



# **Basic Model 仕様書**

**(5/31/98)**

# 目次

1. 概要 .....	3
2. モデル構造体 .....	4
3. モデル構造 .....	10
3.1 メッシュセット .....	11
3.2 テクスチャ構造 .....	16
3.3 Ninja アトリビュート .....	20
3.4 テクスチャフォーマット .....	24

## 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。

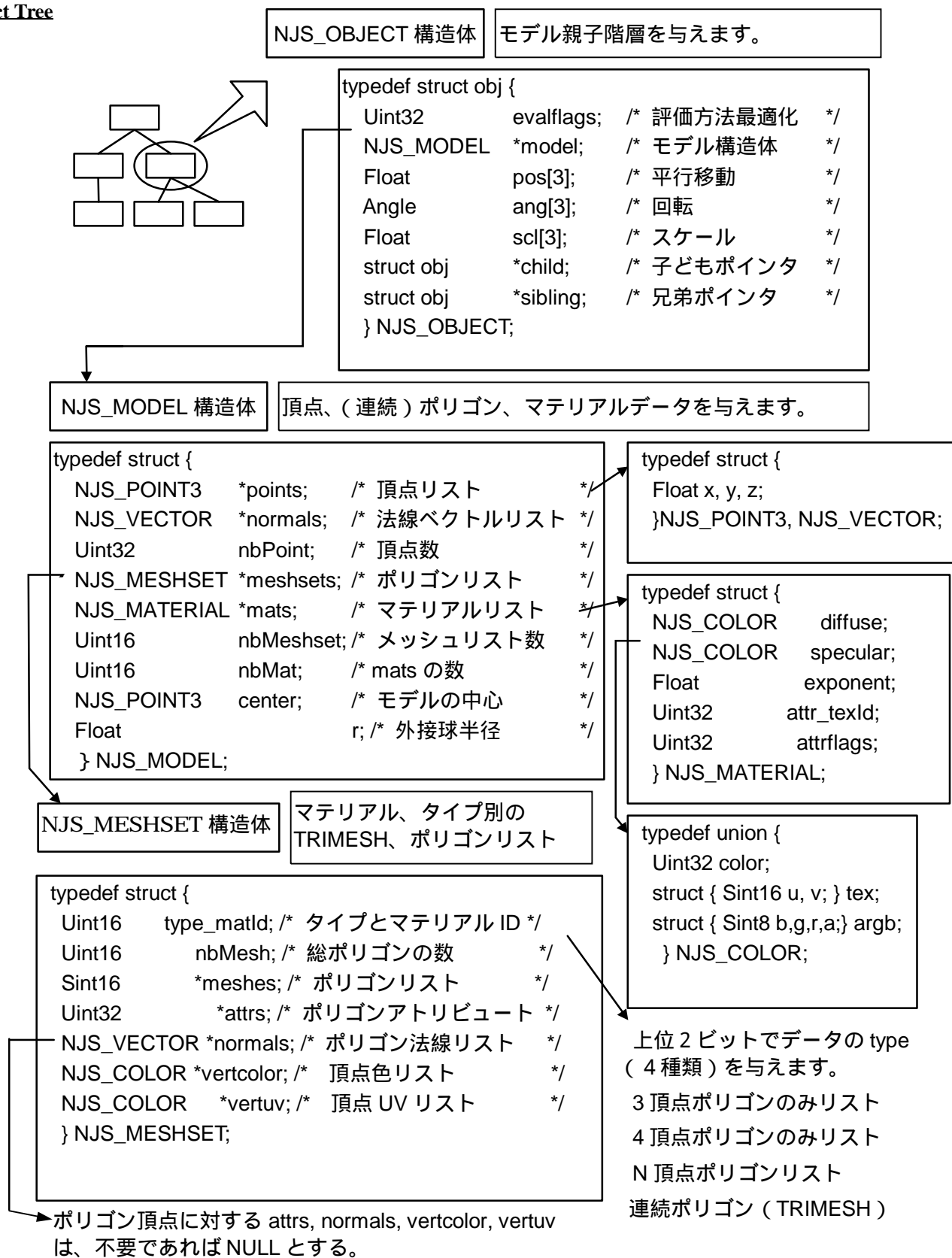
モデル以外のモーションおよびテクスチャについては、引き続き現行の形式が使用されます(ただしカメラ、ライト対応のための構造体メンバの型が変更されます)。

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. モデル構造体

### 構造体図

#### 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使用するをお勧めしますが、ユーザ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ー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)
```

### /\* カテゴリーコード \*/

```
#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)
```

- 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。

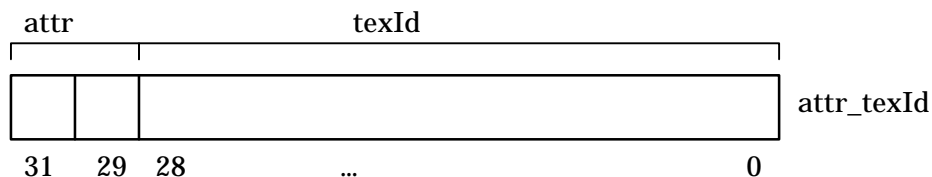
## テクスチャリスト構造体

```
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、『3.3 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 3.モデル構造の『3.1 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、『3.3 Ninja アトリビュート』を参照してください。

## モデル構造体

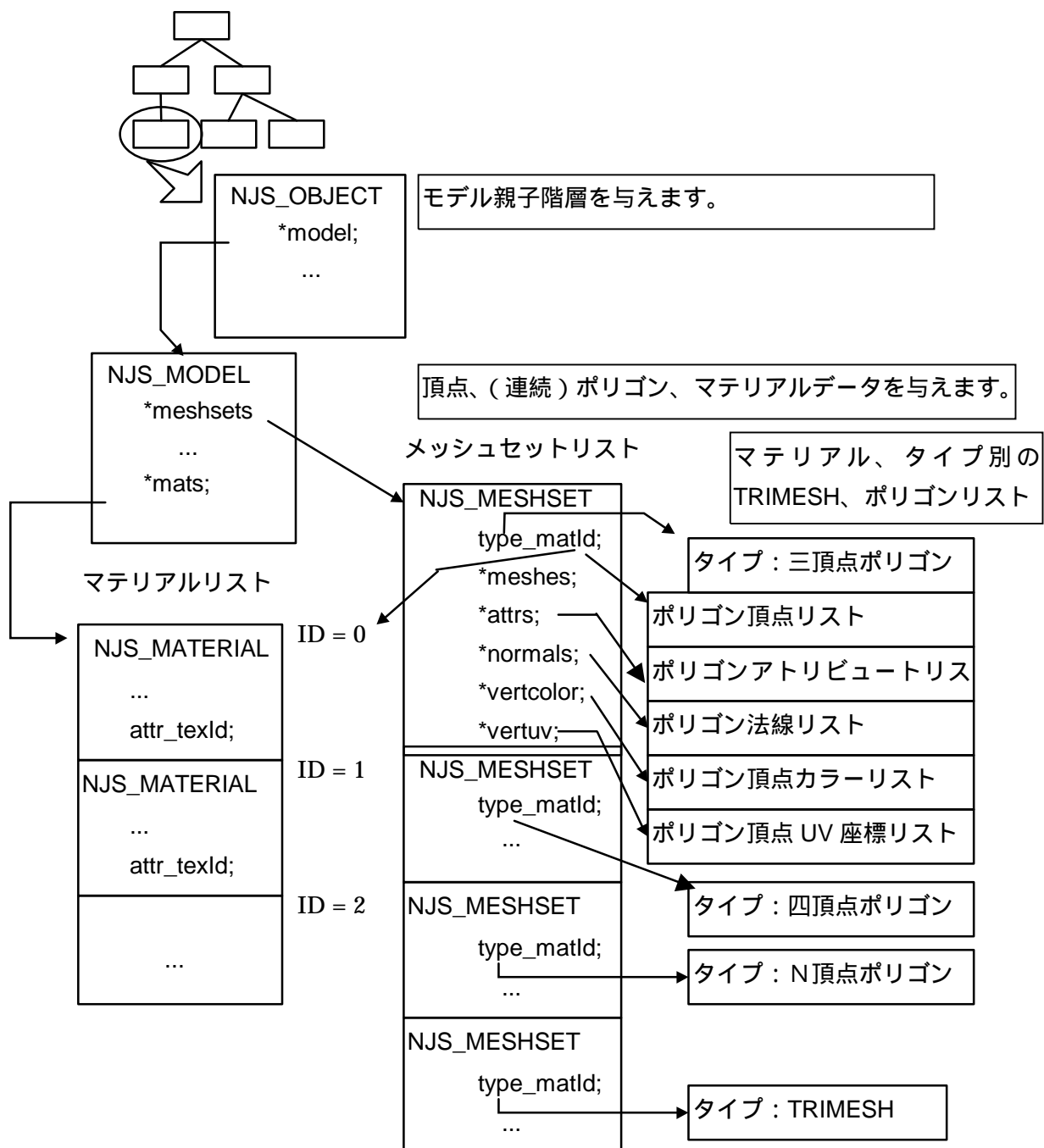
```
typedef struct {
    NJS_POINT3      *points;      /* 頂点リスト                      */
    NJS_VECTOR      *normals;     /* 頂点の法線ベクトルリスト        */
    Uint32          nbPoint;      /* 頂点の数。                      */
    NJS_MESHSET     *meshsets;    /* ポリゴン、TRIMESH リスト        */
    NJS_MATERIAL     *mats;       /* 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。

### 3. モデル構造

構造体関連図

Object Tree

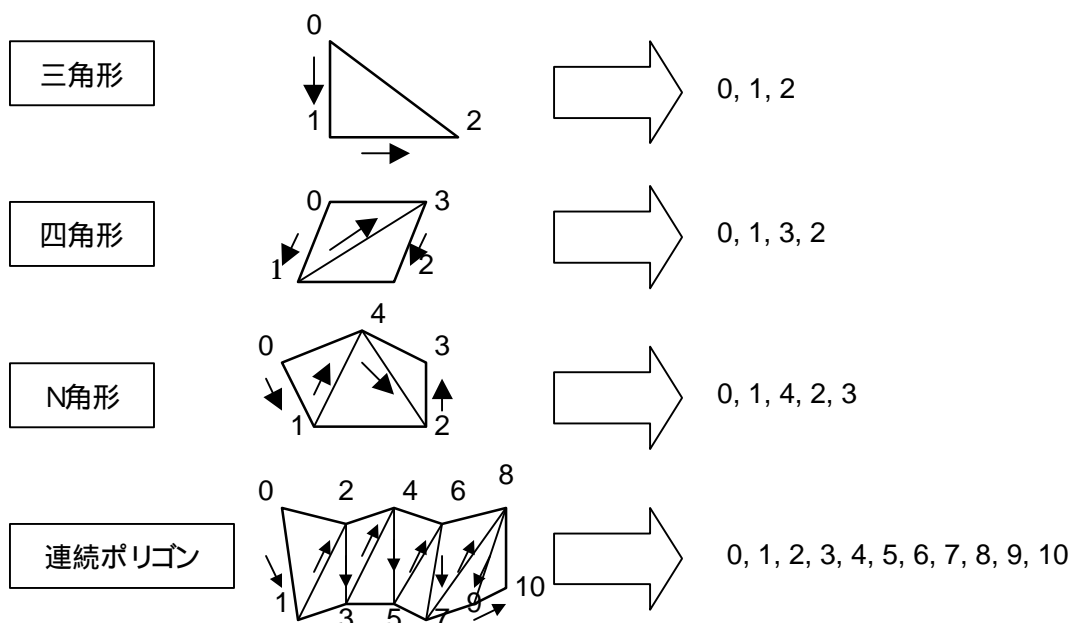


- メッシュセットリストは、使用されているもののみのリストです。  
例えば、三角形ポリゴンのみのリストだけの場合は、nbMeshset=1 です。
- attrs, normals, vertcolor, vertuv は、不要であれば NULL とします。

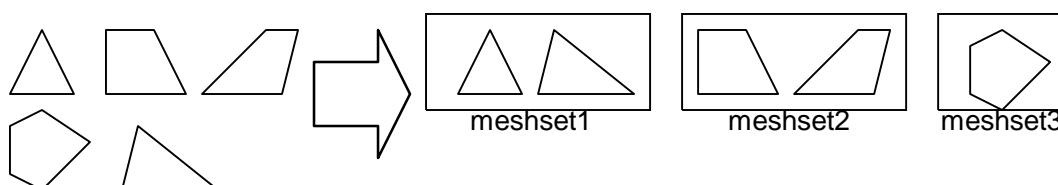
### 3.1 メッシュセット

```
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;
```

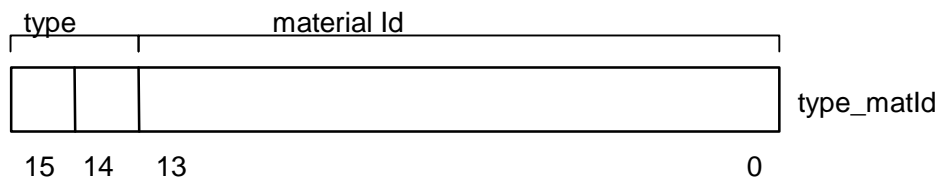
- 三角形ポリゴンのみ、四角形ポリゴンのみ、N角形ポリゴンのみ、TRIMESH (連続ポリゴン) のみのデータ列を格納します。
- 頂点の順番は、すべて連続ポリゴンの描画順番 (ジグザグ) になります。



- \*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[]	$\overline{\{3, 4, 5\}}$	$\overline{\{9, 8, 6\}}$	$\{2, 10, 7, 13, 14, 11, \dots\}$
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[]	$\overline{\{3, 4, 5, 9\}}$	$\overline{\{8, 6, 2, 10\}}$	$\{7, 13, 14, 11, \dots\}$
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}, ...}

vertuv[] = {0xEFAB,0xFF98,0x44FF,...}

vertcolor[] = {0xFFFF,0xEEEE,0CCCC,...}

nbMesh = trimesh 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 )

meshes[] = {  
                    Polygon1                      Polygon2  
                    ┌──────────┴──────────┐  
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。

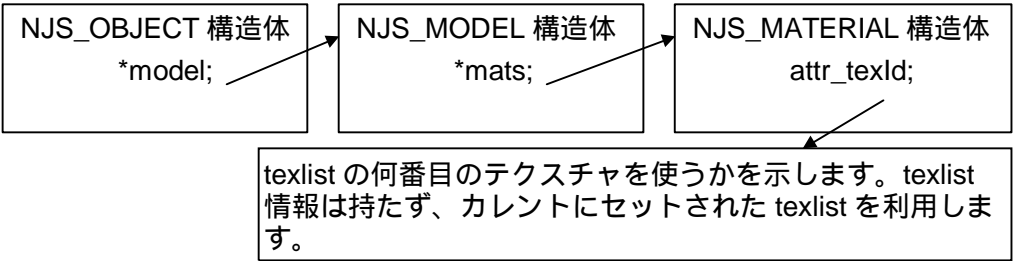
nbMesh = N 角形ポリゴン数

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

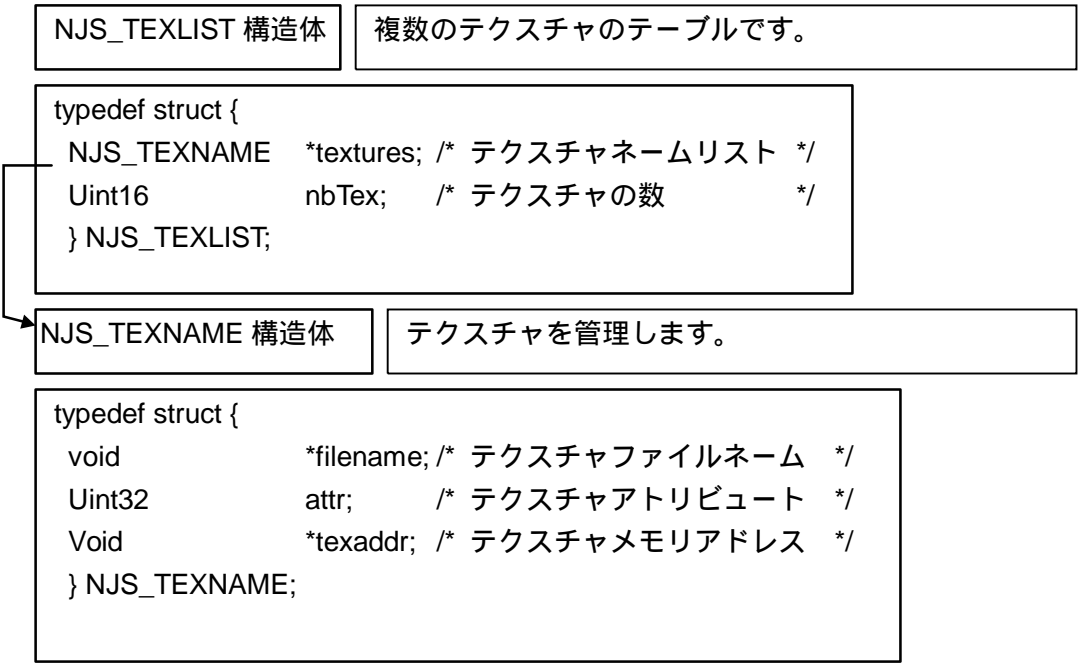
### 3.2 テクスチャ構造

#### 構造体図

#### ObjectTree



#### Texlist



#### texlist

- ・ テクスチャは、実ファイルと texlist で管理されます。
- ・ テクスチャメモリへの書き換えは、texlist 単位で行われます。
- ・ ファイルの代わりにメモリデータを指定できます。

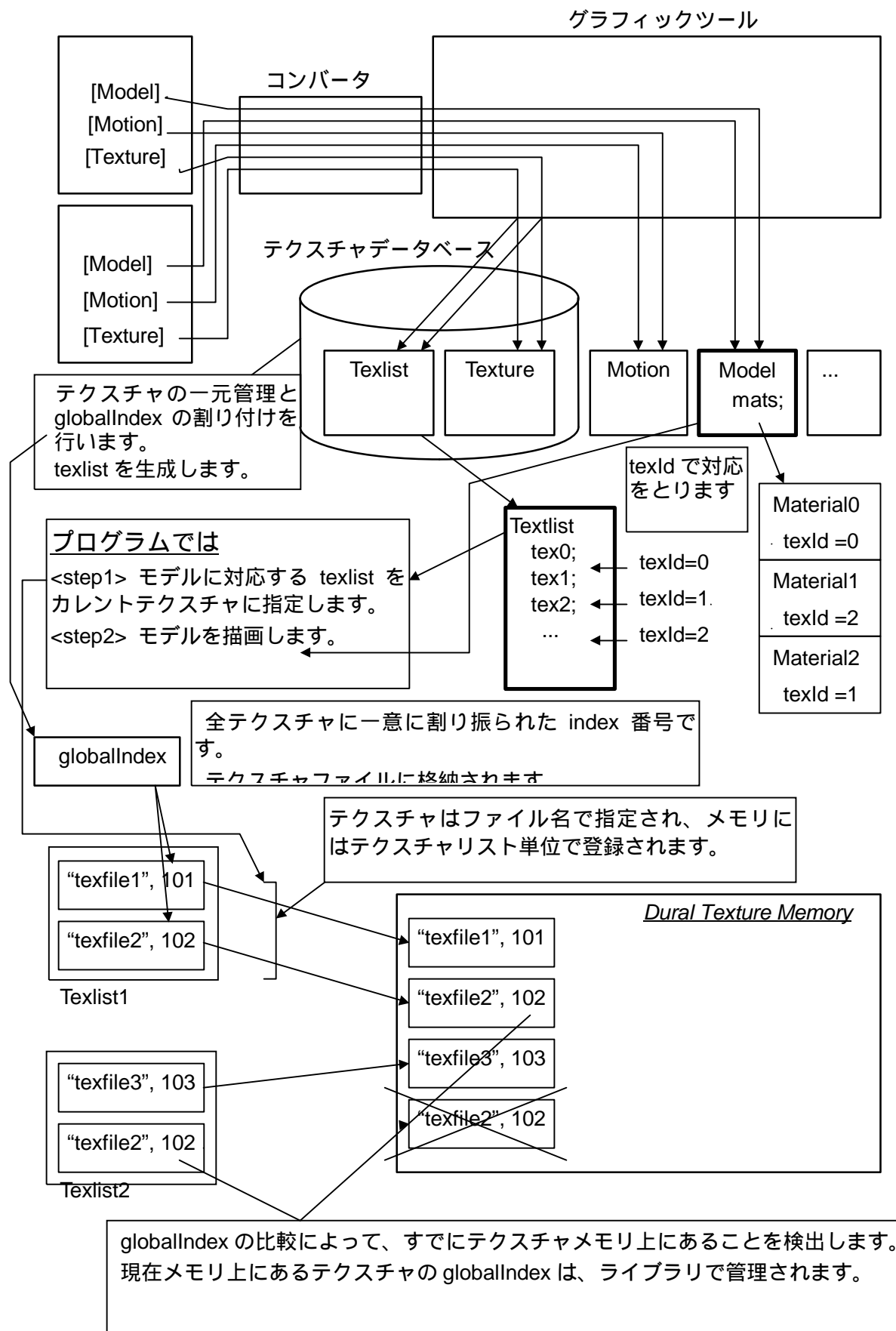
#### globalIndex

- ・ 全テクスチャに対し、一意な番号を割り振ります。
- ・ 番号の比較によりテクスチャ重複をチェックします。
- ・ globalIndex はテクスチャ

#### texaddr

- ・ ライブラリ関数を使用して、カレントテクスチャを texlist 単位でセットします。この時テクスチャメモリに登録されたアドレスを格納します。

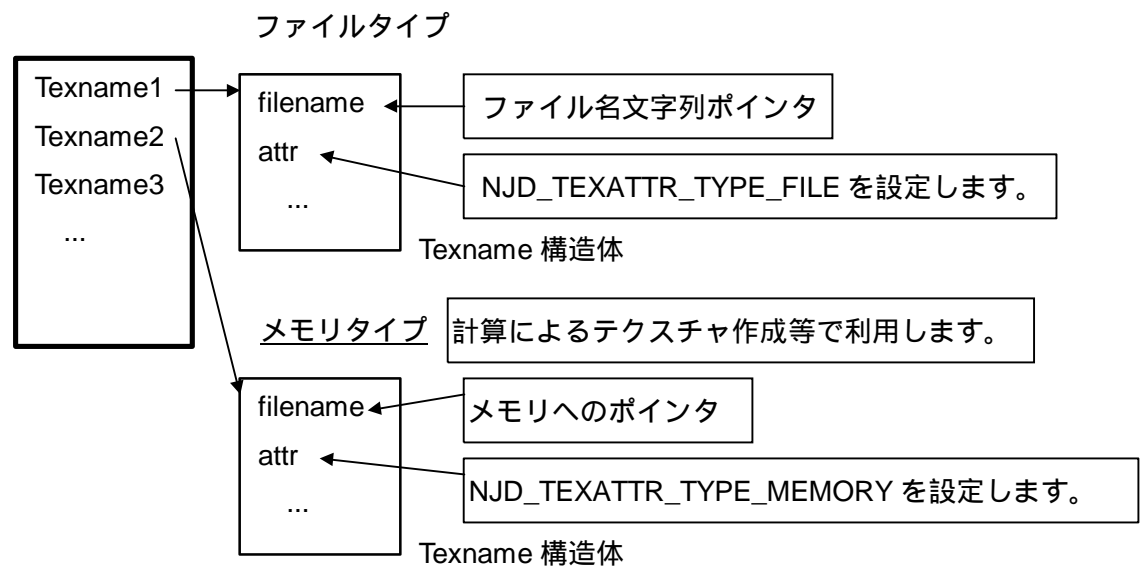
## テクスチャ処理の概要



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。

Texlist



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。

## テクスチャ構造解説

- 通常モデルには複数のテクス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、複数のテクスチャファイル名の配列で構成されています。
- テクスチャファイルには `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、必ずテクス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。
- 同一テクスチャを含む複数の `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クスチャ群が、各テクス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、出現したテクスチャがすでに `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。
- ユーザが独自に `globalIndex` を割り付けたい場合、使われる全テクス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クスチャファイルの `globalIndex` チャンクに書き込みます。
- モデルのテクスチャ情報は `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。
- テクスチャとモデルの対応は、モデル描画の前に利用する `texlist` をカレントテクス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、簡単にテクスチャの差し替えができます。
- `texaddr` には、テクスチャメモリ上のアドレスがカレントテクス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` のテクスチャファイル名の記述において、テクス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。
- テクスチャの拡張子“.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` 内のテクスチャ数とオブジェクトツリーで使われているテクスチャ数の不整合を検出しません。

### 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 attrs 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。

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

<u>31-29</u> :	SRC Alpha Instruction	( ブレンディングパラメータ、後述 )
<u>28-26</u> :	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。)
<u>20</u> :	Use Alpha	( の有効/無効。1 の時有効。)
19:	Ignore Specular	( スペキュラの無視。1 の時無効。)
<u>18-17</u> :	Flip UV	( フリップ制御 )
<u>16-15</u> :	Clamp UV	( クランプ制御 )
14-13:	Filter-Mode	0 ... Point Sampled(hard spec) 1 ... Bilinear Filter(hard spec) 2 ... Tri-liner Filter(hard spec)
<u>12</u> :	Super-Sample Texture	( Anisotropic Filter ON/OFF )
<u>11-8</u> :	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。

```
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	( $O_R$ , $O_G$ , $O_B$ , $O_A$ )
Inverse 'Other' Colour	3	( $1 - O_R$ , $1 - O_G$ , $1 - O_B$ , $1 - O_A$ )
SRC Alpha	4	( $S_A$ , $S_A$ , $S_A$ , $S_A$ )
Inverse SRC Alpha	5	( $1 - S_A$ , $1 - S_A$ , $1 - S_A$ , $1 - S_A$ )
DST Alpha	6	( $D_A$ , $D_A$ , $D_A$ , $D_A$ )
Inverse DST Alpha	7	( $1 - D_A$ , $1 - D_A$ , $1 - D_A$ , $1 - D_A$ )

'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)を掛け合わせるによって行われます。

Example 'D' Adjust bit pattern	Equivalent value
00.00	<i>Illegal</i>
00.01	0.25
01.00	1.0
11.11	3.75

設定不可

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)

```

### 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イ階調で用意します。R G B 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。

#### V Q

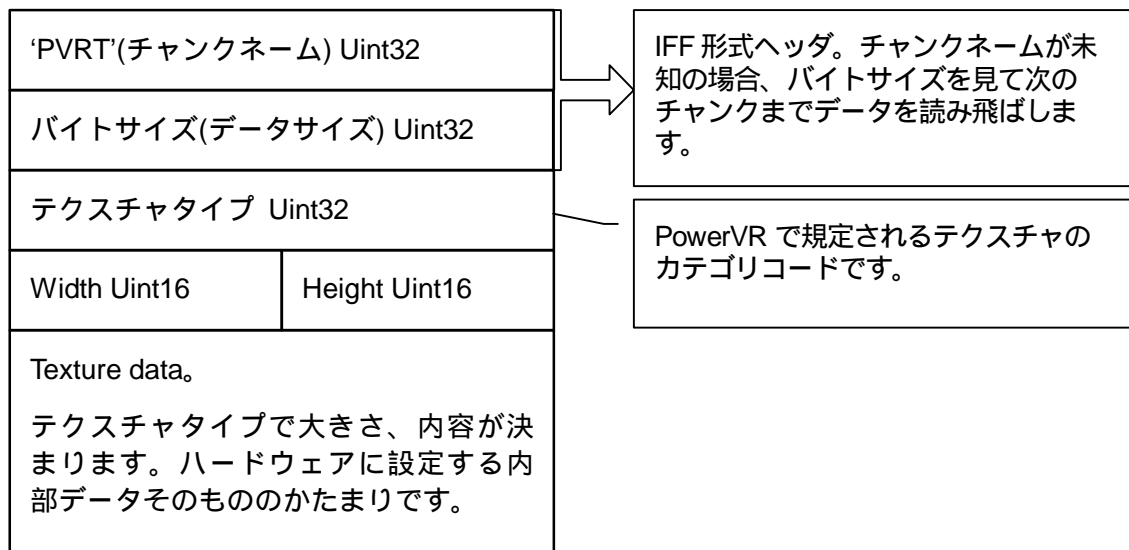
ベクトル量子化によりテクスチャを圧縮します。詳細はV Qの仕様を参照してください。

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。

'PVRI'(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。