



# **Ninja2 CompactShape 仕様書**

**(08/11/2000)**

<b>1. COMPACTSHAPE 仕様</b>	<b>3</b>
1.1 特徴	3
1.2 エンベロープとの共存	3
1.3 VERTEXGROUP (頂点グループ)	4
1.4 データ構造	4
1.5 VRS(VERTEX RESOURCE)ファイル	5
1.6 制限事項	6
<b>2. 構造体とマクロ定義</b>	<b>7</b>
2.1 構造体	7
2.2 COMPACTSHAPE 用 CHUNK VERTEX	9
<i>ChunkName</i> : 'NJD_CV'	10
<i>ChunkName</i> : 'NJD_CV_D8'	11
<i>ChunkName</i> : 'NJD_CV_VN'	12
<i>ChunkName</i> : 'NJD_CV_VN_D8'	13
<i>ChunkName</i> : 'NJD_VN'	14
<i>ChunkName</i> : 'NJD_VN_D8'	15
<i>ChunkName</i> : 'NJD_D8'	16
<b>3. アスキーフォーマット</b>	<b>17</b>
3.1 アスキー出力のマクロ定義	17
3.1.1 <i>Nja</i> の基本定義によるマクロ	17
3.1.2 チャンクフォーマット定義によるマクロ	17
3.1.3 <i>CompactShape</i> 専用のチャンクフォーマット頂点リストマクロ	17
3.1.4 <i>CompactShape</i> 用の <i>ChunkVertex</i> フラグ定義	18
3.1.5 モーション定義によるマクロ	18
3.1.6 <i>CompactShape</i> 定義によるマクロ	19
3.2 アスキー出力の例	21
3.3 アスキー出力解説	24
<b>4. バイナリフォーマット</b>	<b>30</b>
4.1 バイナリ拡張子	30
4.2 バイナリ構造	31
4.3 POF0 のアルゴリズム	32

# 1. CompactShape 仕様

## 1.1 特徴

特徴は次の通り。

チャンクモデル形式による頂点と法線の一本化。

頂点モーションが必要な頂点だけのデータ表現。

同一モデル内でのエンベロープ（頂点ウェイトによる変形）と頂点モーションの共存。

頂点カラーを頂点モーションデータの一部とできる（ただしこれを使用する場合はエンベロープとの共存はできない）。

vrs（vertex resource）ファイルの導入。

モーションリソース（motion resource）に対する mrs ファイルのように頂点グループ定義に vrs ファイルを導入する。vrs ファイルから頂点グループデータを参照することで頂点グループの設定は基本となるモデル階層に対してのみすればよくなる。その他の（モーション生成用）モデル階層は mrs 参照（vrs 参照）を利用することで頂点グループを反映する。

ShapeList の導入。

キーフレーム形状の時間軸上での共有。例えば顔の表情などで4つの基本形状をモーション時間軸の中で繰り返し使うような場合 ShapeList を繰り返し参照することで同じ形状を再利用できる。ユーザの工夫でデータサイズを大幅に小さくできる。

KeyframeGroup の導入。

従来の頂点モーションでは同一 keyframe を持つ複数の頂点リストに対し別々に keyframe を持っていた。これだとデータサイズとキー探索の時間の無駄が多い。また今回エンベロープと同時での実行を可能とするためモデル階層内に分散する頂点リスト群に対し同一の keyframe を適用、表現する必要がある。

モデル階層内で同一の keyframe データを共有可能なモデル階層頂点データを集めて KeyframeGroup を作成する。モーションの中では ShapeId と KeyframeGroup とその中でのエントリ番号を指定することでモデル階層頂点モーションデータを一意に指定する。

一体整形（ワンスキン）モデルにおいて KeyframeGroup は一つであり全体に一つの keyframe しか持たず効率の良い表現が実現できる。

## 1.2 エンベロープとの共存

CompactShape ではエンベロープ（ウェイト値による変形）と頂点モーション共存において単一の頂点に対しこの両方のアルゴリズムを同時に実行することはしない。頂点モーション+エンベロープの頂点は二つのアルゴリズムの実行結果を合成した頂点位置を新たな頂点モーションデータとして処理する。複雑な処理を避け高速化を図る。この方法により

現在のライブラリに実装されるアルゴリズムで実行可能。

頭の先から足の先まで同一モデル（ワンスキンモデル）において体はエンベロープ、顔の表情は頂点モーションによるデータが実現可能。

上腕の筋肉盛り上がりなど現在のエンベロープでは表現できない部分に対し局所的に頂点モーションを適用できる。ただしエンベロープ+頂点モーションの部分は頂点モーションによる近似であるので完全ではない。ユーザレベルでの見た目の調整が必須。

上記項目を前提としてデータに次の最適化をする。

エンベロープ+頂点モーションの頂点はウェイト100%としてエンベロープのウェイト値を削除する（ウェイト値による頂点分散による計算量の増加の抑制）。エンベロープ1の部分的適用。

エンベロープ+頂点モーションの頂点は100%ウェイトの頂点としてモデル頂点リストの先頭(チャンク)に集められる(頂点モーションデータの効率的なデータ表現を実現)。エンベロープ1適用により頂点モーションの適用される頂点が先頭の複数のチャンクに別れる場合もある。

オプションでエンベロープ1を適用せず元モデルに100%の頂点としてグローバルに頂点モーションを実行できる。

以上の方法によりエンベロープ+頂点モーションデータを効率よく同時に実行する。

## 1.3 VertexGroup (頂点グループ)

CompactShape のデータ作成においてモデルデータの一部の頂点に頂点モーションを適用するための頂点グループ指定が必要となる。このために各モデラーの頂点グループ指定機能を利用する。頂点グループ指定はモデラーごとに異なる。例えばLightWave3D(6.0以降)ではクラスタ機能で頂点グループを作成できる。これはモデルデータであるlwo2に格納されSDKからこの情報を読み出せる。Softimage、MAYAもクラスタによる頂点グループ指定が可能。3D Studio MAXではこの目的に適するグループ指定はないが自前のプラグインとして制作が簡単であるため頂点グループを指定できるプラグインを作成する。

ここで頂点グループの自動生成はあえてしない。見た目からのユーザ指定が重要。頂点グループの頂点の追加、削除をした場合再コンバートは必須となる。ただしユーザの利便性を考え次の動作をする。

対象となるモデルがエンベロープを持たず頂点グループ指定がなく頂点モーションをする場合頂点リスト全体を頂点グループとして頂点モーションデータを出力する。

基本となるvrsファイル参照によりすべてのシーンファイルの頂点グループを修正する必要はない。

vrsファイル名はmrsファイルに書き込まれており従来のmrs参照時にvrsファイルがあればこれを間接参照する。mrsファイル、vrsファイルはセットで扱う必要がある。

またモデラーデータに対し頂点グループ情報をシーンレベルで保存するためのnrv(Ninja Resource Vertex)も合わせて定義される。これにはvrs構造を基本としシーンに含まれるすべてのモデル階層の頂点グループデータが保存される。nrvを利用するかどうかは使用されるモデラー及びモデラーのデータ格納能力に依存する。nrvはモデラーが保存できない頂点グループデータを保存するための予備的な手段として利用される。

## 1.4 データ構造

次にデータ構造について説明する。

エンベロープは従来のままの構造とする。ただし頂点グループにより頂点モーションの適用を指定された頂点はエンベロープ1(100%ウェイト)を適用する。Ninjaコンバータではマルチウェイトの元データでもコンバータオプションでもっともウェイトの割合が高いモデルに100%としてデータを生成できる。これを頂点グループに対し部分的に適用する。

更に100%ウェイトの頂点群の中にはもともと100%ウェイトであるもの、頂点グループ指定されたものの2種類があるが頂点グループ指定されたものを先頭に集める。これらは先頭に集められるが一つのチャンクデータの中に出力される。

頂点モーションデータはチャンクフォーマットのNJD\_CVとNJD\_CV\_VN及びNJD\_CV\_D8、NJD\_CV\_VN\_D8を利用する。これらは頂点のみ、頂点+法線、頂点+頂点カラー、頂点+法線+頂点カラーを意味するがさらに効率の良いデータ表現をするためにCompactShape専用として法線のみ、法線+頂点カラー、頂点カラーのみも定義する。以下にCompactShapeで利用されるChunk Vertexの種類をまとめる。

NJD_CV	頂点のみ。
--------	-------

NJD_CV_VN	頂点 + 法線。
NJD_CV_D8	頂点 + 頂点カラー。
NJD_CV_VN_D8	頂点 + 法線 + 頂点カラー。
NJD_VN	法線のみ。CompactShape 専用。
NJD_VN_D8	法線 + 頂点カラー。CompactShape 専用。
NJD_D8	頂点カラーのみ。CompactShape 専用。

データは NJS\_MDATA1 構造体に CompactShape データを格納し従来の NJS\_MOTION 構造体のデータとして設定することで表現される。NJS\_MDATA1 構造体には各モデルに適用される KeyframeGroup 配列を格納する NJS\_CSHAPEDATA 構造体のポインタが格納される。

KeyframeGroup には同一の keyframe を共有可能なモデルの頂点リストが格納される。NJS\_CSHAPEDATA 構造体には ShapeId と時間を指定するキーフレームアニメーションデータに加え KeyframeGroup に格納される何番目の頂点リストを使うかを指定するための EntryId が格納される。

ファイル拡張子は従来の nas, njs を利用する。アスキーフォーマットの nas の一行目のヘッダーには Ninja2AsciiCompactShape の文字列が格納される。頂点カラーアニメーションに関してはコンバータ内部でのプレライト計算機能によりモデラー上でのライトアニメーションの結果を埋め込むことができる。

CompactShape 用頂点データは中間バッファ上のオフセットを与える IndexOffset と頂点数を与える nbIndices データを省略する。その省略されていることを明示するために CompactShape 用データであることを示すフラグがチャンクの ChunkHead に設定される（詳細は後述）。

またオプションで元モデルの頂点としてグローバルに頂点モーションする場合は頂点グループで指定された頂点は元モデルに 100% の頂点として処理される。この場合エンベロープを使っているワンスキンモデルでは関節の動きに頂点は追従しないのでモデルモーションと頂点モーションの同時実行は必須となる。

CompactShape では nas(njs) には ShapeList のみ出力される。モーション構造体はこれに対応する nam(njm) に格納される。つまり頂点モーションをするモデルに関わるモーションファイル nam にはモーション構造体が二つ出力される。バイナリ出力ではモーション構造体を含むチャンクが二つ格納される。ライブラリ関数呼び出し時にユーザにより二つのモーションを同時に使うかが選択される。nam と nas によりモデルがモーションする手順はわからない。

nas ファイルは同一 ShapeList から構成される頂点モーションにおいて共有できる。例えば顔の表情の基本パターンを作成しこれを複数の CompactShape モーション構造体から ShapeId および EntryId で参照できる。

また Ninja2 ではアスキー出力 (nam) においてアクション構造体の出力をしない。また SimpleShape は廃止される。

## 1.5 VRS(Vertex Resource)ファイル

次に vrs ファイルについて説明する。

vrs は vertex resource を意味しユーザが設定したモデル階層分の頂点グループおよび頂点に付随する情報を格納する。

vrs は mrs と同様オプション指定で出力するかどうかを選択できる。

vrs は mrs にファイル名を格納し mrs 経由で参照される。コンバータからの参照は従来どおり mrs のみの参照で良い。そのため vrs ファイルは常に mrs ファイルとセットで扱う必要がある。

mrs に vrs 指定があったとしてもそのファイルが実在しない場合無視されカレントのシーンファイルに格納される頂点グループが利用される。

頂点グループに登録された頂点が CompactShape の対象頂点となる。

vrs データはモデル内部のすべての選択された頂点を格納する必要があるため大きくなるのでバイナリ出力かつ独自の圧縮をする。mrs と統合しない。

vrs 形式の仕様は他のドキュメントを用意する。

## 1.6 制限事項

---

同一頂点にエンベロープ、頂点モーションの両方の影響があるモデル階層を変換する場合基本的に ShapeList 上の一つの形状を複数回使いまわすことはできない。これはエンベロープと頂点モーションの合成結果を ShapeList に埋め込むためである。

顔の変形などではエンベロープの影響のない頂点に頂点モーションを設定することで同一形状の使いまわしを表現する。

## 2. 構造体とマクロ定義

### 2.1 構造体

#### ShapeList 用構造体

ShapeList は頂点モーションで利用される頂点形状を格納する。時間軸は持たずこれからモーションファイルで必要とされるすべての頂点リストをチャンク形式で一次元配列にしさらにそれを並べてリスト化し格納する。nas(njs)に出力される。

```
typedef struct {
    Sint32                *cv; /* Chunk Vertex ポインタ */
} NJS_CSHAPECV;

typedef struct {
    NJS_CSHAPECV          *cvlist; /* KeyframeGroup Chunk Vertexデータ */
    Uint16                groupId; /* KeyframeGroup Id */
    Uint16                reserved; /* リザーブ領域 */
} NJS_CSHAPE_KEYGROUP;

typedef struct {
    NJS_CSHAPE_KEYGROUP    *shapes; /* KeyframeGroupの配列 */
    Uint32                nbShape; /* KeyframeGroupのエントリ数 */
} NJS_CSHAPELIST;
```

#### モーション構造体

モデルモーションと共に nam ( njm ) ファイルに出力される。

```
typedef struct {
    void                *mdata; /* モデル階層分のデータ配列 */
    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、NJS\_CSHAPEDATA 構造体で管理される KeyframeGroup をモデルトレースに対応させるための NJS\_MDATA1 が格納される。

```
#define NJD_MTYPE_LINEAR          0x0000 /* 線形補完 */
#define NJD_MTYPE_SPLINE         0x0040 /* スプライン補完 */
#define NJD_MTYPE_USER           0x0080 /* ユーザ関数補完 */
#define NJD_MTYPE_MASK           0x00c0 /* 抽出マスク */
```

inp\_fn に格納される補完計算方法は線形補完 ( NJS\_MTYPE\_LINER )。要素数は 1 固定。

```

#define NJD_MTYPE_POS_0          (BIT_0)    /* NJS_MKEY_F を利用          */
#define NJD_MTYPE_ANG_1          (BIT_1)    /* NJS_MKEY_A を利用          */
#define NJD_MTYPE_SCL_2          (BIT_2)    /* NJS_MKEY_F を利用          */
#define NJD_MTYPE_VEC_3          (BIT_3)    /* NJS_MKEY_F を利用          */
#define NJD_MTYPE_VEC_0          (BIT_4)    /* NJS_MKEY_F を利用          */
#define NJD_MTYPE_SANG_1         (BIT_5)    /* NJS_MKEY_SA を利用         */
#define NJD_MTYPE_TARGET_3       (BIT_6)    /* NJS_MKEY_F を利用          */
#define NJD_MTYPE_ROLL_6         (BIT_7)    /* NJS_MKEY_SA1 を利用        */
#define NJD_MTYPE_ANGLE_7        (BIT_8)    /* NJS_MKEY_SA1 を利用        */
#define NJD_MTYPE_RGB_8          (BIT_9)    /* NJS_MKEY_F を利用          */
#define NJD_MTYPE_INTENSITY_9    (BIT_10)   /* NJS_MKEY_F2 を利用         */
#define NJD_MTYPE_POINT_9        (BIT_12)   /* NJS_MKEY_F2 を利用         */
#define NJD_MTYPE_QUAT_1         (BIT_13)   /* NJS_MKEY_QUAT を利用       */
#define NJD_MTYPE_SHAPEID        (BIT_14)   /* NJS_MKEY_SHAPEID を利用    */
#define NJD_MTYPE_EVENT_4        (BIT_15)   /* NJS_MKEY_???を利用         */

```

CompactShape では NJD\_MTYPE\_SHAPEID を利用する。他の成分と組み合わせられて使用されることはない。

## NJS\_MKEY\_SHAPEID 構造体

```

typedef struct {
    Uint32          keyframe; /* キーフレーン番号          */
    Uint32          shapeId;  /* ShapeList のエントリ ID    */
} NJS_MKEY_SHAPEID;

```

shapeId は nas ファイルに格納される ShapeList のエントリ ID を指定する。

## NJS\_CSHAPEDATA 構造体

```

typedef struct {
    NJS_MKEY_SHAPEID *keys; /* モーションポインタ          */
    Uint16            nbKey; /* EntryId 及びキーフレーン数 */
    Uint16            entryId; /* KeyframeGroup の EntryId    */
} NJS_CSHAPEDATA;

```

NJS\_MKEY\_SHAPEID 構造体によるキーフレーン頂点モーションデータを格納する。キーの数が nbKey に格納される。entryId は ShapeId で指定された ShapeList の KeyframeGroup に格納される何番目の頂点リストを使うかを指定する。

## NJS\_MDATA1 構造体

```

typedef struct {
    void             *p[1]; /* NJS_CSHAPEDATA 配列ポインタ */
    Uint32            nb[1]; /* NJS_CSHAPEDATA 配列のエントリ数 */
} NJS_MDATA1;

```

p には NJS\_CSHAPEDATA の配列のポインタが格納される。配列のサイズはモデルに適用される KeyframeGroup の数になる。nb には配列の長さが格納される。各 KeyframeGroup はチャンクフォーマットの頂点リストの先頭に配置される頂点チャンクに適用される。二つある場合は先頭の二つのチャンクに配列順にそれぞれ適用される。

頂点モーション不要なモデルのエントリは NULL となる。頂点モーションに関わらないモデルをモーションデータのエントリから削除する EvalShapeSkip が適用される場合適用されたモデルのエントリが MDATA1 の上から消える。



## 2.2 CompactShape 用 Chunk Vertex

---

CompactShape で使用する ChunkVertex について説明する。

通常の ChunkVertex は中間バッファへのオフセットと頂点数を与えるインデックスの数を含んでいる。

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

ここで longsize はロングワード単位 ( 4 バイト ) のオフセットを意味する。

CompactShape データでは中間バッファへ書き込む前のデータに対し計算結果を書き込むため IndexOffset は常に 0 であり nbIndices は longsize から求めることができるため IndexOffset と nbIndices は省略するものとする。

```
[ChunkHead(31-16)|longsize(15-0)][Data]  
ChunkHead:  
    < CompactShape で利用する従来の Chunk Vertex >  
    NJD_CV、NJD_CV_VN、NJD_CV_D8、NJD_CV_VN_D8  
    < CompactShape 専用 Chunk Vertex >  
    NJD_VN、NJD_VN_D8、NJD_D8
```

ChunkHead には CompactShape 用 ChunkVertex を示すためのフラグが設定される。

```
#define NJD_FV_SHAPE    (0x40<<NJD_FV_SHIFT)    /* compact shape flag */
```

略号の意味は以下の通り。

CV	: Vertex
VN	: Vertex normal
D8	: Diffuse ARGB8888

ChunkName : 'NJD\_CV'

(Chunk Vertex)

**概要 :**

vlist において頂点リストを定義する。頂点座標データのみ。

**形式 :**

```
[ChunkHead(31-16) | longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE | NJD_CV
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    x,y,z, ...
```

**説明 :**

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

頂点座標データのみ書き換えて頂点モーションをする。頂点法線変化の少ない頂点モーションで利用する。データサイズがもっとも少ない。エンベロープと同時に使うと連続部分の表現が難しいので CompactShape 単体での使用を推奨。

CompactShape データ内で使用する場合はモデル出力時に出力される IndexOffset と nbIndices フィールドを省略する。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。

ChunkName : 'NJD\_CV\_D8'

(Chunk Vertex Diffuse ARGB8888)

#### 概要：

vlist において頂点リストを定義する。頂点座標データあり。頂点法線なし。Diffuse 頂点カラーあり。

#### 形式：

```
[ChunkHead(31-16)|longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE|NJD_CV_D8
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    x,y,z,D8888,...
```

#### 説明：

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

頂点座標データあり。頂点法線なし。Diffuse 頂点カラーあり。頂点カラーを使い光源計算をしないモデルで利用する。頂点カラーにはコンバータのプレライト機能で頂点カラーアニメーションを埋め込むことができる。

通常モデラーでは頂点カラーアニメーションは作成できない。コンバータのプレライト機能で頂点カラーをコンバート時に埋め込むことで頂点カラーアニメーションを実現する。

CompactShape データ内で使用する場合はモデル出力時に出力される IndexOffset と nbIndices フィールドを省略する。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。

ChunkName : 'NJD\_CV\_VN'  
(Chunk Vertex VertexNormal)

#### 概要 :

vlist において頂点リストを定義する。頂点座標データあり。頂点法線あり。Diffuse 頂点カラーなし。

#### 形式 :

```
[ChunkHead(31-16)|longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE | NJD_CV_VN
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    x,y,z,nx,ny,nz, ...
```

#### 説明 :

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

頂点座標データあり。頂点法線あり。Diffuse 頂点カラーなし。もっとも標準的な頂点リスト。動きの激しい頂点モーションで利用する。法線も再計算されるため陰影も含め綺麗に変形ができる。エンベロープと同時に使う場合はこの形式を推奨。

CompactShape データ内で使用する場合はモデル出力時に出力される IndexOffset と nbIndices フィールドを省略する。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。

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する。頂点座標データあり。頂点法線あり。Diffuse 頂点カラーあり。

#### 形式 :

```
[ChunkHead(31-16)|longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE|NJD_CV_VN_D8
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    x,y,z,nx,ny,nz,D8888,...
```

#### 説明 :

```
#define NJD_CV_VN_D8 (NJD_VERTOFF+10)
```

頂点座標データあり。頂点法線あり。Diffuse 頂点カラーあり。頂点カラーを使いかつ光源計算をする場合に利用する。現時点では使用頻度は低いと考えるが組み合わせ上可能であるため定義する。エンベロープと同時に使うと連続部分の表現が難しいので CompactShape 単体での使用を推奨。

通常モデラーでは頂点カラーアニメーションは作成できない。コンバータのプレライト機能で頂点カラーをコンバート時に埋め込むことで頂点カラーアニメーションを実現する。

CompactShape データ内で使用する場合はモデル出力時に出力される IndexOffset と nbIndices フィールドを省略する。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。

ChunkName : 'NJD\_VN'

(Chunk VertexNormal)

**概要 :**

CompactShape 専用。vlist において頂点リストを定義する。頂点法線のみ。

**形式 :**

```
[ChunkHead(31-16)|longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE|NJD_VN
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    nx,ny,nz,...
```

**説明 :**

```
#define NJD_VN (NJD_SHAPEOFF+0)
```

頂点法線のみアニメーションする場合に使用する。エンベロープと同時に使うと連続部分の表現が難しいので CompactShape 単体での使用を推奨。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。

ChunkName : 'NJD\_VN\_D8'

(Chunk VertexNormal and Diffuse ARGB8888)

#### 概要 :

CompactShape 専用。vlist において頂点リストを定義する。頂点座標データなし。頂点法線あり。Diffuse 頂点カラーあり。

#### 形式 :

```
[ChunkHead(31-16)|longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE|NJD_VN_D8
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    nx,ny,nz,D8888,...
```

#### 説明 :

```
#define NJD_VN_D8 (NJD_SHAPEOFF+1)
```

頂点座標データなし。頂点法線あり。Diffuse 頂点カラーあり。Diffuse 頂点カラーを使いかつ光源計算をする場合に利用する。現時点では使用頻度は低いと考えるが組み合わせ上可能であるため定義する。エンベロープと同時に使うと連続部分の表現が難しいので CompactShape 単体での使用を推奨。

通常モデラーでは頂点カラーアニメーションは作成できない。コンバータのプレライト機能で頂点カラーをコンバート時に埋め込むことで頂点カラーアニメーションを実現する。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。

ChunkName : 'NJD\_D8'

(Chunk Diffuse ARGB8888)

#### 概要 :

CompactShape 専用。vlist において頂点リストを定義する。Diffuse 頂点カラーのみ。

#### 形式 :

```
[ChunkHead(31-16) | longsize(15-0)][Data]
  ChunkHead:
    NJD_FV_SHAPE | NJD_D8
  longsize:
    次のチャンクまでのロングワードオフセット。
  Data:
    D8888, ...
```

#### 説明 :

```
#define NJD_D8 (NJD_SHAPEOFF+2)
```

Diffuse 頂点カラーのみの頂点モーションで使用。コンバータプレライトによる頂点カラーアニメーションの出力が可能。エンベロープと同時に使うと連続部分の表現が難しいので CompactShape 単体での使用を推奨。

通常モデラーでは頂点カラーアニメーションは作成できない。コンバータのプレライト機能で頂点カラーをコンバート時に埋め込むことで頂点カラーアニメーションを実現する。

```
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT)
```

CompactShape データであることを示すために ChunkHead には必ず NJD\_FV\_SHAPE フラグが設定される。



## 3. アスキーフォーマット

### 3.1 アスキー出力のマクロ定義

ここでは CompactShape 頂点モーションに関わる部分だけを説明する。通常のモーション出力に関わる部分は省略する。ただし NJS\_MOTION 構造体の type, inp\_fn に設定されるフラグは視認性を高めるため Ninja2 よりテキストで定義する。この部分に関してはここで説明する。下記の定義は Ninja2 用 NjDef.h の中でされる。

#### 3.1.1 Nja の基本定義によるマクロ

START	データエントリの開始を意味する。
END	データエントリの終了を意味する。

#### 3.1.2 チャンクフォーマット定義によるマクロ

VLIST	頂点リスト。
VERT	頂点座標 (x, y, z) エントリ。
NORM	頂点法線 (nx, ny, nz) エントリ。
CnkEnd	頂点リストの終了を意味する。
CnkV_VN	Chunk Vertex 法線付き。
CnkV_VN_D8	Chunk Vertex 法線、頂点カラー付き。
CnkV	Chunk Vertex 法線なし。
CnkV_D8	Chunk Vertex 法線なし、頂点カラー付き。

#### 3.1.3 CompactShape 専用のチャンクフォーマット頂点リストマクロ

CnkVN	法線のみ。CompactShape 専用。
CnkVN_D8	法線 + 頂点カラー。CompactShape 専用。
CnkD8	頂点カラーのみ。CompactShape 専用。

ライブラリのインクルードファイルに Chunk ナンバーの定義を追加。

```
/* chunk shape */
#define NJD_VN      (NJD_SHAPEOFF+0) /* nx,ny,nz,... for Ninja2 Shape */
#define NJD_VN_D8   (NJD_SHAPEOFF+1) /* nx,ny,nz,D8888,... for Ninja2 Shape */
#define NJD_D8      (NJD_SHAPEOFF+2) /* D8888,... for Ninja2 Shape */
```

NjDef.h に下記定義が追加される。

```
/* Chunk Vertex for CompactShape */
#define CnkVN      (_CkV(_f, _s)|NJD_VN) /* nx,ny,nz, ... */
#define CnkVN_D8   (_CkV(_f, _s)|NJD_VN_D8) /* nx,ny,nz,D8888, ... */
#define CnkD8      (_CkV(_f, _s)|NJD_D8) /* D8888, ... */
```

### 3.1.4 CompactShape 用の ChunkVertex フラグ定義

FV_SHAPE	CompactShape 用に出力された頂点リストであることを示す。CompactShape データとして不要である中間バッファへのオフセットである IndexOffset, 頂点数を与える nbIndices データを省略していることを明示する。
----------	--

ライブラリのインクルードファイルにフラグを追加。

```
/* Flag Compact Shape */  
#define NJD_FV_SHAPE (0x40<<NJD_FV_SHIFT) /* compact shape flag */
```

NjDef.h に下記定義が追加される。

```
#define FV_SHAPE NJD_FV_SHAPE
```

### 3.1.5 モーション定義によるマクロ

MOTION_START	モーションアスキーチャンクの開始を意味する。
MOTION_END	モーションアスキーチャンクの終了を意味する。
DEFAULT_START	デフォルト変数名チャンクの開始を意味する。
DEFAULT_END	デフォルト変数名チャンクの終了を意味する。
MDATA1	モーションデータ成分をくくるための構造体。
MOTION	モーションデータを意味する。
MdataArray	階層分のモーションをくくるためのエントリ。
MFrameNum	モーションのフレーム数。
MotionBitF	MDATA1 に格納されるモーションの種類を格納。CompactShape では NJD_MTYPE_SHAPEID に固定。
InterpolFctF	キーフレーム補完の種類とモーションの要素数を定義。CompactShape ではキーフレーム補完の方法はデフォルトで NJD_MTYPE_LINER。要素数は 1 固定。

モーションの各要素を示すマクロ名は次の通り。

FMK_POS0	移動成分が要素としてあることを示す。
FMK_ANG1	回転成分が要素としてあることを示す。
FMK_SCA2	スケール成分が要素としてあることを示す。
FMK_VEC3	ベクトル成分が要素としてあることを示す。
FMK_VEC0	ライトのベクトル成分が要素としてあることを示す。
FMK_SANG1	ShortAngle による回転成分が要素としてあることを示す。
FMK_TAR3	カメラ用ターゲット成分が要素としてあることを示す。
FMK_ROL6	カメラ用ロール成分が要素としてあることを示す。ShortAngle を使用。
FMK_ANG7	カメラ用画角成分が要素としてあることを示す。ShortAngle を使用。
FMK_RGB8	ライトカラー成分が要素としてあることを示す。
FMK_INT9	ライト Intensity + Ambient 成分が要素としてあることを示す。
FMK_PO9	ポイントライト(nrange, frange)成分が要素としてあることを示す。
FMK_QUA1	クォータニオン成分が要素としてあることを示す。
FMK_SHID	CompactShape データが要素としてあることを示す。
FMK_EVE4	イベントモーションが要素としてあることを示す。

モーション成分に関わる定義。後ろの番号はモーション成分の並び順番を意味する。モーションの中に複数の成分がある場合若い番号順に整列される。同じ番号のデータは同時に存在しない排他的な要素同士を意味する。FMK\_SHID は CompactShape を一意に示し他の成分と同時に使われることはないので後ろの番号を持たない。

```
/* Flag Motion Key */  
#define FMK_POS0 NJD_MTYPE_POS_0  
#define FMK_ANG1 NJD_MTYPE_ANG_1  
#define FMK_SCA2 NJD_MTYPE_SCL_2  
#define FMK_VEC3 NJD_MTYPE_VEC_3
```

```
#define FMK_VEC0      NJD_MTYPE_VEC_0
#define FMK_SANG1     NJD_MTYPE_SANG_1
#define FMK_TAR3      NJD_MTYPE_TARGET_3
#define FMK_ROL6      NJD_MTYPE_ROLL_6
#define FMK_ANG7      NJD_MTYPE_ANGLE_7
#define FMK_RGB8      NJD_MTYPE_RGB_8
#define FMK_INT9      NJD_MTYPE_INTENSITY_9
#define FMK_POI9      NJD_MTYPE_POINT_9
#define FMK_QUA1      NJD_MTYPE_QUAT_1
#define FMK_SHID      NJD_MTYPE_SHAPEID
#define FMK_EVE4      NJD_MTYPE_EVENT_4
```

FMT_L	keyframe の Linear 補完を意味する。
FMT_S	keyframe の Spline 補完を意味する。
FMT_U	keyframe の User 関数補完を意味する。現在未対応。

```
/* Flag Motion Type */
#define FMT_L      NJD_MTYPE_LINEAR
#define FMT_S      NJD_MTYPE_SPLINE
#define FMT_U      NJD_MTYPE_USER
```

FMD_1	NJS_MDATA1 構造体を利用。
FMD_2	NJS_MDATA2 構造体を利用。
FMD_3	NJS_MDATA3 構造体を利用。
FMD_4	NJS_MDATA4 構造体を利用。
FMD_5	NJS_MDATA5 構造体を利用。

```
/* Flag MDATA */
#define FMD_1      (1)
#define FMD_2      (2)
#define FMD_3      (3)
#define FMD_4      (4)
#define FMD_5      (5)
```

### 3.1.6 CompactShape 定義によるマクロ

CSHAPE_LIST_START	ShapeList アスキーチャンクの開始を示す。
CSHAPE_LIST_END	ShapeList アスキーチャンクの終了を示す。
CVLIST	モデル階層の各モデルの頂点データを配列にする。
CSHAPEKGLIST	ShapeList に登録される KeyframeGroup を配列にする。
CSHAPEKG	モデル階層形状の KeyframeGroup 各エントリを示す。
CSHAPELIST	ShapeList を与える。
CShapeKgList	モデル階層形状の KeyframeGroup 配列を与える。
CShapeNum	ShapeList に登録されるモデル階層形状の KeyframeGroup によるエントリ総数。
CSHAPE_MOTION_START	CompactShape アスキーチャンクの開始を示す。
CSHAPE_MOTION_END	CompactShape アスキーチャンクの終了を示す。
CSHAPE	CompactShape モーションデータ。
CSHAPEDATA	モデルに適用される ShapeId で ShapeList を参照する頂点モーション配列。
MKEYSHID	キーフレーム頂点モーションの各エントリを示す。

```
/* *****
 * COMPACTSHAPE
 * ***** */
#define CSHAPE_LIST_START
#define CSHAPE_LIST_END

#define CVLIST      NJS_CSHAPECV
#define CSHAPEKGLIST NJS_CSHAPE_KEYGROUP
#define CSHAPEKG(_e, _g)  (_e), (_g), 0

#define CSHAPELIST  NJS_CSHAPELIST
```

```
#define CShapeKgList
#define CShapeNum

#define CSHAPE_MOTION_START
#define CSHAPE_MOTION_END

#define CSHAPE                NJS_MKEY_SHAPEID
#define CSHAPEDATA            NJS_CSHAPEDATA

#define MKEYSHID(_k, _id)      {((UInt32)(_k)), ((UInt32)(_id))}
```

## 3.2 アスキー出力の例

---

### nas ファイル

```
---
/* NAS 2.00.00 NinjaAsciiCompactShape (SI) */

/* CSHAPELIST : cshapelistsc0_model0 */

CSHAPE_LIST_START

VLIST    cv0_sc0_model0_node0[]
START
    CnkV_VN(FV_SHAPE, 120),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

VLIST    cv0_sc0_model0_node1[]
START
    CnkV_VN(FV_SHAPE, 100),
    ...
    CnkEnd()
END

VLIST    cv0_sc0_model0_node4[]
START
    CnkV_VN(FV_SHAPE, 200),
    ...
    CnkEnd()
END

CVLIST    kg0_cvlist0_sc0_model0[]
START
    cv0_sc0_model0_node0,
    cv0_sc0_model0_node1,
    cv0_sc0_model0_node4,
END

VLIST    cv1_sc0_model0_node0[]
START
    CnkV_VN(FV_SHAPE, 120),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

...

CVLIST    kg0_cvlist1_sc0_model0[]
```

```

START
    cv1_sc0_model0_node0,
    cv1_sc0_model0_node1,
    cv1_sc0_model0_node4,
END

VLIST    cv2_sc0_model0_node0[]
START
    CnkV_VN(FV_SHAPE, 50),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

...

CVLIST    kg0_cvlist2_sc0_model0[]
START
    cv2_sc0_model0_node0,
    cv2_sc0_model0_node1,
    cv2_sc0_model0_node4,
END

VLIST    cv0_sc0_model0_node2[]
START
    CnkV_VN(FV_SHAPE, 100),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

VLIST    cv0_sc0_model0_node3[]
START
    CnkV_VN(FV_SHAPE, 90),
    ...
    CnkEnd()
END

CVLIST    kg1_cvlist0_sc0_model0[]
START
    cv0_sc0_model0_node2,
    cv0_sc0_model0_node3,
END

VLIST    cv1_sc0_model0_node2[]
START
    CnkV_VN(FV_SHAPE, 120),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

```

```

...

CVLIST    kg1_cvlist1_sc0_model0[]
START
    cv1_sc0_model0_node2,
    cv1_sc0_model0_node3,
END

VLIST    cv2_sc0_model0_node0[]
START
    CnkV_VN(FV_SHAPE, 50),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

...

CVLIST    kg1_cvlist2_sc0_model0[]
START
    cv2_sc0_model0_node2,
    cv2_sc0_model0_node3,
END

CSHAPEKGLIST cshapes_sc0_model0[]
START
    CSHAPEKG( kg0_cvlist0_sc0_model0, 0 ), /* 0 */
    CSHAPEKG( kg0_cvlist1_sc0_model0, 0 ), /* 1 */
    CSHAPEKG( kg0_cvlist2_sc0_model0, 0 ), /* 2 */
    CSHAPEKG( kg1_cvlist0_sc0_model0, 1 ), /* 3 */
    CSHAPEKG( kg1_cvlist1_sc0_model0, 1 ), /* 4 */
    CSHAPEKG( kg1_cvlist2_sc0_model0, 1 ), /* 5 */
END

CSHAPELIST cshapelistsc0_model0
START
    CShapeKgList    cshapes_sc0_model0,
    CShapeNum        6,
END

CSHAPE_LIST_END
---
```

## nam ファイル

```

---
/* NAM 2.00.00 NinjaAsciiMotion (SI) */

/* MOTION : motion_sc0_model0 */
/* CSHAPE MOTION : cshape_motion_sc0_model0 */

MOTION_START

<通常のモーション出力>

MOTION_END
```

```

CSHAPE_MOTION_START

CSHAPE cshape_motion_kg0_sc0_model0[]
START
    MKEYSHID(0, 0),
    MKEYSHID(9, 1),
    MKEYSHID(20, 2),
    ...
END

CSHAPE cshape_motion_kg1_sc0_model0[]
START
    MKEYSHID(0, 3),
    MKEYSHID(10, 4),
    MKEYSHID(25, 5),
    ...
END

CSHAPEDATA cshape_data_sc0_model0[]
START
    cshape_motion_kg0_sc0_model0, 6, 0, /* 0 */
    cshape_motion_kg0_sc0_model0, 6, 1, /* 1 */
    cshape_motion_kg0_sc0_model0, 6, 2, /* 1 */
    cshape_motion_kg1_sc0_model0, 5, 0, /* 2 */
    cshape_motion_kg1_sc0_model0, 5, 1, /* 3 */
    cshape_motion_kg0_sc0_model0, 6, 3, /* 4 */
END

MDATA1 mdata_cshape_sc0_model0[]
START
    &cshape_data_sc0_model0[0], 1,
    &cshape_data_sc0_model0[1], 2,
    &cshape_data_sc0_model0[3], 1,
    &cshape_data_sc0_model0[4], 1,
    &cshape_data_sc0_model0[5], 1,
END

MOTION cshape_motion_sc0_model0[]
START
MdataArray    mdata_cshape_sc0_model0,
MFrameNum     100,
MotionBitF    (FMK_SHID),
InterpolFctF  (FMT_L|FMD_1),
END

CSHAPE_MOTION_END

```

---

### 3.3 アスキー出力解説

---

#### nas ファイル

```

---
/* NAS 2.00.00 NinjaAsciiCompactShape (SI) */

```

ヘッダには nas ファイルであること、バージョン、CompactShape データであること、作成されたモデラーを示す情報が記述される。ここで"SI"は Softimage を意味する。  
 "LW"は LightWave3D, "MAX"は 3D Studio MAX, "MAYA"は MAYA を意味する。



```
/* CSHAPELIST : cshapelistsc0_model0 */
```

ユーザが参照すべき変数名をコメントとして表示。

```
CSHAPE_LIST_START
```

ShapeList のアスキーチャンク開始を意味する。

```
VLIST    cv0_sc0_model0_node0[]
```

cv0 は Chunk Vertex の 0 番目のデータ、sc0 は現在のシーンファイル名、model0 は現在のモデル階層名、node0 は現在のモデル (Ninja ではオブジェクトと呼ぶ)。sc0、model0、node0 は実データ名が格納される。

```
START
```

```
    CnkV_VN(FV_SHAPE, 120),  
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),  
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),  
    ...  
    CnkEnd()
```

```
END
```

チャンク形式による頂点データ。アニメーションの種類により形式を変更。

```
VLIST    cv0_sc0_model0_node1[]
```

```
START
```

```
    CnkV_VN(FV_SHAPE, 100),  
    ...  
    CnkEnd()
```

```
END
```

```
VLIST    cv0_sc0_model0_node4[]
```

```
START
```

```
    CnkV_VN(FV_SHAPE, 200),  
    ...  
    CnkEnd()
```

```
END
```

```
CVLIST    kg0_cvlist0_sc0_model0[]
```

```
START
```

```
    cv0_sc0_model0_node0,  
    cv0_sc0_model0_node1,  
    cv0_sc0_model0_node4,
```

```
END
```

頂点リスト cv をくくり KeyframeGroup 単位の cv の配列(cvlist)を作る。

```
VLIST    cv1_sc0_model0_node0[]
```

```
START
```

```
    CnkV_VN(FV_SHAPE, 120),  
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),  
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),  
    ...  
    CnkEnd()
```

```
END
```

```
...
```

```

CVLIST    kg0_cvlist1_sc0_model0[]
START
    cv1_sc0_model0_node0,
    cv1_sc0_model0_node1,
    cv1_sc0_model0_node4,
END

```

```

VLIST    cv2_sc0_model0_node0[]
START
    CnkV_VN(FV_SHAPE, 50),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

```

...

```

CVLIST    kg0_cvlist2_sc0_model0[]
START
    cv2_sc0_model0_node0,
    cv2_sc0_model0_node1,
    cv2_sc0_model0_node4,
END

```

```

VLIST    cv0_sc0_model0_node2[]
START
    CnkV_VN(FV_SHAPE, 100),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

```

```

VLIST    cv0_sc0_model0_node3[]
START
    CnkV_VN(FV_SHAPE, 90),
    ...
    CnkEnd()
END

```

```

CVLIST    kg1_cvlist0_sc0_model0[]
START
    cv0_sc0_model0_node2,
    cv0_sc0_model0_node3,
END

```

```

VLIST    cv1_sc0_model0_node2[]
START
    CnkV_VN(FV_SHAPE, 120),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()

```

```

END

...

CVLIST    kg1_cvlist1_sc0_model0[]
START
    cv1_sc0_model0_node2,
    cv1_sc0_model0_node3,
END

VLIST     cv2_sc0_model0_node0[]
START
    CnkV_VN(FV_SHAPE, 50),
    VERT( 0x41d830c8, 0xbe800e2d, 0x3e2d26e2 ),
    NORM( 0x3f7ffd32, 0xbc178a51, 0x33b873d1 ),
    ...
    CnkEnd()
END

...

CVLIST    kg1_cvlist2_sc0_model0[]
START
    cv2_sc0_model0_node2,
    cv2_sc0_model0_node3,
END

CSHAPEKGLIST cshapes_sc0_model0[]
START
    CSHAPEKG( kg0_cvlist0_sc0_model0, 0 ), /* 0 */
    CSHAPEKG( kg0_cvlist1_sc0_model0, 0 ), /* 1 */
    CSHAPEKG( kg0_cvlist2_sc0_model0, 0 ), /* 2 */
    CSHAPEKG( kg1_cvlist0_sc0_model0, 1 ), /* 3 */
    CSHAPEKG( kg1_cvlist1_sc0_model0, 1 ), /* 4 */
    CSHAPEKG( kg1_cvlist2_sc0_model0, 1 ), /* 5 */
END

```

ShapeList のデータ配列。KeyframeGroup がエントリとして並ぶ。

本例では KeyframeGroup は二つ ( kg0, kg1 ) あり kg0\_cvlist0\_sc0\_model0, kg1\_cvlist0\_sc0\_model0 の二つでモデル階層全体の情報となる。ただし KeyframeGroup 単位で keyframe 位置が異なるため KeyframeGroup 毎に数や時間軸的な配置は異なる。

```

CSHAPELIST cshapelist_sc0_model0
START
    CShapeKgList    cshapes_sc0_model0,
    CShapeNum        6,
END

```

ShapeList 上のエントリをくくりエントリの数を与える。

```
CSHAPE_LIST_END
```

ShapeList のアスキーチャンクの終了を意味する。

```
---
```

## nam ファイル

```
---
```

```
/* NAM 2.00.00 NinjaAsciiMotion (SI) */
```

ヘッダには nam ファイルであること、バージョン、モーションデータであること、作成されたモデラーを示す情報が記述される。ここで "SI" は Softimage を意味する。  
"LW" は LightWave3D, "MAX" は 3D Studio MAX, "MAYA" は MAYA を意味する。

```
/* MOTION : motion_sc0_model0 */
/* CSHAPE MOTION : cshape_motion_sc0_model0 */
```

ユーザが参照すべき変数名をコメントとして表示。

MOTION\_START

<通常のモーション出力>

MOTION\_END

CSHAPE\_MOTION\_START

```
CSHAPE cshape_motion_kg0_sc0_model0[]
START
    MKEYSHID(0, 0),
    MKEYSHID(9, 1),
    MKEYSHID(20, 2),
    ...
END
```

フレーム時間 (左) と ShapeList のエントリを示す ShapeId (右) により頂点モーションの Keyframe データを構成する。  
KeyframeGroup0 に対するデータ。

```
CSHAPE cshape_motion_kg1_sc0_model0[]
START
    MKEYSHID(0, 3),
    MKEYSHID(10, 4),
    MKEYSHID(25, 5),
    ...
END
```

フレーム時間 (左) と ShapeList のエントリを示す ShapeId (右) により頂点モーションの Keyframe データを構成する。  
KeyframeGroup1 に対するデータ。

```
CSHAPEDATA cshape_data_sc0_model0[]
START
    cshape_motion_kg0_sc0_model0, 6, 0, /* 0 */
    cshape_motion_kg0_sc0_model0, 6, 1, /* 1 */
    cshape_motion_kg0_sc0_model0, 6, 2, /* 1 */
    cshape_motion_kg1_sc0_model0, 5, 0, /* 2 */
    cshape_motion_kg1_sc0_model0, 5, 1, /* 3 */
    cshape_motion_kg0_sc0_model0, 6, 3, /* 4 */
END
```

各モデルに必要な KeyframeGroup を一括して一次元配列に並べる。MDATA1 からはエントリのポインタと数を設定することで参照する。コメントの番号は MDATA1 に配置されるモデルの番号が示される。1 が連続して二つ示されているのはモデル階層の二番目のモデルに CSHAPEDATA のエントリ二つが割り当てられていることを示す。

```
MDATA1 mdata_cshape_sc0_model0[]
START
    &cshape_data_sc0_model0[0], 1,
    &cshape_data_sc0_model0[1], 2,
    &cshape_data_sc0_model0[3], 1,
    &cshape_data_sc0_model0[4], 1,
    &cshape_data_sc0_model0[5], 1,
END
```

MDATA1 の各エントリはモデルトレースの順で一次元に並べられた各モデルを意味しそのモデルに対応する CSHAPEDATA の先頭ポインタおよび適用されるエントリの数データとして並べられる。

```
MOTION cshape_motion_sc0_model0[]  
START  
MdataArray      mdata_cshape_sc0_model0,  
MFrameNum       100,  
MotionBitF      (FMK_SHID),  
InterpolFctF    (FMT_L|FMD_1),  
END
```

MotionBitF には CompactShape であることを示す FMK\_SHID が設定されている。補完方式はリニア (FMT\_L) 固定、要素数は 1 (FMD\_1) 固定。MFrameNum にはモーションフレーム数が格納される。

CSHAPE\_MOTION\_END

CompactShape モーションアスキーチャンクの終了を意味する。

---

## 4. バイナリフォーマット

### 4.1 バイナリ拡張子

Ninja2 に関わるファイル拡張子をここで示す。

#### 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
モーションリソース (アスキーのみ)	-	.mrs
モデル階層用頂点グループリソース (バイナリのみ)	.vrs	-
シーン用頂点グループリソース (バイナリのみ)	.nrv	-
汎用リソースファイル (アスキーのみ)	-	.nre

Ninja1 では nas, njs には SimpleShape モーションが格納されたが Ninja2 では SimpleShape は廃止され ShapeList が格納される。

CompactShape 出力を必要とするコンバートにおいて nam, njm には二つのモーションデータが出力される。一つはモデルモーション、もう一つが CompactShape 用頂点モーション。

Chunk モデルバイナリデータ出力ファイルである njd, nj において Ninja1 のバイナリチャンク名と異なるチャンク名を使用する。これは同じ Chunk フォーマットであるが Ninja1 と Ninja2 で仕様が変更されたため同一扱いをしないための処置である。アスキーフォーマットに関してはヘッダなどで内容確認が容易なため特別な処置はしない。

## 4.2 バイナリ構造

Chunkname	bytesize	データバイナリチャンク
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		<div>&lt; 主なチャンクの種類 &gt; 'N2CM' : ninja2 chunk model tree 'NJTL' : ninja texlist 'NJLI' : ninja light 'NJCA' : ninja camera 'NMDM' : ninja model motion 'NLIM' : ninja light motion 'NCAM' : ninja camera motion 'CPSM' : ninja compact shape motion 'NJSL' : ninja compact shape list</div>
Chunk POF0	bytesize	P O F 0 チャンク
ポインタオフセットデータ		<div>ポインタオフセットアルゴリズムタイプ 0</div>

図 1 バイナリ基本構造

POF0 チャンクは直前の Ninja 用バイナリデータに含まれるデータ上のポインタのオフセットを与える。POF0 で与えられたポインタ位置にローダで読み込んだ実アドレスの先頭を足し込むことでデータ内部でのポインタ参照を有効にする。

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。

Ninja2 用の Chunk フォーマット出力を Ninja1 のものと区別するためにチャンク名を変更する。

CompactShape 用の CPSM, NJSL は新規に定義される。

'NJSL'	bytesize	ShapeList データ
ShapeList データ		
Chunk POF0	bytesize	ShapeList の POF
Shapelist POF データ		

図 2 njs バイナリ基本構造

バイナリロードすると NJS\_CSHAPELIST 構造体の先頭ポインタがユーザ領域の先頭となる。

'NMDM'	bytesize	Model Motion データ
Model Motion データ		
Chunk POF0	bytesize	Model Motion の POF0 データ
Chunk Model POF データ		
'CPSM'	bytesize	CompactShape Motion データ
CompactShapeMotion データ		
Chunk POF0	bytesize	CompactShape の POF0 データ
CompactShape POF データ		

図 3 CompactShape Motion を含む njm バイナリ基本構造

バイナリロードすると NJS\_MOTION 構造体の先頭ポインタがユーザ領域の先頭となる。

さらに次のチャンクを発見し新たにバイナリロードすると CompactShape Motion を格納する NJS\_MOTION 構造体の先頭ポインタが二つ目のユーザ領域の先頭になる。

## 4.3 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>

相対値に変換。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>

P O F 0 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 6 4 より小さい値ならば char 値として 1 4 ビットの最大値 1 6 3 8 4 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 P O F 0 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 P O F 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 P O F 内部では char で 0 を書き込んでおけばパディングされる。

以 上