



Ninja2 Motion 仕様書

(08/10/2000)

1. 概要	4
1.0 NINJA1 からの変更点.....	4
1.1 回転成分の扱い及び SHORTANGLE について.....	5
1.2 クォータニオン	5
1.3 イベントモーション(ハイドモーション).....	5
1.4 モーションの種類.....	7
1.5 モーション出力特性	8
1.5.1 フルフレーム形式を使った出力.....	8
1.5.2 キーフレーム形式を使った出力.....	9
1.5.3 mrs からの Null 置き換えの制御.....	10
1.5.4 EvalSkip 及び EvalShapeSkip.....	11
1.6 NULLOBJECT EVALFLAGS	11
2. 構造体とマクロ定義	13
2.1 NJS_OBJECT 構造体およびマクロ定義.....	13
2.1.1 Chunk Object 構造体.....	13
2.1.2 evalflags フラグ説明.....	13
2.2 NJS_CAMERA 構造体及びマクロ定義.....	15
2.2.1 NJS_CAMERA 構造体.....	15
2.3 NJS_LIGHT 構造体及びマクロ定義	15
2.3.1 NJS_LIGHT 構造体.....	15
2.3.2 Ninja2 ライブラリ関数と光源パラメータの関係	17
2.4 NJS_MOTION 構造体及びマクロ定義.....	19
2.4.1 モーション構造体.....	19
2.4.3 キー構造体.....	22
3. オブジェクトモーション	24
構造体関連図	24
3.1 モデルモーション解説.....	25
3.2 モデルモーションアスキーフォーマットのマクロ定義.....	27
3.2.1 モーション構造体マクロ	27
3.2.2 モーションキー構造体マクロ.....	28
3.2.3 TRS モーションマクロ.....	29
3.2.4 イベントモーション(ハイドモーション) マクロ	29
3.2.5 アスキーフォーマットヘッダ情報.....	30
3.3 モデルモーションアスキーフォーマット出力例.....	31
3.3.1 モデルモーション出力例.....	31
3.3.2 出力例の解説	37
4. カメラおよびカメラモーション.....	44

4.1 カメラモーション解説.....	44
4.2 カメラアスキーフォーマットマクロ.....	44
4.3 カメラアスキーフォーマット出力例.....	45
4.3.1 カメラ (nac ファイル) の出力例(VECTOR).....	45
4.3.2 カメラ(nac ファイル)の出力例(TARGET).....	46
4.3.3 カメラモーションの出力例(TARGET).....	47
5. ライトモーション.....	51
5.1 ライトモーション解説.....	51
5.2 ライトアスキーフォーマットマクロ.....	51
5.3 ライトアスキーフォーマット出力例.....	54
5.3.1 ライト (nal ファイル) の出力例(EasyDraw の平行光源).....	54
5.3.2 ライト (nal ファイル) の出力例(SimpleDraw の平行光源).....	55
5.3.3 ライト (nal ファイル) の出力例(EasyMultiDraw の平行光源).....	55
5.3.4 ライト (nal ファイル) の出力例(EasyMultiDraw の点光源).....	56
5.3.5 ライト (nal ファイル) の出力例(EasyMultiDraw のアンビエントライト).....	56
5.3.6 ライト (nal ファイル) の出力例(SimpleMultiDraw の平行光源).....	57
5.3.7 ライト (nal ファイル) の出力例(SimpleMultiDraw の点光源).....	58
5.3.8 ライト (nal ファイル) の出力例(SimpleMultiDraw のアンビエントライト).....	58
5.3.9 ライトモーションの出力例(EasyDraw および SimpleDraw の平行光源).....	59
5.3.10 ライトモーションの出力例(EasyMultiDraw および SimpleMultiDraw の平行光源).....	62
5.3.11 ライトモーションの出力例(EasyMultiDraw および SimpleMultiDraw の点光源).....	65
5.3.12 ライトモーションの出力例(EasyMultiDraw および SimpleMultiDraw のアンビエントライト).....	66
6. バイナリフォーマット.....	68
6.1 バイナリ拡張子.....	68
6.2 バイナリ構造.....	69
6.3 バイナリチャンクヘッダ.....	69
6.4 POF0 のアルゴリズム.....	70

1. 概要

1.0 Ninja1 からの変更点

本仕様書は Ninja2 におけるモーションの仕様について説明する。Ninja1 から Ninja2 へのモーション変更点は次の通り。

コンバータからのアスキーフォーマット出力において NJS_ACTION 構造体の出力をしない。カメラ用の NJS_CACTION 構造体、ライト用の NJS_LACTION 構造体も同様に出力しない。ライブラリ上での構造体定義は維持される。

従来の頂点アニメーションである SimpleShape をデータ表現効率が高くかつ同一モデル内でエンベロープと共存可能な CompactShape に置き換える。これに伴い SimpleShape は廃止される。Ninja2 用の頂点アニメーションデータの nas(njs)、nam(njm) ファイルは無条件に CompactShape データが格納される。

16ビット精度(ShortAngle)による XYZ 回転 (オイラー角) による回転成分形式の追加。

モーションリンクはすべてクォータニオンでされる。クォータニオン形式のモーション出力の場合は当然とし XYZ 回転 (オイラー角) の場合も同様。計算時間は多少増えるがデータサイズの 16ビット化と組み合わせデータサイズの最小化とスムーズなモーションリンクを実現する。モーション作成の生産効率の改善が期待できる。ライブラリ内部の実装でありコンバータのデータ出力にはこれに関わる変更はない。

XYZ 回転 (オイラー角) によるモーションデータを利用しモーションリンクのみクォータニオンを利用することをライブラリに指定するフラグを追加。

NJS_OBJECT 構造体のクォータニオン対応。モデルデータ出力時に回転成分をクォータニオンとして出力できる。クォータニオンを利用したモーションデータが NULL の場合のオブジェクト回転値の参照も可能。

ライト、カメラの構造体変更。ライトはスポットライトがなくなる。Ninja2 ライブラリから NormalDraw 関数がなくなり、描画関数は EasyDraw, SimpleDraw, EasyMultiDraw, SimpleMultiDraw の 4 つのみとなる

カメラの画角、ロールには ShortAngle が適用される。

ハイドアニメーションをサポートする。これは新たにモーション内部で制御可能なフラグ群を用意する。これをイベントモーションと呼ぶ。現在イベントモーションから制御可能なのはハイドのみ。

ライト、カメラモーションにおいてアスキーチャック名を MOTION_START、MOTION_END から CAMERA_MOTION_START、CAMERA_MOTION_END、LIGHT_MOTION_START、LIGHT_MOTION_END にそれぞれ変更。

1.1 回転成分の扱い及び ShortAngle について

Ninja では回転成分の表現に次の二つの形式ある。

- ・ XYZ 回転 (オイラー角) による回転成分表現
- ・ クォータニオンによる回転成分表現

Ninja1 では前者のオイラー角に 32 ビット精度で回転を表現していた。360 度を 16 ビット精度で表現しそれを越える角度を表現するために残る 16 ビットを利用していた。

Ninja2 ではこれに 16 ビット精度表現を追加する。-180 ~ 180 度に対し 16 ビットを割り振る (これを ShortAngle と呼ぶ)。それを越える角度は -180 ~ 180 度の間に変換する。回転方向は前のキーとの差の小さい方向へ回転することで表現される。ユーザは必ず回転方向を指定できる間隔 (180 度以下) でキーを設定する必要がある。

Ninja2 ではデータを XYZ 回転 (オイラー角) で持ちデータサイズを小さく抑え、クォータニオンでスムーズに補完することができる。

Ninja2 では Sangle として typedef される。

1.2 クォータニオン

クォータニオンとは虚数を使った回転成分表現である。XYZ 回転 (オイラー角表現) では回転成分を XYZ 軸方向に分解しこれらをそれぞれ成分別に補完することで本来の回転に無い歪みが発生 (ジンバルロック) しスムーズに補完できない場合がある、また同一の回転を表現する XYZ 値が複数解存在することによる滑らかにつながらない解どうしのリンクによる連結結果の不具合などが多く発生する。

クォータニオンでは最短のパスで補完される特性を持ちオイラー角で発生する問題はなく滑らかなモーションリンクが実現できる。ただしデータがベクトルと回転成分の計 4 つのパラメータから構成されるためオイラー角の XYZ の 3 つのパラメータの場合よりも 1 成分大きなデータとなる。データサイズの効率化は今後の課題である。

クォータニオンはコンバータのオプションで指定されることでデータ出力できる。

1.3 イベントモーション(ハイドモーション)

NJS_MOTION 構造体のモデルモーション要素としてイベントモーションを追加する。イベントモーションは keyframe と 16 ビットのフラグ領域(key)から構成される。

```
typedef struct {  
    Uint16      keyframe;  
    Uint16      key;  
} NJS_MKEY_UI16;
```

keyframe 時間に必要なイベントフラグを key に設定する。今回サポートされるフラグはハイド (モデルの非表示) モーションのみ。今後新たなフラグ制御可能なモーション要素があれば随時追加する。

```
<NjMotion.h>  
#define NJD_EVENT_HIDE          (BIT_0)  
  
<NjDef.h>  
/* Flag Event Motion */  
#define FEM_NONE                (0)  
#define FEM_HD                  NJD_EVENT_HIDE
```

(例) モーションアスキーフォーマット出力の任意のモデルモーションデータにおいて

```
MEYUI16( 0, FEM_NONE );  
MEYUI16( 30, FEM_HD );  
MEYUI16( 60, FEM_NONE );  
MEYUI16( 100, FEM_NONE );
```

この例ではモーションフレーム数最大値が100でkeyframe=30からkeyframe=60までモデルがハイドされることを意味する。ここで注意することはモーションの最初(keyframe=0)と最後のフレーム(keyframe=100)は省略できないことである。これは現在の時間におけるkeyの選択において他のキーと同じく2分探索法が使われるため初期値として最初と最後が必要なためである。データサイズが比較的小さいモーションであるためライブラリに専用キー探索を作らず他のものと同様な扱いをする。

NJS_MOTION 構造体においてモーション要素を示す要素名として NJD_MTYPE_EVENT_4 を定義する。これはモデルモーションにおいて POS_0, ANG_1, SCL_2 の三つの要素の後ろに配置されることを意味する。

```
#define NJD_MTYPE_EVENT_4 (BIT_15)
```

モデルモーションにはタイプ A (TRRR)、B (TRTR)、C (TRS)の三つがあった。これにイベントモーションが付加されることでタイプ A (TRRR)+E、B (TRTR)+E、C (TRS)+E の新たなモーション出力が追加される(ここで E はイベントモーションを意味する)。これに伴い同時に利用するモデルモーションにおける要素数はC (TRS)+E の場合に最大4要素となる。

イベントモーションも mrs (モーションリソース)を参照する。モーションは mrs 経由でモデル階層の持つ evalflags のハイドの有無を参照しイベントモーション側のハイドモーションが変化すること無く不要な場合このモデルに対するイベントモーションデータを NULL にすることでデータの最適化をする。

またイベントモーション実行中はモーションデータに NJD_EVENT_HIDE が検出された場合これを優先し NJS_CNK_OBJECT 構造体の evalflags は無視される。モデルに対するイベントモーションが NULL の場合は NJS_CNK_OBJECT 構造体の evalflags が利用される。

(ハイドモーションに関する注意事項)

モーションの最初と最後のフレームは省略できない。

モデラーからのデータ設定は可視性キー (VisibleKey) に対し割合を設定することで行う。VisibleKey には 1.0~0.0 の値が設定可能。1.0 で表示 0.0 において非表示を意味する。可視性キーの割合が 1.0 (100%) でない場合 Ninja2 コンバータではハイドフラグをオンする。Ninja2 ではハイドをフラグ1ビットで表現するため可視性を 50% などにして半透明状態を再現することはできない。

現在エンベロープのモデルのハイドをサポートしていない。エンベロープモデルに設定されるハイドモーションは無効になる。モデルがエンベロープモデルかどうかは NJS_CNK_OBJECT 構造体の evalflags に設定されるエンベロープフラグ (NJD_EVAL_ENVELOPE、アスキーフォーマット上では FEV_EN) で識別可能。

1.4 モーションの種類

次に Ninja2 のサポートするモーションの種類について説明する。ここで移動成分 Translation を T で回転成分 Rotation を R でスケール成分 Scale を S で表現しこれらを移動、回転、スケール成分を合わせて TRS と表現する。第二列に各データの格納されるファイル拡張子を示す。() はバイナリの場合の拡張子を示す。

モデルモーション	nam(njm)	TRS によるモデル階層に対するモーションを表現する。Ninja2 では新規にイベントモーションを利用したハイドモーションも可能。
頂点モーション (CompactShape)	nas(njs) nam(njm)	頂点モーションもモデルモーションであるが別データとして扱うため項目を分ける。
ライトモーション	nal(njl)	ライトパラメータをモーションする。
カメラモーション	nac(njc)	カメラパラメータをモーションする。

CompactShape では nas ファイルに格納される ShapeList に格納された頂点を nam データにキーフレーム時間と共に埋め込まれた ShapeId と組み合わせることでデータを表現する。ユーザが ShapeList を複数のモーションで共有することでデータサイズ小さくすることが可能になる。これは顔の表情データなどで利用する。

ただし ShapeList を複数モーションで共有するためには同一頂点に対しシェイプとエンベロープが同時にかけられないようにする必要がある。シェイプ+エンベロープの結果はエンベロープが TRS の影響を受けた結果の合成を利用するため使いまわしはできない。また LightWave 以外は直接モデラーの最終頂点位置を入力しているため例えば MAX のバルジの結果を含む頂点位置を取り出すこともできる。筋肉盛り上がりなどが必要な場所のみ頂点グループ指定すればエンベロープ+シェイプによりバルジの再現が可能である。LightWave ではコンバータでエンベロープ+シェイプの結果を計算しているのでその他の成分は含むことはできない。

nas ファイルには ShapeList のみが格納される。キーフレームを含む時間軸データはモデルモーションファイルとして出力される nam ファイルの後半(同一ファイル中)に出力される。例えばバイナリ出力において njm の中にはモデルモーション用の NJS_MOTION 構造体ポインタを先頭とするモーションチャンクと CompactShape 用 NJS_MOTION 構造体ポインタを先頭とするモーションチャンクの二つが格納される。チャンクヘッダは中身を確認するための情報としてそれぞれの識別子を持つ。

各モーションはすべて NJS_MOTION 構造体で管理される。ここに示さない部分として 2D 機能の CellSprite においても NJS_MOTION 構造体を利用する。CellSprite はモデル階層のモーションとは同時に使われることはないため NJS_MOTION 構造体を独自に利用する手法を取っている。CellSprite の仕様に関しては別資料を参照のこと。その他のモーションではモーションの対象となる階層数があれば共通部分は使いまわすことも可能。例えばライトのモーションでカメラを動かすこともできる。

モデルモーションは次の三つの形式をサポートする。ここではこれらの三つを TRRR, TRTR, TRS と表現する。さらにイベントモーションがこれらに組み合わせられる。

TRRR	TRRR はモデル階層のルートモデルのみ移動成分(T)と回転成分(R)を持ちその他は回転成分(R)のみから構成される。関節の移動距離が変化しないキャラクターに利用し通常もっとも基本となる形式。ルートモデルのTRでキャラクターの移動を表現し他のモデルのRでキャラクターを動作させる。NJS_MOTION 構造体メンバーmdata には要素数 2 の NJS_MDATA2 構造体 が利用されルートモデル以外の T には NULL ポインタが格納される。 イベントモーションを組み合わせた時は NJS_MDATA3 構造体を利用する。
TRTR	TRTR はモデル階層のすべてが移動成分(T)と回転成分(R)を持つ。関節の移動を含むキャラクターに利用する。NJS_MOTION 構造体メンバーmdata には NJS_MDATA2 構造体 が利用される。 イベントモーションを組み合わせた時は NJS_MDATA3 構造体を利用する。
TRS	TRS はモデル階層のすべてが移動成分(T)と回転成分(R)とスケール成分(S)を持つ。 NJS_MOTION 構造体メンバーmdata には要素数 3 の NJS_MDATA3 構造体 が利用される。 Ninja2 ライブラリは性能優先のため法線の正規化処理をしません。スケールを使うと法線の長さが変化することで光源計算結果が変化するので注意が必要。イベントモーションを組み合わせた時は NJS_MDATA4 構造体を利用する。

1.5 モーション出力特性

Ninja ではモーション出力においてモーションデータサイズを小さくするためにいくつかの方法を利用している。

あるモデルに関するモーションが無い場合モデル階層 (NJS_CNK_OBJECT 構造体) に格納される値を使うことでモーション側のデータを減らす。モーションデータがないモデルのデータには NULL が設定される。

同一モデルでも姿勢が変わればモデル階層 (NJS_CNK_OBJECT 構造体) の値は変化する。モーションに NULL を入れるためにはモデル階層に基本姿勢を設け (ゲームで実際に使われるモデル階層データ) これとの差分でモーションデータの有り無しを決めモーションデータを生成する必要がある。

全フレームデータから差分の閾値 (イプシロン) 以下のデータを間引くことでデータサイズ削減できる。閾値をあげることでデータサイズは小さくできるが歪みも多くなる。(フルフレーム形式と呼ぶ)

ユーザの設定したキーをそのまま出力できる。(キーフレーム形式と呼ぶ)

その他 Ninja2 から XYZ 回転 (オイラー角) のデータサイズを 32 ビットから 16 ビットに変更しデータサイズを小さくする。

以上の手法について解説する。

モデルモーションには次の二つがある。

フルフレーム形式
キーフレーム形式

どちらも最終的には Ninja のモーションファイルである nam (バイナリは njm) に出力される。上記の二つの形式の選択は最終データまでのデータの加工手順の違いを意味する。

1.5.1 フルフレーム形式を使った出力

フルフレームとはモデラーの中で定義されたキーフレームを一切使用せず全フレームカウント時の値の配列データを基本とする。ただし全フレームデータをすべて使うことはデータサイズが大きすぎて現実的でない。そこで全フレームデータから変化の少ない部分、および動作のない部分のデータを間引き、ライブラリ側で用意されるキーフレーム補完機能でその部分を補うことで実際のモーション出力/動作を実現する。

フルフレーム実装の注意は次の通り。

モデラーの中で定義されたキーフレームの数およびキーに依存した記述はしてはならない。あくまでもフレームカウント値を一つずつ進めて得た値 (フル出力) をある閾値により間引くことで最適化する。つまり『完全なデータ (フルフレーム) から間引く手法』である。

間引きの量を決定する要素として次のパラメータがある。

mrs による Null への置き換え。

ここで mrs はモーションリソースを意味する。モデル出力時のオプション、データを保存しモーション出力時のデータ最適化に利用する。あるキャラクターに対するモデル出力は一回、これによって生成される mrs も一つ。この mrs を複数存在するモーション出力に共通して使うことでキャラクターデータ全体に対し同一の最適化を実現する。

モデルツリーを出力した時のモデルの TRS を保存する mrs の値とモーション用に作られたデータの TRS を比較し最初から最後まで変化がない場合モーションデータを Null とする。モデルツリーには mrs に与えられる TRS と同じ値が格納されているため Null 部分のモーションにこの値を使うことでモーションを実行する。参照する mrs が指定されていない場合は現在コンバート中のモーション内部で最初から最後まで変化がない場

合はこれを Null とする。mrs の利用によりモデルツリーの動かない部分のモーションデータの一括した間引き (Null への置き換え) が実現される。

イプシロン

トランス (T)、ローテーション (R)、スケール (S) のそれぞれに対し出力された一つ前の値との差分が一定以上かを判別する閾値を定義する。この値がイプシロンを越える場合は出力し越えない場合は間引く。デフォルトでは TRS に対し 0.001 になっている。これは経験的に決めた値である。ユーザが希望するデータサイズ、間引きによる歪みの許容範囲、モーションデータの特性に応じてユーザ自身で変更する必要がある。基本的には値を大きくする (0.001 より大) 方向での調整。小さくする方向の調整では float 精度のモデラー内部データの精度を越えるため実質的な効果はない。

コンバータにおいて mrs はファイル名を指定しイプシロンは TRS に対する値を指定することで動作する。イプシロンはフルフレームのモーション出力のみに有効。キーフレームではユーザの定義したキーフレーム位置を忠実に出力することを前提とするためイプシロンは無効である。

上記パラメータによりモーションデータは Null と変化の小さなキーフレームデータが間引かれたデータとして出力されこれがデフォルト出力形式となる。しかしユーザの目的に合わせこれらの間引き動作をしないデータも出力できる。ここでは nre に定義されるリソース名および DscToNj2 に使用されているオプション名を利用しこのオプションの動作について説明する。

次の三つのオプションはモーション出力の Null への置き換えを禁止する。

nre 名: transArray が TRUE オプション名-pt or -printtrans

モーションのトランスに対し mrs 比較もしくは 0 フレーム目との比較による Null への置き換えをしない。モーションによる変化がなくても最初のフレームと最後のフレームの 2 つのデータの配列を出力する。

nre 名: rotateArray が TRUE オプション名-pr or -printrot

モーションのローテーションに対し mrs 比較もしくは 0 フレーム目との比較による Null への置き換えをしない。モーションによる変化がなくても最初のフレームと最後のフレームの 2 つのデータの配列を出力する。

nre 名: scaleArray が TRUE オプション名-ps or -printscales

モーションのスケールに対し mrs 比較もしくは 0 フレーム目との比較による Null への置き換えをしない。モーションによる変化がなくても最初のフレームと最後のフレームの 2 つのデータの配列を出力する。

次の三つのオプションはモーション出力のイプシロンによるフレームの間引きを禁止する。

nre 名: transKey が TRUE オプション名-tk or -transkey

モーションのトランスに対しイプシロンによる間引きをしない。モーションはフレーム数分すべて出力される。

nre 名: rotateKey が TRUE オプション名-rk or -rotkey

モーションのローテーションに対しイプシロンによる間引きをしない。モーションはフレーム数分すべて出力される。

nre 名: scaleKey が TRUE オプション名-sk or -scalekey

モーションのスケールに対しイプシロンによる間引きをしない。モーションはフレーム数分すべて出力される。

上記 6 つのオプションをすべて使用することで Null なし、間引きなしのフルフレームデータが出力できる。

1.5.2 キーフレーム形式を使った出力

キーフレーム形式はモデラー上でユーザが指定したキーフレーム位置を忠実に出力する。

イプシロンによるフレームの間引きはしない。ユーザが意識的にモーションのデータ量を調整でき、モーションの歪みに対するキーフレームの追加などをユーザが自由に設定できる。ただし任意のモデルに関わる

モーション値が変化することなく一定である場合これを一括して Null に置き換えることには対応する。mrs の TRS とモーション値を比較し Null に置き換えることでキーフレームにおいても mrs によるモーションデータサイズの削減が可能となる。キーフレームを一切持たないモデルに関してはデフォルトで Null への置き換えをする。オプションで Null にしないことができる。

キーフレーム形式において mrs との関係を次に説明する。

mrs を参照しない場合の出力動作

キーフレームがないモデルのモーション部分は Null とする。キーフレームがあるモデルはそのまま間引くことなく出力する。

mrs を参照する場合の出力動作

キーフレームがないモデルのモーション部分は Null とする。キーフレームがあるモデルの各フレームのモーション値がすべて参照する mrs の TRS と同じ場合も Null とする。その他はそのまま間引くことなく出力する。

キーフレーム形式では無変化部分の Null への置き換えはあるがイプシロンによるフレーム間の間引きは適用されない。

次の三つのオプションはキーフレーム形式の場合のモーション出力の Null への置き換えを禁止する。

nre 名 : transArray が TRUE オプション名-pt or -printtrans

モーションのトランスに対し mrs 比較もしくは 0 フレーム目との比較による Null への置き換えをしない。フレーム間の間引きはしないでそのまま出力する。キーフレームがないモデルに対しては Null を出力せずモデルツリーに設定される値を最初と最後のフレームに設定しデータを生成する。

nre 名 : rotateArray が TRUE オプション名-pr or -printrot

モーションのローテーションに対し mrs 比較もしくは 0 フレーム目との比較による Null への置き換えをしない。フレーム間の間引きはしないでそのまま出力する。キーフレームがないモデルに対しては Null を出力せずモデルツリーに設定される値を最初と最後のフレームに設定しデータを生成する。

nre 名 : scaleArray が TRUE オプション名-ps or -printscales

モーションのスケールに対し mrs 比較もしくは 0 フレーム目との比較による Null への置き換えをしない。フレーム間の間引きはしないでそのまま出力する。キーフレームがないモデルに対しては Null を出力せずモデルツリーに設定される値を最初と最後のフレームに設定しデータを生成する。

1.5.3 mrs からの Null 置き換えの制御

mrs はモーションリソースを意味する。モデルコンバート時の親子階層、オプション、ポリゴン数などの情報をまとめる。現時点ではサターンの頃の仕様が一部残っており未使用のパラメータも含まれる。

ユーザによる mrs の書き換えは原則禁止ですが一部 mrs からの Null 置き換え禁止の制御が可能であるのでこの部分について説明する。またモデルコンバート時のハイドフラグも mrs に書き込まれイベントモーションの省略が可能な時には Null への置き換えがされる。

mrs において<Tree>の項目で始まるデータ部分はモデルツリーデータの構造を表す。

(例)

```
<Tree> 53
# rot offset x y z      | Id| depth "Name"                      EsSsKtKrKsHd
[ 0.00  0.00  0.00]    | 0| 0 "body_3"                      000000
[ 0.00  0.00  0.00]    | 1| 1 "null"                        000000
...
```

ここで EsSsKtKrKsHd に注目する。並ぶ 5 つのゼロは 1 にすると TRUE, 0 にすると FALSE を意味する。Kt, Kr, Ks により Null への置き換え禁止がモデル単位で制御可能。この部分を手で書きかえる。モーション出力時にこの mrs を参照してコンバートする。

Es は EvalSkip を意味する。値を 1 にすることでこのモデルがモーションに関わらないことを指定する。モーションデータの配列にこのモデルのデータは含まれずモーション実行中はこのモデルをスキップしてモーションを実行することを指定する。

Ss は EvalShapeSkip を意味する。値を 1 にすることでこのモデルがシェイプモーションに関わらないことを指定する。シェイプモーションデータの配列にこのモデルのデータが含まれずモーション実行中はモデルをスキップしてモーションを実行することを指定する。

Kt は KeepTrans を意味する。これは nre の transArray に相当する。トランスモーションの Null の置き換えを禁止する。

Kr は KeepRotation を意味する。これは nre の rotArray に相当する。ローテーションモーションの Null の置き換えを禁止する。

Ks は KeepScale を意味する。これは nre の scaleArray に相当する。スケールモーションの Null の置き換えを禁止する。

Hd は Hide を意味する。これはモデルの非表示を制御する NJS_CNK_OBJECT 構造体の evalflags に NJD_EVAL_HIDE が設定されているかどうかをフラグとして設定する。イベントモーションの最適化に影響する。evalflags に NJD_EVAL_HIDE がなくイベントモーションで NJD_EVENT_HIDE がまったく無い場合、evalflags に NJD_EVAL_HIDE がありイベントモーションで NJD_EVENT_HIDE が最初から最後まで設定されてる場合にはモデルに対するイベントモーションデータは NULL となる。

1.5.4 EvalSkip 及び EvalShapeSkip

EvalSkip とは、モーションデータにおいて常にモーションしないモデルに設定する。マテリアルネームから Es 文字列を設定することでモデル単位での設定ができる。EvalShapeSkip とは、シェイプモーションデータにおいて常にシェイプモーションしないモデルに設定する。ただしどちらもマテリアルネームから設定できるのはポリゴンを持ちマテリアルを持つモデルのみ。

ポリゴンを持たないモデルに関してこのフラグを立てるためには NullObject EvalFlags (後述) を使う。

EvalSkip、EvalShapeSkip はモデル階層に対しモーションを生成したあとその親子階層が一部変更されたような場合でその追加部分をスキップすれば既存のモーションが使えるような場合に利用する。例えばキャラクターに銃を持たせたい場合その銃に EvalSkip、EvalShapeSkip を設定する。これにより銃を持っていない時と持っている時で同じモーションが使いまわせる。

プログラマー用に Null モデルを仕込む必要がある場合は NullObject EvalFlags を利用する。

ただし CompactShape ではエンベロープとシェイプの共存を実現するためエンベロープの頂点が他の Null モデルに移動する場合があります。本来 Null モデルでスキップ可能なモデルもスキップできない場合があります。スキップできない場合 Warning と共に EvalShapeSkip は無視される。

1.6 NullObject EvalFlags

ポリゴンを持たない Null Object はマテリアルを持たないためマテリアルネームエディタでオブジェクト構造体の Evalflags にフラグを立てることができない。そのため Null Object に必要最小限の Evalflags 設定をするための仕組みを用意する。

リソース追加

UseNullEval Null Object に対する Evalflags 設定の有効のオン/オフ。

これに対応するコマンドラインオプションは

-ne or -nulleva Use null object evalflags

UseNullEval リソースがオンの場合に限り Null Object (ポリゴン、マテリアルなし) に対し次の文字列フラグをつけられるようにする。この文字はアスキーフォーマットにそのまま出力されるが何の手当てもしない。本機能により次の二つのフラグ設定を Null Object に対しする。

• NJD_EVAL_SKIP|NJD_EVAL_SHAPE_SKIP (同時に二つ)

・NJD_EVAL_MODIFIER

シンプルにするため現状 Null Object の先頭一文字のみをフラグとする。

設定する文字列も複雑な文字列をやめ一文字とする。

'S'を設定すると NJD_EVAL_SKIP と NJD_EVAL_SHAPE_SKIP の両方が設定される。これはあらかじめ親子階層にモーションに関わらない Null を仕込んでおき銃を持たせたりする場合に利用する。

'M'を設定かつモデル階層のルートであると Modifier Volume 親子階層としてコンバートされルートの Null Object の Evalflags に NJD_EVAL_MODIFIER が設定される。

'M'を階層の途中に設定しても無視され通常に名前の一部として処理される。

'S'と'M'を同時に設定する事はできない。

特殊なエディタは作らない。

ModifierVolume はマテリアルとテクスチャを二組持つデータである Modifier Data において第一のマテリアルとテクスチャが表示される状態の Modifier Data と Modifier Volume と交差した部分を第二のマテリアルとテクスチャに切り替えて表示できる機能。新規に Modeifier Volume であることを示すためのフラグ。

```
#define NJD_EVAL_MODIFIER    BIT_9
```

を定義する。設定を検出した場合オブジェクト構造体に NJD_EVAL_MODIFIER を設定しそれに関わるモデルツリー内のオブジェクトのモデルデータを Modifier Volume として出力する。

(注意事項)

・ライブラリはこのフラグをチェックしない。ユーザはこのフラグを検出しモディファイア専用関数を呼び出す必要がある。

Modifier Data としてチャンクフォーマットの独立三角形コリジョンデータである NJD_CO_P3 形式を利用する (モディファイアボリュームデータにハードウェアの仕様でストリップは利用できない)。NJD_CO_P3 出力が指定されるとすべてのマテリアルが無視される。

NinjaExport の出力チャンク形式でチャンクボリューム Polygon3 を指定することでモディファイアボリュームの出力は可能だがこの場合シーン全体がチャンクボリュームになる。モデラーでテクスチャを二枚貼って Modifier Data を作成しこれと干渉し合うモーション付きを一つのシーンにするためには各親子階層ごとに属性を設定する必要がある。これに NullObjectEvalflags を使用する。2 パラデータはテクスチャ 2 枚貼りを 2 パラデータとして出力するオプションで全体に設定するかマテリアルネームエディタで個々に設定する。

モディファイアボリュームと通常のモデルの親子階層は禁止。モディファイアボリュームのみからなる親子階層は利用可能。

< 注意事項 >

ライブラリは NJD_EVAL_MODIFIER を一切チェックしない。ユーザが確認しフラグが立っていた場合にモディファイア用のルーチンと呼ぶ必要がある。

モディファイアボリューム用のデータは完全に閉じた形状でなければならない。開いたモデルや穴の空いたモデルは利用できない。

使用時は必ず UseNullEval がオンであること。

2. 構造体とマクロ定義

次に Ninja2 で定義される構造体およびそれに設定されるフラグ定義を説明する。ここでは CompactShape に関しては省略しモデル、カメラ、ライトに関して説明する。CompactShape に関しては CompactShape の説明を参照のこと。

2.1 NJS_OBJECT 構造体およびマクロ定義

2.1.1 Chunk Object 構造体

```
typedef struct cnkobj {
    Uint32          evalflags; /* 評価方法の最適化フラグ */
    NJS_CNK_MODEL   *model;    /* モデル構造体ポインタ */
    Float           pos[3];     /* 平行移動 */
    Angle           ang[3];     /* 回転 */
    Float           scl[3];     /* スケール */
    struct obj      *child;     /* 子ども object へのポインタ */
    struct obj      *sibling;   /* 兄弟 object へのポインタ */
    Float           re_quat;     /* クォータニオンの第四成分 */
} NJS_CNK_OBJECT;
```

Ninja2 では回転のタイプにクォータニオンを指定してコンバートすると NJS_CNK_OBJECT 構造体の回転成分にクォータニオンが格納される。具体的にはデータは Angle の ang 及び re_quat の計四つメンバーに格納される。この構造は従来の構造を残しつつ XYZ 回転 (オイラー角) とクォータニオンの場合の両方が表現できるようにするためのものである。

回転成分がクォータニオンの場合 evalflags メンバに NJD_EVAL_QUATERNION (後述) のフラグが設定される。NJS_CNK_MODEL のポインタである model には頂点データとポリゴンデータが格納される。pos にはモデルの親からの相対位置、scl には親からの相対スケール値が格納される。child には親子階層を構成するモデル階層データにおいて自分の子供のモデルが sibling には兄弟のモデルが設定される。

モデル階層描画はルートのモデルから child 優先で child, sibling ポインタにつながるモデルを順次処理していくことで実現される。モーションではこのトレース順に出現したモデルを一次元配列に並べた構造を基本とする。モデル階層データとモーションの一次元配列データを順次並列に参照することでモデル階層はモーションする。

evalflags は各モデルごとの計算に必要な処理を示すためのフラグが設定される。evalflags はモーションの処理に重要なフラグを数多く定義する。次に evalflags を説明する。

2.1.2 evalflags フラグ説明

```
#define NJD_EVAL_UNIT_POS      BIT_0 /* 移動成分が 0 である */
#define NJD_EVAL_UNIT_ANG      BIT_1 /* 回転成分が 0 である */
#define NJD_EVAL_UNIT_SCL      BIT_2 /* スケール成分が 1 である */
#define NJD_EVAL_HIDE          BIT_3 /* モデル描画をしない */
#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 の回転順番指定 */
#define NJD_EVAL_SKIP          BIT_6 /* モーションに関わらないモデルを示す */
#define NJD_EVAL_SHAPE_SKIP    BIT_7 /* 頂点モーションに関わらないモデルを示す */
#define NJD_EVAL_CLIP          BIT_8 /* child のモデルクリップ打ち切ることを示す */
#define NJD_EVAL_MODIFIER      BIT_9 /* モディファイアデータであることを示す */
#define NJD_EVAL_QUATERNION    BIT_10 /* クォータニオンを利用していることを示す */
#define NJD_EVAL_ROTATE_BASE   BIT_11 /* マトリックスを保存する */
#define NJD_EVAL_ROTATE_SET    BIT_12 /* マトリックスを設定する */
```

```
#define NJD_EVAL_ENVELOPE      BIT_13    /* エンベロープに関わるモデルであることを示す */
#define NJD_EVAL_MASK          0x3fff    /* 上記ビットを抽出するマスク */
```

これらのフラグはコンバータで設定される。

NJD_EVAL_UNIT_POS	移動成分が0であることを示す。移動成分のマトリクス計算が省略される。コンバータで自動設定される。
NJD_EVAL_UNIT_ANG	回転成分が0であることを示す。回転成分のマトリクス計算が省略される。コンバータで自動設定される。
NJD_EVAL_UNIT_SCL	スケール成分がすべて1の場合に設定される。スケールのマトリクス計算が省略される。コンバータで自動設定される。
NJD_EVAL_HIDE	モデル描画をしないことを示す。マテリアルネームエディタを利用しモデルからユーザ指定で設定される。
NJD_EVAL_BREAK	child のトレースを打ち切る。複数の child を一括して表示しない場合などに利用する。モーションでこれを利用する場合は対応するモデルとモーションデータがずれる可能性があるので注意が必要である。
NJD_EVAL_ZXY_ANG	LightWave3D の回転の順番は XYZ ではなく ZXY となる。LightWave3D データにおいてフラグを設定しライブラリがこれを参照する。
NJD_EVAL_SKIP	モーションデータに関わらないモデル。モーションデータの一次元配列データには関与しない。モーションデータ上にはデータがなくモーション実行時はスキップされる。
NJD_EVAL_SHAPE_SKIP	シェイプモーションデータに関わらないモデル。シェイプモーションデータの一次元配列データには関与しない。シェイプモーションデータ上にはデータがなくシェイプモーション実行時にはスキップされる。
NJD_EVAL_CLIP	child のモデルクリップトレースを打ち切る。カレントのモデルの半径 R を child を含むサイズにしておくことでモデルクリップ処理を省略する。
NJD_EVAL_MODIFIER	モディファイアボリュームデータであることをユーザに明示する。ライブラリはこれを参照しない。ユーザはこのフラグを発見した場合にモディファイアボリューム用関数を意識的に呼び出す必要がある。
NJD_EVAL_QUATERNION	回転成分の表現にクォータニオンを利用していることを示す。
NJD_EVAL_ROTATE_BASE	カレントのマトリクスを保存する。
NJD_EVAL_ROTATE_SET	保存されたマトリクスをカレントに書き戻す。
NJD_EVAL_ENVELOPE	このモデルがエンベロープに関わるモデルであることを示す。 <u>このフラグがセットされるモデルはハイド設定が無効となる。</u>

NJD_EVAL_ROTATE_BASE, NJD_EVAL_ROTATE_SET は Softimage のトグルエフェクトローテーションに対応するために用意される。100%再現可能かどうかは現在未定。

2.2 NJS_CAMERA 構造体及びマクロ定義

NJS_CAMERA 構造体は次の通り。モーションとして使用するパラメータは、ポジション、ターゲットもしくはベクトル、ロール、画角の4つ。

2.2.1 NJS_CAMERA 構造体

```
typedef struct{
    Float      px, py, pz;      /* カメラのポジション */
    Float      vx, vy, vz;      /* カメラの方向ベクトルまたは、ターゲットポジション */
    Sangle     roll;            /* カメラのロール */
    Sangle     ang;             /* カメラの画角 */
    Uint32     type;            /* カメラのタイプ */
} NJS_CAMERA;
```

- ・ px,py,pz カメラポジション
- ・ vx,vy,vz カメラの方向ベクトルまたは、ターゲットポジション
- ・ roll カメラのロール
- ・ ang カメラの画角
- ・ type カメラ種類

```
#define NJD_CTYPE_VECTOR      (1) /* カメラ方向 */
#define NJD_CTYPE_TARGET      (2) /* ターゲット */
```

2.3 NJS_LIGHT 構造体及びマクロ定義

NJS_LIGHT 構造体は次の通り。Ninja2 ではスポットライトをサポートしない。NormalDraw は削除された。また NJS_LIGHT 構造体のメンバー意味は描画関数の種類 (EasyDraw, SimpleDraw, EasyMultiDraw, SimpleMultiDraw) で意味が変わるので注意。またライトモーションの成分も変化するので注意が必要。

Ninja2 のライトは常に利用する描画関数の種類を意識してコンバートをする必要がある。

2.3.1 NJS_LIGHT 構造体

```
typedef struct{
    Float      x,y,z;           /* ポジション or ベクトル */
    Float      r,g,b;           /* 光源カラー */
    Float      f1,f2;           /* Intensity and Ambient or nrange and frange */
    Sint32     func;            /* 描画関数の種類。Easy, Simple, EasyMulti, SimpleMulti */
    Uint32     type;            /* タイプ点光源、平行光源、アンビエントライト */
}NJS_LIGHT;
```

- ・ x,y,z 点光源のときライトポジション, 平行光源のときライトベクトル。
- ・ r,g,b ライトカラー 0.f - 1.f が通常値。Multi 系のみ、それ以外 (負数) も設定可能
- ・ f1,f2 Easy、Simple ライトのときインテンシティ、アンビエント 0.f - 1.f が通常値。それ以外 (負数) も設定可能。

EasyMultiDraw、SimpleMultiDraw の点光源ライトのときライト範囲 (nrange, frange)

EasyMultiDraw、SimpleMultiDraw の平行光源ライトのとき未使用

EasyMultiDraw、SimpleMultiDraw のアンビエントライトのとき未使用

- ・ func 描画関数設定 描画関数タイプ定義を使用

```
#define NJD_MFUNC_UNDEFINE      (0) /* 未定義関数 */
#define NJD_MFUNC_EASY          (1) /* CnkEasy 関数 */
```

```
#define NJD_MFUNC_SIMPLE      (2)    /* CnkSimple 関数          */  
#define NJD_MFUNC_EASY_MULT I (3)    /* CnkEasyMulti 関数      */  
#define NJD_MFUNC_SIMPLE_MULT I (4)    /* CnkSimpleMulti 関数    */
```

・ type ライト種類定義

```
#define NJD_LTYPE_POINT      (1)    /* 点光源                  */  
#define NJD_LTYPE_VECTOR    (2)    /* 平行光源                */  
#define NJD_LTYPE_AMBIENT    (3)    /* アンビエントライト      */
```


2.3.2 Ninja2 ライブラリ関数と光源パラメータの関係

描画関数とライトの種類によるライト構造体の各データの意味を説明する。

EasyDraw

利用できるのは平行光源一つのみ。

< 平行光源 >

- type NJD_LTYPE_VECTOR (固定)
- x,y,z ライト方向
- r,g,b ライトカラー
- f1,f2 インテンシティ、アンビエント値
- func NJD_MFUNC_EASY

SimpleDraw

利用できるのは平行光源一つのみ。

< 平行光源 >

- type NJD_LTYPE_VECTOR (固定)
- x,y,z ライト方向
- r,g,b ライトカラー
- f1,f2 インテンシティ、アンビエント値
- func NJD_MFUNC_SIMPLE

EasyMultiDraw

利用できるのは平行光源、点光源、アンビエントライトの三つ。

< 平行光源 >

- type NJD_LTYPE_VECTOR
- x,y,z ライト方向
- r,g,b ライトカラー
- f1,f2 無効
- func NJD_MFUNC_EASY_MULTI

< 点光源 >

- type NJD_LTYPE_POINT
- x,y,z ライト位置
- r,g,b ライトカラー
- f1,f2 ライト範囲 nrange, frange
- func NJD_MFUNC_EASY_MULTI

< アンビエントライト >

- type NJD_LTYPE_AMBIENT
- x,y,z 無効
- r,g,b アンビエントカラー
- f1,f2 無効
- func NJD_MFUNC_EASY_MULTI

SimpleMultiDraw

利用できるのは平行光源、点光源、アンビエントライトの三つ。

< 平行光源 >

- type NJD_LTYPE_VECTOR
- x,y,z ライト方向
- r,g,b ライトカラー

- f1,f2 無効
- func NJD_MFUNC_SIMPLE_MULTI

< 点光源 >

- type NJD_LTYPE_POINT
- x,y,z ライト位置
- r,g,b ライトカラー
- f1,f2 ライト範囲 nrange, frange
- func NJD_MFUNC_SIMPLE_MULTI

< アンビエントライト >

- type NJD_LTYPE_AMBIENT
- x,y,z 無効
- r,g,b アンビエントカラー
- f1,f2 無効
- func NJD_MFUNC_SIMPLE_MULTI

2.4 NJS_MOTION 構造体及びマクロ定義

NJS_MOTION 構造体はすべてのモーションデータの基本構造となる。モデル、カメラ、ライト及びモデルの頂点アニメーション（シェイプモーション）、ハイドモーション（イベントモーション）に利用される。

NJS_MOTION 構造体に格納されるデータ種類はモーション要素ビット列で表現される。要素数に合わせてポイント数の異なる NJS_MDATA1 ~ NJS_MDATA4 構造体呼び出される。NJS_MDATA5 構造体はリザーブ。

CompactShape は単独で NJS_MOTION 構造体に格納される。他のモーション成分と同一の NJS_MOTION 構造体に格納されることはない。CompactShape データと他のモーション成分を同時に実行したい場合はモデルモーション用の NJS_MOTION と CompactShape 用の NJS_MOTION を同時に実行する。

2.4.1 モーション構造体

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

mdata には、モデル階層に含まれる全ての NJS_OBJECT に対応する数の NJS_MDATA が一次元配列として格納される。NJS_MDATA では、モーション構成要素数により NJS_MDATA1 ~ 4 構造体をそれぞれ使用する。例えば移動（T）、回転（R）、スケール（S）の三つの成分からなるモーションデータでは NJS_MDATA3 が利用される。

nbFrame には実際のモーションのフレーム数が type にはモーション要素数ビット列が設定される。inp_fn には Linear、Spline による補完方法の定義と MDATA に格納される要素数が格納される。

```
#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_UI16 を利用       */
```

従来 SimpleShape に利用していた NJD_MTYPE_VERT_4、NJD_MTYPE_NORM_5 及びスポットライト用の NJD_MTYPE_SPOT_10 は廃止する。

```
#define NJD_MTYPE_VERT_4          (BIT_4)    < 廃止 >  
#define NJD_MTYPE_NORM_5          (BIT_5)    < 廃止 >  
#define NJD_MTYPE_SPOT_10        (BIT_11)   < 廃止 >
```

ShortAngle 形式サポートのため NJD_MTYPE_NORM_5 を NJD_MTYPE_SANG_1 に変更。NJD_MTYPE_VERT_4 はライトモーション用 VECTER 成分として NJD_MTYPE_VEC_0 を割り付ける。ハイドのためのイベントモーションデータを NJD_MTYPE_EVENT_4 に割り付ける。スポットライトの定義されていた (BIT_11) はリザーブ。NJD_MTYPE_POINT_10 は新仕様で NJD_MTYPE_INTENSITY_9 と同時に使えないことを示すため NJD_MTYPE_POINT_9 に改名する。また NJD_MTYPE_INTENSITY_9 は従来はインテンシティのための float 一つのデータあったのに対し Ninja2 ではインテンシティとアンビエントの float 二つで表現される。

2D 用 CellSprite でも NJS_MOTION 構造体を利用する。ビット割付の節約とほぼ同様の成分に対しビットを使いまわすことを目的とし 3D モデルと 2D 用 CellSprite で同じモーションを共有することはないため 3D モデル用と 2D 用 CellSprite の場合で同一要素を示すビットに違う意味を与えている。CellSprite において下記ビットは次の意味を持つ。

- ・ NJD_MTYPE_POS_0 は CellSprite の位置 (xyz) 。 NJS_MKEY_F を使用する。xy は座標、z はプライオリティに使用される。
- ・ NJD_MTYPE_ANG_1 は CellSprite の Z 回転 (z) 。 NJS_MKEY_A1 を使用する。
- ・ NJD_MTYPE_SCL_2 は CellSprite の 2 D スケール (xy) 。 NJS_MKEY_F2 を使用する。
- ・ NJD_MTYPE_RGB_8 はマテリアルディフューズカラー (rgb) 。 NJS_MKEY_UI32 を使用する。

3D モデルと 2DCellSprite でキーに使う構造体が異なることに注意が必要。CellSprite に関しては CellSprite 仕様書参照のこと。

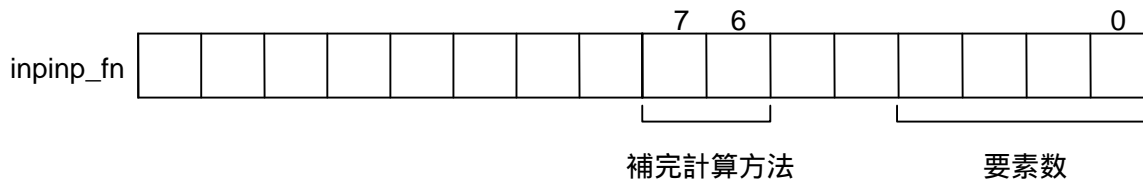
次にモーション要素ビットに関わる説明をする。後ろの文字は複数の要素があるモーションで番号が若い順に要素が並ぶことを規定する。また同じ番号のものは同時に使えないこと意味する。また番号の無い物を他の要素と組み合わせて使われないことを意味する。

NJD_MTYPE_POS_0	モデルの平行移動成分 (xyz) を表現する。NJS_MKEY_F 構造体を使用する。NJS_MTYPE_VEC_0 とは同時に使えない。
NJD_MTYPE_ANG_1	モデルの回転成分 (xyz) を表現する。NJS_MKEY_F 構造体を使用する。モデルモーションにおいて NJD_MTYPE_SANG_1, NJD_MTYPE_QUAT_1 は同時に使えない。
NJD_MTYPE_SCL_2	モデルのスケール成分 (xyz) を表現する。NJS_MKEY_F 構造体を使用する。
NJD_MTYPE_VEC_3	カメラのベクトル成分 (xyz) を表現する。NJS_MKEY_F 構造体を使用する。
NJD_MTYPE_VEC_0	ライトのベクトル成分 (xyz) を表現する。NJS_MKEY_F 構造体を使用する。ライトの場合に使用する NJD_MTYPE_POS_0 と同時には使えない。
NJD_MTYPE_SANG_1	ShortAngle によりモデルの回転成分 (xyz) を表現する。NJS_MKEY_SA 構造体を使用する。
NJD_MTYPE_TARGET_3	カメラのターゲット成分 (xyz) を表現する。NJD_MTYPE_VEC_3 と同時に使えない。
NJD_MTYPE_ROLL_6	カメラのロール成分 (z) を表現する。NJS_MKEY_SA1 構造体を利用する。
NJD_MTYPE_ANGLE_7	カメラの画角 (Sangle) を表現する。NJS_MKEY_SA1 構造体を利用する。
NJD_MTYPE_RGB_8	ライトのカラー (rgb) を表現する。描画関数により格納される rgb の値の意味が変わる。NJS_MKEY_F 構造体を使用する。
NJD_MTYPE_INTENSITY_9	EasyDraw、SimpleDraw で有効。このフラグは要素数 2 であり Intensity (float)、Ambient(float) がセットで扱われる。NJS_MKEY_F2 構造体を利用する。NJD_MTYPE_POINT_9 と同時には使えない。
NJD_MTYPE_POINT_9	EasyMultiDraw、SimpleMultiDraw で有効。ライトの範囲 (nrange, frange) を表現する。NJS_MKEY_F2 構造体を利用する。NJD_MTYPE_INTENSITY_9 と同時には使えない。
NJD_MTYPE_QUAT_1	モデルの回転成分をクォータニオンで表現する。NJS_MKEY_QUAT 構造体を使う。NJD_MTYPE_ANG_1 と NJD_MTYPE_SANG_1 は同時に使えない。
NJD_MTYPE_SHAPEID	CompactShape モーションに利用する。CompactShape は他のデータと組み合わせて NJS_MOTION 構造体には出力されない。CompactShape だけで NJS_MOTION 構造体に格納される。
NJD_MTYPE_EVENT_4	イベントモーションに利用する。現在イベントモーションとしてサポートしているのはハイドのみ。NJD_MTYPE_POS_0、NJD_MTYPE_ANG_1、NJD_MTYPE_SCL_2 と組み合わせてモデルモーションに利用される。

補完計算方法を inp_fn の上位 2 ビットで指定する。

```
#define NJD_MTYPE_LINER          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        */
```

どの構造体を使っているかを示すために、inp_fn の下位 4 ビットには要素数が格納される。



2.4.2 NJS_MDATA1 ~ 5 構造体

```
typedef struct {
    void          *p[1];  /* モーションポインタ */
    Uint32        nb[1];  /* キーフレーム数     */
} NJS_MDATA1;
```

```
typedef struct {
    void          *p[2];  /* モーションポインタ */
    Uint32        nb[2];  /* キーフレーム数     */
} NJS_MDATA2;
```

```
typedef struct {
    void          *p[3];  /* モーションポインタ */
    Uint32        nb[3];  /* キーフレーム数     */
} NJS_MDATA3;
```

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

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

データはキーフレーム構造で表現される。

nb[i]には p[i]要素のモーションキーフレーム数が入る。

2.4.3 キー構造体

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

平行移動 (Pos)、スケール (Scl)、ベクトル (Vec) に利用する。

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

回転 (ANG(Angle)) に利用する。

```
typedef struct {  
    Uint16      keyframe; /* キーフレーム番号          */  
    Sangle      key[3];   /* ShortAngle 型キー値(配列)  */  
} NJS_MKEY_SA;
```

回転 (SANG(ShortAngle)) に利用する。

```
typedef struct {  
    Uint32      keyframe; /* キーフレーム番号          */  
    Float       key[4];   /* Float 型キー値(配列 4)     */  
} NJS_MKEY_QUAT;
```

回転 (quaternion) に利用する。

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

ライト色(RGB)に利用する。

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

従来カメラのロール、ライトの画角に使われていた。Ninja2 では代わりに NJS_MKEY_SA1 が利用される。2D 用の CellSprite で使用。

```
typedef struct {  
    Uint16      keyframe; /* キーフレーム番号          */  
    Sangle      key;      /* ShortAngle                  */  
} NJS_MKEY_SA1;
```

カメラのロール、ライトの画角に利用する。

```
typedef struct {  
    Uint32      keyframe; /* キーフレーム番号          */
```

```

    Float          key;          /* Float 型キー値          */
} NJS_MKEY_F1;

```

従来はライト強度(Intensity)に使われていた。Ninja2 では未使用。リザーブ。

```

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

```

EasyDraw, SimpleDraw では Intensity, Ambient 成分に使用し、EasyMultiDraw, SimpleMultiDraw では nrange, frange に利用する。

```

typedef struct {
    Uint16          keyframe;     /* キーフレーン番号          */
    Uint16          key;         /* 符号無し int16 型キー値    */
} NJS_MKEY_UI16;

```

イベントモーションに利用する。現在はハイドモーションのみ。

```

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

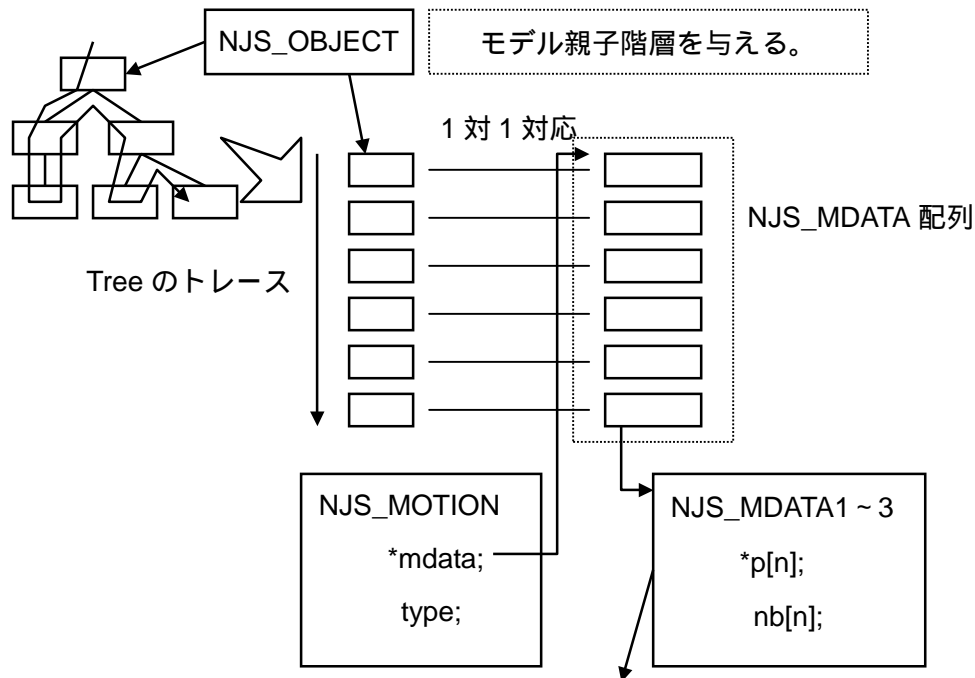
```

CompactShape の nam 出力に利用する。

3. オブジェクトモーション

構造体関連図

Object Tree



(例) pos のみの場合：要素数 1 なので NJS_MDATA1 を使う。

NJS_MOTION 構造体の type = NJD_MKEY_POS_0;

NJS_MKEY_F pos[] = {, , , ...};

NJS_MDATA1 mdata[] = {{pos, poskey_n}, ...};

(例) pos, ang, scl の場合：要素数 3 なので、NJS_MDATA3 を使う。

type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1 | NJD_MTYPE_SCL_2;

MKEY_F pos[] = {, , , ...};

MKEY_A ang[] = {, , , ...};

MKEY_F scl[] = {, , , ...};

MDATA3 mdata[] = {{pos, ang, scl, poskey_n, angkey_n, sclkey_n}, ...};

(例) カメラの pos, vec の場合：要素数 2 なので NJS_MDATA2 を使う。

type = NJD_MTYPE_POS_0 | NJD_MTYPE_VEC_3;

NJS_MKEY_F pos[] = {, , , ...};

NJS_MKEY_F vec[] = {, , , ...};

NJS_MDATA2 mdata[] = {{pos, vec, poskey_n, veckey_n}, ...};

3.1 モデルモーション解説

モーションはすべてキーフレームデータで与えられる。
ユーザはキーフレームデータを線形補完、スプライン補完を使用することにより、モーションを実行する。
キーフレーム番号はゼロから始まる。マイナス値は使用できない。

Translation (Position)	平行移動
Angle	16 ビット (ShortAngle) 32 ビットオイラー角と quaternion が利用可能。
Scale	拡大縮小
Event	イベントモーション。現時点ではハイドのみ。
ShapeId	ShapeList 参照による頂点モーション (CompactShape)

オブジェクトモーションの要素として、上記の 5 つを考える。

CompactShape データは ShapeList を格納する nas ファイルおよび ShapeId と keyframe を格納する nam ファイルから構成される。移動 (T) 回転 (R) スケール (S) イベントモーション (E) から構成されるモデルモーションを合わせて出力する場合は nam ファイルには CompactShape 用の NJS_MOTION と TRS 用の NJS_MOTION データが二つ格納される。そのためオブジェクトモーションにおいて NJS_MOTION 構造体に同時に設定される最大要素数は 4 になる。NJS_MDATA1 ~ 4 構造体が利用される。

オブジェクトモーションに関わる要素

```
#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_SANG_1  (BIT_5)   /* NJS_MKEY_SA 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_UI16 を利用   */
```

後ろの番号は複数の要素が含まれる NJS_MOTION データにおいて要素の並び順番を規定する。要素は若い番号のものから並び同じ番号のものは同時に使われることはない。上記フラグはモーション構造体のメンバ type に設定される。

(例 1) pos と ang の場合

```
type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1;
mdata[] = {pos, ang, ...}
```

(例 2) 例 1 にイベントモーションが組み合わさった場合

```
type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1 | NJD_MTYPE_EVENT_4;
mdata[] = {pos, ang, event, ...}
```

モーション補完方法は、type の上位 2 ビットで指定される。

```

#define NJD_MTYPE_LINER          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        */

```

ルートが pos, ang で他は ang のみのモーションモデルでは、NJS_MDATA2 構造体を使用し、ルート以外の pos には NULL ポインタを使用することにより対応する。

```

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

```

この場合、ang2, ang3 を左につめてはならないことに注意。

3.2 モデルモーションアスキーフォーマットのマクロ定義

3.2.1 モーション構造体マクロ

これらのマクロは NjDef.h に定義されアスキーフォーマット出力においてモデルモーション、CompactShape、ライトモーション、カメラモーションで利用される。

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	要素数 1 のモーションデータ成分をくくるための構造体。
MDATA2	要素数 2 のモーションデータ成分をくくるための構造体。
MDATA3	要素数 3 のモーションデータ成分をくくるための構造体。
MDATA4	要素数 4 のモーションデータ成分をくくるための構造体。
MDATA5	要素数 5 のモーションデータ成分をくくるための構造体。リザーブ。
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	MDATA に格納されるモーションの種類を文字列のフラグで格納。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文字列のフラグで格納。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。
MotionBit	MDATA に格納されるモーションの種類をヘキサの数値で格納。今後は文字列のフラグ設定が可能な MotionBitF を利用する。過去の互換、現時点では MotionBitF に対応していない CellSprite 用に存在する。
InterpolFct	キーフレーム補完の種類とモーションの要素数をヘキサのフラグで格納。今後は文字列のフラグで設定が可能な InterpolFctF を利用する。過去の互換、現時点では MotionBitF に対応していない CellSprite 用に存在する。

```
#define MOTION_START
#define MOTION_END

#define MDATA1      NJS_MDATA1
#define MDATA2      NJS_MDATA2
#define MDATA3      NJS_MDATA3
#define MDATA4      NJS_MDATA4
#define MDATA5      NJS_MDATA5
#define MOTION      NJS_MOTION

#define MdataArray
#define MFrameNum
#define MotionBitF(_f)      (_f)
#define InterpolFctF(_f)    (_f)
#define MotionBit
#define InterpolFct

#define DEFAULT_START
#define DEFAULT_END
```

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_SAN1	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。

```

/* 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_SAN1      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_PO19      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)

```

3.2.2 モーションキー構造体マクロ

これらのマクロは NjDef.h に定義されアスキーフォーマット出力においてモデルモーション、CompactShape、ライトモーション、カメラモーションで利用される。

MKEYF	NJS_MKEY_F 構造体データ出力に対応する。
MKEYF1	NJS_MKEY_F1 構造体データ出力に対応する。
MKEYF2	NJS_MKEY_F2 構造体データ出力に対応する。
MKEYA	NJS_MKEY_A 構造体データ出力に対応する。

MKEYSA	NJS_MKEY_SA 構造体データ出力に対応する。
MKEYA1	NJS_MKEY_A1 構造体データ出力に対応する。
MKEYSA1	NJS_MKEY_SA1 構造体データ出力に対応する。
MKEYQ	NJS_MKEY_QUAT 構造体データ出力に対応する。
MKEYUI32	NJS_MKEY_UI32 構造体データ出力に対応する。
MKEYSI32	NJS_MKEY_SI32 構造体データ出力に対応する。
MKEYUI16	NJS_MKEY_UI16 構造体データ出力に対応する。
MKEYSHID	NJS_MKEY_SHAPEID 構造体データ出力に対応する。

```

#define _KEYVAL          (65535.0f/360.0000F)
#define MKEYF( _k, _x, _y, _z)          ¥
    {((UInt32)(_k)), ((float)(_x)), ((float)(_y)), ((float)(_z)))}
#define MKEYF1( _k, _a)          ¥
    {((UInt32)(_k)), ((float)(_a)))}
#define MKEYF2( _k, _n, _f)          ¥
    {((UInt32)(_k)), ((float)(_n)), ((float)(_f)))}
#define MKEYA( _k, _x, _y, _z)          ¥
    {((UInt32)(_k)), ((long)(_KEYVAL*_x)),          ¥
    ((long)(_KEYVAL*_y)), ((long)(_KEYVAL*_z)))}
#define MKEYSA( _k, _x, _y, _z)          ¥
    {((UInt16)(_k)), ((short)(_KEYVAL*_x)),          ¥
    ((short)(_KEYVAL*_y)), ((short)(_KEYVAL*_z)))}
#define MKEYA1(_k, _v)          {((UInt32)(_k)), ((long)(_KEYVAL*_v)))}
#define MKEYSA1(_k, _v)          {((UInt16)(_k)), ((short)(_KEYVAL*_v)))}
#define MKEYQ( _k, _a, _x, _y, _z)          ¥
    {((UInt32)(_k)), ((float)(_a)),          ¥
    ((float)(_x)), ((float)(_y)), ((float)(_z)))}
#define MKEYUI32(_k, _v)          {((UInt32)(_k)), ((unsigned long)(_v))}
#define MKEYSI32(_k, _v)          {((UInt32)(_k)), ((long)(_v))}
#define MKEYUI16(_k, _v)          {((UInt16)(_k)), ((unsigned short)(_v))}
#define MKEYSHID(_k, _id)          {((UInt32)(_k)), ((UInt32)(_id))}

```

3.2.3 TRS モーションマクロ

POSITION	モデルの移動 (T) 成分データを意味する。
QROTATION	モデルのクォータニオン表現の回転 (R) 成分データを意味する。
ROTATION	モデルの回転 (R) 成分データを意味する。
SROTATION	モデルの ShortAngle 表現の回転 (R) 成分データを意味する。
SCALE	モデルのスケール (S) 成分データを意味する。

```

#define POSITION    NJS_MKEY_F
#define QROTATION  NJS_MKEY_QUAT
#define ROTATION   NJS_MKEY_A
#define SROTATION  NJS_MKEY_SA
#define SCALE      NJS_MKEY_F

```

3.2.4 イベントモーション (ハイドモーション) マクロ

EVENTBITS	モデルのイベントモーション成分データを意味する。TRS モーション成分と組み合わせて利用される。
-----------	--

```

#define EVENTBITS    NJS_MKEY_UI16

```

モーションキーの先頭に利用される。

FEM_NONE	イベントモーションが何も無い場合に利用する。
FEM_HD	ハイドモーションが設定されていることを示すためのフラグ。

```

/* Flag Event Motion */
#define FEM_NONE          (0)
#define FEM_HD            (BIT_0)

```

フラグに利用されるビット列。

FEM_HD は現在一つしか定義されていないが将来二つ目以降が定義された場合それらのフラグはすべて or される。

3.2.5 アスキーフォーマットヘッダ情報

アスキーフォーマットの一行目には内容を示すためのコメントが入る。

(例)

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

ヘッダは次のブロックから構成される。

拡張子文字によるデータの種類の。

モーションに関わるのは次の4つ。

NAM	.nam	モデル、カメラ、ライトモーションで利用する。
NAS	.nas	CompactShape モーションで利用する。
CSNAM	.nam	CellSprite モーションで利用する。
CSA	.csa	CellSprite セルストリームモーションで利用する。

モデル、カメラ、ライト、CellSprite に関わるのは次の6つ。

NJA	.nja	複数の種類の Ninja データを含む場合に利用する。通常は texlist とモデル出力において利用する。
NAD	.nad	モデルデータ。
NAT	.nat	texlist データ。
NAC	.nac	カメラデータ。
NAL	.nal	ライトデータ。
SPA	.spa	CellSprite データ。
NSC	.nsc	出力ファイルをまとめたシーンファイルデータ。Ninja 用のビューアなどで利用する。

出力したコンバータのバージョン名。

文字列によるデータ内容の説明。

"Ninja2AsciiDataMix"	複数の Ninja2 データが格納されていることを示す。例えばテクスチャ (texlist) とモデルデータ。
"Ninja2AsciiDataMix CnkModel"	複数の Ninja2 データが格納されかつ内部データとしてチャンクモデルデータが含まれることを示す。
"Ninja2AsciiModel"	Ninja2 コンバータでは出力されない。リザーブ。
"Ninja2AsciiModel CnkModel"	チャンクモデルデータであることを示す。
"Ninja2AsciiMotion"	モーションデータであることを示す。
"Ninja2AsciiCompactShape"	コンパクトシェイプデータであることを示す。
"Ninja2AsciiTexture"	テクスチャ (texlist) データであることを示す。
"Ninja2AsciiLight"	ライトデータであることを示す。

"Ninja2AsciiCamera"	カメラデータであることを示す。
"Ninja2AsciiCellSprite"	セルスプライトデータであることを示す。
"Ninja2AsciiCellSpriteMotion"	セルスプライトモーションデータであることを示す。
"Ninja2AsciiCellStream"	セルストリームデータであることを示す。
"Ninja2Scene"	シーンデータであることを示す。

出力したモデラーの種類。モデラーの略号は次の通り。

SI	Softimage
LW	LightWave
MAX	3D Studio MAX
MAYA	MAYA

3.3 モデルモーションアスキーフォーマット出力例

ここでは TRS+E つまり移動 (T)、ShortAngle 回転 (R)、スケール (S) 及びイベント (E) モーション (ハイドモーション) を含む場合について説明する。

3.3.1 モデルモーション出力例

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

```
/* MOTION : motion_same_heavywalk_body_3 */
```

```
MOTION_START
```

```
POSITION pos_same_heavywalk_body_3_body_3[]
```

```
START
```

```
    MKEYF( 0,    0.000000F, -0.978237F,  0.058765F ),
```

```
    ... (省略)
```

```
    MKEYF( 120,  0.000000F, -0.978237F,  0.058765F ),
```

```
END
```

```
SROTATION rot_same_heavywalk_body_3_mune[]
```

```
START
```

```
    MKEYSA( 0,   -0.000278F, -0.000212F, 143.120361F),
```

```
    ... (省略)
```

```
    MKEYSA( 120, -0.000278F, -0.000212F, 143.120361F),
```

```
END
```

```
SCALE scl_same_heavywalk_body_3_mune[]
```

```
START
```

```

        MKEYF( 0,    1.000000F,  1.000000F,  1.000000F ),
        ... (省略)
        MKEYF( 120,  1.000000F,  1.000000F,  1.000000F ),
END

```

```

SROTATION rot_same_heavywalk_body_3_kata_r_1[]
START
        MKEYSA( 0,    -0.000001F, -23.736517F, 205.930191F),
        ... (省略)
        MKEYSA( 120,  -0.000001F, -23.736517F, 205.930191F),
END

```

```

ROTATION rot_same_heavywalk_body_3_ude_r_1[]
START
        MKEYSA( 0,    0.000000F, -21.839205F,  7.829935F),
        ... (省略)
        MKEYSA( 120,  0.000000F, -21.839205F,  7.829935F),
END

```

```

SROTATION rot_same_heavywalk_body_3_jnt22_1_1[]
START
        MKEYSA( 0,   -180.000000F, -203.736511F, 25.930185F),
        ... (省略)
        MKEYSA( 120, -180.000000F, -203.736511F, 25.930185F),
END

```

```

SROTATION rot_same_heavywalk_body_3_jnt22_2_1[]
START
        MKEYSA( 0,    0.000000F, 21.839220F,  7.829945F),
        ... (省略)
        MKEYSA( 120,  0.000000F, 21.839220F,  7.829945F),
END

```

```

SROTATION rot_same_heavywalk_body_3_jnt24_1[]
START
        MKEYSA( 0,    0.000101F,  0.000243F, 181.664856F),
        ... (省略)
        MKEYSA( 120,  -0.000124F,  0.000177F, 181.664856F),
END

```

```

SCALE scl_same_heavywalk_body_3_jnt24_1[]

```



```

START
    MKEYF( 0,    0.778696F,  1.000000F,  1.000000F ),
    ... (省略)
    MKEYF( 120,    0.778696F,  1.000000F,  1.000000F ),
END

SCALE scl_same_heavywalk_body_3_eyes[]
START
    MKEYF( 0,    1.000000F,  1.000000F,  1.000000F ),
    ... (省略)
    MKEYF( 120,    1.000000F,  1.000000F,  1.000000F ),
END

SCALE scl_same_heavywalk_body_3_mabuta[]
START
    MKEYF( 0,    1.000000F,  1.000000F,  1.000000F ),
    ... (省略)
    MKEYF( 120,    1.000000F,  1.000000F,  1.000000F ),
END

SRROTATION rot_same_heavywalk_body_3_mabuta_r[]
START
    MKEYSA( 0,   -82.999992F,  -0.000001F,   0.000000F),
    ... (省略)
    MKEYSA( 120,   -82.999992F,  -0.000001F,   0.000000F),
END

SRROTATION rot_same_heavywalk_body_3_mabuta_l[]
START
    MKEYSA( 0,   -82.999992F,  -0.000001F,   0.000000F),
    ... (省略)
    MKEYSA( 120,   -82.999992F,  -0.000001F,   0.000000F),
END

SRROTATION rot_same_heavywalk_body_3_kosi[]
START
    MKEYA( 0,    0.001969F,   0.012202F, -80.812225F),
    ... (省略)
    MKEYA( 120,    0.001969F,   0.012202F, -80.812225F),
END

```

```

SROTATION rot_same_heavywalk_body_3_momo_r[]
START
    MKEYSA( 0,    -0.002560F,    0.002578F, -96.571671F),
    ... (省略)
    MKEYSA( 120,  -0.002566F,    0.002584F, -96.571640F),
END

```

```

SROTATION rot_same_heavywalk_body_3_sune_r[]
START
    MKEYSA( 0,    0.000000F,    0.000000F, 87.133835F),
    ... (省略)
    MKEYSA( 120,  0.000000F,    0.000000F, 87.133827F),
END

```

```

SROTATION rot_same_heavywalk_body_3_foot_r[]
START
    MKEYSA( 0,    0.000000F,    0.001768F, -214.560989F),
    ... (省略)
    MKEYSA( 120,  0.000000F,    0.001764F, -214.560989F),
END

```

```

EVENTBITS event_same_heavywalk_body_3_foot_r[]
START
    MKEYUI16( 0,    FEM_NONE),
    MKEYUI16( 10,    FEM_HD),
    MKEYUI16( 60,    FEM_NONE),
    MKEYUI16( 120,    FEM_NONE),
END

```

```

POSITION pos_same_heavywalk_body_3_item3[]
START
    MKEYF( 0,    -3.021368F,    0.001376F,    0.000000F ),
    MKEYF( 120,  -3.021368F,    0.001376F,    0.000000F ),
END

```

```

SROTATION rot_same_heavywalk_body_3_momo_l[]
START
    MKEYSA( 0,    0.027253F,    -0.029918F, -190.344116F),
    ... (省略)
    MKEYSA( 120,  0.027252F,    -0.029916F, -190.344116F),
END

```

SRotation rot_same_heavywalk_body_3_sune_l[]

START

MKEYSA(0, 0.000000F, 0.000000F, 82.386131F),

... (省略)

MKEYSA(120, 0.000000F, 0.000000F, 82.386147F),

END

SRotation rot_same_heavywalk_body_3_foot_l[]

START

MKEYSA(0, 0.000000F, 0.029117F, -115.490250F),

... (省略)

MKEYSA(120, 0.000000F, 0.029113F, -115.490280F),

END

POSITION pos_same_heavywalk_body_3_item4[]

START

MKEYF(0, -3.018911F, 0.000720F, 0.000001F),

MKEYF(120, -3.018911F, 0.000719F, 0.000001F),

END

MData4 mdata_same_heavywalk_body_3[]

START

pos_same_heavywalk_body_3_body_3, NULL, NULL, NULL, 119, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, rot_same_heavywalk_body_3_mune, scl_same_heavywalk_body_3_mune, NULL, 0, 121, 113, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, rot_same_heavywalk_body_3_kata_r_1, NULL, NULL, 0, 121, 0, 0,

NULL, rot_same_heavywalk_body_3_ude_r_1, NULL, NULL, 0, 121, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, rot_same_heavywalk_body_3_jnt22_1_1, NULL, NULL, 0, 121, 0, 0,

NULL, rot_same_heavywalk_body_3_jnt22_2_1, NULL, NULL, 0, 121, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_jnt24_1, scl_same_heavywalk_body_3_jnt24_1, NULL, 0, 119, 120, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, scl_same_heavywalk_body_3_eyes, NULL, 0, 0, 119, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, scl_same_heavywalk_body_3_mabuta, NULL, 0, 0, 119, 0,
 NULL, rot_same_heavywalk_body_3_mabuta_r, NULL, NULL, 0, 121, 0, 0,
 NULL, rot_same_heavywalk_body_3_mabuta_l, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_kosi, NULL, NULL, 0, 112, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_momo_r, NULL, NULL, 0, 118, 0, 0,
 NULL, rot_same_heavywalk_body_3_sune_r, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_foot_r, NULL, event_same_heavywalk_body_3_foot_r, 0, 115, 0, 4,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 pos_same_heavywalk_body_3_item3, NULL, NULL, NULL, 2, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_momo_l, NULL, NULL, 0, 110, 0, 0,
 NULL, rot_same_heavywalk_body_3_sune_l, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_foot_l, NULL, NULL, 0, 107, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 pos_same_heavywalk_body_3_item4, NULL, NULL, NULL, 2, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,

END

MOTION motion_same_heavywalk_body_3[]

START

MdataArray mdata_same_heavywalk_body_3,

```

MFrameNum      121,
MotionBitF      (FMK_POS0|FMK_SAN1|FMK_SCA2|FMK_EVE4),
InterpolFctF    (FMT_L|FMD_4),
END

MOTION_END

DEFAULT_START

#ifdef DEFAULT_MOTION_NAME
#define DEFAULT_MOTION_NAME motion_same_heavywalk_body_3
#endif

DEFAULT_END

```

3.3.2 出力例の解説

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

NAMファイルであることをバージョン番号、Ninja2用アスキーモーションであることを示す。SIはSoftimageから生成されたことを示す。

```
/* MOTION : motion_same_heavywalk_body_3 */
```

モーション構造体変数名。

```
MOTION_START
```

モーションアスキーチャンク開始。

```

POSITION pos_same_heavywalk_body_3_body_3[]
START
    MKEYF( 0,    0.000000F, -0.978237F,  0.058765F ),
    ... ( 省略 )
    MKEYF( 120,  0.000000F, -0.978237F,  0.058765F ),
END

```

移動成分 (T) キーデータ。

```

SRROTATION rot_same_heavywalk_body_3_mune[]
START

```

```

MKEYSA( 0,   -0.000278F,  -0.000212F, 143.120361F),
... (省略)
MKEYSA( 120,  -0.000278F,  -0.000212F, 143.120361F),
END

```

回転成分 (T) キーデータ。回転表現にShortAngleを使用している。

```

SCALE scl_same_heavywalk_body_3_mune[]
START
MKEYF( 0,   1.000000F,  1.000000F,  1.000000F ),
... (省略)
MKEYF( 120,  1.000000F,  1.000000F,  1.000000F ),
END

```

スケール (S) キーデータ。省略部分にスケール変化が含まれる。

```

SROTATION rot_same_heavywalk_body_3_kata_r_1[]
START
MKEYSA( 0,   -0.000001F, -23.736517F, 205.930191F),
... (省略)
MKEYSA( 120,  -0.000001F, -23.736517F, 205.930191F),
END

```

```

SROTATION rot_same_heavywalk_body_3_ude_r_1[]
START
MKEYSA( 0,   0.000000F, -21.839205F,  7.829935F),
... (省略)
MKEYSA( 120,  0.000000F, -21.839205F,  7.829935F),
END

```

```

SROTATION rot_same_heavywalk_body_3_jnt22_1_1[]
START
MKEYSA( 0,  -180.000000F, -203.736511F,  25.930185F),
... (省略)
MKEYSA( 120,  -180.000000F, -203.736511F,  25.930185F),
END

```

```

SROTATION rot_same_heavywalk_body_3_jnt22_2_1[]
START
MKEYSA( 0,   0.000000F,  21.839220F,  7.829945F),
... (省略)

```

```
        MKEYSA( 120,    0.000000F,  21.839220F,   7.829945F),  
END
```

```
SROTATION rot_same_heavywalk_body_3_jnt24_1[]  
START  
        MKEYSA( 0,    0.000101F,   0.000243F, 181.664856F),  
        ... (省略)  
        MKEYSA( 120,   -0.000124F,   0.000177F, 181.664856F),  
END
```

```
SCALE scl_same_heavywalk_body_3_jnt24_1[]  
START  
        MKEYF( 0,    0.778696F,   1.000000F,   1.000000F ),  
        ... (省略)  
        MKEYF( 120,   0.778696F,   1.000000F,   1.000000F ),  
END
```

```
SCALE scl_same_heavywalk_body_3_eyes[]  
START  
        MKEYF( 0,    1.000000F,   1.000000F,   1.000000F ),  
        ... (省略)  
        MKEYF( 120,   1.000000F,   1.000000F,   1.000000F ),  
END
```

```
SCALE scl_same_heavywalk_body_3_mabuta[]  
START  
        MKEYF( 0,    1.000000F,   1.000000F,   1.000000F ),  
        ... (省略)  
        MKEYF( 120,   1.000000F,   1.000000F,   1.000000F ),  
END
```

```
SROTATION rot_same_heavywalk_body_3_mabuta_r[]  
START  
        MKEYSA( 0,   -82.999992F,  -0.000001F,   0.000000F),  
        ... (省略)  
        MKEYSA( 120,  -82.999992F,  -0.000001F,   0.000000F),  
END
```

```
SROTATION rot_same_heavywalk_body_3_mabuta_l[]  
START  
        MKEYSA( 0,   -82.999992F,  -0.000001F,   0.000000F),
```

```

... (省略)
MKEYSA( 120,  -82.999992F,  -0.000001F,   0.000000F),
END

```

```

SROTATION rot_same_heavywalk_body_3_kosi[]
START
    MKEYA( 0,    0.001969F,   0.012202F, -80.812225F),
    ... (省略)
    MKEYA( 120,   0.001969F,   0.012202F, -80.812225F),
END

```

```

SROTATION rot_same_heavywalk_body_3_momo_r[]
START
    MKEYSA( 0,   -0.002560F,   0.002578F, -96.571671F),
    ... (省略)
    MKEYSA( 120,  -0.002566F,   0.002584F, -96.571640F),
END

```

```

SROTATION rot_same_heavywalk_body_3_sune_r[]
START
    MKEYSA( 0,    0.000000F,   0.000000F,  87.133835F),
    ... (省略)
    MKEYSA( 120,   0.000000F,   0.000000F,  87.133827F),
END

```

```

SROTATION rot_same_heavywalk_body_3_foot_r[]
START
    MKEYSA( 0,    0.000000F,   0.001768F, -214.560989F),
    ... (省略)
    MKEYSA( 120,   0.000000F,   0.001764F, -214.560989F),
END

```

```

EVENTBITS event_same_heavywalk_body_3_foot_r[]
START
    MKEYUI16( 0,    FEM_NONE),
    MKEYUI16( 10,   FEM_HD),
    MKEYUI16( 60,   FEM_NONE),
    MKEYUI16( 120,  FEM_NONE),
END

```

イベントモーション（ハイドモーション）データ。keyframe=10からkeyframe=60までモデルがハイドすることを示す。ライ

ブラリの構造を性能優先で単純化するため最初と最後のフレームは省略できない。

SPOSITION pos_same_heavywalk_body_3_item3[]

START

MKEYF(0, -3.021368F, 0.001376F, 0.000000F),

MKEYF(120, -3.021368F, 0.001376F, 0.000000F),

END

SROTATION rot_same_heavywalk_body_3_momo_l[]

START

MKEYSA(0, 0.027253F, -0.029918F, -190.344116F),

... (省略)

MKEYSA(120, 0.027252F, -0.029916F, -190.344116F),

END

SROTATION rot_same_heavywalk_body_3_sune_l[]

START

MKEYSA(0, 0.000000F, 0.000000F, 82.386131F),

... (省略)

MKEYSA(120, 0.000000F, 0.000000F, 82.386147F),

END

SROTATION rot_same_heavywalk_body_3_foot_l[]

START

MKEYSA(0, 0.000000F, 0.029117F, -115.490250F),

... (省略)

MKEYSA(120, 0.000000F, 0.029113F, -115.490280F),

END

POSITION pos_same_heavywalk_body_3_item4[]

START

MKEYF(0, -3.018911F, 0.000720F, 0.000001F),

MKEYF(120, -3.018911F, 0.000719F, 0.000001F),

END

MDATA4 mdata_same_heavywalk_body_3[]

START

pos_same_heavywalk_body_3_body_3, NULL, NULL, NULL, 119, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,

NULL, rot_same_heavywalk_body_3_mune, scl_same_heavywalk_body_3_mune, NULL, 0, 121, 113, 0,

NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_kata_r_1, NULL, NULL, 0, 121, 0, 0,
 NULL, rot_same_heavywalk_body_3_ude_r_1, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_jnt22_1_1, NULL, NULL, 0, 121, 0, 0,
 NULL, rot_same_heavywalk_body_3_jnt22_2_1, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_jnt24_1, scl_same_heavywalk_body_3_jnt24_1, NULL, 0, 119, 120, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, scl_same_heavywalk_body_3_eyes, NULL, 0, 0, 119, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, scl_same_heavywalk_body_3_mabuta, NULL, 0, 0, 119, 0,
 NULL, rot_same_heavywalk_body_3_mabuta_r, NULL, NULL, 0, 121, 0, 0,
 NULL, rot_same_heavywalk_body_3_mabuta_l, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_kosi, NULL, NULL, 0, 112, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_momo_r, NULL, NULL, 0, 118, 0, 0,
 NULL, rot_same_heavywalk_body_3_sune_r, NULL, NULL, 0, 121, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, rot_same_heavywalk_body_3_foot_r, NULL, event_same_heavywalk_body_3_foot_r, 0, 115, 0, 4, イベント
 モーションあり。
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 pos_same_heavywalk_body_3_item3, NULL, NULL, NULL, 2, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,
 NULL, NULL, NULL, NULL, 0, 0, 0, 0,

```

NULL, rot_same_heavywalk_body_3_momo_l, NULL, NULL, 0, 110, 0, 0,
NULL, rot_same_heavywalk_body_3_sune_l, NULL, NULL, 0, 121, 0, 0,
NULL, NULL, NULL, NULL, 0, 0, 0, 0,
NULL, NULL, NULL, NULL, 0, 0, 0, 0,
NULL, rot_same_heavywalk_body_3_foot_l, NULL, NULL, 0, 107, 0, 0,
NULL, NULL, NULL, NULL, 0, 0, 0, 0,
pos_same_heavywalk_body_3_item4, NULL, NULL, NULL, 2, 0, 0, 0,
NULL, NULL, NULL, NULL, 0, 0, 0, 0,
NULL, NULL, NULL, NULL, 0, 0, 0, 0,

```

END

モデル階層、もしくはmrsとの比較により差分がない（モーションが無い部分）に関してはすべてNULLで表現されている。第四要素としてイベントモーションを一個所だけ持ちそのキーの数は4つ。

MOTION motion_same_heavywalk_body_3[]

START

MdataArray mdata_same_heavywalk_body_3,

MFrameNum 121,

モーションのフレーム数。

MotionBitF (FMK_POS0|FMK_SAN1|FMK_SCA2|FMK_EVE4),

要素として移動（T） ShortAngle回転（SA） スケール（S） イベント（E）がT,SA,S,Eの順番で格納されることを示す。

InterpolFctF (FMT_L|FMD_4),

モーションがリニアで保管され要素数が4つであることを示す。

END

MOTION_END

アスキーチャンクの終了を示す。

DEFAULT_START

#ifndef DEFAULT_MOTION_NAME

#define DEFAULT_MOTION_NAME motion_same_heavywalk_body_3

#endif

DEFAULT_END

4. カメラおよびカメラモーション

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

4.1 カメラモーション解説

カメラには

位置(Pos)

方向(Vec) またはターゲット(Target)

ロール(Roll)

画角(ShortAngle)

の4要素があり、必要に応じて NJS_MDATA_1~4 を使用する。

```
#define NJD_MTYPE_POS_0      (1<<0)   /* NJS_MKEY_F を利用 */
#define NJD_MTYPE_VEC_3      (1<<3)   /* NJS_MKEY_F を利用 */
#define NJD_MTYPE_TARGET_3   (1<<6)   /* NJS_MKEY_F を利用 */
#define NJD_MTYPE_ROLL_6     (1<<7)   /* NJS_MKEY_SA1 を利用 */
#define NJD_MTYPE_ANGLE_7    (1<<8)   /* NJS_MKEY_SA1 を利用 */
```

Vec はベクトル、Target はターゲットのポジションを意味する。

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

フリーカメラ(向きをベクトルで持つカメラ)の場合

type=NJD_MTYPE_POS_0|NJD_MTYPE_VEC_3|NJD_MTYPE_ROLL_6|NJD_MTYPE_ANGLE_7;

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

type=NJD_MTYPE_POS_0|NJD_MTYPE_TARGET_3|NJD_MTYPE_ROLL_6|NJD_MTYPE_ANGLE_7;

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

4.2 カメラアスキーフォーマットマクロ

カメラ定義に関わるマクロを示す。

CAMERA	カメラデータを意味する。
CAMERA_START	カメラアスキーチャンクの開始を意味する。
CAMERA_END	カメラアスキーチャンクの終了を意味する。
CPosition	カメラのポジションデータ。
CVector	カメラの方向データ。
CTarget	カメラの注視点データ。
CRoll	カメラのロールデータ。
CAngle	カメラの画角データ。
CType	カメラのタイプ与える。Vector か Target の二つ。

```
#define CAMERA          NJS_CAMERA
```

```

#define CAMERA_START
#define CAMERA_END
#define CPosition(_x, _y, _z)    (_x), (_y), (_z)
#define CVector(_x, _y, _z)     (_x), (_y), (_z)
#define CTarget(_x, _y, _z)     (_x), (_y), (_z)
#define CRoll(_ang)              ((Sangle)(_KEYVAL*(_ang)))
#define CAngle(_ang)             ((Sangle)(_KEYVAL*(_ang)))
#define CType(_t)                (_t)

```

CVector と CTarget はカメラタイプで使い分ける。

FCT_VECTOR	VECTOR タイプのカメラを指定する。
FCT_TARGET	TARGET タイプのカメラを指定する。

```

/* Flag Camera Type */
#define FCT_VECTOR          NJD_CTYPE_VECTOR
#define FCT_TARGET          NJD_CTYPE_TARGET

```

次にカメラムーブメントに関わるマクロを示す。

CAMERA_MOTION_START	カメラムーブメントアスキーチャンクの開始を意味する。
CAMERA_MOTION_END	カメラムーブメントアスキーチャンクの終了を意味する。
CPOSITION	カメラの移動成分データを意味する。
CVECTOR	カメラの方向成分データを意味する。
CTARGET	カメラの注視点成分データを意味する。
CROLL	カメラのロール成分データを意味する。
CANGLE	カメラの画角成分データを意味する。

```

#define CAMERA_MOTION_START
#define CAMERA_MOTION_END

#define CPOSITION          NJS_MKEY_F
#define CVECTOR            NJS_MKEY_F
#define CTARGET            NJS_MKEY_F
#define CROLL              NJS_MKEY_SA1
#define CANGLE             NJS_MKEY_SA1

```

CVECTOR と CTARGET はカメラタイプで使い分ける。

4.3 カメラアスキーフォーマット出力例

4.3.1 カメラ (nac ファイル) の出力例(VECTOR)

```
/* NAC 2.00.00 Ninja2AsciiCamera (SI) */
```

```
CAMERA_START
```

```
CAMERA    s_untei02_cam_int1[]
```

```
START
```

```
CPosition  ( -6.796906F,  2.220694F,  3.216255F ),
```

```
CVector    (  0.862647F, -0.363019F, -0.352218F ),
```

```
CRoll      (  0.000000F ),
```

```
CAngle     ( 41.539447F ),
```

```
CType      ( FCT_VECTOR ),  
END
```

CAMERA_END

< 解説 >

```
/* NAC 2.00.00 Ninja2AsciiCamera (SI) */
```

Ninja2 用カメラデータであることを示す。SI はデータが Softimage から出力されていることを示す。

CAMERA_START

アスキーカメラチャンクの開始を意味する。

```
CAMERA      s_untei02_cam_int1[]  
START  
CPosition   ( -6.796906F,  2.220694F,  3.216255F ),  位置を示す。  
CVector     (  0.862647F, -0.363019F, -0.352218F ),  方向を示す。  
CRoll       (  0.000000F ),  ロールを示す。  
CAngle      ( 41.539447F ),  画角を示す。  
CType       ( FCT_VECTOR ),  カメラがベクタータイプであることを示す。  
END
```

CAMERA_END

アスキーカメラチャンクの終了を示す。

4.3.2 カメラ(nac ファイル)の出力例(TARGET)

```
/* NAC 2.00.00 Ninja2AsciiCamera (SI) */
```

CAMERA_START

```
CAMERA      s_untei02_cam_int1[]  
START  
CPosition   ( -6.796906F,  2.220694F,  3.216255F ),  
CTarget     ( -5.796906F,  3.220694F,  4.216255F ),  
CRoll       (  0.000000F ),  
CAngle      ( 41.539447F ),  
CType       ( FCT_TARGET ),
```

END

CAMERA_END

解説省略。

4.3.3 カメラモーションの出力例(TARGET)

ここではベクトルタイプのカメラモーションの例を示す。

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

```
/* CAMERA MOTION : motion_s_untei02_cam_int1 */
```

CAMERA_MOTION_START

CPOSITION pos_s_untei02_cam_int1[]

START

MKEYF(0, -6.796906F, 2.220694F, 3.216255F),

... (省略)

MKEYF(62, -19.958214F, 14.943303F, 3.216255F),

END

CTARGET tgt_s_untei02_cam_int1[]

START

MKEYF(0, -1.582013F, 0.026164F, 1.087016F),

... (省略)

MKEYF(62, -1.582013F, 0.026164F, 1.087016F),

END

CROLL roll_s_untei02_cam_int1[]

START

MKEYA1(0, 0.000000F),

MKEYA1(62, 0.000000F),

END

CANGLE angle_s_untei02_cam_int1[]

START

MKEYA1(0, 41.539447F),

MKEYA1(62, 41.539447F),

END

```

MDATA4 mdata_s_untei02_cam_int1[]
START
    pos_s_untei02_cam_int1, tgt_s_untei02_cam_int1, roll_s_untei02_cam_int1, angle_s_untei02_cam_int1, 63, 59,
    2, 2,
END

```

```

MOTION motion_s_untei02_cam_int1[]
START
MdataArray    mdata_s_untei02_cam_int1,
MFrameNum     63,
MotionBitF    (FMK_POS0|FMK_VEC3|FMK_ROL6|FMK_ANG7),
InterpolFctF  (FMT_L|FMD_4),
END

```

CAMERA_MOTION_END

DEFAULT_START

```

#ifndef DEFAULT_CMOTION_NAME
#define DEFAULT_CMOTION_NAME motion_s_untei02_cam_int1
#endif

```

DEFAULT_END

< 解説 >

```

/* NAM 2.00.00 Ninja2AsciiMotion (SI) */

```

Ninja2モーションデータであることを示す。SIはSoftimageから出力されたことを示す。

```

/* CAMERA MOTION : motion_s_untei02_cam_int1 */

```

モーション構造体の変数名を示す。

CAMERA_MOTION_START

アスキーモーションチャンクの始まりを示す。

```

CPOSITION pos_s_untei02_cam_int1[]
START
    MKEYF( 0, -6.796906F, 2.220694F, 3.216255F ),

```



```
... (省略)
MKEYF( 62, -19.958214F, 14.943303F, 3.216255F ),
END
```

移動成分 (T) モーションデータ。

```
CTARGET tgt_s_untei02_cam_int1[]
START
MKEYF( 0, -1.582013F, 0.026164F, 1.087016F ),
... (省略)
MKEYF( 62, -1.582013F, 0.026164F, 1.087016F ),
END
```

ターゲットポジションデータ。

```
CROLL roll_s_untei02_cam_int1[]
START
MKEYA1( 0, 0.000000F ),
MKEYA1( 62, 0.000000F ),
END
```

ロールデータ。

```
CANGLE angle_s_untei02_cam_int1[]
START
MKEYA1( 0, 41.539447F ),
MKEYA1( 62, 41.539447F ),
END
```

画角データ。

```
MDATA4 mdata_s_untei02_cam_int1[]
START
pos_s_untei02_cam_int1, tgt_s_untei02_cam_int1, roll_s_untei02_cam_int1, angle_s_untei02_cam_int1, 63, 59,
2, 2,
END
```

カメラは階層を持たないためエントリは一つのみ。各成分が順に並びキーの数が後に列記される。

```
MOTION motion_s_untei02_cam_int1[]
START
```

MdataArray mdata_s_untei02_cam_int1,

MFrameNum 63,

 モーションのフレーム数。

MotionBitF (FMK_POS0|FMK_TAR3|FMK_ROL6|FMK_ANG7),

 モーション成分に移動成分、ターゲット成分、ロール、画角が含まれることを示す。

InterpolFctF (FMT_L|FMD_4),

 リニア補完で要素数が4つであることを示す。

END

CAMERA_MOTION_END

 アスキーモーションチャンクの終わりを示す。

DEFAULT_START

#ifndef DEFAULT_CMOTION_NAME

#define DEFAULT_CMOTION_NAME motion_s_untei02_cam_int1

#endif

DEFAULT_END

5. ライトおよびライトモーション

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

5.1 ライトモーション解説

ライトモーションは、点光源、平行光源をサポートする。Ninja2 ではスポットライトをサポートしない。Ninja2 は描画関数ごとにライトが異なる。描画関数の種類は光源一つの EasyDraw, SimpleDraw と光源 7 つまでサポートの EasyMultiDraw, SimpleMultiDraw の四種類。

<EasyDraw および SimpleDraw>

使用できるのは平行光源一つのみ。モーション要素は次の 3 要素がある。

方向(Vec)

色(RGB)

光の強度(Intensity+Ambient)。

<EasyMultiDraw 及び SimpleMultiDraw>

平行光源のモーション要素数は 2 要素。f1, f2 は無効でありモーション対象とならない。

方向(Vec)

色(RGB)

点光源のモーション要素数は 3 要素。f1, f2 には Point(nrange, frange) が格納される。

位置(Pos)

範囲(Point)

色(RGB)

アンビエントライトのモーション要素数は 1 要素。アンビエントライトは同時に一つしか使えない。モデルによってはモーションを取り出せない場合がある。

光源では NJS_MDATA_1 ~ 3 を使用する。

```
#define NJD_MTYPE_POS_0          (BIT_0)      /* NJS_MKEY_F を利用 */
#define NJD_MTYPE_VEC_0          (BIT_4)      /* NJS_MKEY_F 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 */
```

Type の設定例

EasyDraw 平行光源

```
type=NJD_MTYPE_VEC_0|NJD_MTYPE_RGB_8|NJD_MTYPE_INTENSITY_9;
```

EasyMultiDraw 点光源の場合

```
type=NJD_MTYPE_POS_0|NJD_MTYPE_RGB_8|NJD_MTYPE_POINT_9;
```

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

5.2 ライトアスキーフォーマットマクロ

ライト定義に関わるマクロ

LIGHT_START	ライトアスキーチャンクの開始を意味する。
LIGHT_END	ライトアスキーチャンクの終了を意味する。
LVector	ライトの方向成分データ。
LPosition	ライトの位置成分データ。
LColor	ライトのカラー成分データ。
LIntensity	ライトの強度成分データ。
LAmbient	ライトのアンビエント強度成分データ。
LNRange	ライトの手前の影響範囲。
LFRange	ライトの遠方の影響範囲。
LFunc	描画関数の種類。
LType	ライトの種類。平行光源 (Vector) 点光源 (Point) 環境光 (Ambient)
LNoopF	このパラメータが無効であることを明示する。3 変数。
LNoopF1	このパラメータが無効であることを明示する。1 変数。

```
#define LIGHT_START
#define LIGHT_END
```

```
#define LVector( _x, _y, _z)      (_x), (_y), (_z)
#define LPosition( _x, _y, _z)   (_x), (_y), (_z)
#define LColor( _x, _y, _z)      (_x), (_y), (_z)
#define LIntensity(_f)           (_f)           /* float */
#define LAmbient(_f)             (_f)           /* float */
#define LNRange(_f)              (_f)           /* float */
#define LFRange(_f)              (_f)           /* float */
#define LFunc(_t)                (_t)           /* func type */
#define LType(_t)                (_t)           /* light type */
#define LNoopF( _x, _y, _z)      (_x), (_y), (_z)
#define LNoopF1( _f)             (_f)
```

FLF_EASY	ライトが EasyDraw 用であることを示す。
FLF_SIMPLE	ライトが SimpleDraw 用であることを示す。
FLF_EASY_MULT I	ライトが EasyMultiDraw 用であることを示す。
FLF_SIMPLE_MULT I	ライトが SimpleMultiDraw 用であることを示す。

```
/* Flag Light Func */
#define FLF_EASY      NJD_MFUNC_EASY
#define FLF_SIMPLE    NJD_MFUNC_SIMPLE
#define FLF_EASY_MULT I  NJD_MFUNC_EASY_MULT I
#define FLF_SIMPLE_MULT I NJD_MFUNC_SIMPLE_MULT I
```

FLT_POINT	点光源データであることを示す。
FLT_VECTOR	平行光源データであることを示す。
FLT_AMBIENT	環境光光源であることを示す。

```
/* Flag Light Type */
#define FLT_POINT      NJD_LTYPE_POINT
#define FLT_VECTOR      NJD_LTYPE_VECTOR
#define FLT_AMBIENT     NJD_LTYPE_AMBIENT
```

ライトモーション定義に関わるマクロを示す。

LIGHT_MOTION_START	ライトモーションアスキーチャンクの開始を意味する。
LIGHT_MOTION_END	ライトモーションアスキーチャンクの終了を意味する。
LPOSITION	ライトの移動成分データを意味する。
LVECTOR	ライトの方向成分データを意味する。
LColor	ライトカラー成分データを意味する。

LINTENSITY	光源強度を意味する。環境光の強度と合わせて扱われる。
LPOINT_FACTORS	光源の影響範囲指定データであることを意味する。f1 が手前の限界位置 (nrange) , f2 が遠方の限界位置 (frange)。

```
#define LIGHT_MOTION_START
#define LIGHT_MOTION_END
```

```
#define LPOSITION          NJS_MKEY_F
#define LVECTOR            NJS_MKEY_F
#define LCOLOR             NJS_MKEY_F
#define LINTENSITY         NJS_MKEY_F2
#define LPOINT_FACTORS    NJS_MKEY_F2
```

5.3 ライトアスキーフォーマット出力例

5.3.1 ライト (nal ファイル) の出力例(EasyDraw の平行光源)

```
/* NAL 2.00.00 NinjaAsciiLight (SI) */
```

```
/* LFunc: EasyDraw LType: Vector */
```

```
LIGHT_START
```

```
LIGHT    s_untei02__inf_light1[]
```

```
START
```

```
LVector    ( 0.880934F, -0.341369F, -0.327754F ),
```

```
LColor     ( 1.000000F, 0.750000F, 1.000000F ),
```

```
LIntensity ( 0.600000F ),
```

```
LAmbient   ( 0.500000F ),
```

```
LFunc      ( FLF_EASY ),
```

```
LType      ( FLT_VECTOR ),
```

```
END
```

```
LIGHT_END
```

< 解説 >

```
/* NAL 2.00.00 NinjaAsciiLight (SI) */
```

Ninja2用ライトデータであることを示す。SIはSoftimageからのデータであることを示す。

```
/* LFunc: EasyDraw LType: Vector */
```

対応するDraw関数名および光源の種類を示す。描画関数タイプがSimpleDrawの場合LFuncのコメントがSimpleDrawに変更される。EasyDrawとSimpleDrawではそれ以外の点は変わらない。

```
LIGHT_START
```

アスキーライトチャンクの始まりを示す。

```
LIGHT    s_untei02__inf_light1[]
```

```
START
```

```
LVector    ( 0.880934F, -0.341369F, -0.327754F ),
```

平行光源の時LVector、点光源の時にLPositionが使われる。

```
LColor      ( 1.000000F, 0.750000F, 1.000000F ),  
LIntensity  ( 0.600000F ),  
LAmbient    ( 0.500000F ),
```

Intensityは色に掛け算されAmbientはそれに足し込まれる。

```
LFunc      ( FLF_EASY ),
```

EasyDraw関数用のライトであることを示す。

```
LType      ( FLT_VECTOR ),
```

平行光源であることを示す。

```
END
```

```
LIGHT_END
```

アスキーライトチャンクの終了を示す。

5.3.2 ライト (nal ファイル) の出力例(SimpleDraw の平行光源)

```
/* NAL 2.00.00 NinjaAsciiLight (SI) */
```

```
/* LFunc: SimpleDraw LType: Vector */
```

```
LIGHT_START
```

```
LIGHT    s_untei02__inf_light1[]
```

```
START
```

```
LVector   ( 0.880934F, -0.341369F, -0.327754F ),
```

```
LColor     ( 1.000000F, 0.750000F, 1.000000F ),
```

```
LIntensity ( 0.600000F ),
```

```
LAmbient   ( 0.500000F ),
```

```
LFunc      ( FLF_SIMPLE ),
```

```
LType      ( FLT_VECTOR ),
```

```
END
```

```
LIGHT_END
```

解説省略。

5.3.3 ライト (nal ファイル) の出力例(EasyMultiDraw の平行光源)

```
/* NAL 2.00.00 Ninja2AsciiLight (SI) */
```

```
/* LFunc: EasyMultiDraw LType: Vector */
```

LIGHT_START

```
LIGHT    s_untei02__inf_light1[]
START
LVector   ( 0.880934F, -0.341369F, -0.327754F ),
LColor    ( 1.000000F, 0.750000F, 1.000000F ),
LNoopF1    ( 0.000000F ),
LNoopF1    ( 0.000000F ),
LFunc     ( FLF_EASY_MULTI ),
LType     ( FLT_VECTOR ),
END
```

LIGHT_END

LNoopF1はこのパラメータが無効であることを意味する。

5.3.4 ライト (nal ファイル) の出力例(EasyMultiDraw の点光源)

```
/* NAL 2.00.00 Ninja2AsciiLight (SI) */
```

```
/* LFunc: EasyMultiDraw LType: Point */
```

LIGHT_START

```
LIGHT    s_untei02__inf_light1[]
START
LPosition ( -8.538586F, 3.308772F, 3.176800F ),
LColor    ( 1.000000F, 0.750000F, 1.000000F ),
LNRange   ( 1.000000F ),
LFRange   ( 20.000000F ),
LFunc     ( FLF_EASY_MULTI ),
LType     ( FLT_POINT ),
END
```

LIGHT_END

解説省略。

5.3.5 ライト (nal ファイル) の出力例(EasyMultiDraw のアンビエントライト)


```
/* NAL 2.00.00 Ninja2AsciiLight (SI) */
```

```
/* LFunc: EasyMultiDraw LType: Ambient */
```

```
LIGHT_START
```

```
LIGHT    s_untei02__inf_light1[]
```

```
START
```

```
LNoopF    ( 0.0F, 0.0F, 0.0F ),
```

```
LColor    ( 1.000000F, 0.750000F, 1.000000F ),
```

```
LNoopF1    ( 0.000000F ),
```

```
LNoopF1    ( 0.000000F ),
```

```
LFunc      ( FLF_EASY_MULT I ),
```

```
LType      ( FLT_AMBIENT ),
```

```
END
```

```
LIGHT_END
```

LNoopF, LNoopF1はそのパラメータが無効であることを示す。

5.3.6 ライト (nal ファイル) の出力例(SimpleMultiDraw の平行光源)

```
/* NAL 2.00.00 Ninja2AsciiLight (SI) */
```

```
/* LFunc: SimpleMultiDraw LType: Vector */
```

```
LIGHT_START
```

```
LIGHT    s_untei02__inf_light1[]
```

```
START
```

```
LVector    ( 0.880934F, -0.341369F, -0.327754F ),
```

```
LColor    ( 1.000000F, 0.750000F, 1.000000F ),
```

```
LNoopF1    ( 0.000000F ),
```

```
LNoopF1    ( 0.000000F ),
```

```
LFunc      ( FLF_SIMPLE_MULT I ),
```

```
LType      ( FLT_VECTOR ),
```

```
END
```

```
LIGHT_END
```

解説省略。

5.3.7 ライト (nal ファイル) の出力例(SimpleMultiDraw の点光源)

```
/* NAL 2.00.00 Ninja2AsciiLight (SI) */

/* LFunc: SimpleMultiDraw LType: Point */

LIGHT_START

LIGHT    s_untei02__inf_light1[]
START
LPosition    ( -8.538586F,  3.308772F,  3.176800F ),
LColor       ( 1.000000F, 0.750000F, 1.000000F ),
LNRange      ( 1.000000F ),
LFRange      ( 20.000000F ),
LFunc        ( FLF_SIMPLE_MULTI ),
LType        ( FLT_POINT ),
END

LIGHT_END
```

解説省略。

5.3.8 ライト (nal ファイル) の出力例(SimpleMultiDraw のアンビエントライト)

```
/* NAL 2.00.00 Ninja2AsciiLight (SI) */

/* LFunc: SimpleMultiDraw LType: Ambient */

LIGHT_START

LIGHT    s_untei02__inf_light1[]
START
LNoopF      ( 0.0F,  0.0F,  0.0F ),
LColor       ( 1.000000F, 0.750000F, 1.000000F ),
LNoopF1      ( 0.000000F ),
LNoopF1      ( 0.000000F ),
LFunc        ( FLF_SIMPLE_MULTI ),
LType        ( FLT_AMBIENT ),
END
```

LIGHT_END

解説省略。

5.3.9 ライトモーションの出力例(EasyDraw および SimpleDraw の平行光源)

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

```
/* LIGHT MOTION : motion_s_untei02_s_kihon2_light6 */
```

```
/* LFunc: EasyDraw LType: Vector */
```

LIGHT_MOTION_START

```
LVECTOR vec_s_untei02_s_kihon2_light6[]
```

START

```
  MKEYF( 0,  0.880934F, -0.341369F, -0.327754F ),
```

```
  ... (省略)
```

```
  MKEYF( 62,  0.880934F, -0.341369F, -0.327754F ),
```

END

```
LCOLOR col_s_untei02_s_kihon2_light6[]
```

START

```
  MKEYF( 0,  1.000000F, 0.750000F, 1.000000F),
```

```
  MKEYF( 62,  1.000000F, 0.750000F, 1.000000F),
```

END

```
LINTENSITY int_s_untei02_s_kihon2_light6[]
```

START

```
  MKEYF2( 0,  1.000000F, 1.000000F),
```

```
  MKEYF2( 62,  1.000000F, 1.000000F),
```

END

```
MDATA3 mdata_s_untei02_s_kihon2_light6[]
```

START

```
  vec_s_untei02_s_kihon2_light6, col_s_untei02_s_kihon2_light6, int_s_untei02_s_kihon2_light6, 4, 2, 2,
```

END

```
MOTION motion_s_untei02_s_kihon2_light6[]
```

START

```
MdataArray    mdata_s_untei02_s_kihon2_light6,
MFrameNum     63,
MotionBitF    (FMK_VEC0|FMK_RGB8|FMK_INT9),
InterpolFctF  (FMT_L|FMD_3),
END
```

```
LIGHT_MOTION_END
```

```
DEFAULT_START
```

```
#ifndef DEFAULT_LMOTION_NAME
#define DEFAULT_LMOTION_NAME motion_s_untei02_s_kihon2_light6
#endif
```

```
DEFAULT_END
```

< 解説 >

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

Ninja2用モーションデータであることを示す。SIはSoftimageからのデータであることを示す。

```
/* LIGHT MOTION : motion_s_untei02_s_kihon2_light6 */
```

モーションの変数名を示す。

```
/* LFunc: EasyDraw LType: Vector */
```

対応するDraw関数名および光源の種類を示す。描画関数タイプがSimpleDrawの場合LFuncのコメントがSimpleDrawに変更される。EasyDrawとSimpleDrawではそれ以外の点は変わらない。

```
LIGHT_MOTION_START
```

アスキーモーションチャンクの始まりを示す。

```
LVECTOR vec_s_untei02_s_kihon2_light6[]
```

```
START
```

```
    MKEYF( 0,  0.880934F, -0.341369F, -0.327754F ),
```

```
    ... (省略)
```

```
    MKEYF( 62, 0.880934F, -0.341369F, -0.327754F ),
```

```
END
```

Vector成分。

```
LCOLOR col_s_untei02_s_kihon2_light6[]  
START  
    MKEYF( 0,  1.000000F, 0.750000F, 1.000000F),  
    MKEYF( 62, 1.000000F, 0.750000F, 1.000000F),  
END
```

RGBカラー成分。

```
LINTENSITY int_s_untei02_s_kihon2_light6[]  
START  
    MKEYF2( 0,  1.000000F, 1.000000F),  
    MKEYF2( 62, 1.000000F, 1.000000F),  
END
```

Intensity+ambient成分。

```
MDATA3 mdata_s_untei02_s_kihon2_light6[]  
START  
    vec_s_untei02_s_kihon2_light6, col_s_untei02_s_kihon2_light6, int_s_untei02_s_kihon2_light6, 4, 2, 2,  
END
```

ライトは階層を持たないためエントリは一つのみ。各成分が順に並びキーの数が後に列記される。

```
MOTION motion_s_untei02_s_kihon2_light6[]  
START  
MdataArray      mdata_s_untei02_s_kihon2_light6,  
MFrameNum       63,  
    モーションのフレーム数。
```

```
MotionBitF      (FMK_VEC0|FMK_RGB8|FMK_INT9),  
    モーションの成分にベクトル、RGB、Intensity+Ambientが含まれることを示す。
```

```
InterpolFctF    (FMT_L|FMD_3),  
    リニア補完で要素数が3つであることを示す。
```

END

LIGHT_MOTION_END

アスキーモーションチャンクの終了を示す。

DEFAULT_START

```
#ifndef DEFAULT_LMOTION_NAME
#define DEFAULT_LMOTION_NAME motion_s_untei02_s_kihon2_light6
#endif
```

DEFAULT_END

5.3.10 ライトモーションの出力例(EasyMultiDraw および SimpleMultiDraw の平行光源)

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

```
/* LIGHT MOTION : motion_s_untei02_s_kihon2_light6 */
```

```
/* LFunc: EasyMultiDraw LType: Vector */
```

LIGHT_MOTION_START

LVECTOR vec_s_untei02_s_kihon2_light6[]

START

MKEYF(0, 0.880934F, -0.341369F, -0.327754F),

... (省略)

MKEYF(62, 0.880934F, -0.341369F, -0.327754F),

END

LCOLOR col_s_untei02_s_kihon2_light6[]

START

MKEYF(0, 1.000000F, 0.750000F, 1.000000F),

MKEYF(62, 1.000000F, 0.750000F, 1.000000F),

END

MDATA2 mdata_s_untei02_s_kihon2_light6[]

START

vec_s_untei02_s_kihon2_light6, col_s_untei02_s_kihon2_light6, 4, 2,

END

MOTION motion_s_untei02_s_kihon2_light6[]

```

START
MdataArray      mdata_s_untei02_s_kihon2_light6,
MFrameNum       63,
MotionBitF      (FMK_VEC0|FMK_RGB8),
InterpolFctF    (FMT_L|FMD_2),
END

```

LIGHT_MOTION_END

DEFAULT_START

```

#ifndef DEFAULT_LMOTION_NAME
#define DEFAULT_LMOTION_NAME motion_s_untei02_s_kihon2_light6
#endif

```

DEFAULT_END

< 解説 >

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

Ninja2用モーションデータであることを示す。SIはSoftimageからのデータであることを示す。

```
/* LIGHT MOTION : motion_s_untei02_s_kihon2_light6 */
```

モーションの変数名を示す。

```
/* LFunc: EasyMultiDraw LType: Vector */
```

対応するDraw関数名および光源の種類を示す。描画関数タイプがSimpleMultiDrawの場合LFuncのコメントがSimpleMultiDrawに変更される。それ以外は変わらない。

LIGHT_MOTION_START

アスキーモーションチャンクの始まりを示す。

```
LVECTOR vec_s_untei02_s_kihon2_light6[]
```

START

```

MKEYF( 0,  0.880934F, -0.341369F, -0.327754F ),
... ( 省略 )
MKEYF( 62, 0.880934F, -0.341369F, -0.327754F ),

```

END

ベクトル成分。

```
LCOLOR col_s_untei02_s_kihon2_light6[]  
START  
    MKEYF( 0, 1.000000F, 0.750000F, 1.000000F),  
    MKEYF( 62, 1.000000F, 0.750000F, 1.000000F),  
END
```

RGB成分。

```
MDATA2 mdata_s_untei02_s_kihon2_light6[]  
START  
    vec_s_untei02_s_kihon2_light6, col_s_untei02_s_kihon2_light6, 4, 2,  
END
```

ライトは階層を持たないためエントリは一つのみ。各成分が順に並びキーの数が後に列記される。

```
MOTION motion_s_untei02_s_kihon2_light6[]  
START  
MdataArray    mdata_s_untei02_s_kihon2_light6,  
MFrameNum     63,  
    モーションのフレーム数。  
MotionBitF     (FMK_VEC0|FMK_RGB8),  
    モーションの成分にベクトル、RGBが含まれることを示す。EasyMultiDraw, SimpleMultiDrawの平行光源はf1, f2が無効なため  
    モーションの要素数は最大 2 となる。
```

```
InterpolFctF   (FMT_L|FMD_2),  
    リニア補完で要素数が 2 つであることを示す。
```

END

LIGHT_MOTION_END

アスキーモーションチャンクの終了を示す。

DEFAULT_START

```
#ifndef DEFAULT_LMOTION_NAME  
#define DEFAULT_LMOTION_NAME motion_s_untei02_s_kihon2_light6  
#endif
```


DEFAULT_END

5.3.11 ライトモーションの出力例(EasyMultiDraw および SimpleMultiDraw の点光源)

```
/* NAM 2.00.00 Ninja2AsciiMotion (SI) */
```

```
/* LIGHT MOTION : motion_s_untei02_s_kihon2_light6 */
```

```
/* LFunc: EasyMultiDraw LType: Point */
```

LIGHT_MOTION_START

LPOSITION pos_s_untei02_s_kihon2_light6[]

START

```
MKEYF( 0, 13.584632F, -8.761686F, 0.000000F ),  
... ( 省略 )  
MKEYF( 62, 13.584632F, -8.761686F, 0.000000F ),
```

END

LCOLOR col_s_untei02_s_kihon2_light6[]

START

```
MKEYF( 0, 1.000000F, 0.750000F, 1.000000F ),  
MKEYF( 62, 1.000000F, 0.750000F, 1.000000F ),
```

END

LPOINT_FACTORS pnt_s_untei02_s_kihon2_light6[]

START

```
MKEYF2( 0, 10.000000F, 10000.000000F ),  
MKEYF2( 62, 10.000000F, 10000.000000F ),
```

END

MDATA3 mdata_s_untei02_s_kihon2_light6[]

START

```
pos_s_untei02_s_kihon2_light6, col_s_untei02_s_kihon2_light6, pnt_s_untei02_s_kihon2_light6, 4, 2, 2,
```

END

MOTION motion_s_untei02_s_kihon2_light6[]

START

```

MdataArray    mdata_s_untei02_s_kihon2_light6,
MFrameNum     63,
MotionBitF    (FMK_POS0|FMK_RGB8|FMK_PO19),
InterpolFctF  (FMT_L|FMD_3),
END

LIGHT_MOTION_END

DEFAULT_START

#ifdef DEFAULT_LMOTION_NAME
#define DEFAULT_LMOTION_NAME motion_s_untei02_s_kihon2_light6
#endif

DEFAULT_END

```

解説省略。

5.3.12 ライトモーションの出力例(EasyMultiDraw および SimpleMultiDraw のアンビエントライト)

```

/* NAM 2.00.00 Ninja2AsciiMotion (SI) */

/* LIGHT MOTION : motion_s_untei02_s_kihon2_light6 */

/* LFunc: EasyMultiDraw LType: Ambient */

LIGHT_MOTION_START

LCOLOR col_s_untei02_s_kihon2_light6[]
START
    MKEYF( 0,  1.000000F, 0.750000F, 1.000000F),
    ... (省略)
    MKEYF( 62, 1.000000F, 0.750000F, 1.000000F),
END

MDATA1 mdata_s_untei02_s_kihon2_light6[]
START
    col_s_untei02_s_kihon2_light6, 4,
END

```

```

MOTION motion_s_untei02_s_kihon2_light6[]

```

START

MdataArray mdata_s_untei02_s_kihon2_light6,

MFrameNum 63,

MotionBitF (FMK_RGB8),

InterpolFctF (FMT_L|FMD_1),

END

LIGHT_MOTION_END

DEFAULT_START

#ifndef DEFAULT_LMOTION_NAME

#define DEFAULT_LMOTION_NAME motion_s_untei02_s_kihon2_light6

#endif

DEFAULT_END

解説省略。

6. バイナリフォーマット

6.1 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
CellSprite	.spj	.spa
CellStream	.csj	.csa

CellSrite、CellSteam に関しては CellSprite 仕様書参照のこと。

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
複数の種類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 Ninja 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。

6.2 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><主なチャンクの種類> '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。

6.3 バイナリチャンクヘッダ

次にチャンク名を表にまとめる。Ninja2 では未使用の Ninja1 用チャンク名、Ninja2 用チャンク名、pvr ファイル用チャンク名、pvm チャンク用ファイル名が説明される。

'NJIN'	Ninja データに任意に書き込むユーザデータ領域。
'NJCM'	Ninja1 用チャンクモデルデータ。過去データ認識用に用意される。Ninja2 で使用不可。
'NJBM'	Ninja1 用ベーシックモデルデータ。過去データ認識用に用意される。Ninja2 で使用不可。
'N2CM'	Ninja2 用チャンクモデルデータ。
'NJTL'	Texlist データ。
'NJLI'	ライトデータ。
'NJCA'	カメラデータ。
'NMDM'	モデルモーションデータ。
'NLIM'	ライトモーションデータ。
'NCAM'	カメラ用モーションデータ。
'NSSM'	Ninja1 用 SimpleShape モーションデータ。過去データ認識用に用意される。Ninja2 で使用不可。
'NJSP'	CellSprite データ。
'NJCS'	CellStream モーションデータ。
'NJSM'	CellSprite モーションデータ。
'CPSM'	CompactShape モーションデータ。
'NJSL'	CompactShape で利用する ShapeList データ。
'NJAD'	絶対アドレスバイナリのアドレス指定チャンク。
'POF0'	Ninja バイナリ復元のためのポインタ情報を格納する。
'POF1'	Ninja バイナリ復元のためのポインタ情報を格納する。データ表現効率の高いアルゴリズムその 2 が必要な時のためのリザーブ。現在は POF0 のみが機能する。
'PVRT'	pvr テクスチャデータ。pvr ファイル、pvm ファイルに格納される。
'PVPL'	パレットデータ。pvp ファイル、pvm ファイルに格納される。
'GBIX'	グローバルインデックスチャンク。パレット使用時はバンク ID も格納する。pvr ファイルに格納される。
'PVRI'	pvm 生成時に利用したテクスチャ元画像の情報、コンバータのオプションを格納する。このチャンクはテクスチャをもう一度コンバートし直す場合の情報として用意される。
'PVMH'	pvm 格納情報をまとめるヘッダ。
'COMM'	pvm に書き込み可能なユーザコメント（ユーザエリア）。
'COMV'	pvm 生成に利用したコンバータ名を保存する。
'MDLN'	pvm に格納されるテクスチャと組み合わせて使われるモデルファイル名。
'PVPN'	pvm に格納されたパレットファイルのファイル名を格納する。
'IMGC'	pvr テクスチャ生成に利用された元画像を格納する。このチャンクを持つ場合 pvm だけからテクスチャの再コンバートができる。

6.4 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