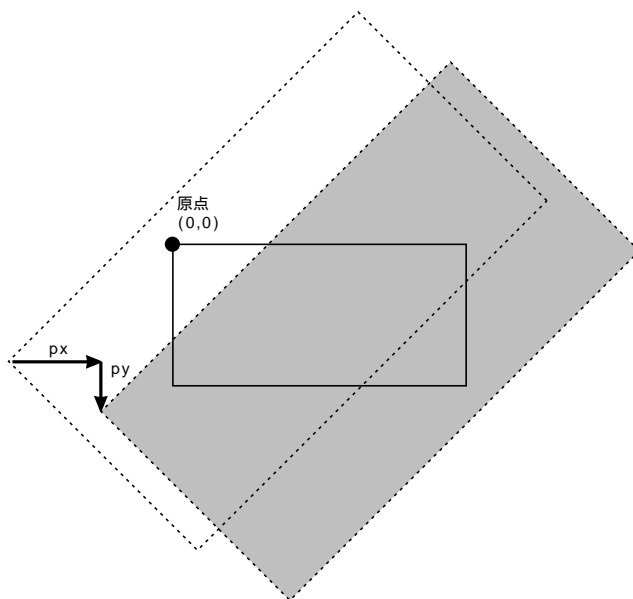
(6) 拡大中心 (spx,spy) を中心に sx,sy だけ拡大する。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \left(m \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix} - \begin{pmatrix} spx \\ spy \end{pmatrix} \right) \begin{pmatrix} sx \\ sy \end{pmatrix} + \begin{pmatrix} spx \\ spy \end{pmatrix}$$



(7) 最後に px,py だけ移動する。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} m & \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix} - \begin{pmatrix} spx \\ spy \end{pmatrix} \end{pmatrix} \begin{pmatrix} sx \\ sy \end{pmatrix} + \begin{pmatrix} spx \\ spy \end{pmatrix} + \begin{pmatrix} px \\ py \end{pmatrix}$$



9.3 スクロールの設定

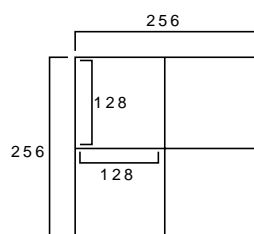
ここでは、実際にセルを描くところから、スクロール面を描画するところまでを説明します。

9.3.1 スクロールの設定例

1. セル画像を描く

セル画像をテクスチャの作成方法にしたがって描きます。ただし、1セルの大きさは8x8、16x16、32x32、64x64、128x128、256x256、512x512、1024x1024になります。

例：256x256 テクスチャに 128x128 テクスチャを 4 枚描く



2. セル画像を PVR 形式に変換する

ツールを使い、テクスチャを PVR フォーマットにします。

3. テクスチャリストを作成する

テクスチャネーム構造体、テクスチャリスト構造体を作成します。詳しい作成の仕方は、[テクスチャのドキュメント](#)を参照してください。

4. マップを作成する

例として以下の図のようなマップを作成します。

0	1	4	5	16
2	3	6	7	17
8	9	12	13	18
10	11	14	15	19

先ほどの

test0.pvr のなかに 0,1,2,3
test1.pvr のなかに 4,5,6,7
test2.pvr のなかに 8,9,10,11
test3.pvr のなかに 12,13,14,15

test4.pvr のなかに 16,17,18,19 が入っているとします。

test0.pvr からリスト作成で登録した順にテクスチャ番号が割り振られます。

test0.pvr が 0 で test4.pvr が 4 になります。これを使ってマップ用マクロ、NJM_MAP を使いマップを作成します。NJM_MAP は、NJM_MAP(テクスチャ番号、テクスチャU、テクスチャV) になっています。

よってマップの 0 の所は NJM_MAP(0,0,0)、1 は NJM_MAP(0,128,0) になります。

こうしてできたマップ配列は、

```
Uint32 map[4][5] ={
{NJM_MAP(0,0,0), NJM_MAP(0,128,0), NJM_MAP(1,0,0),
NJM_MAP(1,128,0), NJM_MAP(4,0,0)},
{NJM_MAP(0,0,128), NJM_MAP(0,128,128), NJM_MAP(1,0,128),
NJM_MAP(1,128,128), NJM_MAP(4,128,0)},
{NJM_MAP(2,0,0), NJM_MAP(2,128,0), NJM_MAP(3,0,0),
NJM_MAP(3,128,0), NJM_MAP(4,0,128)},
{NJM_MAP(2,0,128), NJM_MAP(2,128,128), NJM_MAP(3,0,128),
NJM_MAP(3,128,128), NJM_MAP(4,128,128)}}
};
```

となる。

5. スクロール構造体を設定する

スクロール構造体の各メンバを設定します。

celps	セルのピクセルサイズ 8 ~ 1024 までのいずれかの値を設定します。
mapw	マップの横方向のセルの個数を設定します。
maph	マップの縦方向のセルの個数を設定します。
sw	スクロール面の描画横方向の大きさを設定します。
sh	スクロール面の描画縦方向の大きさを設定します。
list	テクスチャリスト構造体のポインタを指定します。
map	マップ配列の先頭アドレスのポインタを設定します。map[maph][mapw] 以上の大きさのマップを用意すること。それ以下のときは不定
px,py	スクロール画面の移動座標を設定します。
bx,by	マップの書き始めの座標を設定します。
pr	スクロールのプライオリティを設定します。
sflag	拡大縮小をするか、しないか (ON,OFF) を設定します。
sx,sy	x 方向、y 方向の拡大縮小倍率を設定します。
spx,spy	拡大縮小中心座標を設定します。
mflag	回転マトリックスを使用するか、しないか (ON,OFF) を設定します。
cx,cy	回転中心座標を設定します。
m	回転マトリックスを設定します。
colmode	カラーモードを設定します。
colmix	カラー演算を設定 (未定)
clip[2]	現在は使用していません (前バージョンからの変更)
attr	アトリビュート (未定)
sclc	スクロール全体にカラーをかける。カラーモードによって変化します。

6. スクロール関数を使う

最後にスクロール関数を実際に使ってみます (マップ、テクスチャリストは作ってあるものとして)

はじめにテクスチャをロードします。

```
njInitTexture(&texmemlist,5);  
njLoadTexture(&texlist);
```

スクロール構造体を設定します (5 を参照)

```
scl.celops = 128;  
:  
:省略
```

これでスクロール関数を使うとスクロールが描けます。

```
njDrawScroll(&scl);
```

9.4 カラー

スクロール構造体の colmode で使用できるカラーモードについて説明します。

9.4.1 カラーモード

- ・ NJD_COLOR_MODE_FLAT_TEXTURE
すべてのセルのテクスチャに半透明無し (RGB565) のテクスチャを設定するときに使用します。
- ・ NJD_COLOR_MODE_FLAT_TEXTURE_TRANS
セルのテクスチャの中で 1 枚でも半透明 (ARGB1555 又は ARGB4444) がある場合使用します。

9.5 スクロール関数、構造体、定義説明

ここでは、Ninja のスクロール関数とスクロール関連構造体、スクロール関連定義を説明します。

9.5.1 スクロール関連関数

njDrawScroll

概要

2D スクロール面を描画する。

書式

```
#include <Ninja.h>
void njDrawScroll( *scl )
NJS_SCROLL *scl
```

パラメータ

*scl スクロール構造体へのポインタ

戻り値

なし

機能

クリップ画面内に 2D スクロール面を描画する。

備考

テクスチャリストの詳しい作成方法は、[テクスチャドキュメント](#)を参照してください。

9.5.2 スクロール関連構造体

NJS_SCROLL 構造体

```
typedef struct {
    Uint16      celps;          /* セルのピクセルサイズ          */
    Uint16      mapw,maph;      /* セルの個数                      */
    Uint16      sw,sh;          /* スクロール面の描画サイズ        */
    NJS_TEXLIST list;          /* テクスチャリストの先頭ポインタ */
    Uint16      *map;           /* マップ配列の先頭ポインタ        */
    Float       px,py;          /* スクロールの描き始めの座標      */
    Float       bx,by;          /* スクロール原点の描き始めの座標  */
    Float       pr;             /* プライオリティ                  */
    Sint16      sflag;          /* 拡大縮小フラグ ( ON、OFF )      */
    Float       sx,sy;          /* 拡大縮小倍率                    */
    Float       spx,spy;        /* 拡大縮小中心                    */
    Sint16      mflag;          /* 回転フラグ ( ON、OFF )          */
    Float       cx,cy;          /* 回転中心座標                    */
    NJS_SCLMTRX m;             /* 回転マトリックス                */
    Uint16      colmode;        /* カラーモード                    */
    Uint16      colmix;         /* カラー演算 ( 現在使用していません ) */
    NJS_POINT2  clip[2];        /* クリップ点 ( 現在使用していません ) */
    NJS_SCLATTR attr;          /* アトリビュート                  */
    NJS_COLOR   sclc;           /* ITEカラー                        */
}NJS_SCROLL;
```

9.5.3 スクロール関連定義

各値の最大値

```
#define NJD_CELL_NUM_MAX    0xFFFF /* the maximum of cell's number */
#define NJD_MAP_W_MAX       0xFF   /* the maximum of map's width    */
#define NJD_MAP_H_MAX       0xFF   /* the maximum of map's height   */
#define NJD_MAP_MAX         (NJD_MAP_W_MAX*NJD_MAP_H_MAX)
```

カラー定義 (colormode)

```
#define NJD_COLOR_MODE_PACKED_TEXTURE    33
#define NJD_COLOR_MODE_PACKED_TEXTURE_TRANS 41
```

9.5.4 セル設定のためのテクスチャ構造体

NJS_TEXINFO 構造体

```
typedef struct{
    void*      texaddr;          /* texture memory address cache */
    NJS_TEXSURFACE texsurface;
```

```
} NJS_TEXINFO;
```

NJS_TEXNAME 構造体

```
typedef struct{  
    void      *filename; /* ファイルネーム又はNJS_TEXINFO構造体のポインタ */  
    Uint32    attr;      /* テクスチャトリビュート */  
    Uint32    texaddr;   /* テクスチャリストへのアドレス */  
}NJS_TEXNAME;
```

NJS_TEXLIST 構造体

```
typedef struct {  
    NJS_TEXNAME      *textures; /* texture array */  
    Uint32            nbTexture; /* texture count */  
} NJS_TEXLIST;
```

10.1 語句説明

Ninja で使用するテクスチャの語句と意味を説明します。

10.1.1 語句説明

テクスチャ

Ninja でのテクスチャとは、2D グラフィックス、3D グラフィックス、スプライト、スクロール、モデルなどに貼り付ける画像をすべてさします。NINJA で使用できるテクスチャサイズは 8,16,32,64,128,256,512,1024 ドットの縦、横を持つものです。

テクスチャリスト

同時に使用するテクスチャの集合をまとめたものをテクスチャリストと呼びます。Ninja ではテクスチャリスト単位でテクスチャ操作をするのが基本になっています。テクスチャリストの作成方法は『テクスチャのロード』で説明します。

テクスチャ番号

テクスチャリストの先頭から 0,1,2,... と付けられます。詳しくは後述します。

グローバルインデックス番号

ソース内で使用するテクスチャを一意に番号を振ります。

グローバルインデックス番号が同じテクスチャは同一のテクスチャとみなします。

カレントテクスチャリスト

テクスチャ関連関数の操作対象のテクスチャリストをカレントテクスチャリストと呼びます。

カレントテクスチャ

カレントテクスチャリストの中の操作対象テクスチャをカレントテクスチャと呼びます。テクスチャ関連関数の多くが、カレントテクスチャに対してテクスチャ操作をします。

PVR 形式

Ninja でのファイルからロードできるテクスチャファイル形式です。

U, V 座標

テクスチャの内部座標の横方向を U、縦方向を V と呼びます。U, V 座標の範囲は 0 から 1 までです。テクスチャの縦横比 (アスペクト比) が違う場合でも、0 から 1 になります。

アスペクト比

テクスチャの縦横比をアスペクト比と呼びます。

ミップマップ

同じテクスチャを表すテクスチャマップの順位付けされたテクスチャのセットをミップマップと呼びます。

LOD (level of detail) 詳細レベル

ミップマップのレベルを LOD 詳細レベルと呼びます。

テクスチャメモリ

テクスチャを保存しておくためのメモリです。

キャッシュ

テクスチャメモリより多くのテクスチャを使用するため、メインメモリの一部にテクスチャをロードしておきます。この領域をキャッシュと呼びます。頻繁に入れ替えるようなテクスチャは、キャッシュ領域にロードしておくのが効果的です。

テクスチャ情報領域

テクスチャメモリにロードしたテクスチャの各情報を、Ninja が内部で保存しておく領域です。

キャッシュ情報領域

キャッシュ領域にロードしたテクスチャの各情報を、Ninja が内部で保存しておく領域です。

カテゴリコード

Ninja で使用できるテクスチャの形式をカテゴリコードと呼びます。Ninja で使用できるテクスチャの形式は TWIDDLED 形式、TWIDDLED ミップマップ形式、VQ 形式、VQ ミップマップ形式、4 ビットパレット形式、4 ビットパレットミップマップ形式、8 ビットパレット形式、8 ビットパレットミップマップ形式、レクタングル形式、ストライド形式があります。詳しくは『テクスチャ』で説明します。

ストライド値

Ninja で STRIDE 形式のテクスチャを使用する場合、指定します。使用できる値は 32 の倍数で 32 から 992 です。

10.2 テクスチャ

Ninja で使用できる PVR テクスチャのフォーマットとカテゴリコード、カラーフォーマットについて説明します。

10.2.1 PVR フォーマット

グローバル インデックスタグ	識別領域"GBIX"	4バイト
	ネクストタグへのバイト数	4バイト
	グローバルインデックス	4バイト
	ダミー	4バイト

PVRフォーマット タグ	識別領域"PVRT"	4バイト
	ネクストタグへのバイト数	4バイト
	テクスチャアトリビュート	4バイト
	横サイズ	2バイト
	縦サイズ	2バイト
各データ本体		

PVR フォーマットにはグローバルインデックスヘッダがある場合と無い場合があります。テクスチャアトリビュートにはカテゴリコードとカラーフォーマットが入ります。グローバルインデックスタグにダミーが入りました。これのテクスチャをロードできるのは Ninja Ver00050043 からです。

10.2.2 カテゴリコード

Ninja で使用できるテクスチャフォーマットをカテゴリコードと呼びます。ここからは各カテゴリコードを説明します。

TWIDDLED、TWIDDLED MIPMAP フォーマット

TWIDDLED フォーマットは Ninja の基本フォーマットです。このフォーマットは各フィルタやテクスチャのロードを高速に行うためにテクスチャ内部を最適化し、配置し直しています。このため、テクスチャ内部はラスタオーダーでは並んでいません。また、TWIDDLED フォーマットにするためには、テクスチャが正方形でなくてはなりません。

たとえば、ストライドテクスチャとして 1024x1024 のテクスチャ領域に 640x480 のエリアを作成する場合、ストライド値を 640 と指定します。

この場合 UV 値は (U,V)=(0,0)-(0.625f,0.46875f) とすると全画面貼り付けることが出来ます。

10.2.3 カラーフォーマット

Ninja で使用できるカラーフォーマットについて説明します。

通常のテクスチャカラーフォーマット

Ninja で通常使用するカラーフォーマットには ARGB1555、ARGB4444、RGB565 のテクスチャカラーフォーマットがあります。

YUV422 フォーマット

YUV422 のフォーマットです。1 ピクセルあたり 8 ビットで表すことができます。現在の Ninja は対応していません。

Bump フォーマット

バンプマップ用のテクスチャフォーマットです。現在の Ninja では対応していません。

ARGB8888 フォーマット

PALETTIZED 用のフォーマットです。現在の Ninja では対応していません。

Ninja でサポートするテクスチャフォーマット

	ARGB1555	RGB565	ARGB444	YUV422	Bump	ARGB8888
TWIDDLED						×
TWIDDLED MM						×
VQ						×
VQ MM						×
PALETTIZED 4,8				×	×	
PALETTIZED MM				×	×	
RECTANGLE						×
STRIDE						×

現在対応しているもの

今後対応するもの

× 機能的に対応できないもの

10.3 メモリ

Ninja がテクスチャをロードするメモリには、テクスチャメモリとキャッシュメモリがあります。ここでは、この2つのメモリについて説明します。

10.3.1 テクスチャメモリ

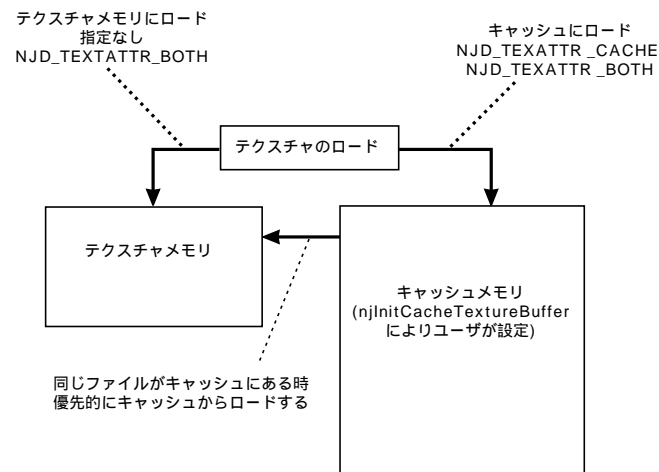
テクスチャをテクスチャとして使用するために、保存しておく領域です。テクスチャメモリ領域をリードすることもできます。

10.3.2 キャッシュ

テクスチャメモリ領域を有効に使用できるよう、メインメモリ上にテクスチャをロードできる領域をユーザーが設定することができます。この領域をキャッシュ領域と呼びます。Ninja では、キャッシュ上にあるテクスチャはキャッシュから優先的にロードするようにしています。テクスチャをキャッシュにロードするためには、テクスチャをロードする際にアトリビュートでキャッシュを指定します。ただし、はじめからメインメモリ上にあるテクスチャはキャッシュにはロードしません（ファイルからのみ）。

注意

Dev.Box の SET4 では、キャッシュテクスチャは使用できません。

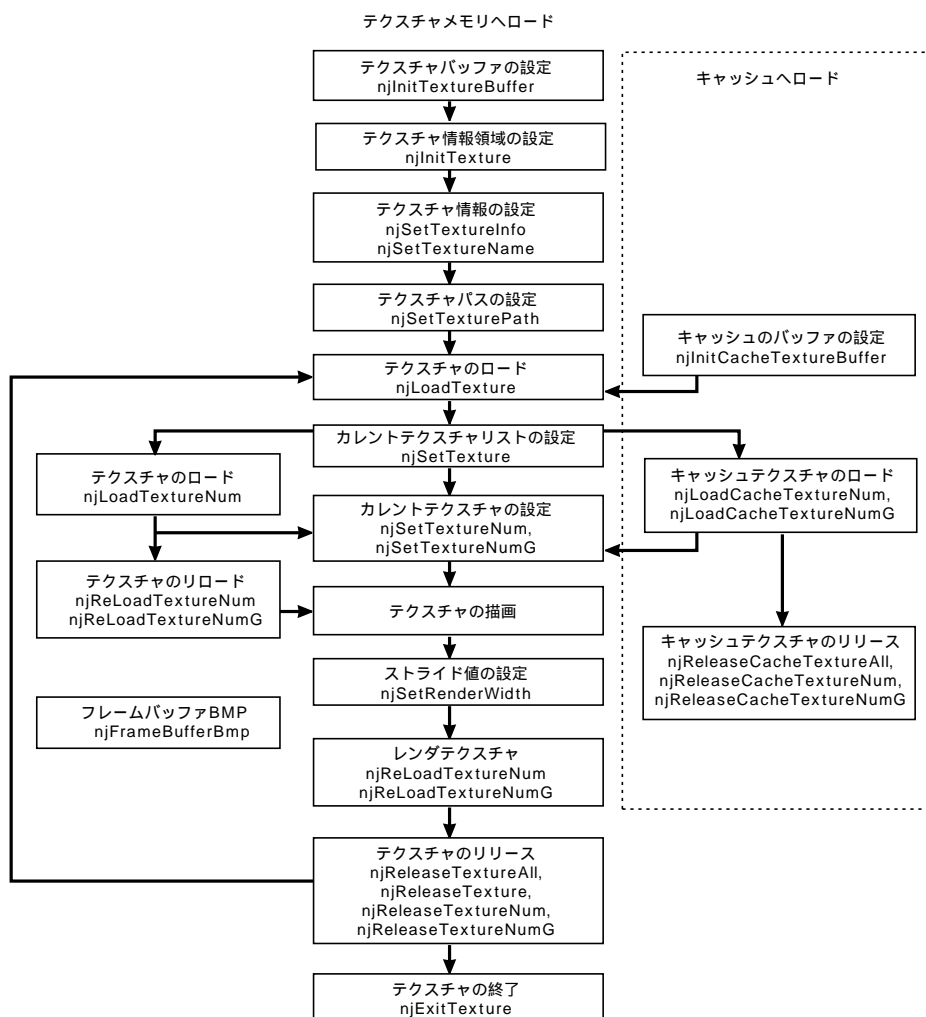


10.4 テクスチャのロード

Ninja のテクスチャ関数を使用してテクスチャのロードをします。

はじめに、テクスチャロードの大まかな流れを説明します。次に、テクスチャリストの作成方法、テクスチャ番号、グローバルインデックス番号について説明します。

10.4.1 テクスチャロードの流れ



`njLoadTextureNum` を行うときは、それ以前に `njSetTexture` を実行してカレントテクスチャリストを設定してください。

`njSetTexturePath`, `njFramebufferBmp` はターゲットでは使用できません。

10.4.2 テクスチャバッファの設定

これまでの古い SET2 用 Ninja では、テクスチャのロード時に必要なワークバッファはテクスチャ関数内部で取得していました。今後、このワーク領域を次の関数でユーザー設定することにしました。

```
void njInitTextureBuffer(Sint8 *addr,Uint32 size)
```

addr はテクスチャワークバッファの先頭ポインタ、size はワークバッファのサイズです。

サイズは、Ninja Ver00050044 から 2048 バイト以上 2048 バイト単位のバッファなら、ファイルのサイズに関係なくテクスチャをロードできるようになりました。2048 バイトのように小さいバッファで大きなファイルをロードすると GD へのアクセスの回数が増えるため、相対的にロード時間は長くなります。その場合はバッファを大きくすることで速度が改善されます。

また、バッファが 32 バイトアラインの場合、DMA を使用します。

Ninja Ver00050044 以前のライブラリでファイルからロードする場合、PVR ファイルの中で最大のサイズを指定してください。メモリからロードの場合はワークを必要としません。

また、Dev.Box SET4 以降のターゲットでは、ファイルを CD などからロードするために、ロード単位は 1 セクタ (2048 バイト) 単位になり、バッファサイズも 1 セクタ単位切り上げになります。

ワークを必要とする期間は、njLoadTexture,njLoadTextureNum が実行されている間だけです。それ以外には必要ありませんので、テクスチャのロードが終わり次第解放しても結構です。

PVR ファイルのテクスチャサイズは次のようになります (Ninja Ver00050044 以前)

TWIDDLED(GLOBALINDEX 12 バイト、ヘッダ 16 バイトを含む)

	SET2		SET4	
サイズ	MIPMAP	NO MIPMAP	MIPMAP	NO MIPMAP
8x8	0xC8	0x9C	0x800	0x800
16x16	0x2C8	0x21C	0x800	0x800
32x32	0xAC8	0x81C	0x1000	0x1000
64x64	0x2AC8	0x201C	0x3000	0x2800
128x128	0xAAC8	0x801C	0xB000	0x8800
256x256	0x2AAC8	0x2001C	0x2B000	0x20800
512x512	0xAAAC8	0x8001C	0xAB000	0x80800
1024x1024	0x2AAAC8	0x20001C	0x2AB000	0x200800

VQ(GLOBALINDEX 12 バイト、ヘッダ 16 バイトを含む)

	SET2		SET4	
サイズ	MIPMAP	NO MIPMAP	MIPMAP	NO MIPMAP
8x8	0x832	0x82C	0x1000	0x1000
16x16	0x872	0x86C	0x1000	0x1000
32x32	0x972	0x91C	0x1000	0x1000
64x64	0xD72	0xC1C	0x1000	0x1000
128x128	0x1D72	0x181C	0x2000	0x2000
256x256	0x5D72	0x481C	0x6000	0x5000
512x512	0x15D72	0x1081C	0x16000	0x11000
1024x1024	0x55D72	0x4081C	0x56000	0x41000

RECTANGLE,STRIDE(GLOBALINDEX 12 バイト、ヘッダ 16 バイトを含む)

サイズ	SET2		SET4	
	MIPMAP	NO MIPMAP	MIPMAP	NO MIPMAP
8x8	X	0x5C	X	0x800
8x16,16x8	X	0x11C	X	0x800
8x32,32x8	X	0x21C	X	0x800
8x64,64x8	X	0x41C	X	0x800
8x128,128x8	X	0x81C	X	0x1000
8x256,256x8	X	0x101C	X	0x1800
8x512,512x8	X	0x201C	X	0x2800
8x1024,1024x8	X	0x401C	X	0x4800
16x16,	X	0x21C	X	0x800
16x32,32x16	X	0x41C	X	0x800
16x64,64x16	X	0x81C	X	0x1000
16x128,128x16	X	0x101C	X	0x1800
16x256,256x16	X	0x201C	X	0x2800
16x512,512x16	X	0x401C	X	0x4800
16x1024,1024x16	X	0x801C	X	0x8800
32x32	X	0x81C	X	0x1000
32x64,64x32	X	0x101C	X	0x1800
32x128,128x32	X	0x201C	X	0x2800
32x256,256x32	X	0x401C	X	0x4800
32x512,512x32	X	0x801C	X	0x8800
32x1024,1024x32	X	0x1001C	X	0x10800
64x64	X	0x201C	X	0x2800
64x128,128x64	X	0x401C	X	0x4800
64x256,256x64	X	0x801C	X	0x8800
64x512,512x64	X	0x1001C	X	0x10800
64x1024,1024x64	X	0x2001C	X	0x20800
128x128	X	0x801C	X	0x8800
128x256,256x128	X	0x1001C	X	0x10800
128x512,512x128	X	0x2001C	X	0x20800
128x1024,1024x128	X	0x4001C	X	0x40800
256x256	X	0x2001C	X	0x20800
256x512,512x256	X	0x4001C	X	0x40800
256x1024,1024x256	X	0x8001C	X	0x80800
512x512	X	0x8001C	X	0x80800
512x1024,1024x512	X	0x10001C	X	0x100800
1024x1024	X	0x20001C	X	0x200800

10.4.3 キャッシュバッファの設定

テクスチャバッファと同様に、これまでキャッシュ関数内部で取得していたキャッシュエリアもユーザー設定になりました。キャッシュバッファはテクスチャバッファとは違い、キャッシュを保存している期間必要です。キャッシュバッファは 32 ビットアラインで取得し、必要サイズはテクスチャの合計サイズ+ リストヘッダ 32 バイト×テクスチャの数です。

```
void njInitCacheTextureBuffer(Sint8 *addr,Uint32 size)
```

10.4.4 テクスチャリストの作成

Ninja ではテクスチャリストを中心にテクスチャの操作をします。テクスチャリストの設定の仕方を書きます。

1. テクスチャの数分だけ、テクスチャネーム構造体を設定します。

NJS_TEXNAME 構造体

```
void          *filename
Uint32        attr
Uint32        texaddr
```

*filename

PVR 形式のテクスチャファイルのときは、ファイル名のストリングを指定します。
メモリからのロードのときは、NJS_TEXINFO 構造体のポインタを指定します。

attr

テクスチャのロード元とロード先を指定します。それぞれのタグの OR をとります。

・ロード元

NJD_TEXATTR_TYPE_FILE

PVR 形式のファイルをロード。 *filename でファイル名を指定します。

NJD_TEXATTR_TYPE_MEMORY

メモリからロード。 *filename で NJS_TEXINFO 構造体のポインタを設定します。
NJS_TEXINFO 構造体の設定の仕方は下を参照のこと

NJD_TEXATTR_TYPE_FRAMEBUFFER (変更)

SET4 より使用できなくなりました。

・ロード先 (指定しないときテクスチャメモリにロードします)

NJD_TEXATTR_CACHE

キャッシュメモリにのみロードします。

NJD_TEXATTR_BOTH

テクスチャメモリとキャッシュメモリの両方にロードします。

・グローバルインデックスの指定

NJD_TEXATTR_GLOBALINDEX

NJS_TEXNAME の texaddr に指定した数字をグローバルインデックスとみなします。
ファイル内のグローバルインデックスより優先されます。

texaddr

テクスチャの設定時、メモリテクスチャのときや NJS_TEXATTR_GLOBALINDEX を attr に設定したときに、グローバルインデックスを指定します。テクスチャロード後は内部のテーブルへのポインタになります。

テクスチャのロード元がメモリの場合、NJS_TEXINFO 構造体の設定が必要です。

NJS_TEXINFO 構造体

```
void*          texaddr;
NJS_TEXSURFACE texsurface;
```

texaddr

メインメモリ内にテクスチャを保存するときに使用します。

texsurface

内部にデータを渡す形式です。

NJS_TEXSURFACE 構造体

```
Uint32      Type;
Uint32      BitDepth;
Uint32      PixelFormat;
Uint32      nWidth;
Uint32      nHeight;
Uint32      TextureSize;
Uint32      fSurfaceFlags;
Uint32      *pSurface;
Uint32      *pVirtual;      <-新規追加
Uint32      *pPhysical;     <-新規追加
```

nWidth

メモリテクスチャの場合、テクスチャの横サイズを設定します。

nHeight

メモリテクスチャの場合、テクスチャの縦サイズを設定します。

その他のメンバについてはロード関数内で設定されます。

2. 1. で設定したテクスチャネーム構造体を使いテクスチャリスト構造体を設定します。

NJS_TEXLIST 構造体

```
NJS_TEXNAME  *textures;
Uint32      nbTexture;
```

textures

各テクスチャ情報を設定した NJS_TEXNAME 構造体のポインタを設定します。

nbTexture

テクスチャの数

例:file01.pvr とメモリテクスチャImage をテクスチャとして設定する場合を示します。

```
extern Uint16 Image[];

NJS_TEXINFO Info;
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist={texname,2};
Sint8* buffer;

/* メモリテクスチャ      Image
   カテゴリコード      TWIDDLED
   カラーフォーマット    ARGB1555
   サイズ                256x256
*/
njSetTextureInfo(&Info,Image,NJD_TEXFMT_TWIDDLED|NJD_TEXFMT_ARGB_1555,256,256);

/* ファイルfile0.pvrをtexname[0] にグローバルインデックス0で設定する */
njSetTextureName(&texname[0],"file0.pvr",0,NJD_TEXATTR_TYPE_FILE|NJD_TEXATTR_GLOBALINDEX);

/* メモリテクスチャImageをtexname[1] にグローバルインデックス1で設定する */
njSetTextureName(&texname[1],&Info,1,NJD_TEXATTR_TYPE_MEMORY|NJD_TEXATTR_GLOBALINDEX);

buffer = syMalloc(0x1000);
/* テクスチャバッファの設定 0x800以上の0x800単位ならロードできる */
njInitTextureBuffer(buffer, 0x1000);

/* テクスチャのイニシャル */
njInitTexture(texmemlist,2);

/* テクスチャをロードする */
njLoadTexture(&texlist);
```

10.4.5 テクスチャ番号

カレントのテクスチャリストに対して、6.3 で説明した NJS_TEXNAME 構造体の設定順にテクスチャ番号を 0,1,2,... と振ります。

```
NJS_TEXNAME texname[] = { {"file0.pvr",,,,}, /*テクスチャ番号0 */
                          {"file1.pvr",,,,}, /*テクスチャ番号1 */
                          {"file2.pvr",,,,}, /*テクスチャ番号2 */
                          {"file3.pvr",,,,}, /*テクスチャ番号3 */
                          :
                          {"filen.pvr",,,,}}; /*テクスチャ番号n */
```

Ninja のテクスチャ関数で使用するテクスチャ番号は、カレントのテクスチャリストのテクスチャ番号を対象にしています。

10.4.6 グローバルインデックス番号

複数のテクスチャリストを使用しても同じテクスチャがテクスチャメモリ上にロードされないように、アプリケーション内で一意に決まる番号を振ります。これをグローバルインデックス番号といいます。グローバルインデックス番号が同じテクスチャは、同一のテクスチャとして扱われます。2D、3D グラフィック、スプライト、スクロール、モデルなどすべてのテクスチャにおいてグローバルインデックス番号が関係します。違うテクスチャに同じ番号が振られることがないように、気を付けてください。また反対に同じテクスチャに違うグローバルインデックス番号を付けると、同一のテクスチャがテクスチャメモリにロードされることになります。

PVR 形式のファイルには内部にグローバルインデックスを入れるチャンクがあります。PVR 形式のテクスチャのグローバルインデックスはツールで管理できます。

また、PVR 形式でもグローバルインデックスのチャンクを持たないものや、ファイル内のグローバルインデックスを使用したくないときは、設定時 NJD_TEXATTR_GLOBALINDEX を指定することでグローバルインデックスを付け直すことができます。

グローバルインデックス番号は 0 から 0xFFFFFFFFF までの数を使用してください。0xFFFFFFFFF から 0xFFFFFFFFF はシステムが使用しますので使用しないでください。

10.4.7 グローバルインデックスの自動割付

Ninja Ver00040032 よりグローバルインデックスのないテクスチャがロードできるようになりました。この場合、デフォルトでグローバルインデックスは 0xFFFFFFFFF より 0xFFFFF FEE、0xFFFFF FED... のように降順にグローバルインデックス番号を割り付けます。自動割付に使用されるグローバルインデックスの初期値は、次の関数で設定できます。

```
void njInitTextureGlobalIndex(Uint32 globalIndex);
```

この関数で指定したグローバルインデックスより、globalIndex, globalIndex-1, globalIndex-2... となります。

グローバルインデックスの自動割付は、降順方向にしか進みませんので、テクスチャをデリートしてもグローバルインデックスは戻らずに次の番号から割り付けられます。

グローバルインデックスを戻したいときには、njInitTextureGlobalIndex 関数で再設定してください。また、通常の方法でつけられたグローバルインデックスとぶつかると、先にロードされた方が優先されますので注意してください。

10.4.8 テクスチャのロードエラー

ユーザーが njInitTexture で設定したテクスチャメモリリスト (NJS_TEXMEMLIST) にテクスチャのロード要求後のデータが保存されます。テクスチャメモリリストには次のようなデータが保存されます。

Uint32	globalIndex;	グローバルインデックス	
Uint32	tspparambuffer;	ハード設定データ	<-新規追加
Uint32	texparambuffer;	ハード設定データ	<-新規追加
Uint32	texaddr;	BIT_0: テクスチャメモリにロード	
		BIT_1: キャッシュにロード	
NJS_TEXINFO	texinfo;	テクスチャインフォ構造体	

Uint16	count;	使用している回数
Uint16	dummy;	エラーコード(新規追加)

テクスチャのロード時にエラーがあった場合、dummy には次のエラーコードが設定されます。

#define NJD_TEXERR_OTHER	(1) //その他のエラー
#define NJD_TEXERR_FILEOPEN	(2) //ファイルオープンエラー
#define NJD_TEXERR_EXTND	(3) //拡張子エラー
#define NJD_TEXERR_HEADER	(4) //ヘッダーエラー
#define NJD_TEXERR_FILELOAD	(5) //ファイルロードエラー
#define NJD_TEXERR_SURFACE	(6) //サーフェス作成エラー
#define NJD_TEXERR_MAINMEMORY	(7) //メインメモリマロックエラー
#define NJD_TEXERR_TEXMEMLOAD	(8) //テクスチャメモリーロードエラー
#define NJD_TEXERR_GLOBALINDEX	(9) //グローバルインデックスエラー(新規)

• NJD_TEXERR_FILEOPEN

ファイルが設定された場所がないため、オープンすることができません。

• NJD_TEXERR_EXTND

拡張子が PVR ファイルではありません。

• NJD_TEXERR_HEADER

テクスチャファイルのヘッダが間違えています。GBIX タグや PVRT タグが違う場合に出ます。

• NJD_TEXERR_FILELOAD

ファイルがロードできません。またはデータが期待していたサイズより小さい場合にこのエラーが出ます。

• NJD_TEXERR_SURFACE

テクスチャメモリーにテクスチャをロードする領域が取れない場合にエラーが出ます。

テクスチャのサイズが大きすぎる場合やテクスチャのロードできないタイプなどが指定されたときにも、このエラーが出ます。

• NJD_TEXERR_MAINMEMORY

テクスチャロード関数内部でワークバッファの領域が確保できない場合、このエラーが出ます。

• NJD_TEXERR_TEXMEMLOAD

テクスチャメモリーにロードできない場合、このエラーが出ます。このエラーは、通常出ることはありません。(このエラーが出る前に NJD_TEXERR_SURFACE でエラーが出るはずです)

• NJD_TEXERR_GLOBALINDEX

無効なグローバルインデックスが指定された場合、またグローバルインデックスが取得できなかったとき、このエラーが出ます。

10.4.9 メモリテクスチャ

メインメモリに展開したテクスチャデータやメインメモリ上で作成したテクスチャデータをロードし、テクスチャデータとして使用することができます。テクスチャデータとして使用できる形式は、PVR 形式のテクスチャデータでヘッダ情報 (グローバルインデックスタグ、PVR ヘッダタグ)+データまたはデータ部分のみでもテクスチャデータとして使用できます。以下は、それぞれの場合についての設定方法です。

ヘッダ情報がメモリテクスチャに入っている場合

```

Uint16 T009[] = {
    0x4247, 0x5849, 0x0004, 0x0000, 0x1d4c, 0x0000,          /* (1) */
                                0x5650, 0x5452, /* (2) */
    0x8008, 0x0000, 0x0101, 0x0000, 0x0080, 0x0080, 0xad20, 0xeeee1,
    0xac40, 0xe5c0, 0xff41, 0xff21, 0xf5e0, 0xe580, 0xd580, 0xf700, /* (3) */
    0xde00, 0xff80, 0xff40, 0fee0, 0ffe0, 0ffe0, 0xff21, 0xf720,

    TWIDDLED テクスチャ
    RGB565
    128x128 のデータが入っている

    :

};

```

(1) グローバルインデックスヘッダ部

(2) PVR ヘッダ部

(3) テクスチャデータ部

```

NJS_TEXINFO          info;   infoをテクスチャロード後に保存する必要は
    ありません
NJS_TEXNAME texname[1];
NJS_TEXLIST texlist = {texname,1};

njSetTextureInfo(&info[0],T009,0,0,0);
njSetTextureName(&texname[0],&info[0],0,NJD_TEXATTR_TYPE_MEMORY);

```

ヘッダ情報にグローバルインデックス情報が入っている場合には、njSetTextureName 関数の第 3、第 4 引数にグローバルインデックス情報を設定する必要はありません。また、PVR ヘッダ情報があるので njSetTextureInfo 関数の第 3、第 4、第 5 引数に情報をセットする必要はありません。

ヘッダ情報がメモリテクスチャに入っていない場合

```
Uint16 T009[] = {
0xac40, 0xe5c0, 0xff41, 0xff21, 0xf5e0, 0xe580, 0xd580, 0xf700,
/* テクスチャデータ部 */
0xde00, 0xff80, 0xff40, 0xee0, 0xfe0, 0xfe0, 0xff21, 0xf720,

TWIDDLEDテクスチャ
RGB565
128x128のデータが入っている
:
};

NJS_TEXINFO info; infoをテクスチャロード後に保存する必要は
ありません
NJS_TEXNAME texname[1];
NJS_TEXLIST texlist = {texname,1};

njSetTextureInfo(&info,T009,NJD_TEXFMT_RGB_565|NJD_TEXFMT_TWIDDLED,128,128);
njSetTextureName(&texname[0],&info,0,
NJD_TEXATTR_TYPE_MEMORY|NJD_TEXATTR_GLOBALINDEX);
```

グローバルインデックス情報、PVR ヘッダ情報の両方が入っていないPVR形式のデータ部分だけの場合、njSetTextureName関数の第3, 第4引数にグローバルインデックス情報を設定し njSetTextureInfo関数の第3にテクスチャのタイプとカラー形式と第4, 第5引数にテクスチャの縦、横のサイズが入ります。

10.4.10 レンダテクスチャ

テクスチャのカテゴリが STRIDE や RECTANGLE のとき、通常のフレームバッファにレンダリングする代わりに、テクスチャにレンダリングすることができます。このテクスチャを使用することで、擬似的な環境マッピングなどができます。レンダテクスチャはテクスチャにレンダリングをし、それを利用してもう一度描画をすることになるので、レンダテクスチャを1フレーム内に複数行くと、行った回数だけレンダリング回数が増え、性能が落ちます。レンダテクスチャで指定するテクスチャがフレームバッファより小さい場合、画面の左上からテクスチャのサイズ分、レンダリングが行われます。また、レンダテクスチャで使用するテクスチャのカラーモードは、フレームバッファのモードと同じでなくてはなりません。レンダテクスチャはフレームバッファテクスチャとは違い、指定したサイズ分の容量をテクスチャ領域にとります。

使用例

```
void njUserInit(void)
{
/*レンダテクスチャを使用する場合、njInitSystemのフレームバッファの
カラーモードとレンダテクスチャで使用するテクスチャのカラーモードを
合わせる */
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
:
/* とりあえずテクスチャのメモリ領域をダミーで確保する */
buff = njMalloc(0x1000);
/* フレームバッファのカラーと合わせる。サイズは512x512とする。 */
njSetTextureInfo(&info,buff,NJD_TEXFMT_STRIDE|NJD_TEXFMT_RGB_565,512,512);
```

```

njSetTextureName(&texname[0],&info,0,NJD_TEXATTR_TYPE_MEMORY|
NJD_TEXATTR_GLOBALINDEX);
/* buffはnjLoadTextureの間だけ存在すればよい */
njInitTextureBuffer(buff,0x1000);
njInitTexture( tex, 100 );
njLoadTexture(&texlist);
/* ダミーで確保した領域はnjLoadTexture後は解放してもよい */
njFree(buff);
/* ストライド値を512にする */
njSetRenderWidth(512);
}

 Sint32 njUserMain(void)
{
    :
    /* モデルなどを描画する */
    njDrawObject( OBJECT );
    :
    njSetTexture(&texlist);
    /* テクスチャ番号0にレンダリングする
    このテクスチャは512x512なので左上から512x512の範囲がレンダリングされる*/
    njRenderTextureNum(0);

    /* レンダリングしたテクスチャを使用して描画する */
    njDrawTexture( poly, 4, 0,TRUE);
    :
}

```

10.5 サンプルプログラム

10.5.1 概要

ここでは、次の例について簡単なサンプルプログラムを示します。

- 例1：PVR ファイルのテクスチャを表示します。
- 例2：メモリからテクスチャをロードし表示します。
- 例3：ファイルからキャッシュにロードし、テクスチャを表示します。

10.5.2 サンプル

例1：PVR ファイルのテクスチャを表示します。

```

1:#include <Ninjawin.h>
2:
3:NJS_TEXNAME texname[2];
4:
5:NJS_TEXLIST texlist ={texname,2};
6:NJS_TEXMEMLIST texmemlist[2]; /* 2つ分のテクスチャ情報領域を確保 */
7:NJS_POINT2COL p[4];
8:Sint8 buffer[0x2B000];
9:
10: void njUserInit(void)
11:{
12:     njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 );
13:     /* 2枚のテクスチャを設定する */

```

```

14:     njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE|
15:         NJD_TEXATTR_GLOBALINDEX);
16:     njSetTextureName(&texname[1], "file1.pvr", 1, NJD_TEXATTR_TYPE_FILE|
17:         NJD_TEXATTR_GLOBALINDEX);
18:     njInitTextureBufer(buffer, 0x2B000);
19:         /* file0, file1は256x256のTwiddled Mipmap */
19:     njInitTexture(texmemlist, 2);
20:     njLoadTexture(&texlist); /* テクスチャをロードする */
21:     njSetTexture(&texlist);
22:         /* カレントテクスチャリストをtexlistにする */
22:     /* カレントテクスチャをtexlistの0番のテクスチャにする*/
23:     njSetTextureNum(0);
24:
25:     /* ポリゴンのデータ入力 */
26:     p[0].x = 100; p[0].y = 100;
27:     p[1].x = 200; p[1].y = 100;
28:     p[2].x = 200; p[2].y = 200;
29:     p[3].x = 100; p[3].y = 200;
30:     p[0].col.tex.u = 0; p[0].col.tex.v = 0;
31:     p[1].col.tex.u = 255; p[1].col.tex.v = 0;
32:     p[2].col.tex.u = 255; p[2].col.tex.v = 255;
33:     p[3].col.tex.u = 0; p[3].col.tex.v = 255;
34: }
35: Sint32 njUserMain(void)
36: {
37:     /* テクスチャのポリゴンを描く */
38:     njDrawPolygon2D(p, 4, -100.f, NJD_FILL|NJD_USE_TEXTURE);
39:     return NJD_USER_CONTINUE;
40: }
41: void njUserExit(void)
42: {
43:     njExitTexture();
44:     njExitSystem();
43: }

```


例2：メモリからテクスチャをロードし表示します。

```

1:#include <Ninjawin.h>
2:
3:extern Uint16 Image[];
    /* 他のファイルに256からのミップマップデータがあると仮定 */
4:
5:NJS_TEXINFO Info;
6:NJS_TEXNAME texname[2];
7:
8:NJS_TEXLIST texlist = {texname,2};
9:NJS_TEXMEMLIST texmemlist[2];    /* 2つ分のテクスチャ情報領域を確保 */
10:NJS_POINT2COL p[4];
11:
12: void njUserInit(void)
13:{
14:    njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 )
15:    /* 2枚のテクスチャを設定する */
16:    njSetTextureInfo(&Info,Image,NJD_TEXFMT_TWIDDLED|NJD_TEXFMT_ARGB
        _1555,256,256);
17:    njSetTextureName(&texname[0],"file0.pvr",0,NJD_TEXATTR_TYPE_FILE|
18:        NJD_TEXATTR_GLOBALINDEX);
19:    njSetTextureName(&texname[1],&Info,1,NJD_TEXATTR_TYPE_MEMORY|
20:        NJD_TEXATTR_GLOBALINDEX);
21:    njInitTexture(texmemlist,2);
22:    njLoadTexture(&texlist);    /* テクスチャをロードする */
23:    njSetTexture(&texlist);
        /* カレントテクスチャリストをtexlistにする */
24:    /* カレントテクスチャをtexlistの1番のテクスチャにする*/
25:    njSetTextureNum(1);
26:
27:    /* ポリゴンのデータ入力 */
28:    p[0].x = 100; p[0].y = 100;
29:    p[1].x = 200; p[1].y = 100;
30:    p[2].x = 200; p[2].y = 200;
31:    p[3].x = 100; p[3].y = 200;
32:    p[0].col.tex.u = 0; p[0].col.tex.v = 0;
33:    p[1].col.tex.u = 255;p[1].col.tex.v = 0;
34:    p[2].col.tex.u = 255;p[2].col.tex.v = 255;
35:    p[3].col.tex.u = 0; p[3].col.tex.v = 255;
36:}
37:
38:Sint32 njUserMain(void)
39:{
40:    /* テクスチャのポリゴンを描く */
41:    njDrawPolygon2D(p,4,-100.f,NJD_FILL|NJD_USE_TEXTURE);
42:    return NJD_USER_CONTINUE;
43:}
44:
45: void njUserExit(void)
46:{
47:    njExitTexture();
48:    njExitSystem();
49:}

```

例3：ファイルからキャッシュにロードし、テクスチャを表示します。

```
1:#include <Ninjawin.h>
2:
3:NJS_TEXNAME texname[2];
4:
5:NJS_TEXLIST texlist ={texname,2};
6:NJS_TEXMEMLIST texmemlist[2]; /* 2つ分のテクスチャ情報領域を確保 */
7:NJS_POINT2COL p[4];
8:Sint8 buffer[0x2B000];
9:Sint8 cbuffer[(0x2AAAC+32)*2];
10:
11:void njUserInit(void)
12:{
13:    njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 )
14:    njInitTexture(texmemlist,2);
15:    /* 2枚のテクスチャを設定する */
16:    njSetTextureName(&texname[0], "file0.pvr",0,NJD_TEXATTR_TYPE_FILE|
17:                    NJD_TEXATTR_CACHE|NJD_TEXATTR_GLOBALINDEX);
18:    njSetTextureName(&texname[1], "file1.pvr",1,NJD_TEXATTR_TYPE_MEMORY|
19:                    NJD_TEXATTR_CACHE|NJD_TEXATTR_GLOBALINDEX);
20:    njInitTextureBuffer(buffer,0x2B000);
21:    njInitCacheTextureBuffer(cbuffer,(0x2AAAC+32)*2);
22:    njLoadTexture(&texlist); /* テクスチャをロードする */
23:    njSetTexture(&texlist);
24:    /* カレントテクスチャリストをtexlistにする */
25:    njLoadCacheTextureNum(0);
26:    /* 0番のテクスチャをキャッシュからロード */
27:    njLoadCacheTextureNum(1);
28:    /* 1番のテクスチャをキャッシュからロード */
29:    /* カレントテクスチャをtexlistの0番のテクスチャにする*/
30:    njSetTextureNum(0);
31:
32:    /* ポリゴンのデータ入力 */
33:    p[0].x = 100; p[0].y = 100;
34:    p[1].x = 200; p[1].y = 100;
35:    p[2].x = 200; p[2].y = 200;
36:    p[3].x = 100; p[3].y = 200;
37:    p[0].col.tex.u = 0; p[0].col.tex.v = 0;
38:    p[1].col.tex.u = 255;p[1].col.tex.v = 0;
39:    p[2].col.tex.u = 255;p[2].col.tex.v = 255;
40:    p[3].col.tex.u = 0; p[3].col.tex.v = 255;
41:}
42:
43:Sint32 njUserMain(void)
44:{
45:    /* テクスチャのポリゴンを描く */
46:    njDrawPolygon2D(p,4,-100.f,NJD_FILL|NJD_USE_TEXTURE);
47:    return NJD_USER_CONTINUE;
48:}
49:
50:void njUserExit(void)
51:{
52:    njExitTexture();
53:    njExitSystem();
54:}
```

10.6 テクスチャ関数の注意点

ここでは、テクスチャ関連関数使用時の注意点について説明します。

10.6.1 Dev.Box の SET2 から SET4、SET5 へ移行時の注意点

1. SET2 ではテクスチャ関数の作業領域は内部でアロケーションしていましたが、SET4 以降では njInitTextureBuffer 関数でテクスチャバッファを確保してください。
2. SET4 以降で njSetTexturePath 関数は使用できません。njSetTexturePath 関数を使用していた所は以下のようにしてください。
SET2 :

```
njSetTexturePath("¥¥image0");
njLoadTexture(&texlist0);
njSetTexturePath("¥¥image1");
njLoadTexture(&texlist1);
```


SET4:

```
gdFsChangeDir("IMAGE0");
njLoadTexture(&texlist0);
gdFsChangeDir("../");
gdFsChangeDir("IMAGE1");
njLoadTexture(&texlist1);
```
3. SET4 では DMA 転送に対応していないため、レンダリング中にテクスチャをロードすることはできません。そのため、キャッシュテクスチャをレンダリング中に使用することや、リロードテクスチャをレンダリング中に使用することはできません。
4. njFramebufferBmp 関数の仕様を変更し、njFramebufferBmp2 関数削除しました。
5. njInitCacheBuffer 関数で与えるキャッシュバッファサイズが間違えていました。詳しくは『テクスチャバッファの設定』をご覧ください。

10.6.2 SET5 での注意点

SET5 ではメインメモリからテクスチャメモリへの転送は、バッファの先頭アドレスが 32 バイトアラインメントの場合 DMA 転送で行い、32 バイトアラインメント以外のときは CPU 転送になります。

実際には、njInitTextureBuffer 関数で指定したバッファの先頭から PVR ファイルをロードするので、ヘッダの次に来るテクスチャデータ部分が 32 バイトアラインメントにある場合、DMA になり、32 バイトアラインメントにない場合 CPU 転送になります。

テクスチャメモリに転送する関数を割り込み禁止状態で実行する場合、32 バイトアラインメントされたバッファのときは DMA 終了割り込みが取れなくなるので、バッファが 32 バイトアラインメントにならないよう注意してください。

11.1 カメラの初期化について

カメラの初期化は下記の関数を用いて行います。

`njlInitCamera (NJS_CAMERA *c)`

カメラの各パラメータを次のように初期化します。

パラメータ	値	備考
位置 (px, py, pz)	(0、0、0)	ワールド座標原点。
視線の向き (vx, vy, vz)	(0、0、- 1)	画面奥。(ローカル Z 軸)
傾き (roll)	0 度	ワールド座標系の Y 軸に対して平行。
画角 (ang)	60 度	視野の水平方向の角度
ニアクリップ (n_clip)	- 1	
ファークリップ (f_clip)	- 60000	
ローカル X 軸 (lx)	(1、0、0)	
ローカル Y 軸 (ly)	(0、1、0)	

注意

視線の向き (ローカル Z 軸)、ローカル X 軸、ローカル Y 軸はそれぞれワールド座標系でのベクトル値で表されます。

11.1.1 カメラ構造体

```
typedef struct {
    Float    px,py,pz;    /* 位置          */
    Float    vx,vy,vz;    /* 視線の向き     */
    Angle    roll;        /* 傾き           */
    Angle    ang;         /* 画角           */
    Float    n_clip;      /* ニアクリップ   */
    Float    f_clip;      /* ファークリップ */
    NJS_VECTOR lx, ly;    /* ローカル座標軸 */
} NJS_CAMERA;
```

11.2 カメラの位置について

カメラは次の関数を用いてワールド空間の好きな座標に配置する事が出来ます（初期化時には(0, 0, 0)に配置されています）。

なお、カメラの位置を移動、回転させても視線の向き、及び傾きが変化する事はありません。

`njTranslateCameraPosition(NJS_CAMERA *c, Float x, Float y, Float z)`

カメラの位置を現在置かれている位置から X、Y、Z 軸に沿って移動します。

`njRotateCameraPositionX(NJS_CAMERA *c, Float ang)`

`njRotateCameraPositionY(NJS_CAMERA *c, Float ang)`

`njRotateCameraPositionZ(NJS_CAMERA *c, Float ang)`

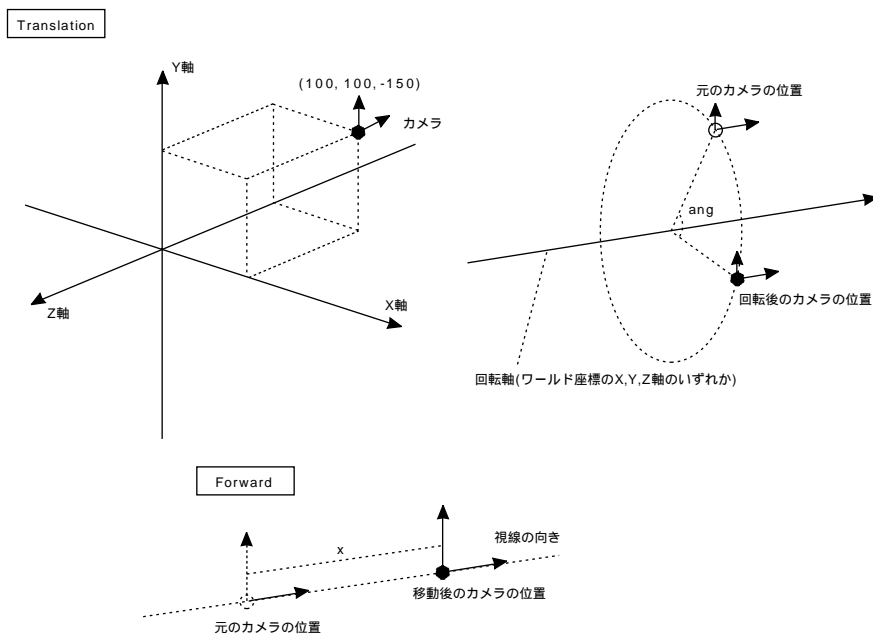
`njRotateCameraPositionXYZ(NJS_CAMERA *c, Float ax, Float ay, Float az)`

カメラの位置を現在置かれている位置からワールド座標の各軸を中心に `ang` だけ回転させた場所に移動します。`njRotateCameraPositionXYZ` 関数は X、Y、Z 軸の順にそれぞれ `ax`、`ay`、`az` だけ回転させます。

`njForwardCameraPosition(NJS_CAMERA *c, Float x)`

カメラを現在置かれている位置から視線の向きに向かって `x` だけ移動します。

黒い点はカメラを表し、点に付いている長い矢印は視線の向き、短い矢印はカメラの傾き（カメラの上方）を表しています。



11.3 カメラの視線について

カメラの視線は向きと傾きの2つのパラメータで表され次の関数を用いて自由に操作する事が出来ます(向きと傾きはお互いに影響を受けません)。

`njPitchCameraInterest(NJS_CAMERA *c, Angle ang)`

カメラのローカル座標系の X 軸に対して視線の向きを `ang` だけ回転します。

`njYawCameraInterest(NJS_CAMERA *c, Angle ang)`

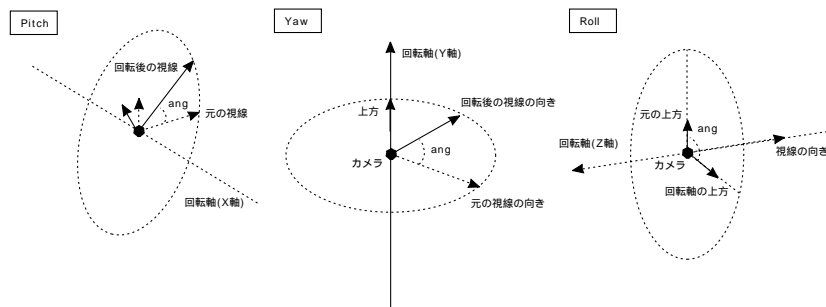
カメラのローカル座標系の Y 軸に対して視線の向きを `ang` だけ回転します。

`NjRollCamera(NJS_CAMERA *c, Angle ang)`

カメラ自体の視線(ローカル座標系の Z 軸)を中心にカメラを回転します

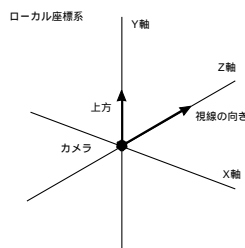
`njPointCameraInterest(NJS_CAMERA *c, Float x, Float y, Float z)`

カメラの視線をワールド空間の座標 (`x`、`y`、`z`) に向けます。



ノート:

カメラのローカル座標系は視線の向きを Z 軸、カメラ自体の上下を Y 軸とした物です。



11.4 カメラの画角について

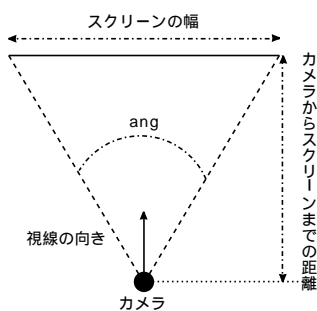
画角は下記の関数を用いる事によって自由に設定する事が出来ます。

`NjSetCameraAngle(NJS_CAMERA *c, Angle ang)`

画角（視野の水平方向の角度）を `ang` に設定します。

カメラの画角はスクリーンと密接な関係に有り、実際にはカメラの位置からスクリーンまでの距離を求め設定されます。そのため `njSetCamera` を実行後、`njSetScreenDist` や `njSetPerspective` 関数を用いてスクリーンまでの距離を変更するとカメラの設定と矛盾が起こりますので注意して下さい。

下記はカメラとスクリーンとの関係を上から見下ろして表したものです。



11.5 カメラの深度について

ここで言うカメラの深度とは表示を行う際のニアクリッピング値とファークリッピング値を指します。

これらの値は下記の関数を用いて設定できます。

`NjSetCameraDepth(NJS_CAMERA *c, Float n, Float f)`

ニアクリッピング値、及びファークリッピング値を設定します。

Ninja の座標系では Z 軸手前が正の値となる為、両値には必ず負の値を設定するよう気を付けて下さい。どちらの値も - 1 より大きい値が設定された場合は - 1 に、 - 65535 より小さな値が設定された場合には - 65535 に値を切り詰め設定します。

これは `njClipZ` 関数で指定するものと全く同じ物です。そのため `NjSetCamera` 関数を実行後に `njClipZ` 関数を用いてニアクリップ値、ファークリップ値を変更するとカメラの設定と矛盾が起こりますので注意して下さい。

11.6 実際のプログラムでの使用方法について

カメラを使用する為の手順は以下の通りです。

- ①マトリクススタックを用意し初期化する。
- ②カメラを用意し初期化する。
- ③カメラの各パラメータを好きな値に設定する。
- ④使用するカメラをセットする。
- ⑤モデル等の描画を行う。

(以下の繰り返しとなります。なお、VIEW との併用は一切の動作を保証しません)
カメラは下記の関数を用いてセットします。

`njSetCamera(NJS_CAMERA *c)`

この関数はカメラ構造体の各パラメータからマトリクスを作成し、マトリクススタックのベースに反映させます。従って、この関数を実行する前に必ずマトリクススタックの初期化を済ませておくよう注意して下さい。

プログラム例)

原点に置いたオブジェクトの回りを視線をオブジェクトに固定したまま回転します。

```
NJS_MATRIX mat[10]
NJS_CAMERA camera;

njInitMatrix(mat, 10, 0);
njInitCamera(&camera);
njTranslateCameraPosition(&camera, 1000.f, 0.f , 0.f);
njPointCameraInterest(&camera, 0.f, 0.f, 0.f);

while(1){
    njSetCamera(&camera);
    /* ここでワールド座標の原点にオブジェクトを描画 */
    njRotateCameraPositionY(&camera, NJM_DEG_ANG(1));
    njPointCameraInterest(&camera, 0.f, 0.f, 0.f);
}
```

	ベーシックフォーマット				チャンクフォーマット		
種類	Normal	Fast	Simple	Easy	Cnk	CnkSimple	CnkEasy
光源の数	複数	複数	単独	単独	複数	単独	単独
光源の種類	平行 / 点 / スポット / スペキュラ	平行 / 点 / スポット / スペキュラ	平行	平行	平行 / 点 / スポット / スペキュラ / アンビエント	平行 / (スペキュラ) + アンビエント	平行 + アンビエント
光源の色	RGB	インテンシティ	インテンシティ	RGB	RGB	RGB	RGB
光源無視				×			×
フラット				×			×
環境				×			×
両面				×			×
スペキュラ				×			×
アンビエント	×	×	×	×			×
マテリアル	ディフーズ / スペキュラ	ディフーズ / スペキュラ	ディフーズ / スペキュラ	無効	ディフーズ / アンビエント / スペキュラ	ディフーズ / スペキュラ	無効
制限		ライトの色はすべて白とみなし、デプスキューは機能しません。	最初に登録したライトを平行光源とみなし、光源の方向だけが影響。光源の色や強さは反映しません。デプスキューは機能しません。	描画属性として、グローテクスチャと、グローポリゴンのみ描画できます。また、多角形ポリゴンには対応していません。			描画属性として、グローテクスチャと、グローポリゴンのみ描画できます。

B.1 ファイル拡張子の説明

Ninja ファイルフォーマットの拡張子を説明します。Ninja はアスキーフォーマットとバイナリフォーマットをサポートします。アスキーフォーマット、バイナリフォーマットはそれぞれに一対一対応します。

拡張子のベース nj という文字列でありこれに中身を意味する 1 文字を付加します。アスキーでは nj の j を a に置き換えて表現します。

データ別拡張子

	バイナリフォーマット拡張子	アスキーフォーマット拡張子
Texlist	.njt	.nat
Model	.njd	.nad
Light	.njl	.nal
Camera	.njc	.nac
Motion	.njm	.nam
Shape Motion	.njs	.nas

その他の拡張子

	バイナリフォーマット拡張子	アスキーフォーマット拡張子
複数のデータを格納する場合	.nj	.nja
シーンファイル(アスキーのみ)	-	.nsc
リソースファイル(アスキーのみ)	-	.nre
テキストチャデータ(バイナリのみ)	.pvr	-
パレットデータ(バイナリのみ)	.pvp	-
マルチテキストチャ(バイナリのみ)	.pvm	-

注意事項

- ・ motion はライト、カメラに関しても Motion 構造体で管理されるため.njm(.nam) で表現されます。ただし shape に関しては通常の motion と組み合わせて使われるため拡張子をつけています。
- ・ 二種類以上のデータを格納した場合.nj(.nja) が使われます。現在のモデルコンバータはすべて.nja を出力していました。これには texlist (.nat) と model (.nad) が含まれています。
- ・ Chunk Model/Basic Model とともに.njd(.nad) を使います。ただし現在は texlist とともにモデルを出力しているため.nj(.nja) で出力されます。

- ・ Ninja におけるシーンファイルはシーンを構成するモデル、カメラ、ライト、モーションを括るためのファイル名リストです。NinjaViewer でのシーン表示のため単位にもなります。アスキー出力のみです。
- ・ リソースファイル.nre は汎用のリソース格納ファイルです。libnre ライブラリにより管理され Ninja グラフィックツールで必要とされるリソースの保存/読み出し/利用を一括管理します。現在テクスチャのコンバータオプション保存に適用されています。今後はコンバータオプションの保存、ユーザー設定データの保存などに拡張していく予定です。
- ・ バイナリーフォーマットはすべて iff チャンク形式で格納されます。各種データを同一のファイルにまとめて格納することが可能です。バイナリローダ nuBinary を使うことでコンパイルなしでユーザー malloc 領域へのデータ格納が可能です。詳細は『[バイナリフォーマット仕様書](#)』を参照のこと。
- ・ アスキーフォーマットはアスキーローダによる読み込み、もしくは NjDef.h をインクルードしてコンパイルすることで利用が可能です。
- ・ pvp ファイルはパレットデータを格納します。
- ・ pvm ファイルは複数のテクスチャを一括して扱うためのファイルフォーマットです。

バイナリフォーマット仕様書

C.1 概要

Ninja はアスキーフォーマットとバイナリフォーマットをサポートします。

データ別拡張子

	バイナリフォーマット拡張子	アスキーフォーマット拡張子
Textlist	.njt	.nat
Model	.njd	.nad
Light	.njl	.nal
Camera	.njc	.nac
Motion	.njm	.nam
Shape Motion	.njs	.nas

その他の拡張子

	バイナリフォーマット拡張子	アスキーフォーマット拡張子
複数のデータを格納する場合	.nj	.nja
シーンファイル(アスキーのみ)	-	.nsc
リソースファイル(アスキーのみ)	-	.nre

- ・ユーザーはコンバート時のオプション指定でアスキーフォーマット、バイナリフォーマットのどちらでも得ることができます。
- ・アスキーフォーマットはアスキーフォーマットローダ (gifts としてソースコード提供中) を使うことでコンパイルなしでロードすること、NjDef.h とともにコンパイルすることで利用できます。
- ・バイナリフォーマットはバイナリフォーマットローダ nuBinary (ライブラリに組み込まれるが gifts としてソースコード提供中) を使うことでコンパイルなしでユーザー malloc 領域にデータをロードできます。
- ・バイナリフォーマットは単一のメモリ領域に必要な構造体データを格納しデータ先頭アドレスを 0 番地としたポインタアドレスをデータ構造に出現するポインタメンバーに埋め込みます。この情報に加えデータの何バイト目にポインタメンバーがいるかを示すオフセット情報を持ちます (ポインタオフセット (POF))。
- ・データおよび POF は独立した iff チャンクとして定義されファイルに連続して配置されます。
- ・ローダはデータチャンクのバイトサイズだけのバッファをメモリをアロケートしここにデータを読み込みます。次に POF で与えられるポインタオフセットにアロケートされたメモ

りの先頭アドレスを足し込みます。これによりデータ内でのポインタのチェーンは正しく実アドレスを示すことになります。

- ・POF はオフセット番号の集合データですが単純にこれを格納するとデータ量が大きくなるため簡単な圧縮をします。現在提供している手法をチャンク POF0 と呼びます。今後より効率のより圧縮方法の適用、データサイズよりも展開速度の速い方法など必要に応じて POF1、POF2 などを必要であれば拡張していきます。POF の圧縮アルゴリズムが複数あってもローダでチャンクネームとして処理を分岐して展開するため互換を保ちつつ古いデータも問題なく展開することができます。
- ・データチャンクのすぐ後ろに格納される POF チャンクを消した場合データのロードはできません。かならずデータと POF チャンクはセットで扱ってください。
- ・バイナリ先頭アドレスにはデータの先頭データがくることを保証します。例えばモデルではモデルツリーのトップのオブジェクト構造体ポインタ、texlist では texlist ポインタ、モーションでは motion 構造体ポインタ。ユーザーは各データのポインタをフリーすることでバイナリデータを破棄できます。
- ・モーションにおいてバイナリフォーマットでは action 構造体の出力はしません。別ファイルに格納されたモデルへのポインタアドレスの解決ができないためです。カメラ、ライトのモーションも同様です。
- ・絶対アドレスのバイナリをサポートします。ユーザーに指定されたアドレス値をあらかじめ足し込みます。これは速度優先のストリーム再生などに利用します。特殊なバイナリなため取り扱いには注意が必要です。
- ・NJACnv というコンバータでアスキーからバイナリフォーマットへコンバートをサポートします。

C.2 バイナリ構造

バイナリ構造

Chunkname	bytesize
0番地からのポインタアドレスを設定したバイナリデータ	
Chunk POF0	bytesize
ポインタオフセットデータ	

データバイナリチャンク
<チャンクの種類>
'NJCM': nina chunk model tree
'NJBM': ninja basic model tree
'NJTL': ninja texlist
'NJLI': ninja light
'NJCA': ninja camera
'NMDM': ninja model motion
'NLIM': ninja light motion
'NCAM': ninja camera motion
'NSSM': ninja simple shape motion
POF0チャンク
ポインタオフセットアルゴリズムタイプ0

モデルデータチャンクと POF チャンクを離してはならない。このペアがセットであれば同一ファイルに複数のデータを格納することもできる。

現在デフォルト出力における texlist 付き Chunk Model の例を以下に示す。

(例) 拡張子.nj

'NJTL'	bytesize	texlistデータ
texlistデータ		
Chunk POF0	bytesize	
texlist POFデータ		Chunk Modelデータ
'NJCM'	bytesize	
Chunk Model データ		
Chunk POF0	bytesize	
Chunk Model POFデータ		

絶対アドレスバイナリ構造

'NJAD'	4(bytesize)	ユーザ指定アドレスを示す
ユーザ指定アドレス		
Chunkname	bytesize	データバイナリチャンク
ユーザ指定番地からのポインタ アドレスを設定したバイナリ データ		
<div><チャンクの種類> 'NJCM': nina chunk model tree 'NJBM': ninja basic model tree 'NJTL': ninja texlist 'NJLI': ninja light 'NJCA': ninja camera 'NMDM': ninja model motion 'NLIM': ninja light motion 'NCAM': ninja camera motion 'NSSM': ninja simple shape motion</div>		

NJAD チャンクとデータチャンクは離してはならない。ただしユーザー責任においてアドレスをあらかじめ知っている場合ははずすこと可能。先頭に NJAD を一つとその後ろに複数の同一種類のバイナリデータが格納することも可能。これはストリーム再生などで利用する。

C.3 Chunk 仕様

C.3.1 Chunk の種類

chunk name	説明
'NJCM'	Chunk Model データを格納します。データポインタの先頭にはトップの NJS_CNKOBJECT 構造体のアドレスが格納されます。
'NJBM'	Basic Model データを格納します。データポインタの先頭にはトップの NJS_OBJECT 構造体のアドレスが格納されます。
'NJTL'	texlist データを格納します。データポインタの先頭には NJS_TEXLIST 構造体の先頭アドレスが格納されます。
'NJLI'	ライトデータを格納します。データポインタの先頭には NJS_LIGHT 構造体の先頭アドレスが格納されます。
'NJCA'	カメラデータを格納します。データポインタの先頭には NJS_CAMERA 構造体の先頭アドレスが格納されます。
'NMDM'	モデルモーションデータを格納します。データポインタの先頭には NJS_MOTION 構造体の先頭アドレスが格納されます。NJS_ACTION データは含まれません。
'NLIM'	ライトモーションデータを格納します。データポインタの先頭には NJS_MOTION 構造体の先頭アドレスが格納されます。NJS_LACTION データは含まれません。
'NCAM'	カメラモーションデータを格納します。データポインタの先頭には NJS_MOTION 構造体の先頭アドレスが格納されます。NJS_CACTION データは含まれません。
'NSSM'	シェイプモーションデータを格納します。データポインタの先頭には NJS_MOTION 構造体の先頭アドレスが格納されます。NJS_ACTION データは含まれません。
'NJAD'	ユーザー指定の絶対アドレスを与えます。これに続くバイナリデータチャンクにはこの絶対アドレスからのポインタデータが格納されています。
'POF0'	ポインタオフセットデータ。バイナリデータをロード時にアロケートされたバッファの実アドレスにデータ内部のポインタ値を更新するためのポインタメンバーのロングワードオフセットを与えます。先頭から順次一つ前のオフセットからの相対オフセットで表現されます。またその値を格納する変数の大きさをオフセット値の大きさにあわせて char (1byte), short (2byte), long (4byte) サイズのを切り替えて格納しデータの表現効率を高めています。

C.3.2 POF0 のアルゴリズム

<step1>

バイナリイメージを作成。この時先頭アドレス (0 番地) からのロングワードオフセットをリストとして格納。

<step2>

値に変換。一つ前のオフセットとの引き算により差分を求める。ポインタは 4 バイトアライメントであるので 4 で割り単位をロングワードにする。

<step3>

オフセット値を表現する char, short, long の上位 2 ビットにフラグをつけ今のデータを char, short, long のいずれのサイズの値として処理すればいいかの設定をする。

<step4>

POF0 チャンクの書き出し上位 2 ビットはフラグとするため相対ロングワードオフセットが 6 ビットの最大値 64 より小さい値ならば char 値として 14 ビットの最大値 16384 より小さければ unsigned short 値としてそれ以上は unsigned long 値として出力する。ここで出力される short, long データはビッグインディアンとする。これにより先頭の 1 バイトにフラグが格納されフラグによるデータタイプ分岐が可能となる。

以下に POF0 データ書き込みのサンプルソースコードを示す。

```
#define NJ_POF_TYPE_PAD    0x00
#define NJ_POF_TYPE_CHAR  0x40
#define NJ_POF_TYPE_SHORT 0x80
#define NJ_POF_TYPE_LONG  0xc0
#define NJ_POF_CHAR_MASK  0xc0
void njPointerCashFlashType0(unsigned long prev, unsigned long current)
{
    unsigned long loffset=(current-prev)>>2;
    if (64 >loffset) {
        char ctmp = NJ_POF_TYPE_CHAR|(char)loffset;
        WriteBytes(&ctmp, sizeof(ctmp));
    } else if (16384 > loffset) {
        unsigned short stmp = (NJ_POF_TYPE_SHORT << 8)|(short)loffset;
        S_SWAP_PC(stmp, stmp); /* keep char order */
        WriteBytes(&stmp, sizeof(stmp));
    } else {
        unsigned long ltmp = (NJ_POF_TYPE_LONG << 24)|loffset;
        L_SWAP_PC(ltmp, ltmp); /* keep char order */
        WriteBytes(&ltmp, sizeof(ltmp));
    }
}
```

次のデータチャンクを 4 バイトアライメント調整するため POF チャンクの最後には最大 3 バイトのパディングが入る。パディングフラグは上位 2 ビットが 00 で示されるため POF 内部では char で 0 を書き込んでおけばパディングされる。



第 2 部

Shinobi 編

第 1 章

Nindows

1.1 概要

Nindows とは、ゲームを開発する上で欠かせないデバッグやパラメータの調整などを実機およびホストマシン上で行うための、簡易的に使用できる GUI システムです。

1.1.1 Nindows の特徴

- ・ Windows などの一般的な GUI と同じコントロールが使用できます。
- ・ Nindows API により、アプリケーションで自由にウィンドウを作成することができます。
- ・ 複雑なプログラミングをすることなく、Texture Viewer 等のデバッグに便利なユーティリティが利用できます。
- ・ Nindows で調整したパラメータ等はリアルタイムに確認できるため、ゲームバランス等の調整が迅速に行えます。

1.1.2 最も簡単な Nindows アプリケーションの作成

ここでは、既存の Ninja アプリケーションに Nindows を組み込む方法を解説します。

ソースファイルにわずか数行の変更を加えるだけで、簡単に Nindows の機能が利用可能になります。

1.1.3 Nindows の組み込み

① Ninja アプリケーションを準備する。

`njUserInit()`、`njUserMain()`、`njUserExit()` の各関数を持つソースファイルを準備します。

② Nindows のヘッダファイルをインクルードする

ソースファイルに次の 1 行を追加します。

```
#include <Nindows.h>
```

③ Nindows の初期化関数をコールする

`njInitTexture()` の呼び出し後に、次の 1 行を追加します。

```
nwInitSystem(numTextures, port);
```

numTextures はテクスチャメモリリストの数です。njInitTexture() で指定した値を指定します。Nindows のフォント用に 3 枚のテクスチャを使用するので、テクスチャメモリリストはアプリケーションで使用する枚数 + 3 を確保し、指定してください。port には、Nindows の操作に使用するコントロールポートを指定します。PDD_PORT_A0/B0/C0/D0 のいずれかを指定してください。

④Nindows の実行関数をコールする

njUserMain の最後の return NJD_USER_CONTINUE を、次のように変更します。

```
return nwExecute();
```

⑤Nindows の終了関数をコールする

njExitSystem() の呼び出し前に、次の 1 行を追加します。

```
nwExitSystem();
```

⑥Nindows のライブラリをリンクする

プロジェクトに Nindows.lib を加えます。

以上の作業により、

- ・ Nindows の Nindows ユーティリティの使用
- ・ Nindows API 関数の呼び出し

が可能になりました。

これらをまとめると、次のようなソースになります。

```
#include <shinobi.h>
#include <Nindows.h>

NJS_TEXMEMLIST tex[1000];

void njUserInit(void)
{
    njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1);
    njInitVertexBuffer(500000, 0, 500000, 0);
    njInitTexture(tex, 1000);
    nwInitSystem(1000, PDD_PORT_A0);
}

Sint32 njUserMain(void)
{
    return nwExecute();
}

void njUserExit(void)
{
    nwExitSystem();
    njExitSystem();
}
```


1.2 Nindows の操作とNindows ユーティリティ

1.2.1 Nindows の操作

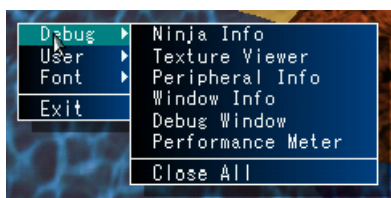
前の節で Nindows の組み込みが終了しました。Nindows を組み込んだアプリケーション（以下 Nindows アプリ）を実行すると、画面上にマウスカーソルが表示され、マウスの操作に応じて画面上を動きます。

システムメニュー

デスクトップ上でマウスの右ボタンをクリックすると、ポップアップメニューが表示されます。これをシステムメニューと呼び、ここからメニューを選択して Nindows の持つ Nindows ユーティリティを使用します。システムメニューには以下の項目があります。

システムメニューのメニュー項目一覧

メニュー項目	説明
Debug	Nindows ユーティリティメニューをポップアップします。
User(Undefined)	ユーザー定義のメニューをポップアップします。Nindows 初期化時には、ユーザーメニューは登録されていないため、淡色表示となり選択できません。
Font	フォントを変更します。
Exit	アプリケーションを終了します。



右ボタンでシステムメニューを表示し、「Debug」を選択した状態

1.2.2 Nindows ユーティリティ

システムメニューから「Debug」を選択した時に表示されるメニュー項目が、Nindows の持つ Nindows ユーティリティです。Nindows ユーティリティには以下のものがあります。

Nindows ユーティリティ一覧

名称	説明
Ninja Info	Ninja ライブラリのバージョン番号、その他情報が表示されます。
Texture Viewer	読み込まれているテクスチャを全て表示することができます。
Peripheral Info	ペリフェラルの情報が表示されます。
Window Info	アクティブなウィンドウの情報が表示されます。
Debug Window	デバッグ文字列を表示するのに便利なウィンドウです。
Performance Meter	アプリケーションの描画パフォーマンスがわかります。

Ninja Info ウィンドウ



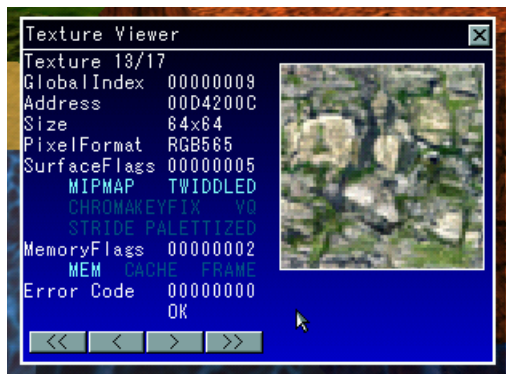
Ninja Info ウィンドウ

Ninja Info ウィンドウには以下の情報が表示されます。

Ninja Info ウィンドウの情報

表示	内容
Ninja Ver.	Ninja ライブラリのバージョン番号です。
Nindows Ver.	Nindows のバージョン番号です。
Model Vertex	投入頂点数です。
Calc Vertex	演算頂点数です。
Set Vertex	描画頂点数です。
Texture Memory	テクスチャメモリの空き容量と全容量です。詳細はグラフィック関連ドキュメントを参照してください。
Vertex Buffer Used	njInitVertexBuffer() で指定された頂点バッファの最大容量と最大使用量です。使用量が最大容量をオーバーするような場合は、描画を減らすか、頂点バッファを多く確保してください。詳細はグラフィック関連ドキュメントを参照してください。
Native Buffer	内部で使用する描画用バッファ (VRAM) の最大容量と、現在の使用量、最大使用量です。使用量が最大容量をオーバーするような場合は、描画を減らしてください。詳細はグラフィック関連ドキュメントを参照してください。

Texture Viewer ウィンドウ



Texture Viewer ウィンドウ

登録されているテクスチャをすべて見ることができます。4つのボタンでテクスチャを切り替えることができます。Texture Viewer ウィンドウには以下の情報が表示されます。

Texture Viewer ウィンドウの情報

表示	内容
Texture	テクスチャ番号と総テクスチャ数
GlobalIndex	グローバルインデックス
Address	テクスチャアドレス
Size	テクスチャサイズ
PixelFormat	ピクセルフォーマット
SurfaceFlags	サーフェースフラグ ハイライト表示されている項目が、そのテクスチャに設定されているフラグです。フラグの詳細はテクスチャ関連ドキュメントを参照してください。
MemoryFlags	メモリフラグ 同じく、そのテクスチャに設定されているフラグです。フラグの詳細はテクスチャ関連ドキュメントを参照してください。
Error Code	テクスチャをロードする際に発生したエラーコードです。ロードに成功した場合は「OK」が表示されます。エラーコードの詳細はテクスチャ関連ドキュメントを参照してください。

Peripheral Info ウィンドウ



Peripheral Info ウィンドウ

ペリフェラル（入力装置）の情報を表示するウィンドウです。「<」、「>」ボタンで表示するコントロールポートを選択できます。Peripheral Info ウィンドウには以下の情報が表示されます。

Peripheral Info ウィンドウの情報

表示	内容
Port	コントロールポート名
Dev	ポートに接続されているペリフェラル名
ON	押されているボタン情報
OFF	押されていないボタン情報
PRESS	押された瞬間のボタン情報
RELEASE	放された瞬間のボタン情報
X1,Y1,X2,Y2	アナログレバー X1,Y1,X2,Y2 の値
L,R	アナログトリガー L,R の値

Window Info ウィンドウ



Peripheral Info ウィンドウ

マウスカーソルの下にあるウィンドウ (アクティブウィンドウ) の情報が表示されます。Nindows API を使用してアプリケーション独自のウィンドウを作成する場合などのデバッグに利用できます。Peripheral Info ウィンドウには以下の情報が表示されます。

Peripheral Info ウィンドウの情報

表示	内容
	アクティブウィンドウのタイトル文字列
x,y	ウィンドウのクライアント領域の左上座標
w,h	ウィンドウのクライアント領域のサイズ

注意

x, y 座標は必ずしも画面上の絶対座標ではなく、ウィンドウスタイルに依存します。

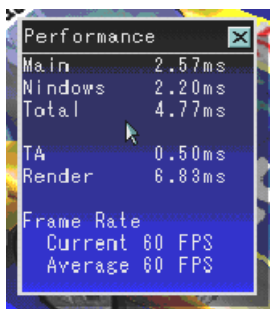
デバッグ用ウィンドウ



Debug 用ウィンドウ

このウィンドウには、最初は何も表示されません。nwDebugPrintf() 関数を用いて、デバッグ文字を表示してください。この関数は 標準の printf() 関数と同様に使用できます。詳細はエディットウィンドウの章を参照してください。

Performance Meter ウィンドウ



Performance Meter ウィンドウ

アプリケーションのパフォーマンスを表示します。ウィンドウには以下の情報が表示されます。

Peripheral Info ウィンドウの情報

表示	内容
Main	njUserMain() 関数の先頭(正しくはnjWaitVSync() 関数の終了時)から、nwExecute() 関数呼び出しまでの時間をミリ秒で表示します。
Nindows	nwExecute() 関数の処理時間をミリ秒で表示します。
Total	Main、Nindows の処理時間の合計を表示します。 この値が約 16.7ms を超えると処理落ちすることになります。
TA	タイル分割処理の時間を示します。 この値が約 16.7ms を超えると処理落ちすることになります。
Render	PowerVR2 の描画時間を示します。 この値が約 16.7ms を超えると処理落ちすることになります。
Frame Rate	現在のフレームレート、過去 1 秒間の平均フレームレートを表示します。

1.2.3 フォントの変更

システムメニューから「Font」を選択することで、通常サイズと大きいサイズのフォントを選択することができます。NTSC モニタに出力した場合など、通常サイズでは文字が読みにくい場合には大きいサイズのフォントを選択してください。

フォントサイズ変更に伴うウィンドウのリサイズには対応していません。フォントを変更する場合、なるべくアプリケーション実行後、他のウィンドウを開く前にフォントを変更するようにしてください。

1.3 ウィンドウ

1.3.1 概要

ウィンドウは、Nindows の最も基本的な要素です。アプリケーションは、まずウィンドウを作成し、描画コールバック関数を指定することにより、ウィンドウのクライアント領域への描画を行います。また、子ウィンドウとして別のウィンドウや、ボタン、スクロールバーなどのコントロールを作成し、制御することができます。

ウィンドウの種類とウィンドウクラス

ウィンドウには次の種類があり、ウィンドウクラスとして区別します。

ウィンドウの種類とウィンドウクラス

ウィンドウクラス	ウィンドウの種類
NWD_WC_WIN	通常のウィンドウ
NWD_WC_SCRWIN	クライアント領域がスクロール可能なウィンドウ
NWD_WC_EDITWIN	エディットウィンドウ
NWD_WC_SCRBAR	スクロールバーコントロール
NWD_WC_BUTTON	ボタンコントロール
NWD_WC_MENUWIN	メニューウィンドウ

ここでは、おもに通常のウィンドウについて解説します。

1.3.2 ウィンドウの作成

例として、クライアント領域にカウンタを表示するウィンドウを、デスクトップに作成します。ウィンドウの作成には、関数 `nwCreateWindow()` を使用します。

```
static void test_window_callback(NHWND hWnd)
{
    static int count = 0;
    char buf[256];
    sprintf(buf, "Count = %d", count++);
    nwTextOut(hWnd, 1, 1, buf);
}

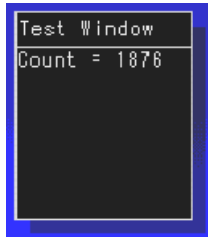
Bool create_test_window(void)
{
    NHWND hWnd = nwCreateWindow(NWD_WC_WIN,
                                "Test Window",
                                NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                                50, 50, 100, 100,
                                NULL);
    if (!hWnd) return FALSE;

    hWnd->clientDraw = test_window_callback;
    return TRUE;
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
}
```

```
        create_test_window();
    }
```

これで、画面の指定位置に次のようなウィンドウが表示され、カウントの表示がインクリメントされていくはずです。なお、ウィンドウが表示されるのは `nwExecute()` の呼び出し、および `njUserMain()` からのリターン後、次のフレームからになります。



カウンタを表示するウィンドウ

また、作成したウィンドウを破棄するには、関数 `nwDestroyWindow()` を呼び出すか、あるいはマウスを操作して、キャプションバーにあるクローズボックスをクリックします（クローズボックスはウィンドウスタイルに `NWD_WS_CONTROL` を指定したウィンドウのみが持つことができます）。

```
nwDestroyWindow(hWnd);
```

1.3.3 子ウィンドウの作成

`nwCreateWindow()` の最後の引数は、親ウィンドウのウィンドウハンドルです。先程の例では、この引数に `NULL` を指定したため、親を持たない（正確にはデスクトップウィンドウを親に持つ）ウィンドウを作成しました。ここでは、親ウィンドウと、その子ウィンドウを作成する例を示します。

```
static void test_window_callback(NHWND hWnd)
{
    static int count = 0;
    char buf[256];
    sprintf(buf, "Count = %d", count++);
    nwTextOut(hWnd, 1, 1, buf);
}

Bool create_test_window(void)
{
    NHWND hWndParent, hWnd;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                               "Parent Window",
                               NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                               50, 50, 100, 100,
                               NULL);
    if (!hWndParent) return FALSE;

    hWndParent->clientDraw = test_window_callback;

    //以下は子ウィンドウの作成
    hWnd = nwCreateWindow(NWD_WC_WIN,
```

```

        "Child Window",
        NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_OFFSET
        20, 20, 60, 60,
        hWndParent);
    if (!hWnd) return FALSE;

    return TRUE;
}

```

親ウィンドウを、関数 `nwDestroyWindow()` またはマウスの操作などにより破棄すると、そのウィンドウを親に持つ全ての子ウィンドウも自動的に破棄されます。この例では親ウィンドウ `hWndParent` を破棄すれば、子ウィンドウ `hWnd` も破棄されます。

```
nwDestroyWindow(hWndParent);
```

なお、スクロールバー (クラス `NWD_WC_SCRBAR`)、ボタン (クラス `NWD_WC_BUTTON`)、およびメニュー (クラス `NWD_WC_MENUWIN`) は、親ウィンドウとして指定することはできません (子ウィンドウを持つことはできません)。ただし、メニューの場合は子メニュー (サブメニュー) を持つことは可能です。詳細は「メニュー」を参照してください。

1.3.4 ウィンドウに関連するパラメータ

ウィンドウハンドル `NWHWND` は、実際には構造体 `NWS_WIN` へのポインタです。

作成したウィンドウのウィンドウハンドルを介してこの構造体のメンバを直接設定することにより、ウィンドウにさまざまな動作をさせることができます。ここでは、知っておくと便利なメンバと、その代表的な利用方法について解説します。

クライアント描画コールバック関数 `hWnd->clientDraw`

ウィンドウ作成例でも使用した、最も代表的なメンバです。

通常、このメンバに何等かのコールバック関数アドレスを設定し、そのコールバック関数の中でクライアント領域への描画等の処理を行います。

デストラクタ `hWnd->destructor`

このメンバに関数アドレスを設定しておくと、ウィンドウが破棄される時にコールバックされます。

ユーザーデータ `hWnd->param1, hWnd->param2`

アプリケーションで自由に設定、参照できる、`Sint32` 型のメンバです。

ユーザーデータ `hWnd->userBuf`

より多くのユーザーデータを格納したい場合、そのデータのアドレスを指定します。データバッファは、アプリケーションで独自に確保してください。

1.3.5 ウィンドウサポート関数のサンプルと説明

Windows API には、上記の `nwCreateWindow()` と `nwDestroyWindow()` の他にも、ウィンドウの管理をサポートする関数が数多く用意されています。以下は、コントローラを使ってウィンドウを動かすサンプルです。


```
Sint32 njUserMain(void)
{
    Sint32 x, y, w, h, style;
    NHHWND hWnd;
    NJS_PERIPHERAL* per = njGetPeripheral(NJD_PORT_A0);

    // ウィンドウが存在するかどうか調べる
    if (!(hWnd = nwFindWindow(NULL, "Test Window"))) goto skip;

    // ウィンドウのクライアント領域の座標とサイズを取得する
    nwGetWindowPos(hWnd, &x, &y);
    nwGetWindowSize(hWnd, &w, &h);
    if (per->press & NJD_DGT_ST) {
        //スタートボタンを押すと、ウィンドウを画面の中心に移動する。
        nwSetWindowPos(hWnd, 320 - w / 2, 240 - h / 2);
    }
    if (per->on & NJD_DGT_TA) {
        // Aボタンを押しているとき、十字キーでウィンドウを移動できる。
        if (per->on & NJD_DGT_KU) y--;
        if (per->on & NJD_DGT_KD) y++;
        if (per->on & NJD_DGT_KL) x--;
        if (per->on & NJD_DGT_KR) x++;
        nwSetWindowPos(hWnd, x, y);
    }
    if (per->on & NJD_DGT_TB) {
        // Bボタンを押しているとき、十字キーでウィンドウのサイズを変更できる。
        if (per->on & NJD_DGT_KU) h--;
        if (per->on & NJD_DGT_KD) h++;
        if (per->on & NJD_DGT_KL) w--;
        if (per->on & NJD_DGT_KR) w++;
        nwSetWindowSize(hWnd, w, h);
    }
    if (per->press & NJD_DGT_TX) {
        // Xボタンを押すと、ウィンドウのキャプションバーのON/OFFを切り替える。
        nwGetWindowStyle(hWnd, &style);
        if (style & NWD_WS_CAPTION) {
            nwSetWindowStyle(hWnd, ~NWD_WS_CAPTION, 0);
        } else {
            nwSetWindowStyle(hWnd, 0xffffffff, NWD_WS_CAPTION);
        }
    }
}

skip:
return nwExecute();
}
```

1.4 スクロールウィンドウ

1.4.1 概要

スクロールウィンドウは、ウィンドウクラス `NWD_WC_SCRWIN` を持つウィンドウです。スクロールウィンドウは、通常のウィンドウにはない、クライアント領域の表示内容をスクロールできるという機能を持っています。

1.4.2 スクロールウィンドウの作成

スクロールウィンドウの作成には、通常のウィンドウの作成と同じ関数 `nwCreateWindow()` を使います。ウィンドウクラスに `NWD_WC_SCRWIN` を指定することにより、スクロールウィンドウが作成できます。

```
static void scroll_window_callback(NWHWND hWnd)
{
    nwTextOut(hWnd, 1, 1, "Scroll Window Sample");
}

Bool create_scroll_window(void)
{
    NWHWND hWnd = nwCreateWindow(NWD_WC_SCRWIN,
                                  "Scroll Window",
                                  NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                                  50, 50, 100, 100,
                                  NULL);
    if (!hWnd) return FALSE;

    hWnd->clientDraw = scroll_window_callback;
    return TRUE;
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    create_scroll_window();
}
```

こうして作成したスクロールウィンドウの外観は通常のウィンドウと同じですが、クライアント領域をマウス操作でスクロールできます。クライアント領域にマウスカーソルを置いて左ボタンを押し、そのままマウスを動かすことにより、自由にスクロールさせることができます。

ウィンドウ作成直後は、クライアント領域は上下左右自由にスクロールできますが、上下方向のみ、左右方向のみスクロール可能にすることもできます。

上下方向のみスクロール可能にする設定

```
nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL);
```

左右方向のみスクロール可能にする設定

```
nwScrWinEnableScroll(hWnd, NWD_ES_HORIZONTAL);
```

上下左右にスクロール可能にする設定

```
nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL | NWD_ES_HORIZONTAL);
```

どの方向にもスクロール不可にする設定

```
nwScrWinEnableScroll(hWnd, 0);
```

1.5 エディットウィンドウ

1.5.1 概要

エディットウィンドウは、ウィンドウクラス `NWD_WC_EDITWIN` を持つウィンドウであり、スクロールウィンドウ (ウィンドウクラス `NWD_WC_SCRWIN`) の機能を含んでいます。エディットウィンドウには、以下の特徴があります。

- ・テキストバッファを持ち、テキストを設定して自動表示させることができる。
- ・複数行のテキストを表示させることができる。
- ・ウィンドウのクライアント領域をスクロールさせることができる。

Nindows の Nindows ユーティリティ「Debug Window」は、このエディットウィンドウとして実現されています。

1.5.2 エディットウィンドウの作成と使用方法

エディットウィンドウを作成するには、関数 `nwCreateEditWindow()` を呼び出します。

```
NWHWND hWnd;  
  
hWnd = nwCreateEditWindow(500, "Edit Window 1",  
    NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_THICKFRAME | NWD_WS_SHADING,  
    50, 50, 150, 100, NULL);
```

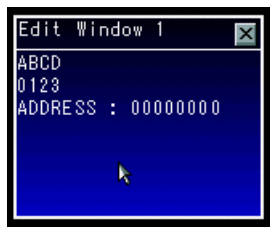
次に、このウィンドウに文字列を追加してみましょう。それには、関数 `nwEditWinAddString()` を使います。

```
nwEditWinAddString(hWnd, "ABCD¥n");  
nwEditWinAddString(hWnd, "0123¥n");
```

さらに、`printf()` と同様に使用できる関数として、`nwEditWinPrintf()` も用意されています。

```
nwEditWinPrintf(hWnd, "ADDRESS : %08X¥n", 0);
```

ここまでの作業で、画面上に次のようなウィンドウが表示されます。



エディットウィンドウの作成例

エディットウィンドウに次々に文字列を追加していくと、ウィンドウのクライアント領域に表示しきれなくなった時点で、自動的にスクロールします。

また、エディットウィンドウはスクロールウィンドウの機能も持っているため、マウスド

ラッグによりクライアント領域の表示内容を自由にスクロールさせることができます。
追加した文字列がバッファに入りきらなくなると、バッファの先頭から消去されていきます。

エディットウィンドウを破棄するには、他のウィンドウ同様、関数 `nwDestroyWindow()` を使います。

```
nwDestroyWindow(hWnd);
```

注意

現バージョンの Ninows では、関数 `nwEditWinAddString()`, `nwEditWinPrint()` で設定する文字列の最後には改行文字 `'\n'` が必要です。改行文字を付加しなかった場合、次にこれらの関数を用いて改行文字を含む文字列を設定するまで、先に設定した文字列は表示されませんので注意してください。

1.6 スクロールバーコントロール

1.6.1 概要

スクロールバーは、各種の数値パラメータを調整するのに非常に適したコントロールです。例えば、背景やモデルの色を変更したり、オブジェクトの移動スピードを調整するなど、さまざまな用途で使用できます。

1.6.2 スクロールバーコントロールの作成

モデルのマテリアル (NJS_ARGB 構造体) を、スクロールバーを使って変更する例を示します。

```
static NJS_ARGB argb = { 0.f, 0.f, 0.f, 0.f};

static NWS_SCROLLBARINFO material_scroll_info[] = {
    /* caption    data    min    max    line page pos */
    {"Alpha Scroll", {&argb.a, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Red Scroll",   {&argb.r, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Green Scroll", {&argb.g, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Blue Scroll",  {&argb.b, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
};

static NWS_SCROLLBARLIST material_scroll_list = {
    sizeof(material_scroll_info) / sizeof(NWS_SCROLLBARINFO),
    NWD_WS_SB_HORZ, 50, 3, 200, 14,
    material_scroll_info,
};

static void draw_material_window_callback(NWHWND hWnd)
{
    nwTextOut(hWnd, 2, 2, "A %.3f", argb.a);
    nwTextOut(hWnd, 2, 16, "R %.3f", argb.r);
    nwTextOut(hWnd, 2, 30, "G %.3f", argb.g);
    nwTextOut(hWnd, 2, 44, "B %.3f", argb.b);
}

static void create_material_window(void)
{
    NWHWND hWnd = nwCreateWindow(NWD_WC_WIN, "Material Window",
                                NWD_WS_CAPTION | NWD_WS_SHADING |
                                NWD_WS_BORDER | NWD_WS_CONTROL,
                                40, 382, 260, 64, NULL);
    hWnd->clientDraw = draw_material_window_callback;

    nwCreateScrollBarArray(&material_scroll_list, hWnd);
}

void njUserInit(void)
{
    :
    create_material_window();
}

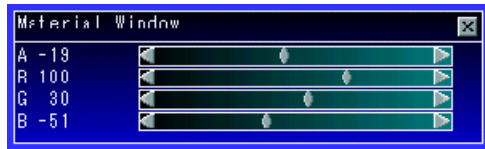
Sint32 njUserMain(void)
{
    :
```

```

        :
        njSetconstantMaterial(&argb);
        njDrawObject(OBJECT);
        return nwExecute();
    }

```

この例では、まずスクロールバーの親となるウィンドウ「Material Window」を作成し、4つのスクロールバーを子ウィンドウとして作成しています。このようにするのが一般的と言えます。これにより、次のようなウィンドウが表示されます。



スクロールバーコントロールの作成例

ここで、スクロールバーつまみを操作すると、それに応じて NJS_ARGB 構造体のメンバ a,r,g,b の値も変化します。

「Material Window」を破棄すると、子ウィンドウである4つのスクロールバーも自動的に破棄されます。

1.6.3 低レベルスクロールバー関数を利用したスクロールバーコントロールの作成

ここでは、既に説明した `nwCreateScrollBarArray()` ではなく、もっと低レベルの関数を使用して、全く同じマテリアルウィンドウを作成する方法を解説します。必要ないと思われる方は次節に進んでください。

```

static NJS_ARGB argb = { 0.f, 0.f, 0.f, 0.f};

static NWS_DATA dt[] = {
    {(void*)&argb.a, NWD_DT_FLOAT},
    {(void*)&argb.r, NWD_DT_FLOAT},
    {(void*)&argb.g, NWD_DT_FLOAT},
    {(void*)&argb.b, NWD_DT_FLOAT},
};

static void draw_material_window_callback(NWHWND hWnd)
{
    nwTextOut(hWnd, 2, 2, "A %.3f", argb.a);
    nwTextOut(hWnd, 2, 16, "R %.3f", argb.r);
    nwTextOut(hWnd, 2, 30, "G %.3f", argb.g);
    nwTextOut(hWnd, 2, 44, "B %.3f", argb.b);
}

static void create_material_window(void)
{
    NWHWND hScl;
    NWHWND hWnd = nwCreateWindow(NWD_WC_WIN, "Material Window",
                                NWD_WS_CAPTION | NWD_WS_SHADING |
                                NWD_WS_BORDER | NWD_WS_CONTROL,
                                40, 382, 260, 64, NULL);
    hWnd->clientDraw = draw_material_window_callback;
}

```

```

hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Alpha Scroll", 80, 3, 200, 11, hWnd);
nwSetScrollBarData(hScl, &dt[0]);
nwSetScrollBarRange(hScl, -256.0f, 255.0f);
nwSetScrollBarPos(hScl, 0.f);
nwSetScrollBarLineMove(hScl, 1.f);
nwSetScrollBarPageMove(hScl, 10.f);

hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Red Scroll", 80, 17, 200, 25, hWnd);
nwSetScrollBarData(hScl, &dt[1]);
nwSetScrollBarRange(hScl, -256.0f, 255.0f);
nwSetScrollBarLineMove(hScl, 1.f);
nwSetScrollBarPageMove(hScl, 10.f);
nwSetScrollBarPos(hScl, 0.f);

hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Green Scroll", 80, 31, 200, 39, hWnd);
nwSetScrollBarData(hScl, &dt[2]);
nwSetScrollBarRange(hScl, -256.0f, 255.0f);
nwSetScrollBarPos(hScl, 0.f);
nwSetScrollBarLineMove(hScl, 1.f);
nwSetScrollBarPageMove(hScl, 10.f);

hScl = nwCreateScrollBar(NWD_WS_SB_HORZ, "Blue Scroll", 80, 45, 200, 53, hWnd);
nwSetScrollBarData(hScl, &dt[3]);
nwSetScrollBarRange(hScl, -256.0f, 255.0f);
nwSetScrollBarPos(hScl, 0.f);
nwSetScrollBarLineMove(hScl, 1.f);
nwSetScrollBarPageMove(hScl, 10.f);
}

```

以上で、先ほどの図と全く同じ機能、外観を持つ「Material Window」が作成できます。

1.7 ボタンコントロール

1.7.1 概要

ボタンは、クリックすることによりコールバックを発生する、様々な目的に利用できるコントロールです。アプリケーションで保持しているフラグ変数のトグル切り替え、複数のオブジェクトから1つを選択する際のインタフェース等、工夫しだいで便利に利用できます。

1.7.2 ボタンコントロールの作成

例として、「Back」、「Next」ボタンにより、環境マッピングに使用するテクスチャを選択するサンプルを作成します。

```
#define TEXTURES 4
static int texno = 0;

static void test_window_callback(NHWND hWnd)
{
    char buf[256];
    sprintf(buf, "Texture=%d", texno);
    nwTextOut(hWnd, 1, 1, buf);
}

static void button_callback_back(NHWND hWnd)
{
    texno--;
    if (texno < 0) texno = TEXTURES - 1;
}

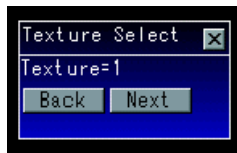
static void button_callback_next(NHWND hWnd)
{
    texno++;
    if (texno >= TEXTURES) texno = 0;
}

Bool create_test_window(void)
{
    NHWND hWndParent;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Texture Select",
                                NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                                50, 50, 128, 48,
                                NULL);
    if (!hWndParent) return FALSE;
    hWndParent->clientDraw = test_window_callback;
    nwCreateButton(button_callback_back, "Back", 3, 20, 48, 13, hWndParent);
    nwCreateButton(button_callback_next, "Next", 56, 20, 48, 13, hWndParent);
    return TRUE;
}

Sint32 njUserMain(void)
{
    :
    :
    njSetTexture(&texlist[texno]);
    njDrawObject(OBJECT);
    return nwExecute();
}
```

```
}
```

以上で、次のようなテクスチャ選択ウィンドウが表示されます。



テクスチャ選択ウィンドウの作成例

「Back」、「Next」ボタンをクリックすると、それぞれ指定のコールバック関数が呼び出され、変数 `texno` の値が変更されます。それに応じて環境マッピングに使用されるテクスチャも変更されます。

テクスチャ選択ウィンドウを破棄すると、子ウィンドウである2つのボタンも自動的に破棄されます。

1.7.3 ボタンの有効化と無効化

先程のサンプルでは、「Back」、「Next」のボタンは常に有効で、クリックすると必ずコールバックが働きます。しかし実際には、状況に応じてボタンを使用不可にしたりする必要があるでしょう。では、先程のサンプルを改造して、そのような処理を追加してみましょう。

追加する処理は、ボタン作成時に、「Back」ボタンを無効にする処理、ボタンの親ウィンドウのコールバック関数中でテクスチャ番号を調べ、それに応じて2つのボタンの有効、無効を切り替える処理です。そしてそのためには、作成したボタンのウィンドウハンドルをグローバル変数に格納しておくといでしょう。

ボタンの有効、無効を設定するには、関数 `nwEnableButton()` を使用します。

ボタンを有効にする処理

```
nwEnableButton(button, TRUE);
```

ボタンを無効にする処理

```
nwEnableButton(button, FALSE);
```

無効化されたボタンは、ボタンの文字が淡色表示され、クリックしてもボタンアニメーションやコールバックが働かなくなります。

```
#define TEXTURES 4
static int texno = 0;
static NWHWND button_back;
static NWHWND button_next;

static void test_window_callback(NWHWND hWnd)
{
    char buf[256];
    sprintf(buf, "Texture=%d", texno);
    nwTextOut(hWnd, 1, 1, buf);
}
```

```
// テクスチャ番号が 0 なら、「 Back 」ボタンを使用不可にする
if (texno == 0) nwEnableButton(button_back, FALSE);
else nwEnableButton(button_back, TRUE);

// テクスチャ番号が 3 なら、「 Next 」ボタンを使用不可にする
if (texno == TEXTURES - 1) nwEnableButton(button_next, FALSE);
else nwEnableButton(button_next, TRUE);
}

static void button_callback_back(NHWND hWnd)
{
    texno--;
    if (texno < 0) texno = 0;
}

static void button_callback_next(NHWND hWnd)
{
    texno++;
    if (texnum >= TEXTURES) TEXTURES - 1;
}

Bool create_test_window(void)
{
    NHWND hWndParent;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Texture Select",
                                NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                                50, 50, 128, 48,
                                NULL);
    if (!hWndParent) return FALSE;

    hWndParent->clientDraw = test_window_callback;

    button_back = nwCreateButton(button_callback_back,
                                "Back", 3, 20, 48, 13, hWndParent);
    button_next = nwCreateButton(button_callback_next,
                                "Next", 56, 20, 48, 13, hWndParent);

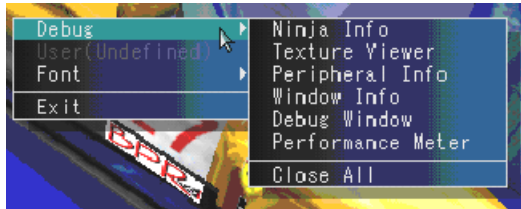
    // 最初はテクスチャ番号が 0 なので、「 Back 」ボタンを使用不可にする
    nwEnableButton(button_back, FALSE);
    return TRUE;
}
```

1.8 メニュー

1.8.1 概要

Nindows は、一般的な GUI システムにおけるポップアップメニューを実現するための API を備えています。

Nindows において、最も代表的なメニューはシステムメニューですが、このメニュー項目の中に、「User(undefined)」という、淡色表示されている項目があります。



システムメニューの「User(undefined)」項目

この項目は、ユーザー定義のメニューを設定するための項目です。メニューテーブルを作成してこの項目に登録することにより、簡単にユーザー定義のメニューが使用できるようになります。この章では、メニューテーブルの作成方法と、作成したメニューテーブルの登録方法を解説します。また、システムメニューに登録せずに直接ポップアップウィンドウを作成する方法も合わせて解説します。

1.8.2 メニューテーブルの作成と登録

メニューテーブルは、構造体 NWS_MENUTABLE の配列として実現します。以下は、1つの項目からなる、最も簡単なメニューテーブルの例です。

```
static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

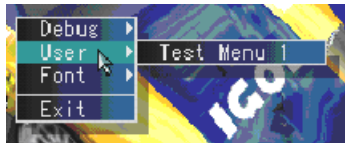
static NWS_MENUTABLE menu_table[] = {
    // 項目 1
    {
        NWD_MF_NORMAL, // ノーマルメニュー項目であることを示すフラグ
        "Test Menu 1", // 項目文字列
        menu_callback, // 項目が選択された時に呼び出されるコールバック関数
        0,             // コールバック関数に渡されるパラメータ
    },
    // テーブルの終了を表す
    {
        NWD_MF_NULL, // テーブル終了
        "",
        NULL,
        0
    },
};
```

このメニューテーブルをシステムメニューの「User(undefined)」項目に登録するには、関数

nwSetUserMenu() を使います。

```
void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    nwSetUserMenu(menu_table);
}
```

この関数の呼び出しにより、淡色だった「User(undefined)」項目は、「User >」という表示に変わり、登録した「Test Menu 1」がサブメニューとしてポップアップします。



ユーザーメニューを登録した状態 (1)

このメニューが選択されると、メニューテーブルで設定したコールバック関数 menu_callback() がコールバックされます。menu_callback() では特に処理をしていないため、このメニューを選択しても何も起こりません。このコールバック関数については次節で解説します。

もう少し複雑なメニューテーブルの例も見てみましょう。

```
static void menu_callback(NHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE menu_table[] = {
    // 項目 1
    {
        NWD_MF_NORMAL, // ノーマルメニュー項目であることを示すフラグ
        "Test Menu 1", // 項目文字列
        menu_callback, // 項目が選択された時に呼び出されるコールバック関数
        0,             // コールバック関数に渡されるパラメータ
    },
    // 項目 2
    {
        NWD_MF_SEPARATOR, // セパレータ
        "",
        NULL,
        0,
    },
    // 項目 3
    {
        NWD_MF_POPUP, // ポップアップメニューであることを示すフラグ
        "Test Popup 1", // 項目文字列
        NULL,
        (Sint32)submenu_table, // サブメニューのメニューテーブルアドレス
    },
    // テーブルの終了を表す
    {
        NWD_MF_NULL, // テーブル終了
        "",
        NULL,
        0,
    },
}
```

```

    }
};

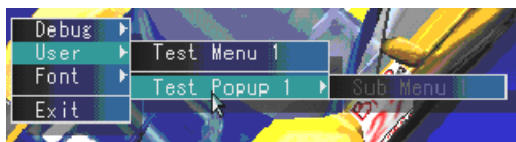
static void submenu_callback(NHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE submenu_table[] = {
    // 項目 1
    {
        NWD_MF_NORMAL | NWD_MF_GRAYED, // ノーマルメニュー項目、選択不可
        "Sub Menu 1",                  // 項目文字列
        submenu_callback,              // コールバック関数
        0,                             // コールバック関数に渡されるパラメータ
    },
    // 項目 2
    {
        NWD_MF_NULL,                  // メニュー終了
        "",
        NULL,
        0
    },
};
};

```

このテーブルを同じように nwSetUserMenu() を使って登録すると、次のようなメニューになります。

なお、nwSetUserMenu() で新しいメニューテーブルを登録すると、今まで登録されていたメニューテーブルは上書きされ、新しいメニューが使用可能になります。



ユーザーメニューを登録した状態 (2)

ユーザーメニューの登録を取り消すには、以下のようにします。

```
nwSetUserMenu(NULL);
```

これでふたたび「User(undefined)」の淡色表示となり、ユーザーメニューは選択できなくなります。

1.8.3 メニューコールバック関数

メニューコールバック関数は、メニューが選択された場合にコールバックされるユーザー定義の関数で、メニューテーブルに登録しておきます。先程の例では、関数 menu_callback() がそれに当たります。

```
static void menu_callback(NHWND hWndMenu, Sint32 idx, Sint32 param)
```

パラメータ hWndMenu には、コールバック元のメニューウィンドウのウィンドウハンドルが渡されてきます。通常は使用する必要はないでしょう。

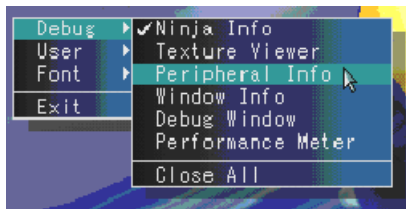
パラメータ idx は、選択されたメニュー項目が、その項目を含むメニューの何番目の項目

であることを示す、0 から始まるインデックス番号です。1 つのコールバック関数で複数のメニュー項目の処理を行いたい場合などに、どのメニュー項目が選択されたかを判定するのに用いることができます。

param は、メニューテーブルでユーザーが定義したパラメータです。これも同様に 1 つのコールバック関数で複数のメニュー項目の処理を行いたい場合などに用いることができます。

1.8.4 チェックマーク

メニュー項目文字列の左側に、チェックマークを表示することができます。チェックマークは、ユーザーに対して、今その項目が有効である、その項目が選択中であることを示すために便利です。図は、Ninows ユーティリティの「Ninja Info」を選択した状態です。「Ninja Info」ウィンドウが表示中であることを示すために、「Ninja Info」の項目名の左にチェックマークを表示しています。「Ninja Info」ウィンドウを閉じると、チェックマークも表示されなくなります。



チェックマークの例

チェックマークの表示、非表示を切り替えるには、メニューテーブルのメンバ type を直接設定します。例は、メニューテーブル menu_table の先頭項目のチェックマークを切り替える例です。

- ・チェックマークを表示する

```
menu_table[0].type |= NWD_MF_CHECKED;
```

- ・チェックマークを消去する

```
menu_table[0].type &= ~NWD_MF_CHECKED;
```

では、具体的な例を見てみましょう。

```
static NWS_MENUTABLE menu_table[] = {
    {NWD_MF_NORMAL, "Test Window", create_test_window, 0},
    {NWD_MF_NULL, "", NULL, 0},
};

static void destroy_test_window_callback(NWHWND hWnd)
{
    menu_table[0].type &= ~NWD_MF_CHECKED;
}

static void create_test_window(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
    NWHWND hWnd;
    // ウィンドウがすでに作成されていたら破棄する
```

```

// ウィンドウデストラクタによりチェックマークが消去される
if (hWnd = nwFindWindow(NULL, "Test Window")) {
    nwDestroyWindow(hWnd);
} else {
    hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window",
        NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_SHADING | NWD_WS_BORDER,
        50, 50, 100, 100, NULL);
    // デストラクタの設定
    hWndNjInfo->destructor = destroy_test_window_callback;
    // 「 Test Window 1 」を作成したので、メニュー項目にチェックマークを付ける
    menu_table[0].type |= NWD_MF_CHECKED;
}
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    nwSetUserMenu(menu_table);
}

```

このメニューテーブルを nwSetUserMenu() で登録し、メニューから「 Test Window 」を選択すると、選択するたびにウィンドウの作成、破棄を行い、それに応じてチェックマークを切り替えます。

ウィンドウデストラクタ (hWnd->destructor) を設定して、その中でチェックマークの消去を行っているのは、メニュー選択以外の方法でウィンドウが破棄される場合があるためです。次のコードは、一見正しいようですが、クローズボックスをクリックしてウィンドウを破棄した場合などに、チェックマークが消去されません。

```

static void create_test_window(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
    NWHWND hWnd;

    if (hWnd = nwFindWindow(NULL, "Test Window")) {
        nwDestroyWindow(hWnd);
        menu_table[0].type &= ~NWD_MF_CHECKED;
    } else {
        hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window",
            NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_SHADING | NWD_WS_BORDER,
            50, 50, 100, 100, NULL);
        menu_table[0].type |= NWD_MF_CHECKED;
    }
}

```

1.8.5 ポップアップメニューの作成

簡単なポップアップメニューの作成

これまでは、システムメニューにユーザーメニューを設定する方法を解説してきましたが、システムメニューに登録せずに、画面にポップアップメニューを作成することもできます。それには関数 nwCreateMenuWindow() を使用します。

```

static void submenu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{

```



```

}

static NWS_MENUTABLE submenu_table[] = {
    {NWD_MF_NORMAL | NWD_MF_GRAYED, "Sub Menu 1", submenu_callback, 0},
    {NWD_MF_NULL, "", NULL, 0},
};

static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE menu_table[] = {
    {NWD_MF_NORMAL, "Test Menu 1", menu_callback, 0},
    {NWD_MF_SEPARATOR, "", NULL, 0},
    {NWD_MF_POPUP, "Test Popup 1", NULL, (Sint32)submenu_table},
    {NWD_MF_NULL, "", NULL, 0},
};

void create_popup_menu(void)
{
    NWHWND hWnd;

    hWnd = nwCreateMenuWindow(menu_table, "Test Menu", 10, 10, NULL);
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    create_popup_menu();
}

```

これで、画面に次のようなポップアップメニューが表示されます。



作成したポップアップメニュー

作成したメニューウィンドウは、メニュー項目が選択されるか、メニューウィンドウ領域外がクリックされると自動的に破棄されます。

画面上に常駐するポップアップメニューの作成

メニューウィンドウは、メニュー項目が選択されるか、メニューウィンドウ領域外がクリックされると自動的に破棄されるため、このままでは最大 1 回しかメニューを選択できません。ふたたび `nwCreateMenuWindow()` を使用して同じメニューを作成することが必要となります。

画面上に常駐するメニューを作成したい場合は、次のようにします。

```

void create_popup_menu(void)
{

```

```

        NWHWND hWnd;

        hWnd = nwFindWindow(NULL, "Test Menu");
        if (!hWnd)
            hWnd = nwCreateMenuWindow(menu_table, "Test Menu", 10, 10, NULL);
    }

    Sint32 njUserMain(void)
    {
        :
        :
        create_popup_menu();
        :
        :
        return nwExecute();
    }

```

すでにメニューウィンドウがあるかどうかを毎フレーム調べて、もしなければメニューウィンドウを作成しなおします。これで、一見画面に常駐しているように見えるポップアップメニューが実現できます。

1.9 マウス

1.9.1 概要

Nindows では、マウスカーソルの座標を取得する関数、マウスでの操作の許可、禁止を切り替える関数を用意しています。マウス操作はコントローラで行うため、ボタンの取得は `pdGetPeripheral()` で行います。

1.9.2 マウス情報の取得

マウスボタンの取得には `pdGetPeripheral()` を、マウス座標の取得には Nindows API の `nwGetMousePosition()` を使用します。次のサンプルを参考にしてください。

```
Sint32 njUserMain(void)
{
    Sint32 x, y;
    PDD_PERIPHERAL* mouse;

    // nwInitSystem()で指定したコントロールポートの情報を取得
    mouse = njGetPeripheral(PDD_PORT_A0);

    // マウス座標の取得
    nwGetMousePosition(&x, &y);

    if (mouse->on & PDD_DGT_TA) {
        // Aボタンが押されている
    }
    if (mouse->on & PDD_DGT_TB) {
        // Bボタンが押されている
    }
    if (mouse->press & PDD_DGT_TA) {
        // Aボタンが押された
    }
    if (mouse->press & PDD_DGT_TB) {
        // Bボタンが押された
    }
    if (mouse->release & PDD_DGT_TA) {
        // Aボタンが放された
    }
    if (mouse->release & PDD_DGT_TB) {
        // Bボタンが放された
    }

    :
    :
    return nwExecute();
}
```

マウスカーソルの座標にあるウィンドウを知りたい場合は、以下のようにします。

```
Sint32 njUserMain(void)
{
    Sint32 x, y;
    NWHWND hWnd;

    nwGetMousePosition(&x, &y);
    hWnd = nwFindWindowByPos(x, y);
```

```
if (hWnd) {  
    // マウスカーソルのある座標に、何かウィンドウが表示されている  
} else {  
    // マウスカーソルのある座標には、ウィンドウは表示されていない  
}  
  
:  
  
return nwExecute();  
}
```

1.10 フォント

1.10.1 概要

Windows では、システムメニューの「Font」項目の選択あるいは API 呼び出しによって、フォントを切り替えることができます。

1.10.2 フォントの変更に関する問題

Windows ではフォントの変更に伴うウィンドウのリサイズには対応していません。また、一部の特殊なウィンドウとコントロール「Property」などは、大きいサイズのフォントでは正しく表示されません。

2.1 概要

この章では、GDFS (GD ファイルシステム) ライブラリについて解説します。

2.1.1 基本機能

GDFS ライブラリは、表 1 の機能を提供します。

表 1 GDFS ライブラリが提供する機能

機能	詳細
GD-ROM の高密エリアのアクセス機能	①ディレクトリのアクセス ②ファイルのオープン / クローズ ③ファイルの同期 / 非同期読み込み
GDDA の再生機能	①トラック指定再生 ②セクタ指定再生

2.1.2 モジュール構成

ファイルシステムを含む、GD アクセスに関するモジュール構成を図 1 に示します。

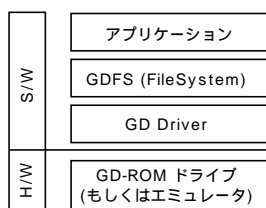


図 1 モジュール構成

- ・ sy_cache.lib(RAW CACHE library)
- ・ sy_chain.lib(interrupt chain handler library)
- ・ kabuto.lib(Access Librar for kabuto)

2.2 機能説明

2.2.1 用語の説明

GDFS ライブラリで使用している用語を解説します (表 2)。

表 2 GDFS ライブラリ用語

用語	意味
FAD	フレームアドレス。GD-ROM 上の物理セクタ
ファイルハンドル	ファイルを操作するために必要な情報 (実際にはそれを指し示すもの)
GD バッファ	GD ドライブが読み込んだデータを一時的に蓄えておく場所
コールバック	条件が満たされたときに、登録してある処理を呼び出す仕組み
セクタ	GD-ROM に格納されているデータを管理する最小単位 (GD-ROM では 2,048 バイト固定)
ディレクトリレコード	GD-ROM のディレクトリ構造を記録してある場所
ディレクトリレコードハンドル	ディレクトリレコードからファイル情報を取り出し、格納した情報 (実際にはそれを指し示すもの)
ライブラリワークエリア	ライブラリの実行時に必要とする作業領域

2.2.2 初期化

GDFS ライブラリの使用にあたっては、はじめに必ず `gdFsInit()` 関数を呼び出さなければなりません。この関数は次の処理を行います。

- ・ライブラリの作業領域などの初期化
- ・デバイスの初期化 (GD-ROM)
- ・マウント処理

ライブラリの初期化時には、ライブラリワークエリアとカレントディレクトリの情報を格納するための領域が必要です。これらはライブラリの利用者が用意しなければなりません。

ライブラリワークエリアは、同時にオープンする最大ファイル数によって必要サイズが変化します。また、カレントディレクトリの情報を格納する領域は、格納するファイル数によって必要サイズが変化します。

これらの必要サイズは、次のマクロで求められます。

- ・`gdFsGetWorkSize(max_open)`
- ・`gdFsGetDirrecSize(max_dirent)`

ライブラリの初期化のサンプルをリスト 1 に示します。ただし、この場合 `gdfswork` で示されるワークエリアの先頭アドレスは 32 バイト境界で整列されていなければなりません。

なお、初期化後、ライブラリ終了処理を行う前に再度初期化処理を行うことはできません。

リスト 1 ライブラリの初期化例

```
UInt32 gdfswork[gdFsGetWorkSize(8)/4];
UInt32 gdfscurdir[gdFsGetDirrecSize(64)/4];

gdFsInit(8, gdfswork, 64, gdfscurdir);
```


ただし、この場合 gdfswork は 32 バイト境界にある必要があります。
一度 Init が完了すると、Finish を呼び出す前に、再度 Init を呼び出しても無条件で成功を返します (これは Overlay 化した時のための対策です)。

2.2.3 ディレクトリ操作

GDFS ライブラリでは、ディレクトリの操作にディレクトリレコードハンドル (図 2) を使用します。

ディレクトリレコードハンドルは、GD-ROM 内のディレクトリのファイル情報を格納し、ファイル名、ファイルサイズなどの必要な情報を、ファイルシステムで使えるようにするものです。通常の OS におけるディレクトリキャッシュに相当します。

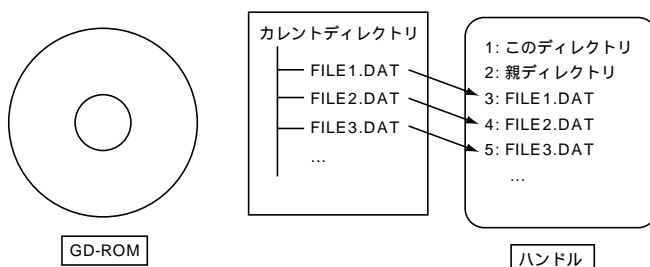


図 2 ディレクトリレコードハンドル

ライブラリ初期化時に、カレントディレクトリの情報を格納するためのディレクトリコードハンドルが自動的に生成されています。カレントディレクトリを作業ディレクトリに移動して作業する場合は、ほかにハンドルを生成する必要はありません。

カレントディレクトリの移動には、gdFsChangeDir 関数を使います。

複数のディレクトリにあるファイルに交互にアクセスするような場合には、その度に GD-ROM にアクセスすることになるので、パフォーマンスが著しく低下することがあります。

これはディレクトリレコードハンドルを生成し、頻繁にアクセスする必要があるディレクトリの情報をあらかじめ読み取っておくことで回避することができます (図 3、リスト 2)。

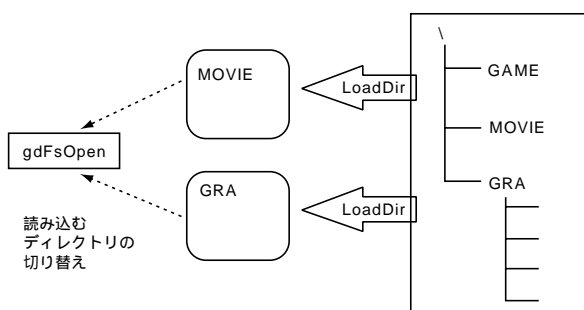


図 3 複数のディレクトリの切り替え

リスト2 複数ディレクトリ切り替え例

```
UInt32 dirbuf1[ gdFsGetDirrecSize( 64 ) ];
UInt32 dirbuf2[ gdFsGetDirrecSize( 64 ) ];
GDFS_DIRREC gf_moviedir;
GDFS_DIRREC gf_gradir;
GDFS gdfs1 ,gdfs2;

gf_moviedir = gdFsCreateDirhn( dirbuf1 ,64 );
gf_gradir = gdFsCreateDirhn( dirbuf2 ,64 );

gdFsLoadDir( "MOVIE", gf_moviedir );
gdFsLoadDir( "GRA" , gf_gradir );

gdfs1 = gdFsOpen( "ABC.MOV" ,gf_moviedir );
gdfs2 = gdFsOpen( "DEF.GRA" ,gf_gradir );

/* ファイルの読み込み等の処理 */

gdFsClose( gdfs1 );
gdFsClose( gdfs2 );
```

2.2.4 ファイル操作

読み込み等のファイル操作を行うためには、ファイルハンドルが必要となります。

ファイルハンドルはgdFsOpen 関数で生成することができます。

ファイルハンドルからファイルサイズ等の情報を取得することができます (gdFsGetFileSize, gdFsFileSctSize, gdFsGetFad)。

また、そのファイルの読み出し位置を変更したり、取得したり、ファイルの終わりにさしかかったかどうかを調べることができます (gdFsSeek, gdFsTell, gdFsCheckEof)。

また、必要が無くなったファイルハンドルは、gdFsClose 関数で、開放しなければなりません。

2.2.5 ファイルへのアクセス

ファイルへのアクセス方式には、次の2つの方法があります。

①完了復帰型アクセス

処理が完了するまで制御をブロックします。

完了復帰型アクセス関数の制御方法は簡単です。しかし、処理が完了するまでブロックされるため、ゲームのようにリアルタイム制を要求される場合には、パフォーマンスを低下させる原因になることがあります。

リスト3 完了復帰型アクセス例

```
GDFS gf;
UInt32 buf[ 2048 / sizeof( UInt32 ) ];
```

```
gf = gdFsOpen( "ABC.DAT" ,NULL );

gdFsRead( gf ,1 ,buf );

gdFsClose( gf );
```

② 時復帰型アクセス

要求を受け付けるとすぐにアプリケーションに制御を移します。

即時復帰型アクセス関数の制御方法は完了復帰型よりも多少複雑です。しかし、完了するまで待たされないの、読み込み中にほかの動作を平行して行うことができます。

リスト 4 即時復帰型アクセス例

```
GDFS gf;
Uint32 buf[ 2048 / sizeof( Uint32 ) ];

gf = gdFsOpen( "ABC.DAT" ,NULL );

gdFsReqRd32( gf ,1 ,buf );

do {
    stat = gdFsGetStat( gf );
} while( stat == GDD_STAT_READ );

gdFsClose( gf );
```

読み込み中の状態変化は、図 4 のようになります。

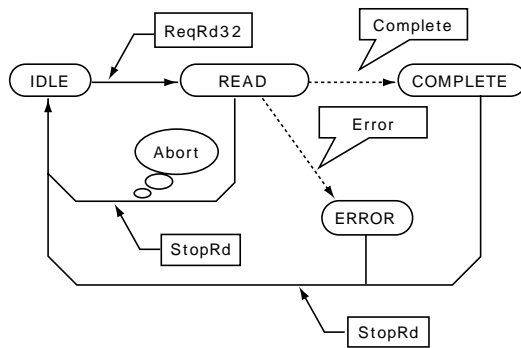


図 4 ファイルハンドルの状態遷移

また、ストリーミング再生などの用途に適した、読み込みと転送を分割して行える GdReqGdRd 関数と GdTrans32 関数が用意されています (図 5、表 4)。

リスト 5 にサンプルを示します。

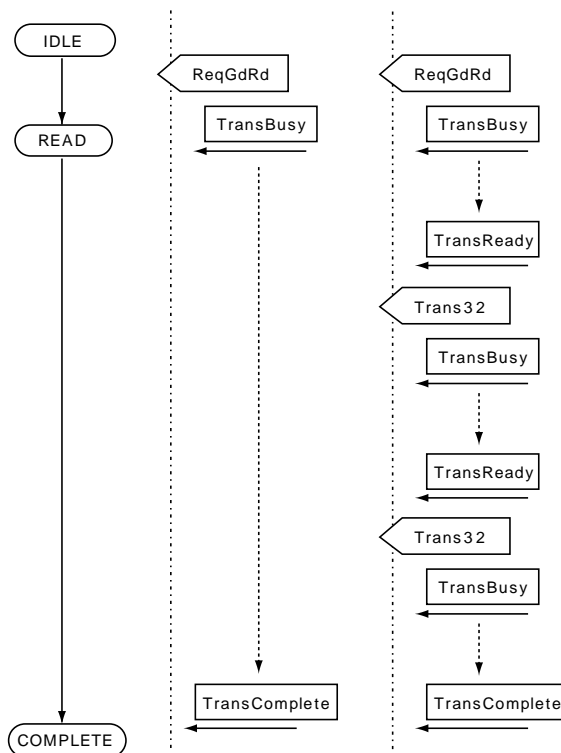


図 5 GdReqGdRd 関数と GdTrans32 関数

表 4 読み込みと転送を分割して処理できる関数

関数	機能
GdReqGdRd	GD バッファへの先読みリクエストの発行
GdTrans32	GD バッファからの転送指定

リスト 5 RegGdRd アクセス例

```
gdFsReqGdRd( gf ,128 );

while( (stat = gdFsGetStat( gf )) == GDD_STAT_READ ) {
    if ( gdFsGetTransStat( gf ) == GDD_TRANS_READY ) {
        gdFsTrans32( gf ,2048 ,buf );
    }
}
```

アクセス中の注意点 キャッシュについて

Dreamcast には、16KB のオペランドキャッシュ(OC)と 8KB のインストラクションキャッシュ(IC)が搭載されています。これらが有効のときは、次のようなキャッシュ問題が発生する場合があります。

- ・オーバーレイモジュールなどのように、プログラムを同一アドレスにモジュールを読み直す場合、メモリ上に次のモジュールを読み込んでも、以前のコードがインストラクションキャッシュ上にあるため、誤動作を起こすことがある。
- ・データを同一アドレスに読み直す場合、たまたまオペランドキャッシュ上に残ったデータがあると、メモリ上のデータを書き換えてもキャッシュ上のデータが読めてしまうことがある。

これらの問題を防ぐには、読み込みを行う前にインストラクションキャッシュ(IC)、またはオペランドキャッシュ(OC) の該当部分をクリアする必要があります。しかし、ファイルシステムは、読むべきファイルがプログラムであるかデータであるかを判断できません。

そこで、ファイルシステムではオペランドキャッシュのコヒーレンシのみを保証するように設計されています。具体的には、ファイルの読み込みリクエストが受理された時点で、該当メモリ空間のキャッシュをインバリデートします。

ファイルの読み込みリクエストが受理された時点で、該当メモリ空間のキャッシュをインバリデート(invalidate)します。ただし、P2 空間(0xa0000000)を指定したときは、インバリデートを行いません。

しかし、インバリデートした後、実データがメモリ上に転送される前に該当メモリにアクセスした場合、データがキャッシュに読み込まれてしまいます。

これらの理由により、読み込みを始めてから完了するまでは、該当メモリへのアクセスを禁止します。

2.2.6 ドライブ情報の取得

ドライブ情報取得関数を利用すると、ドライブ状態や TOC を取得できます。GD トレイが空いているかどうか、ドライブ状態取得関数で取得できます(リスト 6)。

ただし、ドライブ状態はドライブにアクセスするコマンドを発行した後に更新されます。長期間ドライブにアクセスしていない状態にあるときは、ドライブ情報更新関数(gdFsReqDrvStat)を使用して、状態を更新してからドライブ状態を取得してください。

リスト6 ドライブ状態取得例

```
Sint32 dstat;

dstat = gdFsGetDrvStat();

if ( dstat == GDD_DRVSTAT_OPEN ) {
    /* abort 処理 */
}
```

また、TOC を取得することによって、各トラックの情報を取得できます。これを調べると DA(Digital-Audio) トラックが識別できます (リスト7)。

リスト7 TOC 取得例

```
Sint32 tocbuf[ 102 ];

gdFsGetToc( 1 ,&tocbuf );
```

TOC は GD 高密エリアのみ取得できます。ゲームで使う DA トラックは、通常高密エリアのものを 사용합니다。表 5 にその構造を示します。

表 5 TOC データフォーマット

バイト	内容
0 ~ 3	トラック 1 情報
4 ~ 7	トラック 2 情報
...	...
392 ~ 395	トラック 99 情報
396 ~ 399	先頭トラック情報
400 ~ 403	最終トラック情報
404 ~ 407	リードアウト情報

表 6 TOC データフォーマットの各情報の詳細

上位	下位			
第 1 バイト		第 2 バイト	第 3 バイト	第 4 バイト
Ctr	0x1	トラック開始 FAD		
Ctr	0x1	先頭トラック番号	0x00	0x00
Ctr	0x1	終了トラック番号	0x00	0x00
Ctr	0x1	リードアウト開始 FAD		

表 7 Control(表 6 の Ctr) ビットの値

コントロールフィールド	説明
00?0	2 Audio without Pre-emphasis
00?1	2 Audio with Pre-emphasis
0?0?	コピー禁止
0?1?	コピー許可
01?0	デジタルデータ
1???	ブロードキャストデータ

2.2.7 コールバック登録

GDFS ライブラリでは、次のようなコールバックをサポートしています。

- ・読み込み完了コールバック
- ・転送終了コールバック
- ・エラー発生コールバック

ライブラリ利用者が用意したコールバック関数を登録するには、それぞれに応じた `gdFsEntry` ~ 関数を呼び出します。

これらのコールバック関数は、割り込み処理中に呼び出されるため、可能な限り処理を単純で簡潔なものにしてください。コールバック関数が長い時間処理を占有してしまうと、他の処理に悪影響を与える可能性があります。十分注意してください。

2.2.8 デジタルオーディオ再生 (DA 再生)

DA (Digital-Audio) 再生に関しては、次の機能を用意しています。

- ・トラック指定再生
- ・セクタ指定再生
- ・再生停止
- ・再生ポーズ
- ・再生再開

これらを実現する関数名は `gdFsDA` ~ です。

DA 再生中にファイルの読み込みなどの処理を行ったときは、再生が強制中断されます。また、ファイルの読み込み中に DA を再生した場合は無効になります。

2.2.9 サーバー関数

GDFS ライブラリでは、リクエストされた処理をバックグラウンド実行するために、サーバー関数を用意しています。このサーバー関数はライブラリ初期化時に自動的に `VblankIn` 割り込みに登録されます。したがって、ライブラリ利用者は、特に必要がない限り使用する必要はありません。

2.3 定数

表 8 シークモード

定数名	説明
GDD_SEEK_SET	ファイルの先頭
GDD_SEEK_CUR	ファイルの現在の読み込み位置
GDD_SEEK_END	ファイルの終端

表 9 ステータス

定数名	説明
GDD_STAT_IDLE	アイドル
GDD_STAT_COMPLETE	動作完了
GDD_STAT_READ	読み込み中
GDD_STAT_ERR	エラー発生
GDD_STAT_FATAL	致命的エラー発生

表 10 ドライブステータス

定数名	説明
GDD_DRVSTAT_BUSY	稼働中
GDD_DRVSTAT_PAUSE	ポーズ状態
GDD_DRVSTAT_STANDBY	スタンバイ状態
GDD_DRVSTAT_PLAY	再生中
GDD_DRVSTAT_SEEK	シーク中
GDD_DRVSTAT_SCAN	スキャン再生中
GDD_DRVSTAT_OPEN	トレイオープン
GDD_DRVSTAT_NODISC	ディスクがない
GDD_DRVSTAT_RETRY	リトライ
GDD_DRVSTAT_ERROR	エラー

表 11 転送状態

定数名	説明
GDD_FS_TRANS_READY	転送可能
GDD_FS_TRANS_BUSY	転送中
GDD_FS_TRANS_COMPLETE	転送完了
GDD_FS_TRANS_ERROR	転送エラー

表 12 エラーコード

定 数 名	説 明
GDD_ERR_OK	正常終了
GDD_ERR_INIT	ライブラリ初期化に失敗した
GDD_ERR_RESET	ドライブの初期化に失敗した
GDD_ERR_LIBOV	オーバーレイの初期化に失敗した
GDD_ERR_MOUNT	マウントできなかった
GDD_ERR_DISC	不正なディスクを使用した
GDD_ERR_DIRREC	不正なディレクトリレコードハンドルを使用した
GDD_ERR_CANTOPEN	ファイルがオープンできなかった
GDD_ERR_NOTFOUND	ファイルが見つからなかった
GDD_ERR_NOHNDL	未使用のハンドルが見つからなかった
GDD_ERR_ILHNDL	不正なハンドルを使用した
GDD_ERR_NOTDIR	ディレクトリではない
GDD_ERR_DIROVER	格納できるエントリー数を超えた
GDD_ERR_BUSY	実行中
GDD_ERR_32ALIGN	32 バイト境界から外れている
GDD_ERR_SIZE	32 バイト単位ではない
GDD_ERR_SEEK	シークの指定が不正
GDD_ERR_OFS	指定された位置が不正
GDD_ERR_ILLTMODE	転送モードが正しくない
GDD_ERR_READ	読み込みに失敗した
GDD_ERR_NOTREAD	読み込み中ではない
GDD_ERR_TOUT	タイムアウト
GDD_ERR_EOF	ファイルの終端に達した
GDD_ERR_TRAYOPEND	トレイがオープンされている
GDD_ERR_SIZEOVER	要求サイズが大きすぎる
GDD_ERR_FATAL	致命的なエラーが発生した
GDD_ERR_UNDEF	未定義のエラー
GDD_ERR_NOERR	エラーはなかった
GDD_ERR_RECOVER	エラーが発生したが復帰した
GDD_ERR_NOTREADY	ドライブの準備ができていない
GDD_ERR_MEDIA	メディアエラー
GDD_ERR_HWARE	ハードウェアエラー
GDD_ERR_ILLREQ	不正なリクエストを発行した
GDD_ERR_UNITATTENT	媒体の交換を検出した
GDD_ERR_PROTECT	プロテクトされている
GDD_ERR_ABORT	中断された
GDD_ERR_NOREADABLE	読み込めない
GDD_ERR_CHECKBUSY	メディアチェック中

表 13 ファイルフラグ

ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
Multi Extent	-	-	Protection	Record	Associated	Directory	Existence

2.3.1 データ型

表 14 データ型

データ型	GDFS	ファイルハンドル
------	------	----------

データ型	GDFS_DIRREC	ディレクトリレコードハンドル
------	-------------	----------------

データ型	GDFS_DIRINFO	ディレクトリ情報
------	--------------	----------

型	名前	説明
Sint32	fad	先頭 FAD
Sint32	fsize	ファイルサイズ
UInt8	flag	ファイルフラグ
UInt8	pad[3]	予約

データ型	GDFS_FUNC	コールバック関数
------	-----------	----------

型	名前	説明
void *	obj	コールバック関数に渡される第 1 引数
関数例	void gdfs_func(void *obj)	

データ型	GDFS_ERRFUNC	エラーコールバック関数
------	--------------	-------------

型	名前	説明
void *	obj	コールバック関数に渡される第 1 引数
Sint32	errcode	エラーコード
関数例	void gdfs_errfunc(void *obj ,Sint32 errcode)	

2.3.2 関数一覧

表 15 ライブラリ初期化 / 終了関数

関数	機能
gdFsInit	ライブラリの初期化
gdFsReinit	ライブラリの再初期化
gdFsFinish	ライブラリの終了
gdFsGetWorkSize	ライブラリワークサイズ取得 (マクロ)
gdFsGetDirrecSize	ディレクトリレコードバッファサイズ取得 (マクロ)

表 16 ディレクトリ操作関数

関数	機能
gdFsCreateDirhn	ディレクトリハンドルの生成
gdFsLoadDir	ディレクトリの読み取り
gdFsSetDir	カレントディレクトリを設定
gdFsChangeDir	カレントディレクトリの変更
gdFsGetDirInfo	ファイルの情報を取得

表 17 ファイル操作関数

関数	機能
gdFsOpen	ファイルのオープン
gdFsOpenRange	ファイルのセクタ指定オープン
gdFsClose	ファイルのクローズ
gdFsSeek	ファイルの読み出し位置変更
gdFsTell	ファイルの次の読み出し位置を取得
gdFsCheckEof	ファイルが EOF に到達したかを調べる
gdFsGetFileSize	ファイルサイズの取得
gdFsGetFileSctSize	ファイルのセクタサイズ取得
gdFsGetFad	ファイルの FAD を取得
gdFsCalcSctSize	バイト数からセクタ数を算出 (マクロ)

表 18 完了復帰関数

関数	機能
gdFsRead	ファイルの読み込み

表 19 即時復帰関数

関数	機能
gdFsReqRd32	ファイルの読み込み
gdFsGetStat	ファイルの状態取得
gdFsGetErrStat	ファイルのエラー状態取得
gdFsGetNumRd	ファイルの読み込み済みサイズの取得
gdFsStopRd	ファイルの読み込み中断
gdFsMovePickup	ピックアップ (読取装置) の移動
gdFsReqGdRd	GD バッファへの先読みのリクエスト発行
gdFsTrans32	GD バッファからの転送指定
gdFsGetTransStat	転送状態を取得
gdFsGetWorkHn	現在実行中のハンドルの取得
gdFsGetSysHn	システムハンドルの取得

表 20 ドライブ情報取得関数

関数	機能
gdFsGetToc	TOC の取得
gdFsGetDrvStat	ドライブの状態の取得
gdFsReqDrvStat	ドライブの状態を更新させる
gdFsIsTrayOpen	トレイのオープン状態を取得

表 21 コールバック登録関数

関数	機能
gdFsEntryRdEndFunc	読み込み終了時のコールバック関数を登録
gdFsEntryTrEndFunc	転送終了時のコールバック関数を登録
gdFsEntryErrFunc	エラー発生時のコールバック関数を登録

表 22 デジタルオーディオ (DA) 再生関数

関数	機能
gdFsDaPlay	DA のトラック再生
gdFsDaPlaySct	DA のセクタ指定再生
gdFsDaStop	DA の再生停止
gdFsDaPause	DA の再生の一時停止
gdFsDaRelease	DA の再生の一時停止解除

表 23 サーバー関数

関数	機能
gdFsExecServer	サーバー関数

各ライブラリのリファレンスなどに関する補足事項を説明します。なお、この内容は将来変更される可能性があるので注意してください。

3.1 Shinobi 環境の CPU モード

SHINOBI 環境は、SH7091 の特権モードで動作します。初期化時 (AIP) に、SR.MD=1 に設定されます。アプリケーションでのこのビットの変更を禁止します。

3.2 Shinobi 環境でのバンクレジスタの使用等について

ユーザープログラムの一部をアセンブラで記述するときに、バンクレジスタを使う際の注意事項について説明します。

3.2.1 汎用レジスタ

① 初期化時

- ・ SR.MD=1, SR.RB=1 に設定されます。ユーザープログラムがこのビットを変更することを禁止します。
- ・ R0 ~ R7 が R0_BANK1 ~ R7_BANK1 に対応します。
- ・ バンクレジスタは、R0_BANK0 ~ R7_BANK0 に対応します。

② ライブラリの裏レジスタの使用

- ・ R1_BANK0 ~ R7_BANK0 はライブラリでは使用しません。

③ 割り込みライブラリでの保存レジスタ

- ・ 次のレジスタを除くすべてのレジスタが保存されます。
裏レジスタ BANK0_R0 ~ BANK0_R15, DBR, SGR, PC, R15

④ バンクレジスタ使用について

- ・ R0_BANK0 ~ R7_BANK0 はユーザに開放されています。

3.2.2 浮動小数点レジスタ

①初期化時

- ・ `fpscr = 0x00040001` (`FPSCR.FR = 0,FPSCR.SZ=0`) で初期化されます。
- ・ `FR0 ~ FR15` は `FPR0_BANK0 ~ FPR15_BANK0` に対応します。
- ・ `FMOV` 命令のデータサイズは 32 ビットです。

②ライブラリ使用

- ・ バンク浮動浮動小数点レジスタ (`fprn_bank1`) は、Ninja のカレントマトリックスとして使用しています。ユーザープログラムはこのレジスタを破壊してはなりません。
ライブラリを呼び出す場合は、`fpscr.sz=0,fpscr.fr=0` でなければなりません。

③割り込みライブラリでの保存レジスタ

- ・ すべての浮動小数点レジスタ、および `fpsrc,fpul` は保存されます
(「汎用レジスタ ③割り込みライブラリ (`syInt`) での保存レジスタ」参照)

④バンクレジスタの使用について

- ・ `FPRn_BANK1` をユーザープログラムが使用するときは、使用前に保存し、ライブラリ呼び出し前に復帰してください。

3.2.3 GBR レジスタ

- ・ ライブラリでは、GBR レジスタを使用していません。
- ・ ユーザーに開放されています。

3.2.4 その他

- ・ レジスタの詳細については、『SH7091 プログラミングマニュアル』を参照してください。

3.3 キャッシュの操作について

3.3.1 ユーザのキャッシュ操作について

次の①～②の操作を行う場合、現状ではライブラリ利用者側で、CPU(SH7091) のキャッシュをキャッシュライブラリを利用して、明示的に操作する必要があります。

① インストラクションキャッシュの無効化

プログラムを GD から読み込んだ場合、そのプログラム（読み込んだ部分）を実行する前にインストラクションキャッシュ（ICache）を無効化（インバリデート）します。

理由

GD ライブラリでは、オペランドキャッシュのコヒーレンシーは保証していますが、読み込むデータがプログラムなのかデータなのかを区別できません。ICache を一律無効化するのは不適当なので、ユーザー側で対処する必要があります。

② テクスチャ領域のライトバック

メインメモリ上にあるテクスチャを書き換えた直後にテクスチャをロードする場合、該当テクスチャ領域をライトバックします。ファイルからテクスチャをロードする場合は必要ありません。

理由

現状では、njLoadTexture 系の関数では、性能を考慮してライブラリ内で一律にキャッシュをメモリに書き戻し（ライトバック）を行っていません。

3.3.2 ライブラリのキャッシュ操作について

ライブラリ内では次の①～②の時点で、キャッシュの操作を行っています。

- ① 初期化時、全キャッシュをパージ（ライトバック後インバリデート）します。
対象関数：syHwInit
- ② ファイルの読み込み時、および転送の前に、読み込み領域に対してオペランドキャッシュ（OCache）をインバリデート、またはパージします。
対象関数：gdFsRead, gdFsTrans32
- ③ ファイルからテクスチャロード後、該当エリアのライトバックを行います。

3.3.3 メモリマップ

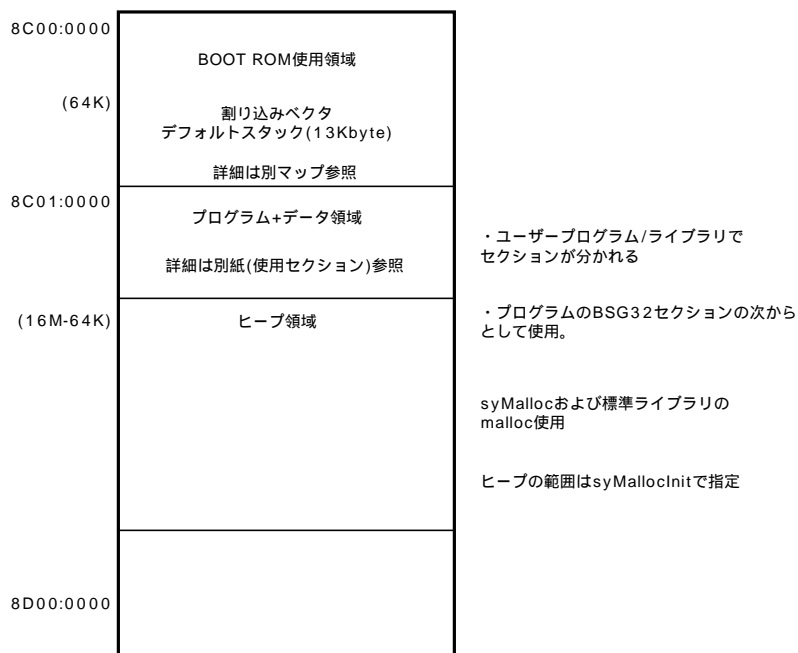


図1 メモリマップ

3.3.4 8C00:0000 ~ 8C01:0000 のメモリマップ

初期化コード領域の一部(8C00:C000~)は実行後破壊可能のため、デフォルトスタックは13K(8C00:C000~8C00:F400)バイトになります。この範囲は、スタートアップルーチンによって「SEGA SEGA...」で埋められます。

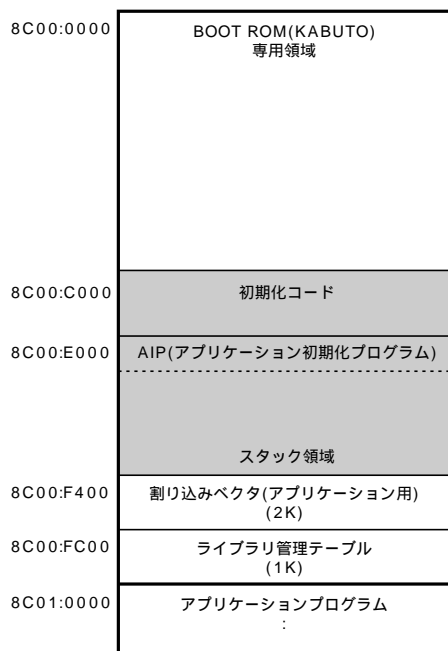


図2 8C00:0000 ~ 8C01:0000 のメモリマップ

3.3.5 使用セクション

セクション名を配置順に記述します。

セクション名は、ライブラリで使うものは「P(コード) C(定数) D(データ) B(BSS)」プリフィックスの後に SG を付加します。さらに、アライメントが 32 バイトのものは、32 を付加します。

表 1 仕様セッション一覧網 (掛け部がライブラリ使用)

#	セクション名	説明	アライメント サイズ	備考
1	DSGLH	・アプリケーションの先頭へJMP ・ライブラリハンドル(*1)領域	4	*1 ライブラリオーバーレイ用 情報
2	DSGLE	ライブラリハンドルの終了	4	
3	P	ユーザ(*2)プログラム領域	4	*2 標準ライブラリ含む。以下同じ
4	C	ユーザ定数領域	4	
5	C32		32	
6	D	ユーザ初期化済みデータ領域	4	
7	D32		32	
8	D_INIT_	グローバルコンストラクタ領域	4	C++用(未実装)
9	D_END_	グローバルデストラクタ領域	4	C++用(未実装)
10	R	(未使用)	4	
11	R32		32	
12	PSG	ライブラリプログラム領域(Ninjaは除く)	4	
13	検討中		4	
14	CSG	ライブラリ定数領域	4	
15	CSG32			
16	DSG	ライブラリ初期化済みデータ領域	32	
17	DSG32			
18	RSG	(未使用)	4	
19	RSG32		32	
20	B	ユーザ未初期化データ領域	4	スタートアップ時にBSG32 セクションまで0クリア
21	B32		32	
22	BSG	ライブラリ未初期化データ領域	4	
23	BSG32		32	この名前セクションの次から ヒープが開始