



Ninja2 CellSprite 仕様書

(08/11/2000)

1. 概要	3
2. CELLSPRITE 構造体	7
3. CELL CHUNK 仕様	12
3.1 CELL CHUNK の種類	12
3.2 CELL CHUNK TIME END	14
<i>ChunkName : 'NJD_CC_TE'</i>	14
3.3 CELL CHUNK TIME STAMP	14
<i>ChunkName : 'NJD_CC_TS'</i>	14
3.4 CELL CHUNK CELL	15
<i>ChunkName : 'NJD_CC_CE'</i>	15
3.5 CELL CHUNK TexID	16
<i>ChunkName : 'NJD_CC_TD'</i>	16
3.6 CELL CHUNK ATTR	16
<i>ChunkName : 'NJD_CC_AT'</i>	16
3.7 CELL CHUNK CZANG	17
<i>ChunkName : 'NJD_CC_CZ'</i>	17
3.8 CELL CHUNK CELL PRIORITY	17
<i>ChunkName : 'NJD_CC_CP'</i>	17
3.9 CELL CHUNK USER EVENT	18
<i>ChunkName : 'NJD_CC_UE'</i>	18
3.10 CELL BURST CHUNK (ADDRESS TYPE)	19
<i>ChunkName : 'NJD_CBC_AD'</i>	19
3.11 CELL BURST CHUNK (ENTRYID TYPE)	20
<i>ChunkName : 'NJD_CBC_ED'</i>	20
4. CELLSTREAM	21
5. NINJAEXPORT フォーマット	22
5.1 TEXLIST アスキーフォーマット NAT ファイル	25
5.2 CELLSPRITE アスキーフォーマット SPA ファイル	29
5.3 MOTION アスキーフォーマット NAM ファイル	35
5.4 CELLSTREAM アスキーフォーマット CSA ファイル	44
5.5 バイナリフォーマット	54
5.6 POF0 のアルゴリズム	56

1. 概要

Ninja2用 CellSprite について説明する。Ninja1 の仕様からアスキー出力に関する一部に変更を加えている。ただしこれらの変更は Ninja2 用であることを示すことと他の Ninja2 データとの表現を合わせるための変更であり現時点では Ninja1 用で作成したデータでも利用可能。

主な変更点は次の通り。

nat ファイルのヘッダ文字列 NinjaAsciiTexture を Ninja2AsciiTexture に変更。

spa ファイルのヘッダ文字列 NinjaAsciiCellSprite を Ninja2AsciiCellSprite に変更。

CellSprite 用 nam ファイルのヘッダ文字列 NinjaAsciiCellSpriteMotion を Ninja2AsciiCellSpriteMotion に変更。

CellSprite 用 nam ファイルの変数名コメントのタイトルを MOTION から CSPRITE_MOTION に変更。

CellSprite 用 nam ファイルの MOTION_START を CSPRITE_MOTION_START、MOTION_END を CSPRITE_MOTION_END に変更。

CellSprite 用 csa ファイルのヘッダ文字列 NinjaAsciiCellStream を Ninja2AsciiCellStream に変更。

CellSprite は複数のパーツ (cell) から構成されるグループをひとまとめにして扱うスプライトを可能とする。各 cell ごとに各種アニメーションが可能であり、ループ時間の異なるアニメーションパターンを同時に管理し実行できる CellStreamList 機構を持つ。スプライト全体に対するモーションでは通常の Ninja の nam、njm ファイルを利用する。ただしモーションの中身は Object データのためのものとは異なりスケールは x, y のみ、回転は z のみ管理する。モーションのフレームは Float でなく Sint32 であり整数でないフレームは指定できない。

CellSprite に使われるテクスチャはすべて texlist で管理され Cell はこの texlist に対する texId でテクスチャを指定する。

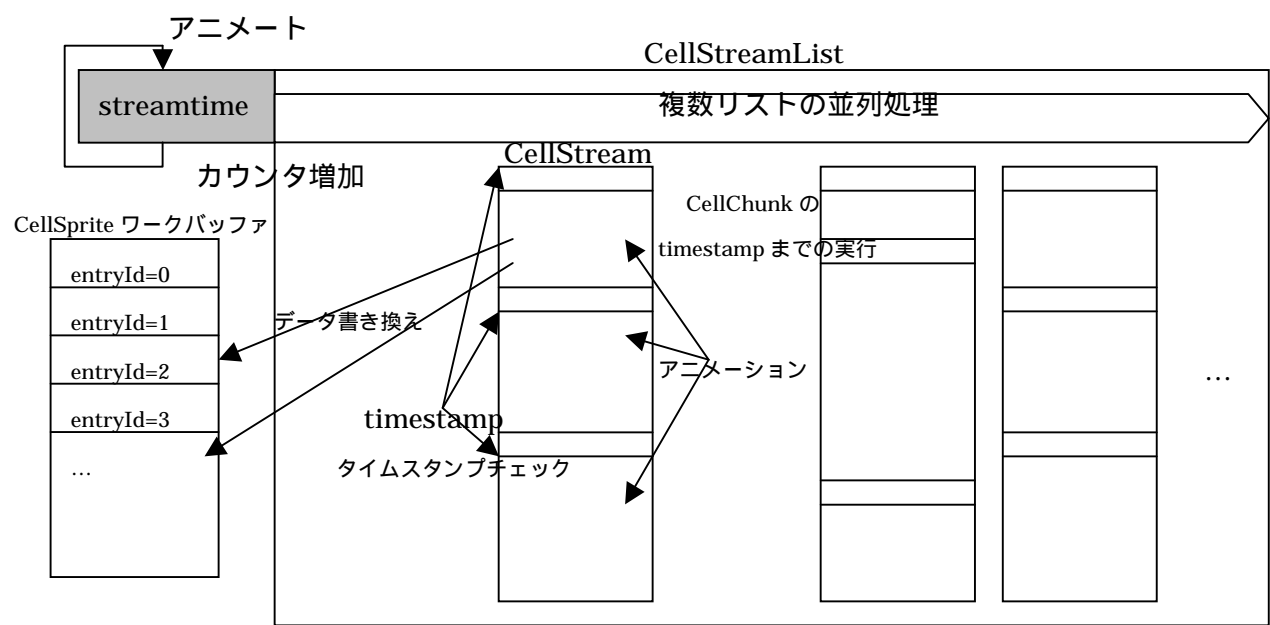
CellSprite のデータ出力はスタジオブルテリアの 2D ツールである Farlux であることを前提する。Ninja2 ライブラリは本仕様の CellSprite データを実行する関数を持つ。

アニメーションパターンは本仕様書に定義される CellChunk 方式で Sint32 の一次元配列に変化する部分だけのデータがタイムスタンプとともに並べられた構造で表現される。CellStreamList はこのアニメーションパターンを複数登録することができる。各アニメーションパターンは任意に基準時間からの開始オフセットとアニメーションのループ時間を持つことができ、CellStreamList 機構は CellStreamList が持つ基準時間に合わせて複数のアニメーションパターンの実行が可能。

cell レベルで可能なアニメーションは次の通り。

Block0	Sint16 texId	cellSprite が利用する texlist のエントリ番号。
	Sint16 attr	ブレンディングアルファを含むアトリビュート群。
Block1	Float cox, coy	cell オフセット。
	Float csx, csy	cell サイズ (スケール)。
	Sint16 czang	cell の回転。
Block2	Sint16 cp	cell 間のプライオリティ。
Block3	Float cent_x, cent_y	cell の中心を与える。回転/移動の中心となる。
Block4	NJS_COLOR argb[4]	頂点カラー、cell を構成する 4 つの頂点のカラーを指定する。
Block5	Float u0, v0	左上頂点の uv 値。
	Float u2, v2	右下頂点の uv 値。

CellChunk 形式では上記データをブロック単位で指定でき必要な部分のみのデータをアニメーションパターンに登録できる。



CellSprite そのもののアニメーションには、ninja の標準のモーションファイル nam、njm を利用する。

煩雑になることを避けるためすでに定義されているモーションのタイプ MTYPE フラグを流用する。nam ファイルは、オブジェクト、シェイプ、カメラ、ライトで利用されているがこれらの一部（例えばポジション）などが共通して使える可能性を考慮した成分別データとしてモーションが格納される。

ここでこの想定を割り切りスプライトモーションと他のモーションが共通して使える可能性はないとしてスプライトモーションと他のモーションで同一フラグに対する意味を二重に持たせる。これは必要以上にフラグ定義が増えることを避けることを目的とする。

NjMotion.h

```
#define NJD_MTYPE_POS_0    BIT_0
#define NJD_MTYPE_ANG_1    BIT_1
#define NJD_MTYPE_SCL_2    BIT_2
#define NJD_MTYPE_RGB_8    BIT_9
```

本来これらの四つは位置（xyz）、角度（xyz）、スケール（xyz）、光源色（rgb）を与えるための定義。しかし CellSprite ではこれらのフラグを次のように利用する。

NJD_MTYPE_POS_0 は、CellSprite の位置（xyz）
NJD_MTYPE_ANG_1 は、CellSprite の Z 回転（z）
NJD_MTYPE_SCL_2 は、CellSprite の 2D スケール（xy）
NJD_MTYPE_RGB_8 は、マテリアルディフューズカラー（argb）

これに伴い利用する MKEY 構造体の種類が変更になる。アスキーの nam ファイルでは利用する文字列に CS をつけることでこれがスプライトモーションであることを示す。バイナリの njm ファイルではチャンク名の違いでそれを識別する。

スプライトアニメーションにおいて cell 単位で設定可能なアトリビュートは次の通り。

ブレンディングアルファ	値を使ったソース、ディスティネーションのブレンディング制御
USE_ALPHA	Cell テクスチャに がある場合に必ず ON する。
PUNCH_THROW	Cell テクスチャをパンチスルーで描画する。
頂点カラー有効	頂点カラーを利用することを意味する。
HFlip/VFlip	Cell テクスチャを縦横にフリップする。
Hide	Cell テクスチャを hide できる。
TimeRepeat	Cell アニメーションを繰り返す。

timestamp、streamtime の時間は単位を持たない。streamtime を 1/60 秒で進めれば 1/60 秒になる。

CellSpriteに関わる出力ファイルは次の通り。

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
CellSprite	.spj	.spa
CellStream	.csj	.csa
CellSprite Motion	.njm	.nam

実使用においてはtexlistとCellSpriteを一つのファイルでCellStreamとCellSpriteMotionを一つのファイルで扱いたいことがある。拡張子の必要以上の増加を防ぐため次の方法を取る。

spa ファイルは、nat ファイルを取り込むことができる。

csa ファイルは、nam ファイルを取り込むことができる。

spj ファイルは、njt ファイルを取り込むことができる。

csj ファイルは、njm ファイルを取り込むことができる。

コンバータはデフォルトで nat を取り込んだ spa、nam を取り込んだ csa を出力する。バイナリ出力では njt を取り込んだ spj、njm を取り込んだ csj を出力する。オプションとして上記成分をバラバラのファイルで出力できる。

2. CellSprite 構造体

NjCSprite.h に定義される。

CellSprite は NJS_CELL 構造体により Cell を表現する。ライブラリはこの構造体の情報を CellStream からアニメーション情報によって書き換えながら Cell 描画する。

```
typedef struct {  
    /* texture Id and attr (block0) */  
    Sint16      texId;  
    Sint16      attr;  
  
    /* texture cell motion (block1) */  
    Float       cox, coy;    /* cell offset      */  
    Float       csx, csy;    /* cell size(scale) */  
    Sint16      czang;  
  
    /* texture cell priority (block2) */  
    Sint16      cp;         /* cell priority    */  
  
    /* texture cell center (block3) */  
    Float       cent_x, cent_y; /* size ratio      */  
  
    /* vertex color (block4) */  
    NJS_COLOR   argb[4];  
  
    /* texture uv (block5) */  
    Float       u0, v0;  
    Float       u2, v2;  
} NJS_CELL;
```

全体は相関性の強いデータをひとまとめにした6つのブロックに分けられる。キャッシュを考慮し64バイトサイズの構造体とする。

ブロック0

texId はテクスチャを指定する。attr はアトリビュート。テクスチャ関連をまとめる。テクスチャの差し替え 値に対する処理の変更、フリップ、ハイドなどができる。

ブロック1

CellSprite 中心座標からのオフセット(cox, coy)、セルのサイズ (csx, csy) これはスケールも兼ねる、z 軸回転 (czang)。Cell の移動に伴うアニメーションをまとめる。

ブロック2

プライオリティチェンジアニメーションをまとめる。Cell 同士のプライオリティを決定する。実際の値は +32767 ~ -32767 で示される。実際には+1.0 ~ -1.0 の z 値に変換される。Cell 間のプライオリティは+1.0 ~ -1.0 の間に設定すること。その結果として二つの CellSprite の Cell 同士の干渉を防ぐためには CellSprite のプライオリティで 2.0 以上はなすことで実現できる。

ブロック3

Cell の中心点 (cent_x, cent_y)。この値はサイズ (csx, csy) に対する比で表現される。つまり Cell の真ん中を設定したい場合は、cent_x=0.5, cent_y=0.5 になる。

ブロック 4

4 頂点の頂点カラーアニメーション。

ブロック 5

スプライト u, v アニメーションをまとめる。

ここではできるだけ自由度を高めるためすべてのパラメータのアニメーションが可能ないように仕様を定義している。

```
typedef struct {  
    NJS_CELL      *cells;  
    Int           nbCell;  
    NJS_POINT3    pos; /* [x, y, z](3D) or [x, y, priority](2D) */  
    Angle         zang; /* zrotation          */  
    Float         sx, sy; /* scale              */  
    NJS_ARGB      diffuse; /* material diffuse */  
    NJS_ARGB      specular; /* material specular */  
} NJS_CELL_SPRITE;
```

CellSprite は NJS_CELL_SPRITE 構造体で管理される。これはモデルで言うところのオブジェクト構造体に相当する。nam によるモーション時にはモーションデータが null の部分に関してはこの構造体上の値が利用される。

Cell には CellSprite を構成するすべての Cell の一次元配列が格納される。nbCell には Cell の数を、pos には x y 座標 (pos.x, pos.y) およびプライオリティ (pos.z) もしくは 3D スプライトの場合は x y z 座標 (pos.x, pos.y, pos.z) を格納、zang は z 軸回転の Angle 値、sx, sy はスケール値 (比でない普通のスケール)、diffuse には、Cell に対するマテリアル diffuse カラー、diffuse の 値で Cell 全体の透明度制御ができる。Specular は sprite の点滅などに対応する。現在 nam による specular のモーションはできない仕様になっている。

```
typedef struct {  
    Sint32      *cellstream;  
    Sint32      *cur_stamp_p;  
    Sint32      timeoffset;  
    Sint32      timemax;  
} NJS_CELL_STREAM;
```

ひとつのアニメーションパターンを与える。cellstream には後述の CellChunk 形式データの一次元配列が格納される。CellChunk にはアニメーションデータとそれを実行すべき時間を与える timestamp データが格納される。cur_stamp_p には cellstream メンバーに格納される現在実行待ちの先頭の timestamp のポインタが格納される。基準時間 (streamtime) とこのポインタ先の timestamp を比較し実行時間を確定する。timeoffset は、CellStreamList 実行中の基準時間における本アニメーションパターンの開始位置をずらす。timemax はアニメーションパターンの時間が格納される。これは cellstream に格納される timestamp チャンクの最後を示す Cell Chunk Time End に格納される時間と一致する。

実際の時間は本アニメーションパターンの中の基本時間を t(t=0 ~ timemax) とすると t+timeoffset が streamtime に一致した部分のアニメーションが実行される。また timeoffset ~ timemax+timeoffset 以外の部分はアニメーションしないが NJD_FCA_TR (time repeat) フラグを Cell の attr に設定しておけば timemax を一周期として自動的に繰り返す。


```
typedef struct {
    Sint32      nbStream;
    Sint32      streamtime;
    NJS_CELL_SPRITE *csp;
    Sint32      *wbuf;
    NJS_CELL_STREAM *streamlist[NJD_CELL_PATTERN_MAX];
} NJS_CELL_STREAM_LIST;
```

一度に実行したい Cell を登録しリストとして実行するために NJS_CELL_STREAM_LIST 構造体を使用する。nbStream は現在登録されている CellStream (アニメーションパターン) の数。streamtime は stream の基準時間。csp はアニメーションさせたい CellSprite のポインタ、wbuf はユーザが用意したワークバッファポインタ、streamlist は登録された CellStream のポインタリスト。ワークバッファサイズは、ライブラリに用意されるワークバッファサイズ取得関数でサイズを得てユーザが確保する。

次にデファインについて説明する。

```
/*-----*/
/* Cell Stream Max */
/*-----*/
#define NJD_CELL_PATTERN_MAX    256
```

NJS_CELL_STREAM 構造体の登録最大数を定義する。

```
/*-----*/
/* Flag Cell Attr */
/*-----*/
/*=====*/
#ifndef NJD_FBS_ZER          /* see NjChunk.h */
/**** Flag Blending Src and Dst ****/
/* ZER : Zero */
/* ONE : One */
/* OC  : `Other' Color */
/* IOC : Inverse `Other' Color */
/* SA  : Src Alpha */
/* ISA : Inverse SRC Alpha */
/* DA  : DST Alpha */
/* IDA : Inverse DST Alpha */

/* Flag Blending Src */

#define NJD_FBS_SHIFT    11
#define NJD_FBS_ZER      (0<<NJD_FBS_SHIFT)
#define NJD_FBS_ONE      (1<<NJD_FBS_SHIFT)
```

```

#define NJD_FBS_OC      (2<<NJD_FBS_SHIFT)
#define NJD_FBS_IOC      (3<<NJD_FBS_SHIFT)
#define NJD_FBS_SA      (4<<NJD_FBS_SHIFT)
#define NJD_FBS_ISA      (5<<NJD_FBS_SHIFT)
#define NJD_FBS_DA      (6<<NJD_FBS_SHIFT)
#define NJD_FBS_IDA      (7<<NJD_FBS_SHIFT)

#define NJD_FBS_MASK      (0x7<<NJD_FBS_SHIFT)

/* Flag Blending Dst */

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

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

/*=====*/

/**** Flag Cell Attr (for attr member) ****/
/* Flag Blending Src(13-11) */
#define NJD_FCS_ZER      NJD_FBS_ZER
#define NJD_FCS_ONE      NJD_FBS_ONE
#define NJD_FCS_OC      NJD_FBS_OC
#define NJD_FCS_IOC      NJD_FBS_IOC
#define NJD_FCS_SA      NJD_FBS_SA
#define NJD_FCS_ISA      NJD_FBS_ISA
#define NJD_FCS_DA      NJD_FBS_DA
#define NJD_FCS_IDA      NJD_FBS_IDA

```

ブレンディングアルファのソースデータに対するファンクションを選択する。

```

/* Flag Blending Dst(10-8) */
#define NJD_FCD_ZER      NJD_FBD_ZER
#define NJD_FCD_ONE      NJD_FBD_ONE
#define NJD_FCD_OC      NJD_FBD_OC

```

```
#define NJD_FCD_IOC      NJD_FBD_IOC
#define NJD_FCD_SA      NJD_FBD_SA
#define NJD_FCD_ISA     NJD_FBD_ISA
#define NJD_FCD_DA      NJD_FBD_DA
#define NJD_FCD_IDA     NJD_FBD_IDA
```

ブレンディングアルファのディスティネーションデータに対するファンクションを選択する。

```
#define NJD_FCA_VC      BIT_7
#define NJD_FCA_AL      BIT_6
#define NJD_FCA_PT      BIT_5
#define NJD_FCA_HF      BIT_4
#define NJD_FCA_VF      BIT_3
#define NJD_FCA_HI      BIT_2
#define NJD_FCA_TR      BIT_1
```

NJS_CELL 構造体の attr メンバーに設定するフラグ。NJD_FCA_VC は頂点カラーを有効にする。NJD_FCA_AL はテクスチャに がある場合に設定する。NJD_FCA_PT はテクスチャをパンチスルーで描画する。NJD_FCA_HF は Cell を横方向にフリップする。NJD_FCA_VF は縦方向にフリップする。NJD_FCA_HI は Cell をハイドし見えなくする。

NJD_FCA_TR はアニメーションパターンを繰り返し実行することを示す。

3. Cell Chunk 仕様

3.1 Cell Chunk の種類

Cell Chunk によるアニメーションパターンは NJS_CELL_STREAM 構造体の cellstream メンバポインタの一次元配列に格納される。

Cell Chunk の名前	略号	サイズ	説明
<u>C</u> ell <u>C</u> hunk <u>T</u> ime <u>E</u> nd	CC_TE	32bit	CellStream に格納されるチャンクの最後を示すタイムスタンプ。
<u>C</u> ell <u>C</u> hunk <u>T</u> ime <u>S</u> tamp	CC_TS	32bit	アニメーションのタイムスタンプ。
<u>C</u> ell <u>C</u> hunk <u>C</u> ell	CC_CE	可変	Cell アニメーションデータ。ブロック単位でデータの有無を管理する。
<u>C</u> ell <u>C</u> hunk <u>T</u> exId	CC_TD	32bit	Cell の TexId のみを書き換える。
<u>C</u> ell <u>C</u> hunk <u>A</u> Ttr	CC_AT	32bit	Cell の attr のみを書き換える。
<u>C</u> ell <u>C</u> hunk <u>C</u> Zang	CC_CZ	32bit	Cell の czang のみを書き換える。
<u>C</u> ell <u>C</u> hunk <u>C</u> ell <u>P</u> riority	CC_CP	32bit	Cell の cp のみを書き換える。
<u>C</u> ell <u>C</u> hunk <u>U</u> ser <u>E</u> vent	CC_UE	可変	ユーザコールバックルーチンの呼び出し。

基本的には Cell Chunk Cell を利用して現在のアニメーションに必要なブロックデータを entryId 分だけ格納しこれらを Cell Chunk Time Stamp で挟むことで時間軸を持つアニメーションデータを構成する。データの最後には Cell Chunk Time End を格納する。場合によっては texId のみの書き換えだけを連続する場合がある。この場合 Cell Chunk Cell を使うのは効率が悪いいため専用書き換えチャンクを用意している。Sint16 領域である texId, attr, czang, cell priority に対し専用書き換えチャンクを用意する。

ゲーム中に絵と連動してユーザイベントを発生させたい場合に対し Cell Chunk User Event を用意する。ユーザデータ領域はロングワード単位の可変サイズ。ライブラリはチャンクに格納されるユーザデータ領域をサイズ n とともにユーザコールバックルーチンに渡す。ユーザデータ領域のロングワードサイズは可変だがツールからの設定 GUI では無制限にできないため本機能に対応するツールにおいて最小で3ロングワードのデータ設定のサポートを必須とする。設定する値はユーザにより任意の順番で整数か浮動小数点値かビット列の可能性はある。

次の仕様は拡張仕様。標準サポート外。

Cell Burst Chunk の名前	略号	サイズ	説明
<u>Cell Burst Chunk</u> (Address type)	CBC_AD	可変	CellStream 一次元配列のデータ書き換えを先頭ポインタアドレスを 0 番地とする相対アドレスで指定します。
<u>Cell Burst Chunk</u> (EntryID type)	CBC_ED	可変	CellStream 一次元配列のデータ書き換えを entryId とそのエントリの Cell 構造体の先頭からのロングワードオフセットで指定します。

Cell Burst Chunk は処理の高速化を実現するための仕様。データの中身を識別せずにデータ領域のコピー操作のみで表現される。Sint32 単位でコピーする。Sint16 領域を書き換えることはできずそれと対となる Sint16 領域を合わせて Sint32 単位で書き込む。

Cell チャンクテーブルは 4 ビットの 16 エントリから構成される。これを事前に明示しこれらのエントリを各チャンクに割り振る。

```
/* table size = 4bits(16 entries) */
```

```
#define NJD_CC_SHIFT      28
```

```
#define NJD_CC_0          (0<<NJD_CC_SHIFT)
```

```
#define NJD_CC_1          (1<<NJD_CC_SHIFT)
```

```
#define NJD_CC_2          (2<<NJD_CC_SHIFT)
```

```
#define NJD_CC_3          (3<<NJD_CC_SHIFT)
```

```
#define NJD_CC_4          (4<<NJD_CC_SHIFT)
```

```
#define NJD_CC_5          (5<<NJD_CC_SHIFT)
```

```
#define NJD_CC_6          (6<<NJD_CC_SHIFT)
```

```
#define NJD_CC_7          (7<<NJD_CC_SHIFT)
```

```
#define NJD_CC_8          (8<<NJD_CC_SHIFT)
```

```
#define NJD_CC_9          (9<<NJD_CC_SHIFT)
```

```
#define NJD_CC_10         (10<<NJD_CC_SHIFT)
```

```
#define NJD_CC_11         (11<<NJD_CC_SHIFT)
```

```
#define NJD_CC_12         (12<<NJD_CC_SHIFT)
```

```
#define NJD_CC_13         (13<<NJD_CC_SHIFT)
```

```
#define NJD_CC_14         (14<<NJD_CC_SHIFT)
```

```
#define NJD_CC_15         (15<<NJD_CC_SHIFT)
```

```
#define NJD_CC_MASK       (0xf<<NJD_CC_SHIFT)
```

3.2 Cell Chunk Time End

ChunkName : 'NJD_CC_TE'

(Cell Chunk Time End)

概要 :

CellStream データの完了を検出する。またその時点の timestamp を格納する。

形式 :

```
[ChunkHead(4)|dammy(12)|timestamp(16)]
```

timestamp:

アニメーションの実行時間。

説明 :

```
#define NJD_CC_TE (NJD_CC_0)
```

3.3 Cell Chunk Time Stamp

ChunkName : 'NJD_CC_TS'

(Cell Chunk Time Stamp)

概要 :

timestamp を格納し CellStream データの時間制御をする。

形式 :

```
[ChunkHead(4)|longsize(12)|timestamp(16)]
```

longsize:

次の timestamp までのロングワードオフセット。

timestamp:

アニメーションの実行時間。

説明 :

```
#define NJD_CC_TS (NJD_CC_1)
```

longsize には次の timestamp までのロングワードでのオフセットが入る。全フレームにおいて固定メンバーの書き換えをした場合 timestamp による時間軸のスキップが可能でありこの longsize の値を利用して次の timestamp までポインタを進める。

3.4 Cell Chunk Cell

ChunkName : 'NJD_CC_CE'

(Cell Chunk Cell)

概要 :

各 Cell のアニメーションパターンデータを格納する。

形式 :

```
[ChunkHead(4) | longsize(4) | headbits(8) | entryId(16)]
[Data]
entryId:
    CellStream に格納される Cell のエントリ番号。
longsize:
    次の Cell チャンクまでのロングワードオフセット。
headbits:
    このチャンクに格納されるブロックデータの有無を示す。
data:
    headbits で示されたブロックを並べたデータ。
```

説明 :

```
#define NJD_CC_CE                (NJD_CC_2)

#define NJS_FCB_SHIFT            16

#define NJS_FCB_IA0              (BIT_7<<NJS_FCB_SHIFT) /* texId and attr */
#define NJS_FCB_MO1              (BIT_6<<NJS_FCB_SHIFT) /* cell motion      */
#define NJS_FCB_PR2              (BIT_5<<NJS_FCB_SHIFT) /* cell proirity    */
#define NJS_FCB_CE3              (BIT_4<<NJS_FCB_SHIFT) /* texture center   */
#define NJS_FCB_VC4              (BIT_3<<NJS_FCB_SHIFT) /* vertex color     */
#define NJS_FCB_UV5              (BIT_2<<NJS_FCB_SHIFT) /* uv               */

#define NJS_FCB_MASK              (0xff<<NJS_FCB_SHIFT)
```

ブロックは 0 ~ 5 の 6 つに分けられる。これらのフラグは headbits に格納される。これらのフラグが立っていない部分を除くデータを並べたものがデータとして格納される。またそのブロック合計ロングワード数が longsize に格納される。
指定された entryId の Cell に対しデータが書き込まれる。

3.5 Cell Chunk TexId

ChunkName : 'NJD_CC_TD'

(Cell Chunk TexId)

概要 :

指定された entryId の cell の texId を書き換える。

形式 :

```
[ChunkHead(4) | entryId(12) | texId(16)]
```

entryId:

CellStream に格納される Cell のエントリ番号。

texId:

書き換える texId。

説明 :

```
#define NJD_CC_TD (NJD_CC_3)
```

例えば texId のみが変化する場合などに利用する。

3.6 Cell Chunk Attr

ChunkName : 'NJD_CC_AT'

(Cell Chunk ATtr)

概要 :

指定された entryId の cell の attr を書き換える。

形式 :

```
[ChunkHead(4) | entryId(12) | attr(16)]
```

entryId:

CellStream に格納される Cell のエントリ番号。

attr:

書き換える attr フラグデータ。

説明 :

```
#define NJD_CC_AT (NJD_CC_4)
```

例えば attr のみが変化する場合などに利用する。

3.7 Cell Chunk Czag

ChunkName : 'NJD_CC_CZ'

(Cell Chunk CZang)

概要 :

指定された entryId の cell の czang を書き換える。

形式 :

[ChunkHead(4) | entryId(12) | czang(16)]

entryId:

CellStream に格納される Cell のエントリ番号。

czang:

書き換える czang。

説明 :

```
#define NJD_CC_CZ (NJD_CC_5)
```

例えば czang のみが変わる場合などに利用する。

3.8 Cell Chunk Cell Priority

ChunkName : 'NJD_CC_CP'

(Cell Chunk Cell Priority)

概要 :

指定された entryId の cell の cp(cell priority)を書き換える。

形式 :

[ChunkHead(4) | entryId(12) | cp(16)]

entryId:

CellStream に格納される Cell のエントリ番号。

cp:

書き換える cp。

説明 :

```
#define NJD_CC_CP (NJD_CC_6)
```

例えば cp のみが変わる場合などに利用する。

3.9 Cell Chunk User Event

ChunkName : 'NJD_CC_UE'

(Cell Chunk User Event)

概要 :

ユーザイベントコールバックルーチンの呼び出し。

形式 :

```
[ChunkHead(4) | longsize(12) | eventId(16)]  
[Data ...]
```

longsize:

ユーザデータ領域のロングワードサイズ。

eventId:

event 登録 Id。

Data:

longsize に指定されるデータサイズ領域。

説明 :

```
#define NJD_CC_UE (NJD_CC_7)
```

ユーザが登録したユーザイベントコールバック関数を呼び出す。ユーザデータ領域はロングワード単位の可変サイズ。ライブラリはチャンクに格納されるユーザデータ領域をサイズ n とともにユーザコールバックルーチンに渡す。コールバックルーチンリストはユーザレベルで eventId で管理される。ライブラリは eventId に対応して登録された複数のコールバックルーチンをリストとして持つ事ができる。コールバックルーチンのテーブルサイズはライブラリで初期化時にユーザ指定で設定される。例えばサイズが 2 5 6 の場合 0 ~ 255 の eventId が利用可能になる。ユーザデータ領域のロングワードサイズは可変だがツールからの設定 GUI では無制限にできないため本機能に対応するツールにおいて最小で 3 ロングワードのデータ設定のサポートを必須とする。設定する値はユーザにより任意の順番で整数か浮動小数点値かビット列の可能性はある。コールバックルーチンの引数は次のようになる。

(例)

```
void UserEvent1CB(Sint32 n, Sint32 *evlist);
```

evlist には n ロングワードのデータ領域が連続するものとする。

ユーザは evlist[0], ..., evlist[n-1] を引数として利用可能。

3.10 Cell Burst Chunk (Address type)

ChunkName : 'NJD_CBC_AD'

(Cell Burst Chunk Address type)

概要 :

面倒なオフセット計算を省略しデータを書き換える最速の構造。拡張仕様。データのメンテナンス性はない。

形式 :

```
[ChunkHead(4) | longsize(12) | longaddress(16)]  
[Data]
```

longsize:

次のチャンクまでのロングワードオフセット。

longaddress:

Cell 構造体ワークエリア先頭からのロングワードオフセット。

Data:

Cell 構造体ワークエリアにおけるデータの連続領域。ブロック間にまたがることも可能。

複数 Cell 構造体にまたがることも可能。

説明 :

```
#define NJD_CBC_AD (NJD_CC_14)
```

性能優先時のアニメーション構造。

3.11 Cell Burst Chunk (entryId type)

ChunkName : 'NJD_CBC_ED'

(Cell Burst Chunk EntryID type)

概要 :

NJD_CBC_AD と同様に速度優先仕様ですがユーザの利便性を多少考慮したタイプ。entryId により書き込みたい Cell を選択しその先頭からロングワードオフセットで書き込む領域を示す。複数 Cell にまたがる書き込みはできない。

形式 :

```
[ChunkHead(4) | longsize(4) | offset(8) | entryId(16)]  
[Data]
```

longsize:

次のチャンクまでのロングワードオフセット。

offset:

構造体先頭からのロングワードオフセット。

entryId:

CellStream に格納される Cell のエントリ番号。

Data:

Cell 構造体ワークエリアにおけるデータの連続領域。ブロック間にまたがることも可能。

説明 :

```
#define NJD_CBC_AD (NJD_CC_15)
```

性能優先時のアニメーション構造。

4. CellStream

CellSprite に関わる出力ファイルは次の通り。

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
Texture	.pvr	-
Palette	.pvp	-
Texlist	.njt	.nat
CellSprite	.spj	.spa
CellStream	.csj	.csa
CellSprite Motion	.njm	.nam

ここで CellStream とはアニメーションパターンデータ。nam および njm ファイルは従来のオブジェクト用のデータ構造をそのまま使うが CellSprite 専用。pvr, pvp, nat および njt は従来のオブジェクト描画に使っているものとまったく同一のもの。texlist は複数の CellSprite に関わる texlist として出力されることが可能。

次に実際の CellStreamList 実行手順を示す。ここではアスキーファイルを出力するものとして説明する。

<step1>

ツールから次のデータが出力される。

CellSprite が利用するテクスチャをリストにまとめた njt, nat ファイルおよび実際のテクスチャである pvr ファイル。パレットテクスチャの場合は pvp ファイルも出力。

NJS_CELL 構造体の一次元配列およびこのポインタをデータとして格納する NJS_CELL_SPRITE 構造体。spa ファイル出力。

Sint32 の一元配列にアニメーションパターンを Cell Chunk で構成した CellStream データとこのポインタを格納する NJS_CELL_STREAM 構造体。csa ファイル。

CellSprite のモーションデータである NJS_MOTION 構造体。nam ファイル。

<step2>

コンパイルもしくはアスキーローダにより各データをプログラムに取り込む。texlist をカレントに設定する。

ライブラリ関数により CellSprite 及び CellStream データを NJS_CELL_STREAM_LIST 構造体に格納する。この時ライブラリから取得したサイズの CellSprite ワークバッファをユーザは用意し NJS_CELL_STREAM_LIST 構造体とともに描画を開始。

モーション全体の時間は NJS_CELL_STREAM_LIST の streamtime で管理される。streamtime を進めて CellStream を実行することで複数のアニメーションパターンを同時に実行する。

<step3>

NJS_CELL_STREAM_LIST の nbStream 分だけの CellStream をループで実行。各 CellStream の cur_stamp_p には現在実行を完了したデータの次の timestamp チャンクのポインタが格納されており、これと timeoffset, timemax から決まる現在の CellStream の時間を比較し一致した場合次の timestamp までの間の Cell データを実行しワークエリアのスプライトデータを書き換える。

この操作により時間軸を考慮した複数の CellStream の同時実行が可能になる。

<step4>

すべての CellStream によるワークバッファ更新が完了したら nam ファイルによって与えられる CellSprite のモーションと組み合わせポリゴンを描画する。

5. NinjaExport フォーマット

CellSprite に関わる Ninja のインクルードファイルは次の通り。

NjCSprite.h	CellSpriteの仕様を定義する。
NjDef.h	アスキーフォーマットのための文字列定義、バイナリチャンク名などを定義する。
NjMotion.h	CellSpriteモーションに関わるデファイン及び構造体を定義する。
(NjChunk.h)	ブレンディングアルファに関わる定義をNjCSprite.hと共有している。NjCSprite.hの中に重複定義しているのでなくても動作する。

再度CellSpriteに関わるファイルを示す。以降これらのファイル出力の使用について説明する。

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
Texture	.pvr	-
Palette	.pvp	-
Texlist	.njt	.nat
CellSprite	.spj	.spa
CellStream	.csj	.csa
CellSprite Motion	.njm	.nam

pvr ファイルにはCellSprite で使用されるテクスチャが出力される。pvp はインデックス画像の格納されたパレットタイプの pvr ファイルとともに使われるパレットデータ。パレットテクスチャの詳細に関してはNinja2 Model 仕様書を参照のこと。またパレットテクスチャ時には pvr ファイルでグローバルインデックスを指定する'GBIX'チャンクのダミー領域（4バイト）にバンク番号を書き込む。この領域をライブラリは参照しない。ユーザが値を読み取り利用する。このダミー領域の本来の目的はテクスチャの DMA 転送のためのアライメント調整。

```
['GBIX'][8(bytesize)][globalIndex][bankId]
```

パレットテクスチャでない場合 bankId 領域には 0 が書き込まれる。

アスキーフォーマットはすべてコンパイルして利用可能。C 言語表記の自由度を抑制しひとつのフォーマットとして固定するため NjDef.h に定義されるデファイン文字列を使った表現を利用する。

アスキーフォーマットには DEFAULT アスキーチャンク出力部分がある。

（例）

```
DEFAULT_START
#ifdef DEFAULT_TEXLIST_NAME
#define DEFAULT_TEXLIST_NAME texlist_testmodel01
#endif
DEFAULT_END
```

これはテストプログラムで複数のファイルをインクルードしてコンパイルして使う場合に先頭にインクルードされるファイルがデフォルトの名前でいつも呼び出せるようにするための記述。必ずしも必須のものではないが現行のアスキー出力ではこれを出力している。この部分に関する説明は以下ではない。アスキー

フォーマットにおける変数名はタイプとシーン名とモデルネームを'_'で構成される。例えばタイプが cell でシーンが test でモデルネームが body の場合変数名は cell_test_body となる。

バイナリフォーマットは他のデータと同様にメモリに格納されたデータダンプとそれらに含まれるポインタデータをロード時に実アドレスに変換するためのポインタリストをそれぞれ iff の Chunk 形式で格納する。

CellSprite 及び CellStream の attr に使われるフラグをアスキーフォーマット用に定義する。

FCS_ZER	Source Zero
FCS_ONE	Source One
FCS_OC	Source 'Other' Color
FCS_IOC	Source Inverse 'Other' Color
FCS_SA	Source SRC Alpha
FCS_ISA	Source Inverse SRC Alpha
FCS_DA	Source DST Alpha
FCS_IDA	Source Inverse DST Alpha
FCD_ZER	Destination Zero
FCD_ONE	Destination One
FCD_OC	Destination 'Other' Color
FCD_IOC	Destination Inverse 'Other' Color
FCD_SA	Destination SRC Alpha
FCD_ISA	Destination Inverse SRC Alpha
FCD_DA	Destination DST Alpha
FCD_IDA	Destination Inverse DST Alpha
FCA_VC	頂点カラー有効。
FCA_AL	USE ALPHA。
FCA_HF	Hフリップ。
FCA_VF	Vフリップ。
FCA_HI	Cell Hideフラグ。ONの時Cellの描画をしない。
FCA_TR	Time Repeat。設定された時間幅を中心としそれ以外の時間も周期を繰り返す。

ブレンディングファンクションは次のファンクションで与えられる。

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


5.1 texlist アスキーフォーマット nat ファイル

nat ファイルでは CellSprite が利用するテクスチャのリストである texlist を出力する。texlist のエントリ番号 (texId) を CellSprite に埋め込む事で CellSprite からテクスチャを指定する。パレットテクスチャの場合のパレットバンク番号も texlist に格納される。この場合のアスキー出力では PTEXN マクロを使う。パレットテクスチャは同じテクスチャでもバンクが異なる場合がありこの場合 texlist に別エントリとして配置する必要がある。texlist では同じパレットテクスチャが存在する場合これを連続で並べ二つ目のデータの一つ前のテクスチャと同じであることを示すフラグを立て二重の読み込みをなくしている。

(パレットテクスチャの出力例)

```
TEXTURENAME textures_MODEL[]
START
    PTEXN( "texturePAL1", 0, 20 ),   バンク番号が指定された。
    PTEXN( "texturePAL1", 1, 21 ),   バンク番号が指定された。
    TEXN( "texture1" ),
    PTEXN( "texturePAL2", 0, 63 ),   バンク番号が指定された。
    TEXN( "texture2" ),
END

TEXTURELIST texlist_MODEL
START
    TextureList textures_MODEL,
    TextureNum 5,
END
```

一つ目と二つ目のテクスチャは同じ ("texturePAL1") ですがバンク番号が異なる。このため連続に並べ二つ目の PTEXN の二つ目の引数にフラグ 1 が立ててある。同じテクスチャが三つあった場合も同様にフラグ 1 が立つ。詳細は Ninja2 Model 仕様書を参照してください。

(ファイルの出力例)

```
/* NAT 2.00.00 Ninja2AsciiTexture (SI) */

/* TEXTLIST      : texlist_same_heavywalk_body_3 n(9) */

TEXTURE_START

TEXTURENAME textures_same_heavywalk_body_3[]
START
    TEXN( "SAME_BETA1" ),      /* 0 32x32 565 TW MIP */
    TEXN( "sameMARK" ),       /* 1 128x128 565 TW MIP */
    TEXN( "same_eye1" ),      /* 2 64x64 565 TW MIP */
    TEXN( "same_eye2" ),      /* 3 64x64 565 TW MIP */
    TEXN( "sameha1" ),        /* 4 128x128 4444 TW MIP */
    TEXN( "sameha3" ),        /* 5 128x128 4444 TW MIP */
    TEXN( "samehada" ),       /* 6 128x128 565 TW MIP */
    TEXN( "samehada2" ),      /* 7 128x128 565 TW MIP */
    TEXN( "samekuti" ),       /* 8 64x64 565 TW MIP */
```

END

TEXTURELIST texlist_same_heavywalk_body_3

START

TextureList textures_same_heavywalk_body_3,

TextureNum 9,

END

TEXTURE_END

DEFAULT_START

#ifndef DEFAULT_TEXLIST_NAME

#define DEFAULT_TEXLIST_NAME texlist_same_heavywalk_body_3

#endif

DEFAULT_END

nat 出力で必要とされる NjDef.h のデファインリスト

TEXTURE_START	texlist アスキーチャンクの記述の始まりを示す。
TEXTURE_END	texlist アスキーチャンクの記述の終わりを示す。
TEXTURENAME	NJS_TEXNAME 構造体一次元配列名。
START	リストの開始。
END	リストの終了。
TEXN	16ビットテクスチャの時に使う NJS_TEXNAME 構造体のテクスチャエントリマクロ。
PTEXN	パレットテクスチャの時に使う NJS_TEXNAME 構造体のテクスチャエントリマクロ。連続フラグと、バンク番号が設定可能。
TEXTURELIST	NJS_TEXLIST 構造体名。
TextureList	texname リスト参照メンバ。
TextureNum	テクスチャの数。

< 解説 >

/* NAT 2.00.00 Ninja2AsciiTexture (SI) */

アスキーローダのための識別子として拡張子を大文字にしたものNATをコメント先頭につける。
次にバージョン名を入れる。ツールによって任意に決めてよい。
Ninja2AsciiTextureによりこのデータがテクスチャに関するものであることを示す。
(SI)はこのデータがSoftimageによって作られたことを示す。この他LW, MAX, MAYA, NJAなどの文字列が定義されている。Ce
llSpriteをサポートするツールはそれ以外の文字列でツールを表す略号を独自に定義し明記のこと。

/* TEXTLIST : texlist_same_heavywalk_body_3 n(9) */

ユーザの利便性を考えファイルの先頭にこのデータ内部のプログラムで使われるトップの変数名を記述する。
n(9)はテクスチャの数。

TEXTURE_START

テクスチャのアスキーチャンクの先頭を意味する。

TEXTURENAME textures_same_heavywalk_body_3[]

START

```
TEXN( "SAME_BETA1" ), /* 0 32x32 565 TW MIP */
TEXN( "sameMARK" ), /* 1 128x128 565 TW MIP */
TEXN( "same_eye1" ), /* 2 64x64 565 TW MIP */
TEXN( "same_eye2" ), /* 3 64x64 565 TW MIP */
TEXN( "sameha1" ), /* 4 128x128 4444 TW MIP */
TEXN( "sameha3" ), /* 5 128x128 4444 TW MIP */
TEXN( "samehada" ), /* 6 128x128 565 TW MIP */
TEXN( "samehada2" ), /* 7 128x128 565 TW MIP */
TEXN( "samekuti" ), /* 8 64x64 565 TW MIP */
```

END

NJS_TEXNAME構造体によるテクスチャの配列。

後ろのコメントはテクスチャの属性を示す。先頭からglobalIndex, テクスチャサイズ、pixelタイプ、TWIDDLED or RECTANGLE、ミップマップ有り無しを示す。

TEXNマクロはNJS_TEXNAME構造体の他のメンバにデフォルト値を設定し隠蔽する。

TEXTURELIST texlist_same_heavywalk_body_3

START

TextureList textures_same_heavywalk_body_3,

TextureNum 9,

END

NJS_TEXLIST構造体データを与える。

TEXTURE_END

テクスチャアスキーチャンクの終わりを示します。

5.2 CellSprite アスキーフォーマット spa ファイル

(ファイル出力例)

```
/* SPA 2.00.00 Ninja2AsciiCellSprite (Farlux) */
```

```
/* CELLSPRITE      : csprite_same_heavywalk_body_3 c(2) */
```

```
CSPRITE_START
```

```
CELLLISTNAME cell_same_heavywalk_body_3[]
```

```
START
```

```
CELLSTART
```

```
CellTexId      (0),  
CellAttr       (FCS_SA|FCD_ISA|FCA_VC|FCA_AL),  
CellOffset     (10.0f, 15.0f),  
CellSize       (64, 64),  
CellZAng       (0),  
CellPriority    (100),  
CellCenter     (0.5f, 0.5f),  
CellIVColor0   (255, 0, 100, 200),  
CellIVColor1   (255, 100, 10, 200),  
CellIVColor2   ( 0, 0, 100, 121),  
CellIVColor3   ( 0, 0, 100, 200),  
CellIUV0       ( 0.3f, 0.2f ),  
CellIUV2       ( 0.7f, 0.9f ),  
CELLEND
```

```
CELLSTART
```

```
< 省略 >
```

```
CELLEND
```

```
END
```

```
CELLSPRITE csprite_same_heavywalk_body_3
```

```
START
```

```
CellList       cell_same_heavywalk_body_3,  
CellNum        2,  
CSPosition     (0.0000000f, 0.000000f, 0.000000f),  
Create Date: 99/02/25 午後 4:52
```

```

CSZAngle      (90.000000f),
CSScale       (1.000000f, 1.000000f),
CSDiffuse     (0.5f, 0.6f, 0.7f, 0.8f),
CSSpecular    (6.0f, 0.7f, 0.78f, 0.78f)
END

```

```
CSPRITE_END
```

```

#ifndef DEFAULT_CSPRITE_NAME
#define DEFAULT_CSPRITE_NAME csprite_same_heavywalk_body_3,
#endif
---
```

spa 出力で必要とされる NjDef.h のデファインリスト

CELLSPRITE	CellSprite データの先頭を示します。
CSPRITE_START	CellSprite アスキーチャンクの記述の始まりを示します。
CSPRITE_END	CellSprite アスキーチャンクの記述の終わりを示します。
CELLLISTNAME	NJS_CELL 構造体一次元配列名。
CELLSTART	Cell データの開始。
CELLEND	Cell データの終了。
START	リストの開始。
END	リストの終了。
CellTexId	Cell の texId。
CellAttr	Cell の attr。
CellOffset	Cell の CellSprite 原点からのオフセット。
CellSize	Cell のサイズ。スケールも兼ねる。
CellZAng	Cell の Z 軸回転。Sint16。
CellPriority	Cell 間のプライオリティ。Sint16。
CellCenter	Cell の回転中心。CellSize に対する比で
CellVColor0	Cell の第 0 頂点カラー。
CellVColor1	Cell の第 1 頂点カラー。
CellVColor2	Cell の第 2 頂点カラー。
CellVColor3	Cell の第 3 頂点カラー。
CellUV0	Cell の第 0 頂点 UV。
CellUV2	Cell の第 2 頂点 UV。
CellList	NJS_CELL 構造体一次元配列名。
CellNum	Cell の数。
CSPosition	CellSprite のポジション。
CSZAngle	CellSprite の z 軸回転。
CSScale	CellSprite のスケール。
CSDiffuse	CellSprite の diffuse カラー。 あり。
CSSpecular	CellSprite の Specular カラー。アルファの代わりに exponent0 ~ 16 が入る。

CSDiffuse による アニメーションをする場合は Cell の attr に USE ALPHA フラグである NJD_FCA_AL を設定する必要がある。

< 解説 >

```
/* SPA 2.00.00 Ninja2AsciiCellSprite (Farlux) */
```

アスキーローダのための識別子として拡張子を大文字にしたものSPAをコメント先頭につける。

次にバージョン名を入れる。ツールによって任意に決めて問題なし。

Ninja2AsciiCellSpriteによりこのデータがセルスプライトに関するものであることを示す。

(Farlux)はこのデータがFarluxによって作られたことを示す。

```
/* CELLSPRITE      : csprite_same_heavywalk_body_3 c(2) */
```

ユーザの利便性を考えファイルの先頭にこのデータ内部のプログラムで使われるトップの変数名を記述する。

c(2)はCellSpriteの数。

CSPRITE_START

CellSpriteのアスキーチャンクの先頭を意味する。

CELLLISTNAME cell_same_heavywalk_body_3[]

START

CELLSTART

```
CellTexId      (0),
CellAttr       (FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
CellOffset     (10.0f, 15.0f),
CellSize       (64, 64),
CellZAng       (0),
CellPriority    (100),
CellCenter     (0.5f, 0.5f),
CellIVColor0   (255, 0, 100, 200),
CellIVColor1   (255, 100, 10, 200),
CellIVColor2   ( 0, 0, 100, 121),
CellIVColor3   ( 0, 0, 100, 200),
CellIUV0       ( 0.3f, 0.2f ),
CellIUV2       ( 0.7f, 0.9f ),
```

CELLEND

CELLSTART

< 省略 >

CELLEND

END

NJS_CELL構造体によるCellの配列。

CELLSPRITE csprite_same_heavywalk_body_3

START

```

CellList      cell_same_heavywalk_body_3,
CellNum       2,
CSPosition    (0.0000000f, 0.000000f, 0.000000f),
CSZAngle      (90.000000f),
CSScale       (1.000000f, 1.000000f),
CSDiffuse     (0.5f, 0.6f, 0.7f, 0.8f),
CSSpecular    (6.0f, 0.7f, 0.78f, 0.78f)
END

```

NJS_CELLSPRITE構造体データを与える。

CSPRITE_END

CellSpriteのアスキーチャンクの終わりを意味する。

```

#ifndef DEFAULT_CSPRITE_NAME
#define DEFAULT_CSPRITE_NAME csprite_same_heavywalk_body_3,
#endif

```

spa ファイルは、nat ファイルを取り込むことができる。

(nat を取り込んだファイル例)

```

/* SPA 2.00.00 Ninja2AsciiCellSprite (Farlux) */

/* TEXTLIST      : texlist_same_heavywalk_body_3 n(9) */
/* CELLSPRITE    : csprite_same_heavywalk_body_3 c(2) */

TEXTURE_START

TEXTURENAME textures_same_heavywalk_body_3[]
START
    TEXTN( "SAME_BETA1" ),      /* 0 32x32 565 TW MIP */
    TEXTN( "sameMARK" ),       /* 1 128x128 565 TW MIP */
    TEXTN( "same_eye1" ),      /* 2 64x64 565 TW MIP */
    TEXTN( "same_eye2" ),      /* 3 64x64 565 TW MIP */
    TEXTN( "sameha1" ),        /* 4 128x128 4444 TW MIP */
    TEXTN( "sameha3" ),        /* 5 128x128 4444 TW MIP */
    TEXTN( "samehada" ),       /* 6 128x128 565 TW MIP */
    TEXTN( "samehada2" ),      /* 7 128x128 565 TW MIP */

```



```

        TEXN( "samekuti" ),          /* 8 64x64 565 TW MIP */
END

TEXTURELIST texlist_same_heavywalk_body_3
START
TextureList textures_same_heavywalk_body_3,
TextureNum 9,
END

TEXTURE_END

CSPRITE_START

CELLLISTNAME cell_same_heavywalk_body_3[]
START

CELLSTART
CellTexId      (0),
CellAttr       (FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
CellOffset     (10.0f, 15.0f),
CellSize       (64, 64),
CellZAng       (0),
CellPriority    (100),
CellCenter     (0.5f, 0.5f),
CellIVColor0   (255, 0, 100, 200),
CellIVColor1   (255, 100, 10, 200),
CellIVColor2   ( 0, 0, 100, 121),
CellIVColor3   ( 0, 0, 100, 200),
CellIUV0       ( 0.3f, 0.2f ),
CellIUV2       ( 0.7f, 0.9f ),
CELLEND

CELLSTART
<省略>
CELLEND

END

CELLSPRITE csprite_same_heavywalk_body_3
START
CellList       cell_same_heavywalk_body_3,

```

```
CellNum      2,  
CSPosition   (0.000000f, 0.000000f, 0.000000f),  
CSZAngle     (90.000000f),  
CSScale      (1.000000f, 1.000000f),  
CSDiffuse    (0.5f, 0.6f, 0.7f, 0.8f),  
CSSpecular   (6.0f, 0.7f, 0.78f, 0.78f)  
END
```

```
CSPRITE_END
```

```
DEFAULT_START
```

```
#ifndef DEFAULT_TEXLIST_NAME  
#define DEFAULT_TEXLIST_NAME texlist_same_heavywalk_body_3  
#endif
```

```
#ifndef DEFAULT_CSPRITE_NAME  
#define DEFAULT_CSPRITE_NAME csprite_same_heavywalk_body_3,  
#endif
```

```
DEFAULT_END
```

5.3 Motion アスキーフォーマット nam ファイル

モーション構造体は次の通り。NjMotion.h に定義される。通常の 3D モデルに対するモーションと同じデータ構造を利用する。

```
typedef struct {
void          *mdata;      /* モーションデータ          */
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 にはモーションデータの要素数に従い NJS_MDATA1 ~ 5 構造体が設定される。CellSprite ではモーションの要素数は最大四つなので NJS_MDATA1 ~ 4 が使われる。Ninja のモーション構造は単一のデータ構造でさまざまなデータを扱うことができるように void のポインタにフックされたデータを type に設定されたフラグで識別する方法を取っている。この方法でモデル、ライト、カメラ、CellSprite のモーションを表現する。nbFrame にはモーションのフレーム数が入る。type には次に定義される四つの成分のモーションフラグが設定される。CellSprite には モーションを可能とするための専用の argb モーション成分が用意される。

```
#define NJD_MTYPE_POS_0          (1<<0) /* NJS_MKEY_F を利用 */
#define NJD_MTYPE_ANG_1         (1<<1) /* NJS_MKEY_A を利用 */
#define NJD_MTYPE_SCL_2         (1<<2) /* NJS_MKEY_F を利用 */
#define NJD_MTYPE_RGB_8         (1<<9) /* NJS_MKEY_UI32 を利用 */
```

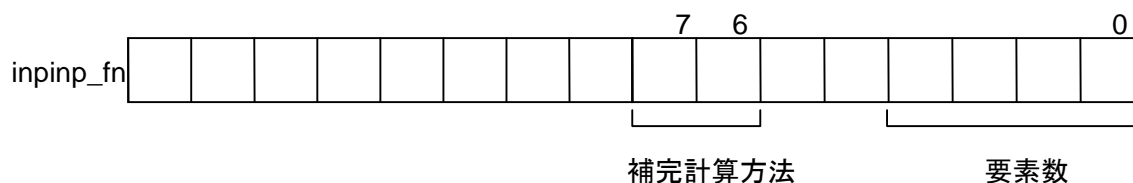
NJD_MTYPE_ANG_1、NJD_MTYPE_SCL_2 はモデルの場合と CellSprite の場合で使うキー構造体が異なる。NJD_MTYPE_0 の Z 値は CellSprite のプライオリティになる。これは CellSprite のモーションをモデルで利用することはないであろうという観点から同一定義でそれぞれの場合に使い分けている。これは必要以上に type のビット定義を消費しないことを目的とする。上記の define のコメントはモデルの場合のもの。CellSprite では下記のようになる。NJD_MTYPE_RGB_8 は CellSprite 専用のため定義を使い分けることなく単一の定義。

pos	NJD_MTYPE_POS_0	(1<<0)	NJS_MKEY_F構造体 xyz成分。zは2Dの場合はプライオリティ。
angle	NJD_MTYPE_ANG_1	(1<<1)	NJS_MKEY_A1構造体 z軸回転。
sx, sy	NJD_MTYPE_SCL_2	(1<<2)	NJS_MKEY_F2構造体。x, y成分のみのスケール。
argb	NJD_MTYPE_RGB_8	(1<<9)	NJS_MKEY_UI32構造体。ARGB成分。

inp_fn には補完方法とモーション要素数が格納される。補完計算方法を inp_fn のビット 7～8 で指定する。デフォルトは LINER。NjMotion.h にフラグが定義される。

```
#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          */
```

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。



NJS_DATA1～4 構造体を以下に示す。

```
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;
```

すべての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。下記四つのデータが NJS_MDATA1～4 の p[i] にフックされる。

NJS_MKEY_F 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;
```

xyz トランスレーションモーション成分に使います。z はプライオリティ。

NJS_MKEY_A1 構造体

```

typedef struct {
    Uint32    keyframe;    /* キーフレーム番号          */
    Sint32    key;         /* 符号有り int32 型キー値    */
} NJS_MKEY_A1;

```

CellSprite の z 軸回転モーションに使う。

NJS_MKEY_F2 構造体

```

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

```

二次元の x, y 成分のみのスケールモーション成分に使う。

NJS_MKEY_UI32 構造体

```

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;

```

ARGB モーション成分に使用します。

モーションの詳細については Ninja2 Motion 仕様書を参照のこと。モーションアスキー出力では NjDef.h に定義されるマクロで上記構造体が整形されて出力される。

(ファイル出力例) 4 つの成分がある場合

```
/* CSNAM 2.00.00 Ninja2AsciiCellSpriteMotion (Farlux) */
```

```
/* CSPRITE MOTION : motion_csprite_spec */
```

```
CSPRITE_MOTION_START
```

```
CSPOSITION pos_csprite_spec[]
```

```
START
```

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

```
    .....
```

```
END
```

```
CSANGLE angle_csprite_spec[]
```

```
START
```

```
    MKEYA1( 0,    90.00000F ),
```

```
    .....
```

```
END
```

```
CSSCALE scl_csprite_spec[]
```

START

MKEYF2(0, 1.000000F, -1.000000F),

.....

END

CSCOLOR col_csprite_spec[]

START

MKEYUI32(0, ARGB(255, 255, 255, 255)),

.....

END

MDATA4 mdata_csprite_spec01[]

START

pos_csprite_spec, angle_csprite_spec, scl_csprite_spec, col_csprite_spec, 100, 100, 10, 10,

END

CSMOTION motion_csprite_spec[]

START

MdadaArray mdata_csprite_spec01,

MFrameNum 100,

MoitonBit 0x0107,

InterpolFct 0x0000,

END

CSPRITE_MOTION_END

(ファイル出力例) ローテーションと のモーシヨンの場合

/* CSNAM 2.00.00 Ninja2AsciiCellSpriteMotion (Farlux) */

/* CSPRITE MOTION : motion_csprite_spec */

CSPRITE_MOTION_START

CSANGLE angle_csprite_spec[]

START

MKEYA1(0, 90.00000F),

.....

END

CSCOLOR col_csprite_spec[]

START

```

MKEYUI32( 0,  ARGB(255, 255, 255, 255) ),
.....
END

MDATA2 mdata_csprite_spec01[]
START
    angle_csprite_spec, col_csprite_spec, 100, 10,
END

CSMOTION motion_csprite_spec[]
START
MdataArray    mdata_csprite_spec01,
MFrameNum     100,
MotionBit     0x0102,
InterpolFct   0x0000,
END

CSPRITE_MOTION_END

DEFAULT_START

#ifdef DEFAULT_MOTION_NAME
#define DEFAULT_MOTION_NAME motion_csprite_spec
#endif

DEFAULT_END

```

CellSprite 用 nam 出力で必要とされる NjDef.h のデファインリスト

CSPRITE_MOTION_START	モーションアスキーチャンクの記述の始まりを示します。
CSPRITE_MOTION_END	モーションアスキーチャンクの記述の終わりを示します。
CSMOTION	NJS_MOTION 構造体。
CSPOSITION	CellSprite の位置モーション。
CSANGLE	CellSprite の z 軸回転。
CSSCALE	CellSprite のスケール。
CSCOLOR	CellSprite の diffuse color
MDATA1	モーション成分が 1 つの場合
MDATA2	モーション成分が 2 つの場合
MDATA3	モーション成分が 3 つの場合
MDATA4	モーション成分が 4 つの場合
START	リストの開始。
END	リストの終了。
MdataArray	モーションデータポインタ。
MframeNum	モーションフレーム数。
MotionBit	格納されるモーション成分の種類。この部分のフラグが数値べた書きで文字列で表現されていないのは従来の仕様を変更しないための措置です。
InterpolFct	モーション補完の種類

MKEYF	Float の xyz 成分用のキーフレームマクロ。
MKEYA1	Angle 成分用のキーフレームマクロ。
MKEYF2	Float の xy 成分用のキーフレームマクロ。
MKEYUI32	Uint32 成分用のキーフレームマクロ。

< 解説 >

(ファイル出力例) 4 つの成分がある場合

```
/* CSNAM 2.00.00 Ninja2AsciiCellSpriteMotion (Farlux) */
```

アスキーローダのための識別子として拡張子を大文字にしたもののNAMをコメント先頭につける。しかしここでは通常のnamとCellSprite用のnamを識別するためにCSNAMという文字列を置く。

次にバージョン名をいれます。ツールによって任意に決めて問題なし。

Ninja2AsciiCellSpriteMotionによりに関するものであることを示す。

(Farlux)はこのデータがFarluxによって作られたことを示す。

```
/* CSPRITE MOTION : motion_csprite_spec */
```

ユーザの利便性を考えファイルの先頭にこのデータ内部のプログラムで使われるトップの変数名を記述する。

CSPRITE_MOTION_START

CellSprite用Motionのアスキーチャンクの先頭を意味する。

CSPOSITION pos_csprite_spec[]

START

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

.....

END

position成分データ。CellSpriteのx, y座標を決める。z値はプライオリティ。

CSANGLE angle_csprite_spec[]

START

MKEYA1(0, 90.00000F),

.....

END

z 軸回転成分。

CSSCALE scl_csprite_spec[]

START

MKEYF2(0, 1.000000F, -1.000000F),

.....

END

スケール成分。

```
CSCOLOR col_csprite_spec[]
START
    MKEYUI32( 0,  ARGB(255, 255, 255, 255) ),
    .....
END
```

diffuse成分。

```
MDATA4 mdata_csprite_spec01[]
START
    pos_csprite_spec, angle_csprite_spec, scl_csprite_spec, col_csprite_spec, 100, 100, 10, 10,
END
```

モーション成分が四つなのでMDATA4を使う。各成分を格納する。

```
CSMOTION motion_csprite_spec[]
START
MdadaArray    mdata_csprite_spec01,
MFrameNum     100,
MoitonBit     0x0107,
InterpolFct    0x0000,
END
```

モーションデータを格納する。MotionBitはNJD_MTYPE_POS_0, NJD_MTYPE_ANG_1, NJD_MTYPE_SCL_2, NJD_MTYPE_RGB_8の四つのフラグをセットしている。InterpolFctは補完が線形にされることを示す。

CSPRITE_MOTION_END

CellStreamのアスキーチャンクの終わりを意味する。

```
( ファイル出力例 ) ローテーションと のモーションの場合
/* CSNAM 2.00.00 Ninja2AsciiCellSpriteMotion (Farlux) */

/* CSPRITE MOTION : motion_csprite_spec */
```

CSPRITE_MOTION_START

```
CSANGLE angle_csprite_spec[]
START
```

```

MKEYA1( 0, 90.00000F ),
.....
END

CSCOLOR col_csprite_spec[]
START
MKEYUI32( 0, ARGB(255, 255, 255, 255) ),
.....
END

```

```

MDATA2 mdata_csprite_spec01[]
START
angle_csprite_spec, col_csprite_spec, 100, 10,
END

```

モーション成分が二つなのでMDATA2を使う。

```

CSMOTION motion_csprite_spec[]
START
MdadaArray    mdata_csprite_spec01,
MFrameNum     100,
MoitonBit     0x0102,
InterpolFct   0x0000,
END

```

モーションデータを格納する。MotionBitはNJD_MTYPE_ANG_1, NJD_MTYPE_RGB_8の二つのフラグをセットしている。Interpol Fctは補完が線形にされることを示す。

```

CSPRITE_MOTION_END

```

5.4 CellStream アスキーフォーマット csa ファイル

出力データは Sint32 の配列に配置される都合により Float データはすべてヘキサの値になる。コンバータはアスキー出力においてコメントとして Float 値を出力できる必要がある。Float 値は次の C プログラムで簡単にヘキサに変更できる。

```
#include <stdlib.h>

main(Int argc, char **argv)
{
    if (1 < argc) {
        Float      f = (Float)atof(argv[1]);
        unsigned long ul = *((long *)&f);
        Float      fb = *((Float *)&ul);

        printf("argv %s f %f hex %x fb %f\n", argv[1], f, ul, fb);
    } else {
        printf("Usage : %s <Float value>\n", argv[0]);
    }
}
```

(ファイル出力例)

```
/* CSA 2.00.00 Ninja2AsciiCellStream (Farlux) */

/* CELLSTREAM      : cstream_same_heavywalk_body_3 */

CSTREAM_START

CELLSTREAMDATA csdata_same_heavywalk_body_3[]
START
    CCnk_TS(0, 27), /* timestamp 0 */
    CCnk_CE(0, 16, FCB_IA0|FCB_MO1|FCB_PR2|FCB_CE3|FCB_VC4|FCB_UV5),
    Bk_IA0(10, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
    Bk_MO1PR2(0x41200000, 0x41200000, /* coxy 10.0F, 10.0F */
              0x42800000, 0x42800000, /* csxy 64, 64 */
              90.0F, 1),
    Bk_CE3(0x3f000000, 0x3f000000), /* 0.5F, 0.5F */
    Bk_VC4_0(255, 255, 90, 100),
    Bk_VC4_1(255, 200, 130, 100),
    Bk_VC4_2(255, 150, 190, 100),
```

```

Bk_VC4_3(255, 100, 200, 100),
Bk_UV5(0x3e99999a, 0x3ecccccd, /* csuv0 0.3F, 0.4F */
        0x3f000000, 0x3f19999a), /* csuv2 0.5F, 0.6F */
CCnk_CE(10, 6, FCB_IA0|FCB_MO1),
Bk_IA0(11, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
Bk_MO1(0x41200000, 0x41200000, /* coxy 10.0F, 10.0F */
        0x42800000, 0x42800000, /* csxy 64, 64 */
        90.0F),
CCnk_CE(1, 2, FCB_IA0|FCB_PR2),
Bk_IA0(12, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
Bk_PR2(2),
CCnk_TS(1, 4), /* timestamp 1 */
CCnk_TD(99, 10),
CCnk_AT(100, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
CCnk_CZ(100, 45.0f),
CCnk_CP(101, 10),
CCnk_TS(2, 4), /* timestamp 2 */
CBCnk_AD(16, 1), /* laddr=16, lsize=1 */
0x000100ff,
CBCnk_ED(2, 0, 7),
0x000200ff, 0x0300ffff, 0x0ff0ff00, 0xffffffff, (この例では数字は適当。)
0x0f0f0f0f, 0xeeeeeeee, 0x1e1e1e1e,
CCnk_UE(3, 0), /* user event longsize=3, eventId=0 */
0x000000ff, 0xffffffff, 0xf0f0f0f0f,
CCnk_TE(100), /* timestamp 3 */

```

END

CELLSTREAM cstream_same_heavywalk_body_3

START

```

CStreamData      csdata_same_heavywalk_body_3,
CSCurrent        csdata_same_heavywalk_body_3,
CSTimeOffset     0,
CSTimeMax        100,

```

END

CSTREAM_END

```
#ifndef DEFAULT_CSTREAM_NAME
```

```
#define DEFAULT_CSTREAM_NAME cstream_same_heavywalk_body_3,
```

```
#endif
```

csa 出力で必要とされる NjDef.h のデファインリスト

フォーマットに関わるもの

CELLSTREAM	CellStream データの先頭を示す。
CSTREAM_START	CellStream アスキーチャンクの記述の始まりを示す。
CSTREAM_END	CellStream アスキーチャンクの記述の終わりを示す。
CELLSTREAMDATA	Sint32 一次元配列に CellChunk によって構成された CellStream データ。
START	リストの開始。
END	リストの終了。
CStreamData	CellStream データ。
CSCurrent	CellStream データのカレントタイムスタンプ CellChunk のポインタ。CellStream 実行により書き換えられるので注意が必要。
CSTimeOffset	開始時間のシフト。マイナス時間は現時点では考えない。
CSTimeMax	最大フレーム時間。フレームは 0 から始まるものとし 0 ~ 100 フレームの場合 100 が入る。フレーム数 101 ではない。

CellChunk timestamp に関するもの

CCnk_TE(_time)	16bit _time : 1 フレームを基準とする時間 タイムスタンプエンド。CellStream データ最後に配置されるタイムスタンプチャンク。
CCnk_TS(_time, _lsize)	16bit _time : 1 フレームを基準とする時間 12bit _lsize : 次のタイムスタンプチャンクまでのロングワードオフセット。 ロングワードオフセットは、タイムスタンプ単位でデータをスキップする場合に使う。 例えばある CellStream (アニメーションパターン) においてすべて同じ Cell ブロックの書き換えをすれば任意のタイムスタンプに移動してもアニメーションは正常に実行可能であるためこの場合に利用する。

CellChunkCell のデータブロックフラグ。CellChunkCell のアスキーデファインである CCnk_CE の_hbits に格納される。

FCB_IA0	texId と attr のブロック 0
FCB_MO1	モーションのブロック 1
FCB_PR2	プライオリティのブロック 2
FCB_CE3	センターのブロック 3
FCB_VC4	頂点カラーのブロック 4
FCB_UV5	UV 値のブロック 5

CellChunk Cell に関するもの

ここで Floathex とは Sint32 配列上に Float 値を格納するため Float 値をヘキサ表現した数字。

CCnk_CE(_eid, _lsize, _hbits)	16bit _eid : Cell のエントリ ID。 4bit _lsize : ロングワードでのデータサイズ。 8bit _hbits : データブロックフラグ。 CellStream の Sint32 一次元配列上でアニメーションデータを与える。
Bk_IA0(_tid, _att)	16bit _tid : texlist に対する texId。 16bit _att : アトリビュートフラグ。
Bk_M01(_cox, _coy, _csx, _csy, _czang)	Floathex _cox : CellSprite 原点からのオフセット x。 Floathex _coy : CellSprite 原点からのオフセット y。 Floathex _csx : Cell のサイズ x。 Floathex _csy : Cell のサイズ y。 16bit _czang : z 軸回転。Float 値で設定するとこのマクロの中で 16 ビット精度に変換。 データはロングワード表現のため _czang の上位 16bit のパディングが付く。マクロ上からは _czang の値は度で表現される。
Bk_M01PR2(_cox, _coy, _csx, _csy, _czang, _cp)	Floathex _cox : CellSprite 原点からのオフセット x。 Floathex _coy : CellSprite 原点からのオフセット y。 Floathex _csx : Cell のサイズ x。 Floathex _csy : Cell のサイズ y。 16bit _czang : z 軸回転。Float 値で設定するとこのマクロの中で 16 ビット精度に変換。マクロ上からは _czang の値は度で表現される。 16bit _cp : cell 間のプライオリティ設定。 _cp, _czang を同時に使う場合にこのマクロを利用する。_cp を同時に設定することでパディングは発生しない。
Bk_PR2(_cp)	16bit _cp : cell 間のプライオリティ設定。
Bk_CE3(_cx, _cy)	Floathex _cx : cell のセンター x。 Floathex _cy : cell のセンター y。 cell size に対する比でセンターを表す。中心は 0.5f, 0.5f となる。
Bk_VC4_0(_a, _r, _g, _b)	8bit _a, _r, _g, _b : 頂点 0 の頂点カラー。
Bk_VC4_1(_a, _r, _g, _b)	8bit _a, _r, _g, _b : 頂点 1 の頂点カラー。
Bk_VC4_2(_a, _r, _g, _b)	8bit _a, _r, _g, _b : 頂点 2 の頂点カラー。
Bk_VC4_3(_a, _r, _g, _b)	8bit _a, _r, _g, _b : 頂点 3 の頂点カラー。
Bk_UV5(_u0, _v0, _u2, _v2)	Floathex _u0, _v0, _u2, _v2 頂点 0 の頂点 UV と頂点 2 の頂点 UV。

CellChunk Sint16 Data に関するもの

ロングワードバウンダリ CellStream データに 16 ビットデータを書き込む。

CCnk_TD(_eid, _tid)	16bit _eid : cell のエントリ ID。 16bit _tid : texlist の texId。
CCnk_AT(_eid, _att)	16bit _eid : cell のエントリ ID。 16bit _att : アトリビュートフラグ。
CCnk_CZ(_eid, _cz)	16bit _eid : cell のエントリ ID。 16bit _cz : z 軸回転。
CCnk_CP(_eid, _cp)	16bit _eid : cell のエントリ ID。 16bit _cp : cell priority。

Cell Chunk User Event に関するもの

CCnk_UE(_longsize, _eventId)	12bit _longsize : ユーザデータ領域のロングワードサイズ。 16bit _eventId : コールバックテーブルに登録される eventId。
------------------------------	---

Cell Burst Chunk に関するもの

CBCnk_AD(_laddr, _lsize)	16bit _laddr :ワークバッファ先頭からのロングワードアドレス。 12bit _lsize :格納される書き込みデータのロングワードサイズ。
CBCnk_ED(_eid, _off, _lsize)	16bit _eid : エントリ ID。 8bit _off : Cell 構造体の先頭からのオフセット。 4bit _lsize : 格納される書き込みデータのロングワードサイズ。

< 解説 >

Sint32 配列にすべてのデータを格納するため Float 値をアスキー表現するためにヘキサを使う必要がある。そのためその値そのものが読めないものとなっているため対象となるデータの後ろには実際の Float 値のコメントがつけられる。

```
/* CSA 2.00.00 Ninja2AsciiCellStream (Farlux) */
```

アスキーローダのための識別子として拡張子を大文字にしたものCSAをコメント先頭につける。

次にバージョン名をいれる。ツールによって任意に決めて問題なし。

Ninja2AsciiCellStreamによりこのデータがテクスチャに関するものであることを示す。

(Farlux)はこのデータがFarluxによって作られたことを示す。

```
/* CELLSTREAM      : cstream_same_heavywalk_body_3 */
```

ユーザの利便性を考えファイルの先頭にこのデータ内部のプログラムで使われるトップの変数名を記述する。

CSTREAM_START

CellStreamのアスキーチャンクの先頭を意味する。

CellStreamデータのための一次元配列。

```
CELLSTREAMDATA csdata_same_heavywalk_body_3[]
```

START

```
CCnk_TS(0, 27), /* timestamp 0 */
```

0フレームのタイムスタンプ、次のタイムスタンプまでのロングワードオフセットが27であることを示す。

```
CCnk_CE(0, 16, FCB_IA0|FCB_MO1|FCB_PR2|FCB_CE3|FCB_VC4|FCB_UV5),
```

entryId=0、16ロングワードのデータサイズ、データブロックはブロック 0 ~ 5 までであることを示す。

```
Bk_IA0(10, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
```

texId = 10, プレンディングアルファモードの設定と頂点カラー、 ありを示す。

```
Bk_MO1PR2(0x41200000, 0x41200000, /* coxy 10.0F, 10.0F */
```

```
0x42800000, 0x42800000, /* csxy 64, 64 */
```


90.0F, 1),

ブロック 1, 2 を同時に出力するケースのためBk_M01PR2を利用する。coxy, csxyは、ヘキサ表示されているデータの意味を示す。zangは、Floatで度で与えられこれは内部で整数 16 ビットに変換格納される。

Bk_CE3(0x3f000000, 0x3f000000), /* 0.5F, 0.5F */

Cellのセンターを与える。

Bk_VC4_0(255, 255, 90, 100),

Bk_VC4_1(255, 200, 130, 100),

Bk_VC4_2(255, 150, 190, 100),

Bk_VC4_3(255, 100, 200, 100),

頂点カラーを与える。4つのマクロで必ず構成され一つだけ等の使い方はできない。

Bk_UV5(0x3e99999a, 0x3ecccccd, /* csuv0 0.3F, 0.4F */

0x3f000000, 0x3f19999a), /* csuv2 0.5F, 0.6F */

CellのUV値を与える。コメントのcsuv0, csuv2はヘキサ表示されているデータの意味を与える。

次のCellデータの処理に入る。

CCnk_CE(10, 6, FCB_IA0|FCB_M01),

entryId=10, データサイズが6ロングワードでブロック0, 1が格納されることを示す。

Bk_IA0(11, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),

texId=11, ブレンディングアルファモードの設定と頂点カラー、ありを示す。

Bk_M01(15.0F, 15.0F, 64, 64, 85.0F),

ブロック2のCell priorityがないためBk_M01を使う。

次のCellデータの処理に入る。

CCnk_CE(1, 2, FCB_IA0|FCB_PR2),

entryId=1, データサイズ2ロングワード、ブロック0、2があることを示す。

Bk_IA0(12, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),

texId=12, ブレンディングアルファモードの設定と頂点カラー、ありを示す。

Bk_PR2(2),

ブロック 1 がないのでBk_PR2を使う。

以下Sint16 dataのためのCell Chunk及びCell Burst Chunkの例。

```
CCnk_TS(1, 4), /* timestamp 1 */
```

1フレームのタイムスタンプ、次のタイムスタンプまでのロングワードオフセットが4であることを示す。

```
CCnk_TD(99, 10),
```

entryId=99のCell構造体のtexIdを10にする。

```
CCnk_AT(100, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
```

entryId=100のCell構造体のattrブレンディングアルファモードの設定と頂点カラー、 ありを設定する。

```
CCnk_CZ(100, 45.0f),
```

entryId=100のCell構造体のczangに45度を設定する。

```
CCnk_CP(101, 10),
```

entryId=101のCell構造体のcell priorityに10を書き込む。

```
CCnk_TS(2, 4), /* timestamp 2 */
```

2 フレームのタイムスタンプ、次のタイムスタンプまでのロングワードオフセットが4であることを示す。

```
CBCnk_AD(16, 1), /* laddr=16, lsize=1 */
```

```
0x000100ff,
```

ワークバッファ先頭より16ロングワードのアドレスの位置に 1 ロングワードを書き込む。

```
CBCnk_ED(2, 0, 7),
```

```
0x000200ff, 0x0300ffff, 0x0ff0ff00, 0xffffffff, (この例では数字は適当。)
```

```
0x0f0f0f0f, 0xeeeeeeee, 0x1e1e1e1e,
```

entryId=2のCell構造体のオフセット 0 から 7 ロングワードを書き込む。

```
CCnk_UE(3, 0), /* user event longsize=3, eventId=0 */
```

```
0x000000ff, 0xffffffff, 0xf0f0f0f0f,
```

UserEventの登録。longsize=3のため三つの値が並ぶ。eventIdが0に登録されたコールバックルーチンが呼び出される。

```
CCnk_TE(100), /* timestamp 3 */
```

100フレームのタイムスタンプ。CellStreamデータの最後であることを示す。

```
END
```

```
CELLSTREAM cstream_same_heavywalk_body_3
```

```
START
```

```
CStreamData      csdata_same_heavywalk_body_3,
```

```
CSCurrent        csdata_same_heavywalk_body_3,
```

```
CSTimeOffset      0,
```

```
CSTimeMax         100,
```

```
END
```

NJS_CELL_STREAM構造体を与える。CSCurrentは現在実行中のタイムスタンプのチャンクのポインタを格納する。初期値はCellStreamデータの先頭となるためCStreamDataのものと同一になる。

```
CSTREAM_END
```

CellStreamのアスキーチャンクの終わりを意味する。

csa ファイルは、nam ファイルを取り込むことができる。

(nam を取り込んだファイル出力例)

```
/* CSA 2.00.00 Ninja2AsciiCellStream (Farlux) */
```

```
/* CSPRITE MOTION : motion_csprite_spec */
```

```
/* CELLSTREAM      : cstream_same_heavywalk_body_3 */
```

```
CSPRITE_MOTION_START
```

```
CSANGLE angle_csprite_spec[]
```

```
START
```

```
    MKEYA1( 0,  90.00000F ),
```

```
    .....
```

```
END
```

```

CSCOLOR col_csprite_spec[]
START
    MKEYUI32( 0,  ARGB(255, 255, 255, 255) ),
    .....
END

MDATA2 mdata_csprite_spec01[]
START
    angle_csprite_spec, col_csprite_spec, 100, 10,
END

CSMOTION motion_csprite_spec[]
START
MdadaArray    mdata_csprite_spec01,
MFrameNum     100,
MoitonBit     0x0102,
InterpolFct   0x0000,
END

CSPRITE_MOTION_END

CSTREAM_START

CELLSTREAMDATA csdata_same_heavywalk_body_3[]
START
    CCnk_TS(0, 27), /* timestamp 0 */
    CCnk_CE(0, 16, FCB_IA0|FCB_M01|FCB_PR2|FCB_CE3|FCB_VC4|FCB_UV5),
    Bk_IA0(10, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
    Bk_M01PR2(0x41200000, 0x41200000, /* coxy 10.0F, 10.0F */
              0x42800000, 0x42800000, /* csxy 64, 64 */
              90.0F, 1),
    Bk_CE3(0x3f000000, 0x3f000000), /* 0.5F, 0.5F */
    Bk_VC4_0(255, 255, 90, 100),
    Bk_VC4_1(255, 200, 130, 100),
    Bk_VC4_2(255, 150, 190, 100),
    Bk_VC4_3(255, 100, 200, 100),
    Bk_UV5(0x3e99999a, 0x3ecccccd, /* csuv0 0.3F, 0.4F */
           0x3f000000, 0x3f19999a), /* csuv2 0.5F, 0.6F */
    CCnk_CE(10, 6, FCB_IA0|FCB_M01),
    Bk_IA0(11, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
    Bk_M01(0x41200000, 0x41200000, /* coxy 10.0F, 10.0F */

```

```

        0x42800000, 0x42800000, /* csxy 64, 64 */
        90.0F),
    CCnk_CE(1, 2, FCB_IA0|FCB_PR2),
    Bk_IA0(12, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
    Bk_PR2(2),
    CCnk_TS(1, 4), /* timestamp 1 */
    CCnk_TD(99, 10),
    CCnk_AT(100, FCS_SA|FCD_ISA|FCA_VC|FCA_AL),
    CCnk_CZ(100, 45.0f),
    CCnk_CP(101, 10),
    CCnk_TS(2, 4), /* timestamp 2 */
    CBCnk_AD(16, 1), /* laddr=16, lsize=1 */
    0x000100ff,
    CBCnk_ED(2, 0, 7),
    0x000200ff, 0x0300ffff, 0x0ff0ff00, 0xffffffff, (この例では数字は適当です。)
    0x0f0f0f0f, 0xeeeeeeee, 0x1e1e1e1e,
    CCnk_UE(3, 0), /* user event longsize=3, eventId=0 */
    0x000000ff, 0xffffffff, 0xf0f0f0f0f,
    CCnk_TE(100), /* timestamp 3 */

```

END

CELLSTREAM cstream_same_heavywalk_body_3

START

```

CStreamData    csdata_same_heavywalk_body_3,
CSCurrent      csdata_same_heavywalk_body_3,
CSTimeOffset   0,
CSTimeMax      100,

```

END

CSTREAM_END

DEFAULT_START

```

#ifndef DEFAULT_MOTION_NAME
#define DEFAULT_MOTION_NAME motion_csprite_spec
#endif

#ifndef DEFAULT_CSTREAM_NAME
#define DEFAULT_CSTREAM_NAME cstream_same_heavywalk_body_3,
#endif

```

DEFAULT_END

5.5 バイナリフォーマット

データ別拡張子

	バイナリフォーマット拡張子	バイナリチャンク名
Texlist	.njl	'NJTL'
CellSprite	.spj	'NJSP'
CellStream	.csj	'NJCS'
CellSprite Motion	.njm	'NCSM'

実使用においてはtexlistとCellSpriteを一つのファイルでCellStreamとCellSpriteMotionを一つのファイルで扱いたいことがある。拡張子の必要以上の増加を防ぐため次の方法を取る。

spj ファイルは、texlist チャンク NJTL を取り込むことができる。
csj ファイルは、motion チャンク NCSM を取り込むことができる。

コンバータは、バイナリ出力時にデフォルトで texlist 付きの spj と motion 付きの csj を出力する。オプションで.njl, .spj, .csj, .njm の四つの出力も可能。

njl、spj、csj、njm の出力について説明する。

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>< CellSprite に関わるチャンクの種類 > 'NJSP' : ninja cell sprite 'NJCS' : ninja cell stream 'NJTL' : ninja texlist 'NCSM' : ninja cell sprite motion</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>

Chunkname は 4 文字のロングワード。bytesize もチャンクデータのサイズをバイトで表現する。
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。

バイナリデータはユーザが利用するデータのトップ(例えばモーションではNJS_MOTION 構造体データ先頭)をアドレスの先頭とするかたまりとしてデータを配置する。内部に含まれるポインタ位置を POF0 チャンクに保存しロードでメモリに取り込んだ時にポインタに実アドレスを加算する。

POF0 チャンクはデータメモリイメージ中のポインタの位置を保存する。オフセットは一つ前からのオフセット値で表現しデータサイズが小さくなるようにする。

データ内部の成分の配置は任意ですがキャッシュなどを考慮し利用するデータができるかぎり連続するようにする。

複数の異なるタイプのデータをまとめた場合拡張子.nj を使う。

(例) 拡張子.nj

'NJTL'	bytesize	texlist データ
texlist データ		
Chunk POF0	bytesize	
texlist POF データ		
'NJSP'	bytesize	CellSprite データ
Cell Sprite データ		
Chunk POF0	bytesize	
Cell Sprite POF データ		
'NJCS'	bytesize	CellStream データ
Cell Stream データ		
Chunk POF0	bytesize	
Cell Stream POF データ		

chunk name	説 明
'NJSP'	CellSprite データを格納します。データポインタの先頭にはトップの NJS_CELL_SPRITE 構造体のアドレスが格納される。
'NJCS'	CellStream データを格納します。データポインタの先頭にはトップの NJS_CELL_STREAM 構造体のアドレスが格納される。
'NJTL'	texlist データを格納します。データポインタの先頭には NJS_TEXLIST 構造体の先頭アドレスが格納される。
'NCSM'	CellSprite 用のモーションデータを格納します。データポインタの先頭には NJS_MOTION 構造体の先頭アドレスが格納される。

5.6 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>

POF0 チャンクの書き出し上位2ビットはフラグとするため相対ロングワードオフセットが6ビットの最大値64より小さい値ならば char 値として14ビットの最大値16384より小さければ unsigned short 値としてそれ以上は unsigned long 値として出力する。ここで出力される short, long データはビッグインディアンとする。これにより先頭の1バイトにフラグが格納されフラグによるデータタイプ分岐が可能となる。

以下に POF0 データ書き込みのサンプルソースコードを示す。

```
#define NJ_POF_TYPE_PAD    0x00
#define NJ_POF_TYPE_CHAR  0x40
#define NJ_POF_TYPE_SHORT 0x80
#define NJ_POF_TYPE_LONG  0xc0
#define NJ_POF_CHAR_MASK  0xc0
```

```
void njPointerCashFlashType0(unsigned long prev, unsigned long current)
{
    unsigned long loffset=(current-prev)>>2;
    if (64 > loffset) {
        char ctmp = NJ_POF_TYPE_CHAR|(char)loffset;
        WriteBytes(&ctmp, sizeof(ctmp));
    } else if (16384 > loffset) {
        unsigned short stmp = (NJ_POF_TYPE_SHORT << 8)|(short)loffset;
        S_SWAP_PC(stmp, stmp); /* keep char order */
        WriteBytes(&stmp, sizeof(stmp));
    } else {
        unsigned long ltmp = (NJ_POF_TYPE_LONG << 24)|loffset;
        L_SWAP_PC(ltmp, ltmp); /* keep char order */
        WriteBytes(&ltmp, sizeof(ltmp));
    }
}
```

次のデータチャンクを4バイトアライメント調整するためPOFチャンクの最後には最大3バイトのパディングが入る。パディングフラグは上位2ビットが00で示されるためPOF内部ではcharで0を書き込んでおけばパディングされる。

以上