

テクスチャガイド

VERSION : 0.10

変更履歴

ver0.04

- ・njhitCacheTexture関数の削除
- ・njLoadCacheTexture関数の追加
- ・njCalcTexture関数の変更
- ・テクスチャキャッシュの使用方法変更
- ・PVR テクスチャフォーマットの追加

ver0.05

- ・2.2 PVR フォーマット図変更
- ・4.2 テクスチャロードの流れの図バグを修正
- ・4.6 テクスチャのロードエラー追加
- ・5.4 テクスチャ関連定義 テクスチャエラー定義追加
- ・6.2 例3 (キャッシュサンプル) のバグを修正

ver0.06

- ・1.1 ストライド値の説明追加
- ・2.3.4 RECTANGLE フォーマットの説明変更
- ・2.3.5 STRIDE フォーマットの説明変更
- ・2.4 図変更
- ・4.2 テクスチャロードの流れの図修正
- ・4.3 テクスチャリストの作成のロード元に NJD_TEXATTR_TYPE_FRAMEBUFFERを追加
- ・4.6 テクスチャのロードエラーに NJD_TEXERR_GLOBALINDEXを追加
- ・4.7 フレームバッファテクスチャ追加
- ・4.8 レンダテクスチャ追加
- ・5.2 テクスチャ関連関数 njCalcTexture 変更ほか 6関数追加
- ・5.4 テクスチャ関連定義変更、追加

ver0.07

- ・2.4 カラーフォーマットの NJJA でサポートしているフォーマットに VQ、VQMMを追加
- ・3.3 キャッシュ領域はユーザー設定に変更
- ・4.2 テクスチャロードの流れに njhitTextureBuffer njhitCacheTextureBufferを追加、その他
- ・4.3 テクスチャバッファの設定を新規追加
- ・4.4 キャッシュバッファの設定を新規追加
- ・5.2 テクスチャ関連関数に njhitTextureBuffer njhitCacheTextureBufferを追加
- ・5.2 テクスチャ関連関数 njSetTexturePath njFramebufferBmpはターゲットでは使用不可になる
- ・5.3 テクスチャ関連構造体の NJS_TEXNAME 構造体を修正
- ・各サンプルに njhitTextureBuffer njhitCacheTextureBufferを追加

ver0.08

- ・4.5 テクスチャリストの作成のロード元からフレームバッファテクスチャを使用不可。
- ・4.9 フレームバッファテクスチャの章を削除。
- ・5.4 テクスチャ関連定義でテクスチャロード元より NJD_TEXATTR_FRAMEBUFFERをSET4より使用不可。

ver0.09

- ・2.3.2 VQ テクスチャにおいて「現在の N N J A では対応していません」の表現を削除
- ・3.3 キャッシュは「SET4 では使用できない」を追加
- ・4.5 テクスチャリストの作成で NJS_TEXSURFACE 構造体に pV irtualとpPhy sicalメンバを追加
- ・4.8 グローバルインデックスの自動割付追加
- ・4.10 メモリテクスチャを追加
- ・5.2 テクスチャ関連関数 njhitTextureGlobalIndex njFrameBufferBmp2関数の追加
- ・5.3 テクスチャ関連構造体 NJS_TEXTURESURFACE 構造体に pV irtualとpPhy sicalメンバを追加、NJS_TEXMEMLIST 構造体 tspparam buffer、texparam bufferメンバを追加
- ・5.4 テクスチャ関連定義 NJS_TEXTURE_WIDTH,NJS_TEXTURE_HEIGHT, NJS_TEXTURE_GLOBALINDEX マクロを追加
- ・7 テクスチャ関数の注意点を追加

ver0.10

- ・2.2 PVR フォーマットのグローバルインデックスタグにダミーを追加
- ・4.3 テクスチャバッファの設定方法の仕様変更
- ・4.4 キャッシュバッファの設定のサイズのバグを修正
- ・5.2 テクスチャ関連関数で njhitTextureBuffer,njFrameBufferBmp 関数を仕様変更し、njFrameBufferBmp2関数を削除
- ・5.4 テクスチャ関連定義でテクスチャデータ取得用マクロのバグ修正

目次

第1章 語句説明

- 1.1 概要
- 1.2 語句説明

第2章 テクスチャ

- 2.1 概要
- 2.2 PVR フォーマット
- 2.3 カテゴリーコード
 - 2.3.1 TW DDLED、TW DDLED M PMAP フォーマット
 - 2.3.2 VQ VQ M PMAP フォーマット (Vector Quantizationベクトル量子化)
 - 2.3.3 PALETT ZE4、PALETT ZE4 M PMAP フォーマット
 - PALETT ZE8、PALETT ZE8 M PMAP フォーマット
 - 2.3.4 RECTANGLE フォーマット
 - 2.3.5 STRIDE フォーマット

2.4 カラーフォーマット

第3章 メモリ

- 3.1 概要
- 3.2 テクスチャメモリ
- 3.3 キャッシュ

第4章 テクスチャのロード

- 4.1 概要
- 4.2 テクスチャロードの流れ
- 4.3 テクスチャバッファの設定
- 4.4 キャッシュバッファの設定
- 4.5 テクスチャリストの作成
- 4.6 テクスチャ番号
- 4.7 グローバルインデックス番号
- 4.8 グローバルインデックスの自動割付
- 4.9 テクスチャのロードエラー
- 4.10 メモリテクスチャ
- 4.11 レンダテクスチャ

第5章 テクスチャ関数、構造体、定義説明

- 5.1 概要
- 5.2 テクスチャ関連関数
- 5.3 テクスチャ関連構造体
- 5.4 テクスチャ関連定義

第6章 サンプルプログラム

- 6.1 概要
- 6.2 サンプル

第7章 テクスチャ関数の注意点

- 7.1 概要
- 7.2 SET2 から SET4、SET5 へ移行時の注意点

7.3 SET5での注意点

テクスチャ

第 1 章 語句説明

1.1 概要

ここでは NINJA で使用するテクスチャの語句と意味を説明します。

1.2 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第 4 章で説明します。

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、スライド形式があります。詳しくは第2章テクスチャで説明します。

スライド値

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。

第2章 テクスチャ

2.1 概要

この章では、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。

2.2 PVR フォーマット

グローバル インデックス タグ	識別領域 "GBX"	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。

2.3 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。

2.3.1 TWIDDLED、TW DDLED M PMAP 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。

0	2	8	10	32				128	130			
1	3	9	11					129	131			
4	6		12	14								
5	7	13	15					47				
16	18			48								
17	19											
20												
			31				63					
64								192				

232 VQ, VQ M PMAP フォーマット (Vector Quantizationベクトル量子化)

VQ テクスチャは高圧縮率の圧縮テクスチャフォーマットです。VQ テクスチャはコードブックと呼ばれるカラーテーブルとコードブックの位置を示す Index からデータを伸長し画像を作成します。

233 PALETTE4, PALETTE4 M PMAP フォーマット PALETTE8, PALETTE8 M PMAP フォーマット

PALETTIZED テクスチャは 4bpp モードと 8bpp モードのがあり、同時に使用することができます。メモリ上では TWIDDED フォーマットと同じ形式を取ります。

現在の NINJA では対応していません。

234 RECTANGLE フォーマット

RECTANGLE テクスチャは縦横のサイズに違うサイズを指定することができます。RECTANGLE はミップマップを使用することはできません。また、TWIDDED テクスチャに比べて、性能が低下します。RECTANGLE 形式の場合もレンダリングすることが出来ます。

235 STRIDE フォーマット

RECTANGLE の特別な形で、この領域にレンダリングすることができ、それをテクスチャとして使用することができます。STRIDE 形式のテクスチャを使用する場合、ストライド値を指定しなくてはなりません。NINJA では njSetRenderWidth 関数を使用します。ストライド形式のテクスチャは以下のようなアドレッシングを用いてテクセルを決定しています。

$$\text{Addr} = U + V * \text{stride}$$

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。

2.4 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。**

・ARGB 8888 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
TW DDLED						×
TW DDLED MM						×
VQ						×
VQ MM						×
PALETTIZED 48				×	×	
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

第3章 メモリ

3.1 概要

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。

3.2 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。

3.3 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）。

SET4 でキャッシュテクスチャは使用できません。

テクスチャメモリにロード

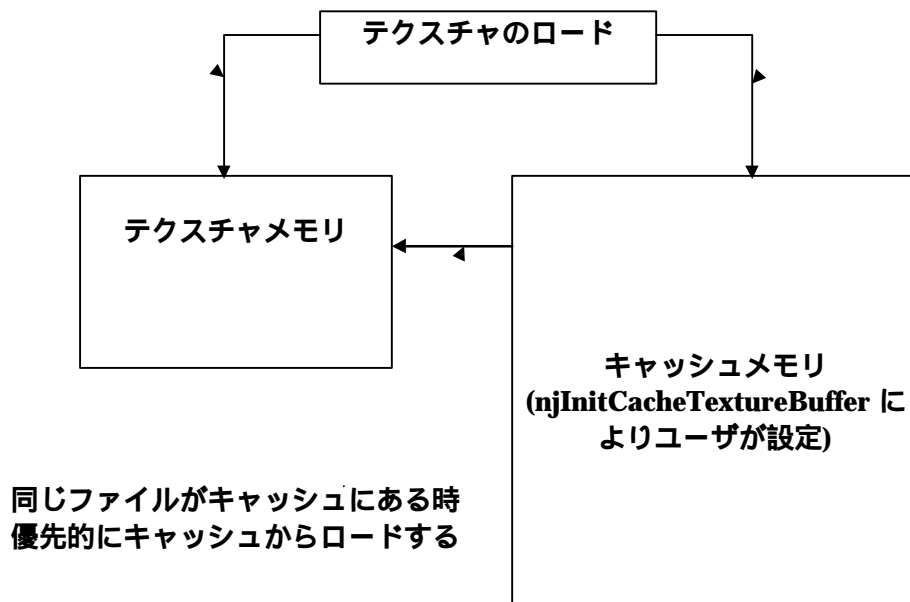
指定なし

NJD_TEXTATTR_BOTH

キャッシュにロード

NJD_TEXATTR_CACHE

NJD_TEXATTR_BOTH



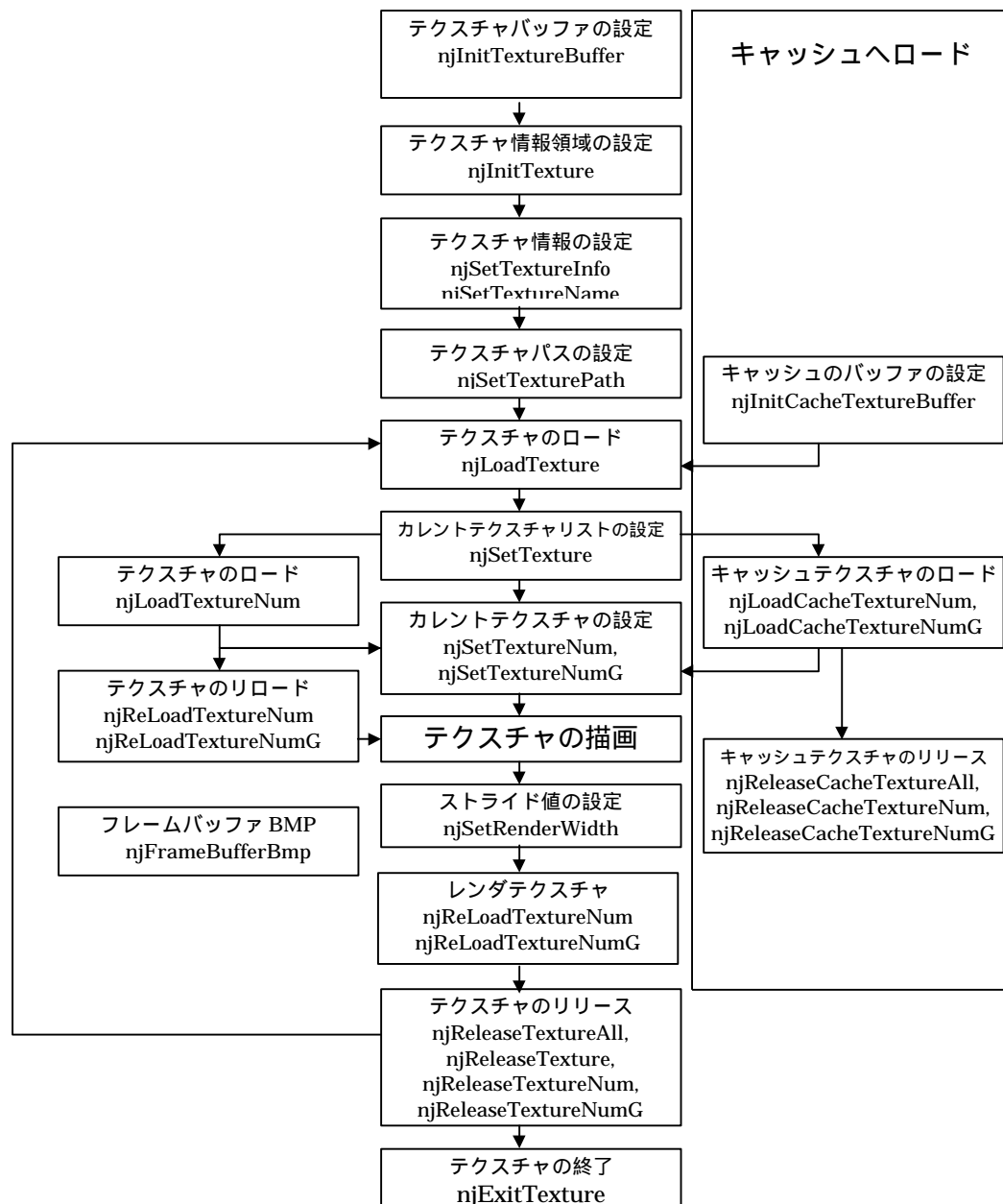
第4章 テクスチャのロード

4.1 概要

次に 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てみます。
はじめに、テクスチャロードの大まかな流れを書きます。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書きます。

4.2 テクスチャロードの流れ

テクスチャメモリヘロード



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。

4.3 テクスチャバッファの設定

これまでの 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設定することになりました。

```
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 2048Byte 以上 2048Byte 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。2048Byte 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 32Byte 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、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 (2048Byte) 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開放しても結構です。

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 (GLOBAL INDEX 12Byte、ヘッダ 16Byte を含む)

サイズ	SET2		SET4	
	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 (GLOBAL INDEX 12Byte、ヘッダ 16Byte を含む)

サイズ	SET2		SET4	
	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 (GLOBAL INDEX 12Byte、ヘッダ 16Byte を含む)

	SET2		SET4	
サイズ	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

4.4 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)
```

4.5 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

NJS_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);
```

4.6 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。

4.7 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ので、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 0xFFFFFFFF までの数を使用してください。0xFFFFFFFF0 から 0xFFFFFFFF はシステムが使用しますので使用しないでください。

4.8 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 0xFFFFFFFF より 0xFFFFFFFFE、0xFFFFFFFFD... 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 njlInitTextureGlobalIndex(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、njlInitTextureGlobalIndex 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。

4.9 テクスチャのロードエラー

ユーザーが njlInitTexture 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。GBK タグや PVRT タグが違う場合にします。

•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)

•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。

4.10 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, 0x5650, 0x5452,  
0x8008, 0x0000, 0x0101, 0x0000, 0x0080, 0x0080, 0xad20, 0xeeee,  
0xac40, 0xe5c0, 0xff41, 0xff21, 0xf5e0, 0xe580, 0xd580, 0xf700,  
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

```
};  
:  グローバルインデックスヘッダ部  
PVR ヘッダ部  
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,  
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

:

};

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,T009,NJD_TEXFMT_RGB_565|NJD_TEXFMT_TWIDDLED,128,128);

njSetTextureName(&texname[0],&info,0,NJD_TEXATTR_TYPE_MEMORY|NJD_TEXATTR_GLOBAL IN
DEX);

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。

4.11 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 後は開放してもよい */
    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);
    :
}
```

第5章 テクスチャ関数、構造体、定義説明

5.1 概要

ここでは、NINJA のテクスチャ関連関数とテクスチャ関連構造体、テクスチャ関連定義を説明します。

5.2 テクスチャ関連関数

njhitTexture

概要

テクスチャ情報領域を設定する

書式

```
#include <Ninja.h>
void njhitTexture(*addr,n);
NJS_TEXMEMLIST *addr
Uint32 n
```

パラメタ

*addr NJS_TEXMEMLIST 構造体 n 個分の領域へのポインタ
n テクスチャ数

戻り値

なし

機能

使用するテクスチャ数 n 個分の NJS_TEXMEMLIST 構造体領域のポインタを addr に設定することにより、テクスチャ情報を格納する領域とする。
テクスチャをロードする前に必ず実行する。

備考

ここで設定する領域はテクスチャ関連の関数が内部で使います。

njhitTextureBuffer(仕様変更)

概要

テクスチャのワークバッファを設定します

書式

```
#include <Ninja.h>
void njhitTextureBuffer(addr,size);
Sint8* addr
Uint32 size
```

パラメタ

*addr ワークバッファの先頭ポインタ
size ワークバッファのサイズ

戻り値

なし

機能

テクスチャのワークとして必要なメモリを指定します。
NINJA Ver00050044 から 2048Byte 以上 2048Byte 単位のサイズでテクスチャをロードできるようになりました。ここで指定したメモリは njLoadTexture njLoadTextureNum njReloadTextureNum njReloadTextureNumG 終了後開放してもかまいません。詳しくは、4.3 章をご覧ください。

備考

njhitCacheTextureBuffer(新規関数)

概要

キャッシュテクスチャバッファを設定します

書式

```
#include <Nj.h>
void njhitCacheTextureBuffer(addr, size);
Sint8* addr
Uint32 size
```

パラメタ

*addr	キャッシュテクスチャバッファの先頭ポインタ
size	キャッシュテクスチャバッファのサイズ

戻り値

なし

機能

キャッシュテクスチャとして必要なメモリを指定します。ここで指定したメモリは、キャッシュテクスチャを使用している間必要です。詳しくは、44 章をご覧ください。

備考

njhitTextureGlobalIndex(新規関数)

概要

グローバルインデックスの自動割付の初期値を設定します

書式

```
#include <Nj.h>
void njhitTextureGlobalIndex(globalIndex);
Uint32 globalIndex
```

パラメタ

globalIndex	グローバルインデックスの自動割付の初期値
-------------	----------------------

戻り値

なし

機能

PVR テクスチャ内にグローバルインデックスがない場合、または、NJS_TEXATTR_GLOBALINDEX が指定されていない場合に使用するグローバルインデックスの自動割付の初期値を設定します。この関数を使用していない場合、グローバルインデックスの自動割付の初期値は 0xFFFFFFFF です。詳しくは、48 章をご覧ください。

備考

nLoadTexture

概要

テクスチャをロードします

書式

```
#include <Nj.h>
Sint32 nLoadTexture(texlist);
NJS_TEXLIST *texlist
```

パラメタ

*texlist NJS_TEXLIST 構造体のポインタ

戻り値

成功 1
失敗 -1

機能

texlist 構造体に設定したテクスチャをテクスチャメモリ、キャッシュメモリ、またはフレームバッファテクスチャとしてロードします。

備考

この関数を実行する前に、必ず njhitTexture を実行しておく必要があります。
SET5 以降ではテクスチャメモリへの転送にテクスチャバッファのアドレスが 32 バイトアラインになる場合 DMA を使用するので、割り込み禁止で nLoadTexture を使用する場合は 32 バイトアラインにならないように注意してください。

nLoadTextureNum

概要

テクスチャ番号のテクスチャをロードします

書式

```
#include <Nj.h>
Sint32 nLoadTextureNum(n);
Uint32 n
```

パラメタ

n カレントテクスチャリストのテクスチャ番号

戻り値

成功 1
失敗 -1

機能

カレントテクスチャリストのテクスチャ番号 n のテクスチャを、テクスチャメモリまたはキャッシュメモリにロードします。カレントテクスチャリストにテクスチャ番号 n がないときエラーを返します。

備考

この関数を実行する前に必ず、njhitTexture、nSetText を実行しておく必要があります。
SET5 以降では、テクスチャメモリへの転送に、テクスチャバッファのアドレスが 32 バイトアラインになる場合 DMA を使用するので、割り込み禁止で nLoadTextureNum を使用する場合は 32 バイトアラインにならないように注意してください。

nSetTexture

概要

カレントテクスチャリストを設定します

書式

```
#include <Ninja.h>
Sint32 nSetTexture(texlist);
NJS_TEXLIST * texlist
```

パラメタ

*texlist NJS_TEXLIST 構造体のポインタ

戻り値

成功 1
失敗 -1

機能

カレントテクスチャリストに texlist を設定します。

備考

指定されたテクスチャはこれ以降、次に nSetTexture を行うまでカレントテクスチャリストになります。
テクスチャ関連関数、nXXXXNum、nXXXXNumG などの関数は、カレントテクスチャリストに対して操作します。

nSetTextureNum

概要

カレントテクスチャをテクスチャ番号に設定します

書式

```
#include <Ninja.h>
Sint32 nSetTextureNum(n);
Uint32 n
```

パラメタ

n テクスチャ番号 n

戻り値

成功 1
失敗 -1

機能

カレントテクスチャリストのテクスチャ番号 n をカレントテクスチャとして設定します。指定されたテクスチャはこれ以降、次に nSetTextureNum、nSetTextureNumG を行うまでカレントテクスチャになります。

備考

指定するテクスチャがテクスチャメモリにある必要があります。

nSetTextureNum G

概要

カレントテクスチャをグローバルインデックス番号に設定します

書式

```
#include <Ninja.h>
Sint32 nSetTextureNum G (gbbaIndex);
Uint32 gbbaIndex
```

パラメタ

gbbaIndex グローバルインデックス番号

戻り値

成功 1

失敗 -1

機能

グローバルインデックス番号 gbbaIndex のテクスチャをカレントテクスチャとして設定します。これ以降、次に nSetTextureNum、nSetTextureNum G を行うまでカレントテクスチャになります。

備考

指定するテクスチャがテクスチャメモリにある必要があります。

nLoadCacheTexture

概要

キャッシュメモリからテクスチャメモリへロードします

書式

```
#include <Ninja.h>
Sint32 nLoadCacheTexture(tex list);
NJS_TEXLIST *tex list
```

パラメタ

*tex list NJS_TEXLIST 構造体のポインタ

戻り値

成功 1

失敗 -1

機能

テクスチャリスト内のキャッシュテクスチャをキャッシュメモリからテクスチャメモリへロードします。

備考

キャッシュメモリに指定のテクスチャがロードされている必要があります。

nLoadCacheTextureNum

概要

キャッシュメモリからテクスチャメモリへロードします

書式

```
#include <Ninja.h>
Sint32 nLoadCacheTextureNum (n);
Uint32 n
```

パラメタ

n カレントテクスチャリストのテクスチャ番号

戻り値

成功 1

失敗 -1

機能

テクスチャ番号 n のテクスチャをキャッシュメモリからテクスチャメモリへロードします。

備考

nSetTexture によりカレントテクスチャリストが指定されている必要があります。
キャッシュメモリに指定のテクスチャがロードされている必要があります。

nLoadCacheTextureNumG

概要

グローバルインデックス番号のテクスチャを
キャッシュメモリからテクスチャメモリへロードします

書式

```
#include <Ninja.h>
Sint32 nLoadCacheTextureNumG(gbbaIndex);
Uint32 gbbaIndex
```

パラメタ

gbbaIndex グローバルインデックス番号

戻り値

成功 1
失敗 -1

機能

グローバルインデックス番号 gbbaIndex のテクスチャを、キャッシュメモリからテクスチャメモリへロードします。

備考

キャッシュメモリに指定のテクスチャがロードされている必要があります。
キャッシュメモリを開放してもテクスチャメモリ内のテクスチャは開放されません。

nReleaseTextureAll

概要

テクスチャメモリを全て開放します

書式

```
#include <Ninja.h>
void nReleaseTextureAll(void);
```

パラメタ

なし

戻り値

なし

機能

テクスチャメモリをすべて開放します。

備考

もう一度テクスチャを使用したいときは、nLoadTexture等でロードする必要があります。

nReleaseTexture

概要

テクスチャリスト内のテクスチャをテクスチャメモリから開放します

書式

```
#include <Ninja.h>
Sint32 nReleaseTexture(*texlist);
NJS_TEXTURET *texlist
```

パラメタ

*texlist NJS_TEXTURET 構造体のポインタ

戻り値

成功 1

失敗 -1

機能

テクスチャリスト texlist 内のテクスチャをテクスチャメモリから開放します。

備考

ロードした他のテクスチャリストで同じテクスチャが登録されているときは、すべてのテクスチャリストで開放しない限り、テクスチャメモリ上から開放されることはありません。

グローバルインデックスが同一のものを同一のテクスチャとしています。

nReleaseTextureNum

概要

テクスチャ番号のテクスチャメモリを開放する

書式

```
#include <Ninja.h>
Sint32 nReleaseTextureNum(n);
Uint32 n
```

パラメタ

n テクスチャ番号

戻り値

成功 1

失敗 -1

機能

カレントテクスチャリストのテクスチャ番号 n 番のテクスチャをテクスチャメモリから開放します。

備考

ロードした他のテクスチャリストで同じテクスチャが登録されているときは、すべてのテクスチャリストで開放しない限り、テクスチャメモリ上から開放されることはありません。

グローバルインデックスが同一のものを、同一のテクスチャとしています。

nReleaseTextureNumG

概要

グローバルインデックス番号のテクスチャメモリを開放します

書式

```
#include <Ninja.h>
Sint32 nReleaseTextureNumG(gbballIndex);
Uint32 gbballIndex
```

パラメタ

gbballIndex グローバルインデックス番号

戻り値

成功 1
失敗 -1

機能

カレントテクスチャリストからグローバルインデックス番号 gbballIndex のテクスチャをテクスチャメモリから開放する。

備考

ロードした他のテクスチャリストで同じテクスチャが登録されているときは、すべてのテクスチャリストで開放しない限り、テクスチャメモリ上から開放されることはありません。
グローバルインデックスが同一のものを同一のテクスチャとしています。

nReleaseCacheTextureAll

概要

キャッシュメモリを全て開放します

書式

```
#include <Ninja.h>
void nReleaseCacheTextureAll(void);
```

パラメタ

なし

戻り値

なし

機能

キャッシュメモリをすべて開放します。

備考

キャッシュメモリを開放しても、テクスチャメモリ内のテクスチャは開放されません。

nReleaseCacheTextureNum

概要

テクスチャ番号のテクスチャをキャッシュメモリから開放します

書式

```
#include <Ninja.h>
Sint32 nReleaseCacheTextureNum (n);
Uint32 n
```

パラメタ

n カレントテクスチャリストのテクスチャ番号

戻り値

成功 1
失敗 -1

機能

テクスチャ番号 n のテクスチャをキャッシュメモリから開放します。

備考

nSetTextureによりカレントテクスチャリストを指定している必要があります。
キャッシュメモリに指定のテクスチャがロードされている必要があります。
キャッシュメモリを開放してもテクスチャメモリ内のテクスチャは開放されません。

nReleaseCacheTextureNum G

概要

グローバルインデックス番号のテクスチャを
キャッシュメモリから開放します

書式

```
#include <Ninja.h>
Sint32 nReleaseCacheTextureNum G (globalIndex);
Uint32 globalIndex
```

パラメタ

globalIndex グローバルインデックス番号

戻り値

成功 1
失敗 -1

機能

グローバルインデックス番号 globalIndex のテクスチャを
キャッシュメモリから開放します。

備考

キャッシュメモリに指定のテクスチャがロードされている必要があります。
キャッシュメモリを開放しても、テクスチャメモリ内のテクスチャは開放されません。

nGetTextureNumG

概要

カレントテクスチャのグローバルインデックス番号を取得します

書式

```
#include <Ninja.h>
U int32 nGetTextureNumG (void);
```

パラメタ

なし

戻り値

成功 グローバルインデックス番号 0 ~ 0xFFFFFFFF0 の値
失敗 0xFFFFFFFF

機能

カレントテクスチャのグローバルインデックス番号を取得します。

備考

これ以前に nSetTextureNum nSetTextureNumG でカレントテクスチャを設定していない場合、意味がありません。

nCalcTexture

概要

テクスチャメモリの残量を計算します

書式

```
#include <Ninja.h>
U int32 nCalcTextureFlag);
```

パラメタ

U int32	flag	NJD_TEXMEM_FREESZE、 NJD_TEXMEM_MAXBLOCK、 NJD_TEXMEM_MAXSZEを指定
---------	------	---

戻り値

テクスチャメモリの全空き領域または最大連続領域を返します

機能

テクスチャメモリの残量を計算します。

NJD_TEXMEM_FREESZE	テクスチャメモリの全空き領域
NJD_TEXMEM_MAXBLOCK	テクスチャメモリの最大連続空き領域
NJD_TEXMEM_MAXSZE	テクスチャメモリの全容量

備考

nInitTexture を先に実行しておく必要があります。

njExitTexture

概要

テクスチャの使用を終了します

書式

```
#include <Nj.h>
void njExitTexture(void);
```

パラメタ

なし

戻り値

なし

機能

テクスチャの使用を終了します。キャッシュ領域が開放されずに残っている場合開放します。

備考

テクスチャ使用を止めるとき、必ず実行してください。

njSetTexturePath (ターゲットでは使用できません)

概要

テクスチャのあるディレクトリのパスを設定します

書式

```
#include <Nj.h>
void njSetTexturePath(path);
```

パラメタ

UInt8 *path ディレクトリへのパス

戻り値

なし

機能

テクスチャのあるディレクトリのパスを設定します。njLoadTexture njLoadTextureNum でファイルからテクスチャをロードするときに有効になります。ここで設定されたパスは、変更されるまで有効です。

備考

njInitTexture の後、njLoadTexture njLoadTextureNum を実行する前に設定してください。
ターゲットでディレクトリを使用する場合、GDFS ライブラリを使用してください。

n_SetTextureInfo

概要

テクスチャインフォ構造体に情報をセットする

書式

```
#include <Ninja.h>
void n_SetTextureInfo(NJS_TEXINFO *pInfo, Uint16 *pTex, Sint32 Type, Sint32 Width, Sint32 Height)
```

パラメタ

NJS_TEXINFO *pInfo	テクスチャ情報 (出力)
Uint16 *pTex	メモリテクスチャのポインタ
Sint32 Type	テクスチャのタイプ
Sint32 Width	テクスチャの横サイズ
Sint32 Height	テクスチャの縦サイズ

戻り値

なし

機能

メモリテクスチャのとき、テクスチャ情報構造体 pInfo にテクスチャ情報を設定します。
Typeにはカラーフォーマットとカテゴリコードを入れます。ここで設定した pInfo を n_SetTextureName の addr に設定します。

備考

使用方法是サンプルプログラムをご覧ください

カラーフォーマット

NJD_TEXFORMAT_RGBA_1555

NJD_TEXFORMAT_RGB_565

NJD_TEXFORMAT_RGBA_4444

NJD_TEXFORMAT_YUV_422

現在は使用できません

NJD_TEXFORMAT_BUMP

現在は使用できません

カテゴリコード

NJD_TEXFORMAT_TWDDLED

NJD_TEXFORMAT_TWDDLED_MM

NJD_TEXFORMAT_VQ

NJD_TEXFORMAT_VQ_MM

NJD_TEXFORMAT_PALETTE4

現在は使用できません

NJD_TEXFORMAT_PALETTE4_MM

現在は使用できません

NJD_TEXFORMAT_PALETTE8

現在は使用できません

NJD_TEXFORMAT_PALETTE8_MM

現在は使用できません

NJD_TEXFORMAT_RECTANGLE

NJD_TEXFORMAT_STRIPE

nSetTextureName

概要

テクスチャネーム構造体にデータをセットする

書式

```
#include <Ninja.h>
void nSetTextureName(texname *, addr, globalIndex, attr)
```

パラメタ

NJS_TEXNAME	*texname	テクスチャネーム構造体 (出力)
void	*addr	ファイル名又は NJD_TEXINFO 構造体へのポインタ
UInt32	globalIndex	グローバルインデックス
UInt32	attr	テクスチャのアトリビュート

戻り値

なし

機能

ファイルからテクスチャをロードするとき addr にファイル名を設定します。
また、attr には NJD_TEXATTR_TYPE_FILE を指定してください。PVR 形式のテクスチャで、グローバルインデックスをファイル内のグローバルインデックスを使用しない場合、または PVR 形式のテクスチャにグローバルインデックスのチャンクが無い場合、attr に NJD_TEXATTR_GLOBALINDEX を設定し、globalIndex にグローバルインデックスを設定してください。
メモリテクスチャの場合は、addr に nSetTextureInfo で設定した NJS_TEXINFO 構造体のポインタを設定します。また、attr には、NJD_TEXATTR_TYPE_MEMORY を globalIndex にはグローバルインデックスを指定してください。

備考

使用方法はサンプルプログラムをご覧ください。

nReloadTextureNum

概要

テクスチャ番号のテクスチャをリロードします

書式

```
#include <Ninja.h>
SInt32 nReloadTextureNum(n, texaddr, attr, bd);
```

パラメタ

UInt32	n	カレントテクスチャリストのテクスチャ番号
void	*texaddr	ファイル名又テクスチャメモリのアドレス
UInt32	attr	テクスチャのアトリビュート
UInt32	bd	ミップマップのレベル

戻り値

成功 1
失敗 -1

機能

カレントテクスチャリストのテクスチャ番号 n のテクスチャをロードし直します。リロードするテクスチャは前にロードしていたテクスチャと同じものとします。
ファイルからテクスチャをロードするときは attr に NJD_TEXATTR_TYPE_FILE を、メモリからテクスチャをロードするときは attr に NJD_TEXATTR_TYPE_MEMORY を指定します。
また、ミップマップテクスチャの場合、bd を指定することで各ミップマップレベルのみ、リロードすることができます。たとえば、bd に 128 と設定することで、128x128 のレベルのテクスチャレベルのみリロードすることができます。ミップマップのテクスチャをすべてのレベルをリロードする場合、bd に 0 を指定しています。
メモリからロードする場合、texaddr で指定したアドレスから bd のレベルにリロードします。

備考

メモリテクスチャの場合、bd で設定したテクスチャの先頭を指定します。

nReloadTextureNumG

概要

グローバルインデックス番号のテクスチャをリロードします

書式

```
#include <Ninja.h>
Sint32 nReloadTextureNumG(gbbaIndex, texaddr, attr, bd);
```

パラメタ

UInt32	gbbaIndex	グローバルインデックス番号
void	*texaddr	ファイル名又テクスチャメモリのアドレス
UInt32	attr	テクスチャのアトリビュート
UInt32	bd	ミップマップのレベル

戻り値

成功 1
失敗 -1

機能

グローバルインデックス番号 gbbaIndex のテクスチャをロードし直します。リロードするテクスチャは前にロードしていたテクスチャと同じものとします。
ファイルからテクスチャをロードするときは attr に NJD_TEXATTR_TYPE_FILE を、メモリからテクスチャをロードするときは attr に NJD_TEXATTR_TYPE_MEMORY を指定します。
また、ミップマップテクスチャの場合、bd を指定することで各ミップマップレベルのみ、リロードすることができます。たとえば、bd に 128 と設定することで、128x128 のレベルのテクスチャレベルのみリロードすることができます。ミップマップのテクスチャをすべてのレベルをリロードする場合、bd に 0 を指定しています。
メモリからロードする場合、texaddr で指定したアドレスから bd のレベルにリロードします。

備考

メモリテクスチャの場合、bd で設定したテクスチャの先頭を指定します。

nRenderTextureNum

概要

テクスチャ番号のテクスチャ領域にレンダリングする

書式

```
#include <Ninja.h>
void nRenderTextureNum(n);
```

パラメタ

UInt32	n	カレントテクスチャリストのテクスチャ番号
--------	---	----------------------

戻り値

なし

機能

カレントテクスチャリストのテクスチャ番号 n 番のテクスチャにレンダリングします。レンダリングできるテクスチャは NJD_TEXFMT_RECTANGLE または NJD_TEXFMT_STRIDE です。テクスチャ領域が画面より小さい場合、画面の左上からの座標までの領域をレンダリングします。NJD_TEXFMT_STRIDE の場合、nSetRenderWidth 関数でストライド値を設定する必要があります。ストライド値は通常、レンダリング領域の横サイズです。

備考

テクスチャ領域にレンダリングすると、フレームバッファへの通常のレンダリングと合計して、2 回のレンダリングをかけることになります。詳しくは、テクスチャドキュメントを参照してください。

nRenderTextureNum G

概要

グローバルインデックス番号のテクスチャ領域にレンダリングする

書式

```
#include <Ninja.h>
void nRenderTextureNum G (gbbalIndex);
```

パラメタ

U int32 gbbalIndex グローバルインデックス番号

戻り値

なし

機能

グローバルインデックス番号 gbbalIndex のテクスチャにレンダリングします。
レンダリングできるテクスチャは NJD_TEXFMT_RECTANGLE または NJD_TEXFMT_STR DE です。
テクスチャ領域が画面より小さい場合、画面の左上からの座標までの領域をレンダリングします。
NJD_TEXFMT_STR DE の場合、nSetRenderW idth関数でストライド値を設定する必要があります。
ストライド値は通常、レンダリング領域の横サイズです。

備考

テクスチャ領域にレンダリングすると、フレームバッファへの通常のレンダリングと合計して、2 回のレンダリングをかけることになります。詳しくは、テクスチャドキュメントを参照してください。

nSetRenderW idth

概要

ストライド値を設定する

書式

```
#include <Ninja.h>
void nSetRenderW idth(nW idth);
```

パラメタ

U int32 nW idth ストライド値

戻り値

なし

機能

ストライドテクスチャ形式を使用する場合のストライド値を設定します。レンダテクスチャなどでストライドテクスチャを指定した場合、レンダリング領域よりテクスチャが小さい場合は、テクスチャの横サイズを、レンダリング領域がテクスチャのサイズより小さい場合は、レンダリング領域の横サイズを設定します。設定できる値は 32 の倍数で 32 から 992 までです。

備考

nFrameBufferBmp (仕様変更)

概要

フレームバッファをビットマップにする

書式

```
#include <Ninja.h>
void nFrameBufferBmp(bmp buffer);
```

パラメタ

bm p ビットマップ保存用バッファ
buffer 作業用バッファ

戻り値

なし

機能

フレームバッファを 24 ビットBMP にします。
ビットマップバッファサイズ、作業用バッファサイズとして、次のサイズが必要です。

ビットマップバッファサイズ (bm p) = 画面縦サイズ x 画面横サイズ x3 バイト+54 バイト

作業用バッファサイズ (buffer) = 画面縦サイズ x 画面横サイズ x 画面モードのバイト数
画面モードのバイト数とは NJD_FRAMEBUFFER_MODE_RGB565のとき 2 バイトと
NJD_FRAMEBUFFER_MODE_ARGB8888は 4 バイトになります
nJhitTextureBufferで作業用バッファを指定する必要はありません。

この関数で指定したビットマップバッファを CodeScape のバイナリセーブするとビットマップになります。
現在は、フレーム 0 番しかテクスチャに出来ないため、フレーム 0 番が描画中にこの関数を使用すると
レンダリング途中の絵になります。(今後変更予定)

備考

今後は表示中のフレームを BMP にするよう改造します。デバッグなどに使用してください。

削除された関数

nJhitCacheTexture

nLoadTextureNumG

nFrameBufferBmp2

5.3 テクスチャ関連構造体

NJS_TEXSURFACE 構造体

```
ty pedef struct{
    Uint32      Type;           /**/
    Uint32      BitDepth;       /**/
    Uint32      PixelFormat;     /**/
    Uint32      nWidth;         /**/
    Uint32      nHeight;        /**/
    Uint32      TextureSize;     /**/
    Uint32      fSurfaceFlags;   /**/
    Uint32      *pSurface;       /**/
    Uint32      pVirtual;        /**/←新規メンバ
    Uint32      pPhysical;       /**/←新規メンバ
}NJS_TEXSURFACE;
```

NJS_TEXINFO 構造体

```
ty pedef struct{
    void*      texaddr;          /* texture buffer address */
    NJS_TEXSURFACE texsurface;   /* texture surface address */
} NJS_TEXINFO;
```

NJS_TEXNAME 構造体

```
ty pedef struct {
    void      *filename;        /* texture filename strings */
    Uint32    texaddr;          /* texture memory address cache*/
} NJS_TEXNAME;
```

NJS_TEXLIST 構造体

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

NJS_TEXMEMLIST 構造体

```
ty pedef struct {
    Uint32      globalIndex;     /* global unique texture ID */
    Uint32      tspparambuffer;   /* TSPParambuffer */
    Uint32      texparambuffer;   /* TextureParambuffer */
    Uint32      texaddr;         /* texture Flag */
    NJS_TEXINFO texinfo;         /* texinfo */
    Uint16      count;           /* texture count */
    Uint16      dummy;          /* texture error */
} NJS_TEXMEMLIST;
```

5.4 テクスチャ関連定義

nWidthHeightで使用

#define NJD_TEXSIZE_1	1
#define NJD_TEXSIZE_2	2
#define NJD_TEXSIZE_4	4
#define NJD_TEXSIZE_8	8
#define NJD_TEXSIZE_16	16
#define NJD_TEXSIZE_32	32
#define NJD_TEXSIZE_64	64
#define NJD_TEXSIZE_128	128
#define NJD_TEXSIZE_256	256
#define NJD_TEXSIZE_512	512
#define NJD_TEXSIZE_1024	1024

attrで使用

テクスチャロード元

#define NJD_TEXATTR_TYPE_FILE	0	ファイルからロード
#define NJD_TEXATTR_TYPE_MEMORY	BIT_30	メモリからロード
#define NJD_TEXATTR_TYPE_FRAMEBUFFER	BIT_28	SET4 より使用不可

テクスチャロード先

#define NJD_TEXATTR_CACHE	BIT_31	キャッシュへロード
#define NJD_TEXATTR_BOTH	BIT_29	キャッシュ、テクスチャメモリへロード
#define NJD_TEXATTR_READAREA_MASK	(BIT_31 BIT_29)	
#define NJD_TEXATTR_READTYPE_MASK	(BIT_30 BIT_28)	

#define NJD_TEXATTR_GLOBALINDEX	BIT_23	変更
#define NJD_TEXATTR_AUTOMPMAP	BIT_22	次期対応
#define NJD_TEXATTR_AUTODITHER	BIT_21	次期対応
#define NJD_TEXATTR_MASK	0xFFFF0000	

Typeで使用

カラーフォーマット

#define NJD_TEXFMT_ARGB_1555	(0x00)	
#define NJD_TEXFMT_RGB_565	(0x01)	
#define NJD_TEXFMT_ARGB_4444	(0x02)	
#define NJD_TEXFMT_YUV_422	(0x03)	現在は使用できません
#define NJD_TEXFMT_BUMP	(0x04)	現在は使用できません
#define NJD_TEXFMT_COLOR_MASK	(0xFF)	

カテゴリコード

#define NJD_TEXFMT_TW_DDLED	(0x0100)	
#define NJD_TEXFMT_TW_DDLED_MM	(0x0200)	
#define NJD_TEXFMT_VQ	(0x0300)	
#define NJD_TEXFMT_VQ_MM	(0x0400)	
#define NJD_TEXFMT_PALETTE4	(0x0500)	現在は使用できません
#define NJD_TEXFMT_PALETTE4_MM	(0x0600)	現在は使用できません
#define NJD_TEXFMT_PALETTE8	(0x0700)	現在は使用できません
#define NJD_TEXFMT_PALETTE8_MM	(0x0800)	現在は使用できません
#define NJD_TEXFMT_RECTANGLE	(0x0900)	
#define NJD_TEXFMT_STRIDE	(0x0B00)	
#define NJD_TEXFMT_TW_DDLED_RECTANGLE	(0x0D00)	現在は使用できません
#define NJD_TEXFMT_ABGR	(0x0E00)	現在は使用できません
#define NJD_TEXFMT_ABGR_MM	(0x0F00)	現在は使用できません

```
#define NJD_TEXFMT_TYPE_MASK (0xFF00)
```

テクスチャエラーコード

#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

テクスチャメモリサイズ取得 (nCabTextureで使用)

#define NJD_TEXMEM_FREESIZE	(0x00000000)
#define NJD_TEXMEM_MAXBLOCK	(0x00000001)
#define NJD_TEXMEM_MAXSIZE	(0x00000002)

テクスチャデータ取得用マクロ (新規追加)

```
#define NJM_TEXTURE_WIDTH(texlist,n) ¥  
    (((NJS_TEXMEMLIST*)(texlist)->textures[(n)].texaddr)->texinfo.texsurface.nWidth)  
#define NJM_TEXTURE_HEIGHT(texlist,n) ¥  
    (((NJS_TEXMEMLIST*)(texlist)->textures[(n)].texaddr)->texinfo.texsurface.nHeight)  
#define NJM_TEXTURE_GLOBALINDEX(texlist,n) ¥  
    (((NJS_TEXMEMLIST*)(texlist)->textures[(n)].texaddr)->globalIndex)
```

第6章 サンプルプログラム

6.1 概要

この章では、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。

6.2 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);/* 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); /* 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();
45:}
```

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_TWIDDED|NJD_TEXFMT_ARGB_1555, 256, 256);
17:    njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE|
18:                                                             NJD_TEXATTR_GLOBAL INDEX);
19:    njSetTextureName(&texname[1], &Info, 1, NJD_TEXATTR_TYPE_MEMORY|
20:                                                             NJD_TEXATTR_GLOBAL INDEX);
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); /* カレントテクスチャリストを texlist にする */
24:    njLoadCacheTextureNum(0); /* 0 番のテクスチャをキャッシュからロード */
25:    njLoadCacheTextureNum(1); /* 1 番のテクスチャをキャッシュからロード */
26:    /* カレントテクスチャを texlist の 0 番のテクスチャにする */
27:    njSetTextureNum(0);
28:
29:    /* ポリゴンのデータ入力 */
30:    p[0].x = 100; p[0].y = 100;
31:    p[1].x = 200; p[1].y = 100;
32:    p[2].x = 200; p[2].y = 200;
33:    p[3].x = 100; p[3].y = 200;
34:    p[0].col.tex.u = 0; p[0].col.tex.v = 0;
35:    p[1].col.tex.u = 255; p[1].col.tex.v = 0;
36:    p[2].col.tex.u = 255; p[2].col.tex.v = 255;
37:    p[3].col.tex.u = 0; p[3].col.tex.v = 255;
38:}
39:
40:Sint32 njUserMain(void)
41:{
42:    /* テクスチャのポリゴンを描く */
43:    njDrawPolygon2D(p, 4, -100.f, NJD_FILL | NJD_USE_TEXTURE);
44:    return NJD_USER_CONTINUE;
45:}
46:
47:void njUserExit(void)
48{
49:    njExitTexture();
50:    njExitSystem();
51:}
```

第7章 テクスチャ関数の注意点

7.1 概要

この章では、テクスチャ関連関数使用時の注意点について書きます。

7.2 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。必要サイズについては 4.3 章を参照してください。

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 4.4 章をご覧ください。

7.3 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場合は 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。