



Ninja2 Model 仕様書

(08/21/2000)

1. 概要	6
1.0 NINJA1 からの変更点	6
1.1 BASIC MODEL の廃止	6
1.2 SIMPLESHAPE 及び COMPACTSHAPE について	6
1.3 NINJA2 RESOURCE(NRE)	7
1.4 NINJA2 OPEN LAYER(NOL)	7
1.5 NINJA2 MOTION RESOURCE(MRS)及び VERTEX RESOURCE(VRS)及び NINJA2 RESOURCE VERTEX(NRV)	7
1.6 エンベロープウェイトの16ビット化	8
1.7 トライリニアフィルタ出力	8
1.8 EVALCLIP	9
1.9 DIVIDE POLYGON、DIVIDE FLAT、DIVIDE EDGE VERTEX、DIVIDE VERTEX COLOR	9
1.10 WEIGHT EPSILON	10
1.11 モデルのインスタンス	10
1.12 モデルの PUSHPOP と EVALFLAGS との関係	10
1.13 2パラメータモディファイア	11
2. PVR 仕様	13
2.1 PVR 概要	13
2.2 PVR	14
2.3 PVP	16
2.4 PVR コンバート	17
2.5 VQ	18
3. PVM 仕様	20
3.1 PVM 概要	20
3.2 PVM チャンクレイアウト	22
3.2 PVM チャンク仕様	23
<i>ChunkName : 'PVMH'</i>	<i>23</i>
<i>ChunkName : 'COMM'</i>	<i>26</i>
<i>ChunkName : 'MDLN'</i>	<i>27</i>
<i>ChunkName : 'CONV'</i>	<i>28</i>
<i>ChunkName : 'PVMI'</i>	<i>29</i>
<i>ChunkName : 'IMGC'</i>	<i>30</i>
<i>ChunkName : 'PVRT'</i>	<i>31</i>
<i>ChunkName : 'PVPN'</i>	<i>32</i>
<i>ChunkName : 'PVPL'</i>	<i>33</i>
3.3 インフォレベル	34
3.3.1 <i>pvmstatus</i>	34
3.4 オペレーション	36
3.4.1 規約	36

3.4.2 デフォルトインフォレベル	36
3.4.3 インフォレベルの変更	36
3.4.4 テクスチャの順番の入れ替え	37
3.4.5 pvm のマージ	37
3.4.6 PVMConv	37
3.4.7 pvmUtil	37
4. マテリアルネーム (サーフェスネーム) 仕様	38
4.1 マテリアルネーム (サーフェスネーム) 概要	38
4.2 マテリアルネーム一覧	39
4.3 補足説明	41
4.3.1 <i>Es, Ss, Kt, Kr, Ks, Hd</i> オプション	41
4.3.2 'G' オプション	41
4.3.3 'Axx' オプション	41
4.3.4 'Bxx' オプション	41
4.3.5 'En' オプション	41
4.3.6 'Dxxx' オプション	42
5. FLATBUMP 仕様	43
5.1 FLATBUMP 概要	43
5.2 FLATBUMP フォーマット	43
6. ENVELOPE 仕様	45
6.1 ENVELOPE 概要	45
6.2 基本的な構造	46
6.3 頂点リストの再構成	46
6.4 頂点リストグループ	47
6.5 共有頂点バッファ	47
6.6 ENVELOPE フォーマット	48
7. TEXTLIST 仕様	52
7.1 TEXTLIST 構造	52
7.2.1 構造体図	52
7.1.2 テクスチャ処理の概要	53
7.1.3 メモリ形式テクスチャ、テクスチャキャッシュの概要	54
7.2 マクロ及び構造体定義	54
7.3 TEXTLIST のパレット対応について	55
8. チャンクモデル (CHUNK MODEL) 仕様	58
8.1 削除されるチャンク	58
8.2 利用可能なチャンク	58
8.3 チャンクモデルの特徴	61

8.4 チャンクモデル構造体.....	62
8.4.1 NJS_CNK_OBJECT 構造体図.....	62
8.4.2 NJD_CNK_OBJECT 構造体.....	63
8.4.3 evalflags の説明.....	64
8.4.4 evalflags のユーザフラグ領域 (上位 8 ビット)	64
8.5 チャンクの種類	65
8.6 CHUNK の構造	66
8.7 CHUNK NULL.....	67
ChunkName : 'NJD_CN'.....	67
8.8 CHUNK END.....	68
ChunkName : 'NJD_CE'.....	68
8.9 CHUNK BITS	68
ChunkName : 'NJD_CB_BA'.....	69
ChunkName : 'NJD_CB_DA'.....	71
ChunkName : 'NJD_CB_EXP'.....	72
ChunkName : 'NJD_CB_CP'.....	73
ChunkName : 'NJD_CB_DP'.....	74
8.10 CHUNK TINY.....	75
ChunkName : 'NJD_CT_TID', 'NJD_CT_TID2'.....	76
8.11 CHUNK MATERIAL	78
ChunkName : 'NJD_CM_D', 'NJD_CM_D2'.....	79
ChunkName : 'NJD_CM_A', 'NJD_CM_A2'.....	80
ChunkName : 'NJD_CM_DA', 'NJD_CM_DA2'.....	81
ChunkName : 'NJD_CM_S', 'NJD_CM_S2'.....	82
ChunkName : 'NJD_CM_DS', 'NJD_CM_DS2'.....	83
ChunkName : 'NJD_CM_AS', 'NJD_CM_AS2'.....	84
ChunkName : 'NJD_CM_DAS', 'NJD_CM_DAS2'.....	85
8.12 CHUNK VERTEX.....	87
ChunkName : 'NJD_CV'.....	89
ChunkName : 'NJD_CV_D8'.....	90
ChunkName : 'NJD_CV_UF'.....	91
ChunkName : 'NJD_CV_NF'.....	92
ChunkName : 'NJD_CV_VN'.....	93
ChunkName : 'NJD_CV_VN_D8'.....	94
ChunkName : 'NJD_CV_VN_UF'.....	95
ChunkName : 'NJD_CV_VN_NF'.....	96
8.13 CHUNK VOLUME.....	97
ChunkName : 'NJD_CO_P3'.....	98
ChunkName : 'NJD_CO_P4'.....	99
ChunkName : 'NJD_CO_ST'.....	100
8.14 CHUNK STRIP	101

<i>ChunkName</i> : 'NJD_CS', 'NJD_CS_2'.....	103
<i>ChunkName</i> : 'NJD_CS_UVN', 'NJD_CS_UVN2'.....	104
<i>ChunkName</i> : 'NJD_CS_UVH', 'NJD_CS_UVH2'.....	105
9. NINJA2 フォーマット	106
9.1 拡張子一覧.....	106
10. アスキーフォーマット.....	108
10.1 アスキーフォーマット概要	108
10.2 CHUNK MODEL アスキーフォーマットのマクロ定義.....	110
10.2.1 汎用マクロ.....	111
10.2.2 <i>texlist</i> マクロ	112
10.2.3 インスタンスマクロ	113
10.2.4 <i>EvalFlags</i> マクロ	113
10.2.5 <i>Blending Alpha</i> マクロ.....	114
10.2.6 <i>'D' Adjust</i> マクロ.....	115
10.2.7 <i>exponent</i> マクロ.....	115
10.2.8 <i>Chunk Null</i> マクロ	116
10.2.9 <i>Chunk End</i> マクロ.....	117
10.2.10 <i>Chunk Bits</i> マクロ.....	117
10.2.11 <i>TexId</i> マクロ (<i>Chunk Tiny</i>)	117
10.2.12 <i>Chunk Material</i> マクロ.....	118
10.2.13 <i>Chunk Vertex</i> マクロ.....	120
10.2.14 <i>Chunk Volume</i> マクロ	122
10.2.15 <i>Chunk Strip</i> マクロ	122
10.2.16 <i>NJS_CNK_MODEL</i> 構造体マクロ.....	124
10.2.17 <i>NJS_CNK_OBJECT</i> 構造体マクロ.....	124
10.3 モデルアスキーフォーマット出力例.....	125
10.3.1 エンベロープ <i>nja</i> ファイルの出力例.....	125
10.3.2 テクスチャ2枚貼り <i>nja</i> ファイルの出力例.....	135
10.3.3 トライリニアフィルタ <i>nja</i> ファイルの出力例.....	137
10.3.4 モデル単位のインスタンス <i>nja</i> ファイルの出力例	143
10.3.5 モデル階層のインスタンス <i>nja</i> ファイルの出力例	149
10.3.6 2パラメータモディファイア <i>nja</i> ファイルの出力例.....	155
10.3.7 <i>FlatBump</i> <i>nja</i> ファイルの出力例	157
10.3.8 <i>nsc</i> ファイルの出力例.....	159
11. バイナリフォーマット.....	161
11.1 バイナリフォーマット概要	161
11.2 バイナリチャンクの種類	162
11.3 バイナリ構造	163
11.4 POF0 のアルゴリズム.....	164

1. 概要

1.0 Ninja1 からの変更点

本仕様書はNinja2におけるモデル及びテクスチャの仕様について説明する。カメラ、ライトはNinja2Motion仕様書を参照のこと。Ninja1 から Ninja2 へのモデルの変更点は次の通り。

BasicModel の廃止。

SimpleShape の廃止。

ライブラリの NormalDraw の廃止。性能面で実用頻度が低いためなくなった。そのためマテリアルのスペキュラーの Exponent 値は NormalDraw のみ有効なため事実上設定しても無効となる。

使用頻度の低いチャンク形式の削除。

D8_S8 チャンクによるスペキュラ頂点カラー対応強化（現在はライブラリ上でのダイレクト出力のみ）。

エンベロープのウェイト値の精度向上（8ビットから16ビットへ）。

SimpleShape よりも効率の良い CompactShape 対応（CompactShape データは vrs と共にモデルエンベロープデータ出力に深い関係を持つ）。

頂点グループ指定のための vrs (vertex resource) 機能の追加。

モデラー上でシーン全体の頂点グループを保存する nrv (Ninja2 resource vertex) 機能の追加。ただし nrv を利用可能なのは現在 Softimage のみ。

未対応であったトライリニアフィルタ出力に対応。

モデルクリップ判定用の中心と半径をユーザが設定できる EvalClip 機能に対応。

両面ポリゴンを2枚のポリゴンに複製する DividePolygon 機能に対応。

エンベロープの微細なウェイト値をカットし描画性能を調整可能な weightEpsilon 機能に対応。

PList におけるアライメント合わせのための ChunkNull の挿入は不要となった。もちろんあっても動作する。Ninja2 ではコンバータは基本的に ChunkNull の出力をしない。

（注意事項）

Ninja2 では SimpleShape は使用できない。Ninja2 用頂点モーションデータ nas ファイルは無条件に CompactShape ファイルを意味する。

1.1 Basic Model の廃止

Basic Model 構造はDreamcast ハードウェア開発前に Saturn との互換性を考慮して設計された。Dreamcast の特性及びエンベロープなどに対応することを前提に設計された Chunk Model を全面に利用し Basic Model は廃止とする。これによりライブラリをコンパクト化できる。

コリジョンデータ等で作りのシンプルな Basic Model を利用していた場合 Chunk Model フォーマットに用意される Chunk Volume を使ってください。描画はできませんが三角、四角、ストリップでのコリジョンデータが出力可能。三角形ポリゴンの場合はモディファイアボリュームデータとしても利用する。

1.2 SimpleShape 及び CompactShape について

SimpleShape はNinja 開発段階で最小限の仕様で頂点モーションを実現した。現在利用可能なエンベロープとは同時に利用できない。またモデル全体しか頂点モーションの対象とできず処理の最適化が難しかった。

CompactShape はエンベロープと同時に利用できる。モデラー上で指定した頂点グループに含まれる頂点だけの頂点モーションをコンバートでき必要な部分だけの頂点モーションさせることができる。エンベロープと

頂点モーションを同時に実行した場合でもその合成結果を頂点モーションとして利用可能であり頂点グループ指定により任意の頂点に対しエンベロープアルゴリズムを適用するか頂点モーションアルゴリズムを適用するかが指定できる。見た目で頂点グループを調整することで効果的にデータコンバートが可能である。

また CompactShape では頂点リストを ShapeList に登録しエントリ ID である ShapeId で間接参照するためターゲット形状の使いまわしが可能である。ただし頂点の使いまわしをする場合は単一の頂点に対し頂点モーションのみが影響するデータとして作成する必要がある。頂点モーション、エンベロープの両方が影響する場合頂点モーション結果は TRS の影響によるエンベロープの変形と頂点モーションの合成結果になるため使いまわしはできない。主な注意事項は次の通り。

Ninja2 の頂点モーションは CompactShape となる。SimpleShape はサポートされない。

nas(njs)ファイルには ShapeList のみが格納される。従来の頂点モーションの NJS_MOTION 構造体はモデルの TRS モーションの格納される nam(njm)ファイルに格納される。

また CompactShape では KeyframeGroup を導入する。KeyframeGroup とはモデル間で同一のキーフレームを共有可能な場合これをグループ化しデータ上に表現する。この方法により KeyframeGroup 単位でキーフレーム探索をすればよくなる。例えば 3D Studio MAX のデータ出力では全モデルにおいて同一キーフレームを保持するがこの場合 KeyframeGroup が一つとなりキーフレームの探索はモデル階層全体で一回でよくなる。また KeyframeGroup の概念はエンベロープでモデル階層に分配される頂点に対し効果的にキーフレームを割り振る概念として非常に重要である。

詳細は Ninja2 CompactShape 仕様書参照のこと。

1.3 Ninja2 Resource(NRE)

Ninja1 の頃より利用していた Ninja 用のリソース管理機構を nre と呼ぶ。nre ファイルはコンバータのオプションその他の情報を保存することを目的とするアスキー形式のファイルである。nre は開発環境に必要なリソースタイプを定義しこれを利用可能な形で提供している。ユーザ利用可能な nre としては ConvertOptions がある。これはすべてのモデルコンバータにおいて利用されコンバートオプションを保存する。Ninja2 データオプティマイザーにバンドルのリソースエディタを使うと任意の nre の中身を書き換えることができる。nre をリソースエディタで直接書き換えることでコンバート時のオプションを変更できる。

nre は複数のリソースを複数個同一のファイルに格納することができる。

1.4 Ninja2 Open Layer(NOL)

コンバータコアを SDK 化した。その SDK 使用し入力データを設定していくことで Ninja2Export 出力を得ることができる。一般開示は現在していないがこれに沿ってオプション群を整理した。SDK を NOL (Ninja Open Layer) と呼ぶ。NOL にはコマンドラインオプションを表現する LToken、nre を基本とし Export のオプションを表現する RToken がある。オプションは LToken、RToken で表現される。Ninja2 Options 仕様書を参照のこと。

1.5 Ninja2 Motion Resource(MRS)及び Vertex Resource(VRS)及び Ninja2 Resource Vertex(nrv)

mrs は Ninja1 の頃から利用されるモーションリソースファイルである。これは nre とは関係なく独自の仕様を持つ。Ninja のモーションデータ構造はモデル階層側の TRS を参照しこの TRS から変化の無い部分はデータを NULL とすることでデータサイズを小さくする。ライブラリはモーションデータが NULL の場合モデル階層側の TRS を参照し利用する。モデルの持つ TRS とモーションの NULL 部分が期待する TRS の値が同じでないとこの方式は利用できない。mrs は基本となるモデルをコンバートした時に生成しこれをモーションコンバート時に参照しながらデータを生成する。これにより基本モデルの TRS による NULL 生成を実現する。mrs にはモ

デル階層の TRS、階層構造情報、コンバートオプションの一部、モデルのポリゴン数、vrs ファイル名などが情報として格納される。

vrs とは CompactShape に関わり導入された概念であり頂点グループデータを保存するファイルである。CompactShape では選択した頂点のみの頂点モーションデータ出力できる。この頂点選択情報を保存する。モーションコンバート時に mrs を参照するが mrs ファイルは内部に vrs ファイル名の情報を持つ。つまり mrs を参照することで vrs ファイルの存在の確認及びデータ参照を同時にする。原則 mrs と vrs は同一のパスに存在する必要があり移動する場合はこの二つをセットで扱われなければならない。例えば a.nja の mrs、vrs は

(例)
a.nja
a.mrs
a.vrs

となり a.mrs を他のフォルダに移動する場合 a.vrs も一緒に移動する。

vrs はモデルに関わる頂点情報なためデータサイズが大きくなるためバイナリ形式を採用している。内部では頂点番号を圧縮して保存している。

またモデラー側での頂点グループを保存するためシーンレベルの vrs データ保存機構を用意する。これを nrv (Ninja2 resource Vertex) と呼ぶ。モデラー側で頂点グループを保存できない場合などに利用する。

またコンバータは OpenGL ビューアによる preview 機能を持つがこれを改良し pick が可能とした。これを使うことで CompactShape データを実行しながらの pick ができ頂点グループの頂点の動的な変更をしながら頂点モーション、エンベロープアルゴリズムの適用の調整が可能である。編集データは nrv もしくはモデラーの頂点グループ情報に保存でき利用可能。

Pick モードで OpenGL ビューアを利用する場合はそこからのコンバートはできない。

1.6 エンベロープウェイトの 16 ビット化

Ninja1 では 256 段階でウェイトを表現しエンベロープを実現していた。しかし実際の利用において 3D 空間で 256 段階の分解能の値を乗算することは誤差が大きくエンベロープでない (256 段階が乗算されてない) モデルを同時に表示した場合に Z 値が近いと Z 値の上下反転するケースが発生した。

実験によれば 12 ビット程度の精度があればこの Z 値の上下反転は防止できるが今後誤差によるトラブルを防ぐため分解能アップのためにリザーブされていた 8 ビットすべてを利用し 16 ビット精度とした。

1.7 トライリニアフィルタ出力

Ninja1 の頃よりマテリアルネームエディタから設定可能であったが正しい動作を実装していなかった。今回これを実装した。トライリニアフィルタとは距離により選ばれた 2 段階のミップマップの間を補完したテクスチャをポリゴン貼る機能でありバイリニアフィルタ使用時に見える距離変化によるミップマップの切り替えの瞬間をなくしスムーズなミップマップ変化を実現できる。

ただし処理は非常に重いためすべてのポリゴンに利用することは推奨しない。処理は 2 パスなし 3 パスで実現される。2 パスとは同一ポリゴンを二回描画する動作、3 パスとは三回描画する動作である。2 パスは透明でないポリゴン、3 パスは半透明ポリゴン時に利用する。

設定はマテリアルネームエディタから文字列 TI を指定することでできる。2 パス、3 パスの描画データ生成はコンバータ内部で処理される。3 パスの場合ポリゴンデータは 3 回繰り返される形式で出力される。

テクスチャ 2 枚貼りの場合にトライリニアフィルタを利用する場合は一枚目に 2 パス、二枚目に半透明なため 3 パスが必要となる。つまりテクスチャ 2 枚貼りは 5 パスで実現される。絵は綺麗であるが処理が重いことを理解した上で利用する必要がある。

トライリニアフィルタはミップマップ間の補完をするためミップマップ無しのテクスチャには適用できないので注意が必要。コンバータはミップマップ無しのテクスチャを使ったマテリアルを見つけた場合ログへのワーニングメッセージと共にトライリニア設定を無視する。

1.8 EvalClip

Ninja では各モデルが画面上に表示されるかどうかをモデルに外接する球体の中心点と半径を利用して判定している。モデル階層は通常数十以上のモデル階層で構成されるがこれらすべてにおいて半径を利用した判定をすることは階層が細かい場合非常に重たい処理になる。

EvalClip は NJS_CNK_OBJECT 構造体の evalflags に指定された NJD_EVAL_CLIP フラグが設定されたモデルの中心点と半径で画面上に表示されない場合そのモデルの子供の探索を打ち切り描画をしない機能である。各モデルごとの表示判定の省略を可能とする。NJD_EVAL_CLIP が設定されるべきモデルはモデル階層の上位層で動きの基本となる中心のモデルである必要があり半径はそのモデルの子供のすべてのモデルが包含されるだけの十分な大きさを持つ必要がある。

Ninja2 では次のような手法によりこの設定をモデラーから可能なようにした。

EvalClip 指定のみのためのモデルを親子階層の一部として組み込むことを可能とする。コンバータは EvalClip 用のモデルを判断し中心点と半径をそのモデルの親に設定しモデルを階層から消去する。この手法によりユーザは任意のサイズの中心点と半径を設定できる。

EvalClip 用のモデルであることはマテリアルネームエディタ (LightWave ではサーフェスネームエディタ) から設定する。文字列 "Ec" を設定することで指定。Ec が検出された場合他の設定はすべて無効。更に Ec の設定されたモデルはコンバート出力結果からは削除される。

注意事項は次の通り。

エンベロープに関わるモデルに Ec を設定しても Ec の設定は無視されます。具体的には他のモデルの頂点ウェイト値を持つもしくは他のモデルが自分の頂点のウェイト値を持つ状態のモデル。

Ec を設定されたモデルの子供のすべてのモデルは合わせて削除されます。また子供モデルは中心点、半径の計算には使われない。

Null モデルの子供に EvalClip 用のモデルを作成した場合 Null モデルの NJS_CNK_OBJECT 構造体の model ポインタにはダミーの NJS_CNK_MODEL 構造体が設定される。ダミーは頂点データ (vlist)、ポリゴンデータ (plist) とともに Null となり中心点と半径指定のみが使われる。

EvalClip モデルに指定されるモデル形状は任意の形状でよい。その外接する球の中心点と半径が生成される。

< 作成手順 >

- ・モデルを作成する。
- ・例えばトップの Null モデルの子供モデルとして球を生成する。キャラクターが動き回ってもはみ出ない位置と大きさを確保する。
- ・球にマテリアルを設定しマテリアルネームエディタから Ec を設定する。
- ・コンバートする。

球の中心点と半径がトップの Null モデルのダミー NJS_CNK_MODEL 構造体データに格納され球が削除されたデータがコンバートされる。

1.9 Divide Polygon、Divide Flat、Divide Edge Vertex、Divide Vertex Color

Divide Polygon はマテリアルに両面ポリゴン属性が設定されている場合にコンバート時に両面ポリゴンを二枚のポリゴンに分割する。この機能が有効であるのは両面ポリゴンがサポートされていない Draw 関数である SimpleMultiDraw である。

Divide Flat はマテリアルに Flat 属性が設定されている場合にコンバート時にポリゴンの面法線を設定し隣接するポリゴンと共有される頂点を分割することで Flat ポリゴンを生成する。

Divide Edge Vertex は隣接するポリゴンに共有される頂点においてポリゴンごとに頂点法線が異なる場合異なる数だけ頂点を複製し法線を出力することでポリゴンのエッジの出力を可能とする。

Divide Vertex Color は隣接するポリゴンに共有される頂点においてポリゴンごとに頂点カラーが異なる場合に異なる数だけ頂点を複製し頂点カラーを出力することで頂点カラーによる陰影の出力を可能とする。Divide Vertex Color は Divide Edge Vertex に付属する。頂点カラーの設定されたモデルにおいて Divide Edge Vertex を指定し頂点カラー出力のチャンクを選ぶと動作する。

Divide Polygon, Divide Flat, Divide Edge Vertex, Divide Vertex Color は同時に利用できるがその分だけ頂点が複製されるので注意が必要である。簡単に頂点数は 1.5 倍以上になる可能性がある。また頂点が分割されることによりトライアングルストリップの連結率も下がるので注意が必要。

1.10 weight Epsilon

エンベロープのウェイト値はモデラーウェイト計算機能により連続的に生成される場合がある。この場合漸近線的に値が小さくなっていく結果値の小さなウェイト値の部分が発生する。実際モデルの出力結果を見てみると 99% のウェイト、1% 未満のウェイト値が多く存在する。複数のウェイト値が存在するということは頂点が複数に分割されていることを意味し見た目の影響の少ないウェイト値に対し重たい計算がされていることになる。

weightEpsilon は設定されたウェイト値に閾値を設けこの値より小さなウェイト値を削除する。また削除したウェイト値を最大のウェイト値に加算する。

値は float 値の % で指定される。1.0f% が指定された場合 1.0% 未満のウェイト値は他のそれよりも大きなウェイト値に加算され削除される。これは例えばある頂点のウェイト値が 99.1%、0.9% の二つのから構成される場合 100.0% の一つのウェイト値にまとめられこれは通常の頂点計算がされる。

weightEpsilon を大き目にするすることである程度頂点計算を抑えたエンベロープを生成可能である。最大 100% を設定するとすべてのウェイト値が削除され No Envelope 状態となる。

weightEpsilon が 0.0f の場合 (デフォルト) ウェイト値の最適は一切されず従来のエンベロープが生成される。

1.11 モデルのインスタンス

マップの森の木などを作る場合一つのモデルを繰り返し参照することでモデルを使いまわせるとデータ効率、性能面で有利である。Ninja ではモデル、モデル階層 (Softimage のみ) のインスタンスをサポートする。モデル単位のインスタンスは NJS_CNK_MODEL 構造体のポインタを複数の NJS_CNK_OBJECT 構造体から参照すること実現される。モデル階層のインスタンスはインスタンス元の Child の NJS_CNK_OBJECT 構造体ポインタを他の NJS_CNK_OBJECT 構造体の Child に設定することで実現される。

C 言語仕様を基本とするアスキーフォーマットにおいて各インスタンス参照される変数は他の変数からの参照のため nja ファイルの先頭で extern 宣言される。

注意事項としてはインスタンスされるモデルは同一モデル階層内のものとする。これはインスタンスをポインタレベルで実現しているため参照不能な場合インスタンスを実現できないためである。ただしアスキー出力においては出力される変数名が同一になることで同一階層内でなくともコンパイルすることでポインタをつなぐことができるため動作する場合がある。これは動作保証外ではあるがユーザ責任において利用も可能。バイナリ出力は同一階層内でないとできない。

1.12 モデルの PushPop と Evalflags との関係

通常モデル階層トレースは構造体の Child, Sibling のポインタを参照しこのポインタが NULL でない場合 Child, Sibling をトレースすることで実現される。

Ninja ではモデル階層トレース最適化のために Evalflags を持つ。Evalflags は移動成分が 0 である、回転成分が 0 である、スケール成分がすべて 1.0 であるなどの状態を示すフラグを持つ。Ninja2 ではこの Evalflags

をトレースに使う。例えば移動成分が0の場合 NJD_EVAL_UNIT_POS フラグが設定される。このフラグが設定されている場合ライブラリは移動成分のマトリクス演算を省略する。

また Evalflags はモデル描画を禁止する Hide (NJD_EVAL_HIDE)、Child のトレースを中止する Break (NJD_EVAL_BREAK) を持つがこれをトレース時に参照することで判定処理を単純化している。モデル描画をしないもしくはポリゴンを持たない NULL モデルは NJD_EVAL_HIDE が設定される。NULL モデルは常に NJD_EVAL_HIDE が設定されることでモデルポインタが NULL かどうかの判定は不要になる。

また Child がない場合常に NJD_EVAL_BREAK が設定される。これにより Child のポインタが NULL かどうかの判定は不要になる。

Ninja2 コンバータはモデル階層トレースのために次の Evalflags をデータの状態に合わせて設定している。これらのフラグ設定は Ninja2 のモデルトレースに重要である。

NJD_EVAL_UNIT_POS	移動成分がすべて0の場合に設定。
NJD_EVAL_UNIT_ANG	回転成分がすべて0の場合に設定。
NJD_EVAL_UNIT_SCL	スケール成分がすべて1.0の場合に設定。
NJD_EVAL_HIDE	モデルがポリゴンなしの NULL モデルの場合に設定。
NJD_EVAL_BREAK	child ポインタが NULL の場合に設定。

NJD_EVAL_HIDE、NJD_EVAL_BREAK に関してはユーザがマテリアルネームから設定した場合にも有効となる。またユーザがユーザの利用目的に合わせて NJD_EVAL_UNIT_POS、NJD_EVAL_UNIT_ANG、NJD_EVAL_UNIT_SCL の設定を変更してもよい。

1.13 2 パラメータモディファイア

2 パラメータモディファイアは次の二つのデータから構成される。

モディファイアボリューム
モディファイアデータ

モディファイアボリュームとは独立三角形から構成される閉じた閉空間であり、モディファイアデータとはテクスチャとマテリアルを二重に持つデータで、モディファイアボリュームとの空間的な論理積 (and) 部分に対し第2のテクスチャとマテリアルを描画することのできるデータである。

モディファイアデータはテクスチャとマテリアルを二重に持つことから2パラメータモディファイアデータと呼ばれる。

2パラメータモディファイア機能はモデラーで設定された二枚のテクスチャに対しモディファイアボリュームの内側と外側で二つのテクスチャ、マテリアルを切り替えることができるハードウェア機能である。影に使われているチープシャドウと同時に使うことができないこと、2パラメータモディファイア機能は処理が重いことに注意が必要。

モディファイアボリュームにはチャンクフォーマットのチャンクボリューム Polygon3(NJD_C0_P3)が利用される。チャンクボリューム出力はNinjaExportのオプションで指定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。また一つのシーン内にモディファイアボリュームと2パラメータモディファイアデータを作成しモーションをつけられるようにするためにルートのNull Objectにフラグ設定機能作り親子階層単位でモディファイアボリュームを設定できる(これをNullObject機能と呼ぶ)ようにしている。

ここでモディファイアボリュームに関するNullObject機能の使用方法について説明する。

コンバータオプションのUseNullEvalをオンする(コマンドラインの場合は-ne)。これはNullモデルの先頭一文字をマテリアルとして利用することを示す。このUseNullEvalオン状態で親子階層のルートのNullモデルの名前の先頭に'M'をつけるとその親子階層がモディファイアボリュームになる。また同時にルートのNullモデルのデータを格納するNJS_CNK_OBJECT構造体のevalflagsメンバにNJD_EVAL_MODIFIERフラグが設定される。

ライブラリはNJD_EVAL_MODIFIERフラグをチェックしない。ユーザがこれをチェックしモディファイア用のルーチン呼び出す必要がある。

テクスチャが二枚貼ってあるモデルにおいてこれをテクスチャ二枚貼りとするか 2 パラメータモディファ
イアデータとするかを指定するオプション Use2Para (コマンドラインの場合は -2p) を用意する。これは nre
の ConvertOptions リソースにもなる。またマテリアル単位で 2 パラメータボリュームデータを指定するた
めにマテリアルネームエディタから 2P 文字列を設定できる。

2. pvr 仕様

2.1 pvr 概要

Dreamcast の基本テクスチャ構造として pvr 形式を定義する。これはハードウェア仕様から決定されたテクスチャフォーマットを IFF 形式のチャンクに格納したものである。テクスチャは主に次の種類が利用できる。

flatbump
16 ビットカラー (twiddled dma, dwiddled, rectangle, twiddled rectangle)
YUV422
VQ (small VQ)
パレット
ストライド

Ninja ではハードウェアの持つバンプ機能を flatbump と呼ぶ。これはバンプ機能に対するハードウェアの制限が多く曲面形状に貼って使用するのには難しいためである。使用に関しては特定の用途ワンポイントで利用することを推奨。処理も重いので積極的な利用は推奨しない。

16 ビットカラーテクスチャを基本とし RGB565, ARGB1555, ARGB4444 が利用できる。

YUV422 形式は U 値、V 値を隣あう 2 つのピクセルで共有することで 16 ビット × 2 に RGB 各 8 ビット相当のデータを格納する。2 ピクセルデータに Y1+U, Y2+V の二つのデータを格納する。ここで U, V は二つのピクセルの U1, U2, V1, V2 の平均値。色復元時に Y1, U, V 及び Y2, U, V を利用する。隣あうピクセルで同一の U, V を使うため横方向の分解能が低下するがグラデーションの階調表現に優れている。RGB から YUV の変換式は Dreamcast 独自の係数を持つので注意が必要。通常の係数を利用すると白っぽくなる。

16 ビットカラーのテクスチャを圧縮する独自の VQ 形式をサポートする。実用範囲で最大 1/7 程度の圧縮率が期待できる。テクスチャの絵柄によっては VQ 圧縮の画質劣化に耐えられない場合もある。特にアニメ調の絵柄や文字、エッジの多いテクスチャは VQ に不向きである。

パレットは 1024 エントリが可能。16 色を 1 バンクとし 64 バンクを持つことができる。1024 エントリに対しパレットの形式は一種類しか指定できない。つまり RGB565, ARGB1555, ARGB4444, ARGB8888 のいずれかである。4 ビットの 16 色テクスチャ、8 ビットの 256 色のテクスチャがあるが 8 ビットの場合 1024 エントリを 4 つに分けた 256 エントリの境界からしかパレットを格納できずまたその境界をまたぐことはできない。Ninja ではバンク 64 個に 0~63 絶対的な通し番号を振る。4 ビットの 16 色テクスチャは 0~63 のバンク指定が可能。8 ビットの 256 色テクスチャは 0, 16, 32, 48 のバンク番号指定が有効である。パレットデータの保存には独自の pvp ファイル形式が利用される。

Ninja では同じテクスチャの重複したメモリ登録などを避けるために Global Index を導入している。これはテクスチャに通し番号を振るものでライブラリは同一の番号を持つテクスチャは同一のテクスチャであると判断し重複登録をしない。コンバータは出力時にモデル全体に使われるテクスチャに対し Global Index の通し番号を設定する。ユーザ責任において同時に使われる可能性のあるデータ群に通しの Global Index を設定する必要がある。この機能により例えば複数の texlist の中に同一のテクスチャがある場合でも重複してテクスチャメモリに格納されない制御ができる。Global Index は pvr ファイルの先頭に GBIX チャンクとして定義される。また GBIX チャンクはパレット使用時に pvr が参照するバンク番号を格納するためにも利用される。

pvr 形式として使用できるテクスチャサイズ 8, 16, 32, 62, 128, 256, 512, 1024 を基本とし 8x8 ~ 1024x1024 が利用可能。正方形の場合これをメモリアクセス効率の高いピクセル配置の twiddled 形式で表現する。また正方形の場合ミップマップが利用可能でありトライリニアフィルタが利用可能である。正方形でない場合ミップマップを持つことができない。twiddled 形式を取らない普通のピクセル形式を rectangle と呼ぶ。自分でテクスチャを加工するような用途では rectangle 形式が便利。また矩形の中の正方形単位で twiddled を適用する twiddled rectangle 形式 (ミップマップは不可) も利用できる。twiddled を適用した形式の方が高速である。

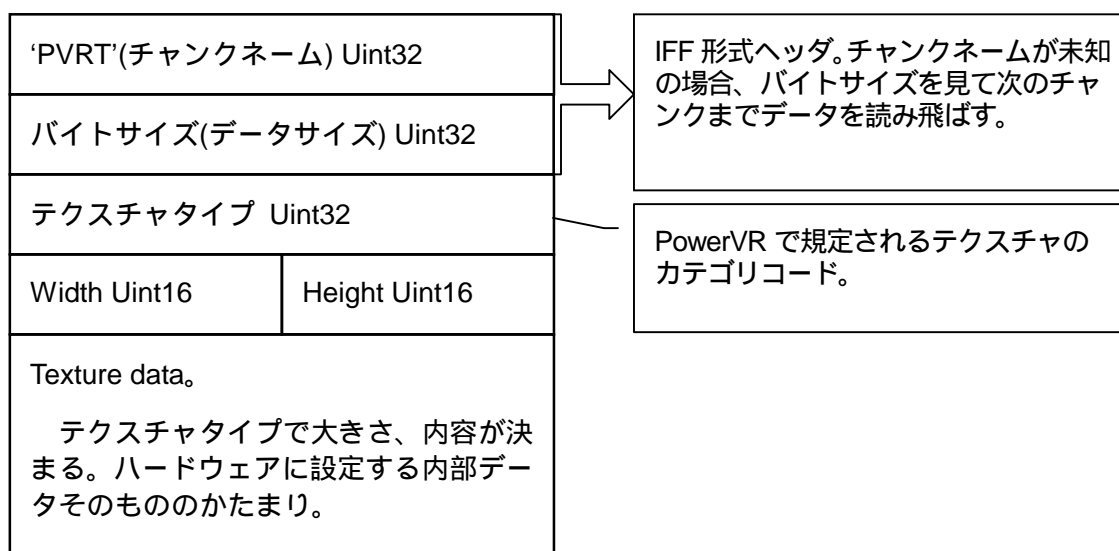
またミップマップありの正方形にしか使えないがもっとも高速である twiddled dma 形式がある。これは DMA 転送可能なバウンダリにテクスチャの先頭アドレスが配置されるようにダミーのデータを入れたものである。Ninja2 では正方形でミップマップありの場合デフォルトで Twiddled dma を利用する。

ストライドは 32 を基本とする横幅を指定できる。これはハードウェアに一つしかないレジスタ指定を利用するため同時には一種類の幅しか利用できない。例えば 92x64 のテクスチャが作れる。

単一のテクスチャをバラバラのファイルで配置することは効率が悪くまたあるグループのテクスチャ群を一括して扱いたい場合がよくある。これに対応するために Ninja では pvm 形式を用意する。pvm はテクスチャ情報に加え関わったモデルデータ名、コンバータオプション、元画像なども格納できる。最終的にゲームで使用する段階で不要な情報を削除しゲームデータを作成する。

2.2 PVR

PVR テクスチャフォーマット概要は次の通り。IFF によるチャンク形式(ヘッダ+サイズ+データ)。データ部はハードウェアが期待する内部データ構造そのまま。詳細はここでは説明しない。



テクスチャタイプにはカラータイプとカテゴリコードの OR をとったビット列が設定される。

```
/* カラータイプ */
#define NJD_TEXFMT_ARGB_1555      (0x00)
#define NJD_TEXFMT_RGB_565       (0x01)
#define NJD_TEXFMT_ARGB_4444     (0x02)
#define NJD_TEXFMT_YUV_422       (0x03)
#define NJD_TEXFMT_BUMP          (0x04)
#define NJD_TEXFMT_RGB_555       (0x05)
#define NJD_TEXFMT_ARGB_8888     (0x06)
#define NJD_TEXFMT_YUV_420       (0x06)
#define NJD_TEXFMT_COLOR_MASK    (0xFF)

/* カテゴリコード */
#define NJD_TEXFMT_TWIDDLED       (0x0100)
#define NJD_TEXFMT_TWIDDLED_MM   (0x0200)
#define NJD_TEXFMT_VQ            (0x0300)
#define NJD_TEXFMT_VQ_MM        (0x0400)
#define NJD_TEXFMT_PALETTE4      (0x0500)
#define NJD_TEXFMT_PALETTE4_MM   (0x0600)
#define NJD_TEXFMT_PALETTE8      (0x0700)
#define NJD_TEXFMT_PALETTE8_MM   (0x0800)
#define NJD_TEXFMT_RECTANGLE     (0x0900)
```

```
#define NJD_TEXFMT_STRIDE          (0x0B00)
#define NJD_TEXFMT_TWIDDLED_RECTANGLE (0x0D00)
#define NJD_TEXFMT_ABGR          (0x0E00)
#define NJD_TEXFMT_ABGR_MM       (0x0F00)
#define NJD_TEXFMT_SMALLVQ       (0x1000)
#define NJD_TEXFMT_SMALLVQ_MM    (0x1100)
#define NJD_TEXFMT_TWIDDLED_MM_DMA (0x1200)
#define NJD_TEXFMT_TYPE_MASK      (0xFF00)
```

PVRT チャンク以外に GBIX, PVRI の 2 つのチャンクを定義する。

'GBIX'(チャンクネーム) Uint32
8 (byte) Uint32
globalIndex Uint32
dammy(bankId) Uint32

テクスチャの globalIndex を記述する。Ninja2 で pvr ファイルを使う場合はこのチャンクがファイルの先頭にくるようにする。テクスチャの DMA 転送のためのアライメント調整のために最後にダミーの 4 バイトを入れる。パレットテクスチャ出力コンバータ (Farlux) ではこのダミー領域にパレットの bankId を出力する。情報として格納するがライブラリはこれの直接の利用はしない。

'PVRI'(チャンクネーム) Uint32
バイトサイズ Uint32

Data。テクスチャ制御情報。

テクスチャ制御情報を格納する。現在はあまり利用していない。

2.3 PVP

pvp ファイルはチャンクネーム 'PVPL'を持つパレットチャンクを格納する。

'PVPL'(4)	PowerVR 用パレットデータチャンクであることを示す。
bytesize(4)	次のチャンクまでのバイトサイズ。
category code(2)	カラータイプ、4bpp、8bpp を指定。
bankId(2)	バンク指定。0 ~ 63、パレットでない場合 - 1 を設定。
entryoffset(2)	バンク内のエントリオフセット。
entrycount(2)	palette data に格納される色の数。
<div>palette data</div> <div><u>RGB565 の場合</u> [16bitRGB]の一次元配列。</div> <div><u>ARGB1555、ARGB4444 の場合</u> [16bitARGB]の一次元配列。</div> <div><u>ARGB8888 の場合</u> [32bitARGB]の一次元配列。</div>	

category code:

```
<palette color format>
#define NJD_TEXFMT_ARGB1555      (0x0000)
#define NJD_TEXFMT_RGB565        (0x0001)
#define NJD_TEXFMT_ARGB4444      (0x0002)
#define NJD_TEXFMT_ARGB8888      (0x0006)
```

texture format は設定しません。color format のみカテゴリコードとして設定。このカテゴリコードが ARGB8888 の場合は Data のピクセルサイズが 32 ビットそれ以外の場合は 16 ビットになる。

bank:

0-63 の数字で示す。4bpp の場合は 0-63 ですが 8bpp の場合はこの数字の上位 2 ビットのみが有効。バンク番号は 0, 16, 32, 48 のみが利用できる。これ以外のバンク番号はまるめられる。例えば 1 ~ 15 は 0 になる。

entryoffset:

バンクの先頭からのオフセット。バンクの先頭のエントリを 0 番として指定。

entrycount:

データに格納される色のエントリ数。

Data:

エン트리数分の色データ。32 ビット、16 ビットの 2 タイプ。RGB565、ARGB1555、ARGB4444 の場合は 16 ビットカラーデータの一次元配列。ARGB8888 の場合は 32 ビットカラーデータの一次元配列がデータとなる。エントリ数は entrycount で与えられる。

pvp ファイル内部において PVPL チャンクは複数定義することができる。複数バンク分のパレットデータを一つのファイルの中で定義でき一括してライブラリでロードできる。デフォルト動作では PVPL は pvp ファイルにひとつ。パレットテクスチャは正方形のみで twiddled になる。ミップマップを使う場合は親画像のパレット色が共有されるのでピクセルの単純な間引きによりミップマップ画像を得るため 16 ビットカラーテクスチャに比べミップマップが汚くなることに注意。

2.4 PVR コンバート

Ninja2 のモデルコンバータもしくは pvrconv を利用して pvr に変換する。現在変換可能な元画像形式は次の通り。

pic

Softimage で利用する。

bmp

LightWave, 3D Studio MAX, MAYA 等で利用。 値を指定する場合はもう一つの bmp を 値用として利用。R 値が として利用される。パレットにも使用。

tga

LightWave, 3D Studio MAX, MAYA 等で利用。

pix

コンバータは元の画像の 値をチェックし自動的に次の三つのフォーマットを切り替えて出力する。

がない場合： RGB565 で出力。

がある場合： ARGB4444 で出力。

が 0 , 255 の二値の場合： ARGB1555 で出力。

テクスチャが正方形でミップマップありの場合 twiddled dma 形式がミップマップ無しの場合 tiddled 形式が長方形の場合 twiddled rectangle 形式がコンバータで自動選択される。オプション指定で rectangle 形式も出力可能。

twiddled 形式

テクスチャのピクセルを高速にメモリから読み出せる順番に並べ替えたテクスチャ。ミップマップが利用可能。表示が高速。ミップマップありの場合にトライリニアフィルタが利用可能。

rectangle 形式

ピクセルの順番をイメージそのままとしているテクスチャ。表示が twiddled に比べ低速。ミップマップが使用できない。

twiddled rectangle 形式

長方形領域の中を正方形領域に分割しその単位で twiddled にする。Mipmap できない。rectangle 形式よりも高速。

flatBump 形式

バンプマッピングテクスチャをグレイ階調(RGB 同色)で用意する。R G B カラー画像のバンプは扱えない。コンバータでハードウェアが期待するデータに変換する。flatBump は低速かつ制限が多くあるので注意が必要。

VQ

2x2 のブロックを 256 個持つコードブックを基本とする独自のベクトル量子化によりテクスチャを圧縮する。

small VQ

256 エントリのコードブックを小さな VQ テクスチャで共有する事で小さなテクスチャを圧縮する。

YUV422

RGB 値を YUV 値に変換する。8 ビット階調を持ち綺麗なグラデーションを出せるが横 2 ピクセルで UV 値を共有するため横の解像度が悪く縦線に弱い。

パレットテクスチャ

モデラーから直接パレットテクスチャを出力することはない。16 ビットカラーのテクスチャでモデルおよび texlist を生成し後でその 16 ビットカラーの pvr ファイルをパレットのテクスチャに置き換える。pvr の使用するバンク番号はマテリアルネームエディタで設定する。このバンク番号は PTEXN マクロで texlist 上に出力される。texlist とパレット PVR 差し替えの方法については texlist の説明です。

2.5 VQ

VQ は 2x2 のブロックを 256 個とこれを参照する 8 ビットのインデックス画像から構成される。ブロックが 2x2 に対し 1 インデックスを持つためインデックス画像は本来の解像度の縦横半分になることに注意が必要。

small VQ とはテクスチャサイズが小さく普通の VQ をしても圧縮率が得られない場合において 256 個より少ないコードブックでテクスチャを圧縮し複数の小さなテクスチャの VQ データを 256 個のコードブックにまとめて格納する方式である。VQ のハードウェアは 256 個のコードブックサイズの後ろにインデックス画像が存在することを前提とし例えば最初の 64 個のコードブックだけを使用し 256 個のコードブックサイズの後ろにインデックス画像を格納し次に先ほどの 64 個のコードブックの後ろに再び 64 個のコードブックを格納し 256 個のコードブックサイズ後ろにインデックス画像を格納する。メモリ上ではコードブックとインデックス画像が楕上に配置される。この方法により 256 個を基本とする VQ 構造の中に小さな VQ コードブックを格納することを実現する。small VQ が使えるかつ効果があるのは小さなテクスチャのみである。

現在 Ninja2 で利用可能なアルゴリズムは VQ3, VQ4 である。VQ1, VQ2 は画質で劣るのでサポートを停止した。VQ4 は VQ3 の改良版であるが 256 階調のプライオリティ画像を同時に指定することで VQ 圧縮時の 2x2 のブロック保存優先順位を作成できる。例えば文字の輪郭等を優先的に残したい場合この部分をプライオリティ画像上で塗りつぶす。これにより優先的にその部分にコードブックが割り当てられる。その分他の部分が劣化するので注意が必要。現在この機能は PhotoShop プラグインなどで利用可能。また VQ4 は VQ3 よりも 値に強い。

次に VQ 圧縮率を示す。256 個のコードブック以外は small VQ の使用を意味する。ミップマップありとなしがある。

Non-MIPMAP:

Image Size / Pixels	256 codes	128 codes	64 codes	32 codes	16 codes
16x16				1.6 : 1	2.7 : 1
32x32			2.7 : 1	4.0 : 1	
64x64		4.0 : 1	5.3 : 1		
128x128	5.3 : 1				
256x256	7.1 : 1				
512x512	7.8 : 1				
1024x1024	7.9 : 1				

MIPMAP:

Image Size / Pixels	256 codes	128 codes	64 codes	32 codes	16 codes
16x16				2.0 : 1	3.2 : 1
32x32			3.2 : 1	4.6 : 1	5.8 : 1
64x64		4.6 : 1	5.8 : 1	6.7 : 1	
128x128	5.8 : 1				
256x256	7.3 : 1				
512x512	7.8 : 1				
1024x1024	8.0 : 1				

512x512、1024x1024 においてもコードブックの数は 256 エントリである。画像上の 2x2 ブロック数に対し相対的な数が圧倒的に少ないので画質維持は難しいので注意が必要。

3. pvm 仕様

3.1 pvm 概要

pvm (PVR Manipulation) は pvr ファイルの拡張である。複数の pvr を格納しかつその pvr が生成された過程、元画像、方法、配置情報を保存することによりツールからの最適化操作を可能とする。pvm に含まれるデータチャンクは作業の工程、目的によって選択可能。最終的にゲームで利用する段階ではツール加工に必要な情報を無くし最小のサイズにできる。以下に特徴をまとめる。

- IFF チャンク形式を採用。

- 複数のテクスチャを一括して扱える。

- 情報チャンクをファイルの先頭に置き加工時の操作をファイルの先頭情報だけでできる。

- 元画像情報とコンバータ、コンバータオプション情報を持ちテクスチャの再変換が可能。

- ユーザが編集したテクスチャの上書きを禁止することができる。

- モデル出力ファイル名を持ちモデルデータの texId の更新が可能。

- pvr ファイルへの分解が可能。編集後再び pvm へのパックができる。また pvr データを格納しない情報部分のみの pvm と pvr ファイル群の状態でも同じディレクトリにあれば利用可能。

- 元画像そのものを pvm に格納し他のマシンへの取り込みができる。ただしデータ量は大きくなるので注意が必要。

- 利用するコンバータを指定できる。オプションも保存できる。

- Ninja2 ライブラリは pvm から texlist への変換をサポートする。

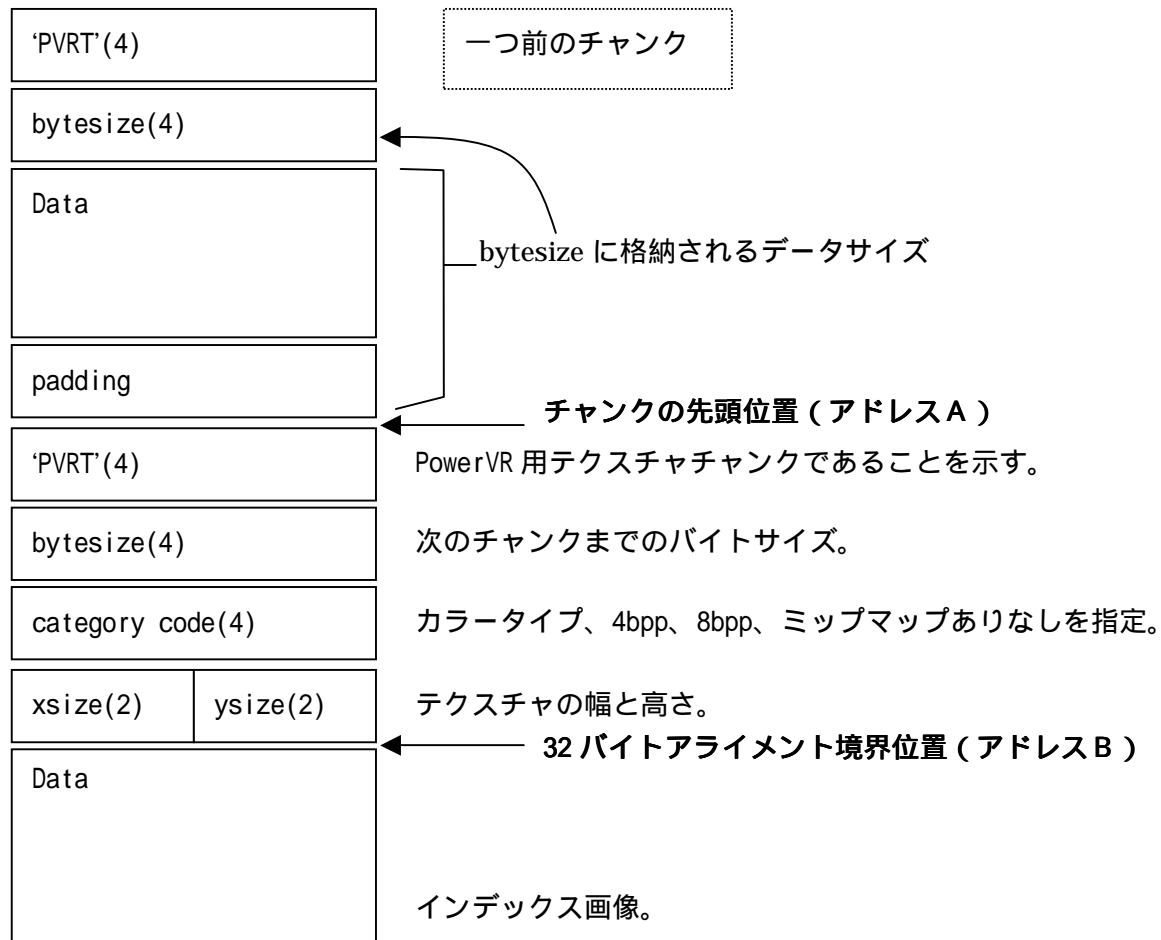
- パレットバンク ID を格納できる。globalIndex フィールドの上位 6 ビットを利用。

- パレットデータ PVPL チャンクを格納できる。ロード時にパレットの設定が可能。

(注意事項)

データの読み出し効率を高めるためにデータアドレスのアライメント調整が重要になる。PVRT チャンクの場合はデータの先頭が 32 バイトアライメント、その他のチャンクはチャンクの先頭が 4 バイトアライメントになるようにする。アライメント調整は性能に重要な意味を持つため pvm の仕様の一部とする。チャンクのデータサイズはアライメント調整分のダミーデータ込みのサイズになるので注意が必要。

PVRT における 3 2 バイトアライメント調整は次の通り。



3 2 バイトアライメント位置はチャンクの先頭でないことに注意。3 2 バイトアライメント位置をアドレス B とするチャンク先頭アドレス A は $A = B - 16$ になる。一つ前のチャンクデータの終了アドレスがアドレス A のバウンダリ位置にあわない場合はアドレス A までのパディングが必要となる。一つ前のチャンクの bytesize にはこのパディングサイズが含まれる。

それ以外のチャンクに関してはチャンクの先頭アドレスを 4 バイトアライメント境界にあわせる。この時生成されるパディングデータサイズは一つ前のチャンクの bytesize に含まれる。

パレット対応は texlist の構造の変更をしないでする必要から globalIndex の上位 6 ビットを利用している (そのためパレットを使う場合のみ globalIndex の最大番号が 26 ビットの最大値になる)。構造を複雑にしないため pvm も同様の構造をとることとする。次節で説明する PVMH チャンクは pvm 全体の情報を管理するがここで globalIndex データの上位 6 ビットにパレットのバンク ID を格納する。パレットを使う時は PVMH の pvmstatus フィールドに PVMH_PS_BANKID フラグを設定することで globalIndex の上位 6 ビットをバンク ID として利用していることを示す。このフラグがない場合は globalIndex フィールドは 32 ビットサイズとして扱われる。

PVMH_PS_BANKID フラグがオンの状態において pvm が globalIndex フィールドを持たない場合でもバンク ID を持つ場合は globalIndex フィールドは存在し上位 6 ビットにバンク ID が格納される。上位 6 ビット以外は無効。

テクスチャがパレットタイプかどうかはテクスチャデータである PVRT チャンクのカテゴリコードから決まる。パレットは総エントリ数が 1024 エントリ。これを 4bpp の 16 色単位で 64 (0 ~ 63) バンクに分割する。このバンク ID が globalIndex の上位 6 ビットに格納される。8bpp の場合は 256 色 4 バンクで扱われますがバンク番号は 4bpp の場合のものと共通化されており 0, 16, 32, 48 のみが 8bpp の場合のバンク ID として有効。モデルコンバータからのバンク ID 指定はマテリアルネームからできる。

3.2 PVM チャンクレイアウト

pvm ファイルのチャンクレイアウトは次の通り。括弧内はバイト数を示す。各チャンクの bytesize は次のチャンクヘッダまでのバイトオフセット。

```
<Pvm format>
--
['PVMH'(4)][bytesize(4)]    IFF ヘッダ
[PVMH チャンクデータ](PVM 情報チャンク)
['MDLN'(4)][bytesize(4)]    IFF ヘッダ
[MDLN チャンクデータ](モデルネームリストチャンク)
['CONV'(4)][bytesize(4)]    IFF ヘッダ
[CONV チャンクデータ](コンバータ)
['PVMI'(4)][bytesize(4)]    IFF ヘッダ
[PVMI チャンクデータ](マニピュレーション情報チャンク)
['IMGH'(4)][bytesize(4)]    IFF ヘッダ
[IMGH チャンクデータ](オリジナル画像チャンク)
['PVPN'(4)][bytesize(4)]    IFF ヘッダ
[PVPN チャンクデータ](パレットネームリストチャンク)
['PVPL'(4)][bytesize(4)]    IFF ヘッダ
[PVPL チャンクデータ](パレットデータ)
['PVRT'(4)][bytesize(4)]    IFF ヘッダ
[PVRT チャンクデータ](PVR データチャンク)
['COMM'(4)][bytesize(4)]    IFF ヘッダ
[COMM チャンクデータ](コメントチャンク)
--
```

IMGH チャンクと PVRT チャンクはエントリ分繰り返される。

PVPL チャンクは任意の数を並べることができる。PVRT チャンク群の前に配置される。

PVPN は PVRL のファイル名を格納する。PVPL の数は任意のためそれぞれの PVPL の前にそれぞれのファイル名である PVPN を配置する。

pvm ファイルの先頭には必ず PVMH チャンク配置される。ただしコメントチャンク COMM は PVMH の前にあってもよい。(COMM チャンクを除いて)先頭が PVMH でない場合 pvm ファイルとして扱われない。PVMH 以外のチャンクはデータを省略可能(この省略による情報量の変化をインフォレベルと呼ぶ)。ただしチャンクの組み合わせによりあまり意味を持たないチャンクデータの省略の組み合わせもありうる。

IFF 方式を採用しているためユーザは独自のチャンクを定義し情報を加えることができる。PVM ツールは自分の知らないチャンクヘッダとデータは読み飛ばして処理をする。

3.2 PVM チャンク仕様

ChunkName : 'PVMH'

(PVM Header Information)

概要 :

pvm ファイルのヘッダ情報。pvm のファイルの先頭には必ず PVMH チャンクを置く。PVM のインフォレベルを示すフラグ、格納されている pvr の数、texId を制御するための entryId、pvr の名前、パレットデータの有無、タイプ情報を持つ。またパレット使用時には globalIndex の上位 6 ビットにはパレットバンク ID を格納する。

形式 :

```
[ 'PVMH' (4) ][byteSize(4)]
[pvmstatus(2)][entrycount(2)]
{ [entryId(2)][pvrname(28)][categoryCode(2)]
[entryStatus(2)][BankId|globalIndex(4)] }
...
```

説明 :

{ } は entrycount 分だけ繰り返される。entrycount はエントリ数を利用する他のチャンクでのエントリ数でもある。pvrname、categoryCode、entryStatus、globalIndex フィールドは pvmstatus のフラグによりデータを省略することができる。

pvmstatus :

2 バイトフラグデータ。現在の pvm ファイルに含まれるチャンクの種類を与える。エントリで使われるフィールド、チャンクを指定できる。

```
#define PVMH_PS_PVRNAME          (1<<0)
#define PVMH_PS_CATEGORYCODE    (1<<1)
#define PVMH_PS_ENTRYINFO       (1<<2)
#define PVMH_PS_GLOBALINDEX     (1<<3)
#define PVMH_PS_CHUNK_MDLN      (1<<4)
#define PVMH_PS_CHUNK_CONV      (1<<5)
#define PVMH_PS_CHUNK_PVMI      (1<<6)
#define PVMH_PS_CHUNK_IMGC      (1<<7)
#define PVMH_PS_CHUNK_PVRT      (1<<8)
#define PVMH_PS_CHUNK_COMM      (1<<9)
#define PVMH_PS_BANKID          (1<<10)
#define PVMH_PS_CHUNK_PVPL      (1<<11)
#define PVMH_PS_CHUNK_PVPN      (1<<12)
```

PVMH_PS_PVRNAME pvrname フィールドがあることを示す。

PVMH_PS_CATEGORYCODE CategoryCode フィールドがあることを示す。

PVMH_PS_ENTRYINFO EntryStatus フィールドがあることを示す。

PVMH_PS_GLOBALINDEX GlobalIndex フィールドがあることを示す。

(注意) Ninja2 ではこれらの四つのフィールドは必ずあるものとして pvm を扱う。

PVMH_PS_CHUNK_MDLN チャンク MDLN があることを示す。

PVMH_PS_CHUNK_CONV チャンク CONV があることを示す。

PVMH_PS_CHUNK_PVMI チャンク PVMI があることを示す。

PVMH_PS_CHUNK_IMGC チャンク IMGC があることを示す。IMGC がある場合は PVMH の entrycount の値と同じ個数の IMGH チャンクが連続して格納される。

PVMH_PS_CHUNK_PVRT チャンク PVRT があることを示す。PVRT がある場合は PVMH の entrycount の値と同じ個数の PVRT チャンクが連続して格納される。

以下はパレットが使われる場合のみに利用される。

PVMH_PS_BANKID globalIndex フィールドの上位 6 ビットをバンク ID として利用することを示す。

PVMH_PS_CHUNK_PVPL PVPL チャンクが pvm 内部にあることを示す。PVPL チャンクはテクスチャ PVRT チャンクよりも前にパレットは格納される。

PVMH_PS_CHUNK_PVPN PVPL チャンクの前にファイル名である PVPN チャンクを持つことを示す。

entrycount :

pvm ファイルに格納される pvr データの数を与える。他のチャンクにおけるエントリ数としても使われる。

entryId :

データ生成時のエントリのシリアルナンバー。通常は 0 から始まる。pvm が生成された時点では EntryId とモデルの TexId は一致しかつ PVRT の格納順も EntryId に一致する。

例えばエントリの順番の変更 (texlist 上でのテクスチャ順番の変更と等価) は PVMH 上でされるが i 番目と j 番目の位置を変えたい場合、{ } で括られた単位でデータの中身をスワップする。

Swap(data[i], data[j])

他のチャンクのスワップはしなくても PVMH の EntryId を見ればそのデータが他のチャンクの何番目に対応する情報がわかる。

pvrname :

Texlist のテクスチャの名前に相当する。例えば GlobalIndex 設定ツール Gigen により globalIndex を生成する場合この文字列が一致したエントリを同じテクスチャと考え番号が振られる。

categoryCode :

PVR のフォーマットの種類を示すカテゴリコード。本来 PVRT の中を見ればこのコードは格納されているが現在のテクスチャのタイプを簡単に確認するために先頭の PVMH にも持たせている。

entryStatus :

pvm に格納されている pvr データの情報。CategoryCode とともにユーザに現在のテクスチャの状態を示す。テクスチャのサイズ、変換時にディザを使ったかなどを格納する。またツールからの操作で各エントリに対して操作を制限する禁止ビットも用意する。

```
#define PVMH_ES_W8           (1)
#define PVMH_ES_W16          (2)
#define PVMH_ES_W32          (3)
#define PVMH_ES_W64          (4)
#define PVMH_ES_W128         (5)
#define PVMH_ES_W256         (6)
#define PVMH_ES_W512         (7)
#define PVMH_ES_W1024        (8)
#define PVMH_ES_H8           (1<<4)
#define PVMH_ES_H16          (2<<4)
#define PVMH_ES_H32          (3<<4)
```



```
#define PVMH_ES_H64          (4<<4)
#define PVMH_ES_H128         (5<<4)
#define PVMH_ES_H256         (6<<4)
#define PVMH_ES_H512         (7<<4)
#define PVMH_ES_H1024        (8<<4)
#define PVMH_ES_LOCK_ORDER   (1<<8)
#define PVMH_ES_LOCK_PVR     (1<<9)
#define PVMH_ES_DITHER       (1<<10)
#define PVMH_ES_ADITHER       (1<<11)
```

PVMH_ES_LOCK_ORDER	一つ前のテクスチャとの順番をロックする。
PVMH_ES_LOCK_PVR	すでに存在する pvr ファイルの上書きを禁止する。
PVMH_ES_DITHER	変換時にディザを使ったことを示す。
PVMH_ES_ADITHER	変換時に ディザを使ったことを示す。
PVM_ES_W8	テクスチャの幅 8 ピクセル
PVM_ES_W16	テクスチャの幅 16 ピクセル
PVM_ES_W32	テクスチャの幅 32 ピクセル
PVM_ES_W64	テクスチャの幅 64 ピクセル
PVM_ES_W128	テクスチャの幅 128 ピクセル
PVM_ES_W256	テクスチャの幅 256 ピクセル
PVM_ES_W512	テクスチャの幅 512 ピクセル
PVM_ES_W1024	テクスチャの幅 1024 ピクセル
PVM_ES_H8	テクスチャの高さ 8 ピクセル
PVM_ES_H16	テクスチャの高さ 16 ピクセル
PVM_ES_H32	テクスチャの高さ 32 ピクセル
PVM_ES_H64	テクスチャの高さ 64 ピクセル
PVM_ES_H128	テクスチャの高さ 128 ピクセル
PVM_ES_H256	テクスチャの高さ 256 ピクセル
PVM_ES_H512	テクスチャの高さ 512 ピクセル
PVM_ES_H1024	テクスチャの高さ 1024 ピクセル

globalIndex :
テクスチャの通し番号。

ChunkName : 'COMM'

(pvm COMMent)

概要 :

情報を文字列で格納 (コメント) できる。pvm ファイルの任意の位置に配置できる。pvm ロードはデフォルト動作でこの領域を無視して次のチャンクへ処理を進める。

形式 :

```
['COMM'(4)][bytesize(4)]  
[free comment string]
```

説明 :

free comment string :

任意の文字列からなるコメント。Bytesize が string の長さになる。

ChunkName : 'MDLN'
(pvm MoDeL Name list)

概要 :

カレントの pvm ファイルを利用して描画されるモデルデータ群のファイル名。pvm 側からモデルを見つけ texId の変更ができる。複数ファイルがあってもよく数は他のチャンクと無関係に設定できる。ツールで pvm のマージをする場合などに複数のモデルファイル名が格納される可能性がある。ファイル名がパスを持つ場合他のマシンへコピーしたりした場合はモデルファイルが参照できなくなる。パスの先にモデルがない場合はカレントディレクトリ、ユーザ指定パスからの検索もできる。

形式 :

```
[ 'MDLN' (4) ][byteSize(4)]  
[modelcount(2)]  
{ [model name string] }  
...
```

説明 :

{ } は参照されるモデルの数だけ繰り返される。TexId 更新のためのモデル群を与える。

modelcount :

pvm のテクスチャを使うモデル群の数。テクスチャは各モデルデータの texId から参照される。

model name string :

モデルの名前。通常絶対パス付きで記述する。変換したマシン以外に持っていくとパスは無効になってしまうので注意が必要。この場合 pvm のあるカレントディレクトリが利用される。

ChunkName : 'CONV'
(pvm CONverter)

概要 :

デフォルトの pvm 生成コンバータは PVMConv。pvm 出力可能な他のコンバータを使う場合このチャンクにコンバータ名を記述する。絶対パスでもいいがパスが設定されている場合はコンバータ名だけでも可能。またコンバータ名の後ろにはデフォルトで必ずつけるオプションがあればそれを書いてよくコンバータ名を省略してオプションだけが書かれた場合は PVRConv で必ずつけるオプションとして利用される。

(例)

/usr/local/project/pvmconv2 -di pvmconv2 -di を使って変換。
pvmconv2 パスがはられている場合に pvmconv2 を使って変換。
-vq4 -t PVMConv に -vq4 -t オプションをつけて変換。

また各テクスチャごとに指定されたオプションは PVMi チャンクの option string に格納され再変換時に再度コンバータに引き渡されて実行される。

PVMi のオプションデータはコマンドラインのコンバータのオプション文字列 (例えば -vq4) で保存される。この方法により構造体で管理するより簡単にさまざまなコンバータのオプションの保存/再利用が可能となる。新たな pvm コンバータを作成する場合 PVMConv のオプションを利用できるように作ることを推奨。

形式 :

```
[ 'CONV' (4) ][bytesize(4)]  
[converter name string(pathbytesize)]
```

説明 :

pathbytesize : converter name データのバイトサイズ。

converter name string : コンバータの名前、オプション文字列。

ChunkName : 'PVMl'
(PVM Information)

概要 :

pvm 生成時に利用した元画像ファイルの名前とパス、コンバータで利用したオプション文字列を保存する。これによりオプション変更をしてテクスチャの再変換ができる。また各テクスチャごとに指定されたオプションを保存できる。バッファサイズは必要に応じてツールが変えられるようにバイトサイズをデータとして格納している。元画像のパスは他のマシンにコピーされると無効になるので注意が必要。エントリ数は PVMH チャンクの entrycount と同じでなければならない。オプション文字列フィールドは単にバイトサイズを指定しているだけなので他の PVMConv 以外のコンバータでオプションを構造体で持ちたい場合は構造体のバイトサイズだけ領域を確保し書き込むことも可能。ただし CONV チャンクは必ず定義しそのデータを解釈できるコンバータを定義する必要がある。

形式 :

```
[ 'PVMl' (4) ][byteSize(4)]  
[pathbyteSize(2)][optionbyteSize(2)]  
{ [texture name string(pathbyteSize)]  
  [option string(optionbyteSize)] }  
...
```

説明 :

{ } は PVMH チャンクの entrycount 分だけ繰り返される。テクスチャパスの深さの最大値、オプション数の最大値を確認し適切なバッファサイズを確保し文字列を格納する必要がある。

pathbyteSize : texture name string バッファのバイトサイズ。

optionbyteSize : option string バッファのバイトサイズ。

texture name string :

元画像のパスと名前を格納する。他のマシンにコピーされると無効となるので注意が必要。

option string :

コンバータのコマンドライン上のオプションを文字列として格納する。

ChunkName : 'IMGC'
(Original IMaGe Container)

概要 :

コンバータで変換する前の画像を格納する。他のマシンにコピーしても再変換が可能。ただしファイルサイズが大きくなるので注意が必要。元画像が必要無くなった時点でインフォレベルを下げ IMGC チャンクを削除することでデータサイズを小さくしてゲームに使う。

形式 :

```
[ 'IMGC' (4) ][byteSize(4)][type(2)][image data(byteSize-2)]
```

説明 :

取り扱う元画像のフォーマットはバイナリであれば何でもよくタイプは type から確認される。

type :

IMGC_FMT_ARGB ARGB8888 でテクスチャを格納する。

image data : [ARGB8888]...

IMGC_FMT_PAL8 palette 8 でテクスチャを格納する。

image data : [palette256 エントリ][index]...

IMGC_FMT_BMP bmp を格納する。

IMGC_FMT_TGA tga を格納する。

IMGC_FMT_PIC pic を格納する。

IMGC_FMT_PIX pix を格納する。

ChunkName : 'PVRT'
(PVR Texture)

概要 :

テクスチャデータ。

形式 :

['PVRT'(4)][bytesize(4)][pvr data]

説明 :

PVRConv が出力する pvr テクスチャデータ。

ChunkName : 'PVPN'
(PVP Name)

概要 :

パレットの PVPL のファイル名を格納する。

形式 :

```
[ 'COMM' (4) ][ bytesize (4) ]  
[ pallete name string ]
```

説明 :

pallete filename string :

パレットの PVPL のファイル名を格納する。Bytesize が string の長さになる。このファイル名はパレットの pvm からの抜き出し時のファイル名として利用される。パレットデータがある場合にデフォルトインフォレベルで存在する。

ChunkName : 'PVPL'
(PVr PaLette)

概要 :

パレットデータ。

形式 :

```
[ 'PVPL' (4) ][ bytesize(4) ][ category code(2) ][ bankId(2) ]  
[ entryoffset(2) ][ entrycount(2) ][ palette data ]
```

palette data:

RGB565 の場合

[16bitRGB] の一次元配列。

ARGB1555、ARGB4444 の場合

[16bitARGB] の一次元配列。

ARGB8888 の場合

[32bitARGB] の一次元配列。

説明 :

PVRConv が出力するパレットデータ。パレット総エントリ数は1024、これをバンクに区切りバンク番号と画像のインデックス番号でテクスチャを描画する。バンク番号は4bppの場合に0~63まで8bppの場合は0, 16, 32, 48の四つだけが指定可能。パレットに対するbank指定、そのバンクの書き込み開始オフセット(entryoffset)とエントリ数(entrycount)によりパレットカラーを書き込む。パレットデータがある場合にデフォルトインフォレベルで存在する。

3.3 インフォレベル

pvm ファイルはその中に含まれるチャンクデータで複数の形態を持つ。これをインフォレベルと呼ぶ。インフォレベルが高ければより多くのことができる。最終的にはインフォレベルを下げゲームに必要なデータ部分のみを残す。この状態では再加工できなくなる。インフォレベルの変更は PVMConv もしくは他のツールでされる。

3.3.1 pvmstatus

PVMH チャンクはファイルの先頭に必ず置かれ、COMM チャンクは単なるコメントであるためインフォレベルに影響しません。それ以外の PVMH チャンクの pvm status に設定されるフィールドとチャンクの有無がインフォレベルに影響する。

PVMH_PS_PVRNAME がないと名前による重複チェックができないためグローバルインデックスの振り直しができない。

PVMH_PS_CATEGORYCODE、PVMH_PS_ENTRYINFO がないとユーザに対し現在のテクスチャのタイプなどの情報が示せない（ただしツールが対応すれば PVRT チャンクから直接このカテゴリーコードを読み出すことも考えられる）。またスモールVQだけを拾い出して並べるなどのタイプ別ソートができない。

PVMH_PS_GLOBALINDEX がないと Global Index を設定できない。

以上の理由により上記の四つのデータは Ninja 用では必須。Ninja 以外の用途で使う場合で上記のデータが不要な場合フィールドを切り捨ててください。

PVMH_PS_CHUNK_MDLN がないと pvm からモデルの texId の更新ができない。

PVMH_PS_CHUNK_CONV がいない場合は PVMConv が利用される。独自の使い方をユーザが望む場合、PVMConv 以外のコンバータを作った場合には PVMConv を置き換えてください。

PVMH_PS_CHUNK_PVMI がないとテクスチャの再変換ができない。例えば元画像を変更し再度コンバートしたい場合など pvm ファイルからはできなくなる。

PVMH_PS_CHUNK_IMGC がないと他のマシンへ pvm ファイルをコピーした場合に再変換ができない。通常はデータサイズが大きくなるので格納しないことを推奨。

PVMH_PS_CHUNK_PVRT がないと基本的にはゲームには使えない。ただし pvm と同じディレクトリ pvr が展開されている場合に限り pvm ファイル内に PVRT チャンクが無くても動作する。この状態であればゲームで使うこともできる。この場合 pvm は texlist 生成のための情報ファイルとして動作する。

PVMH_PS_CHUNK_PVPL は pvm にパレット情報が含まれることを示す。pvr にパレットタイプを使っている場合に関連するパレットデータをテクスチャと一緒に扱うことができる。

PVMH_PS_CHUNK_PVPN は pvm にパレットファイルのファイル名が含まれることを示す。パレットの pvm からの抜き出し時のファイル名として利用される。

最終的にゲーム用にインフォレベルを下げた状態は PVMH チャンクとグローバルインデックス (PVMH_PS_GLOBALINDEX) PVRT チャンク (PVMH_PS_CHUNK_PVRT) だけの構成となる。これにより texlist を生成し従来どおりにテクスチャが利用可能。

```
['PVMH'][bytesize]
[pvmstatus][entrycount]
{ [entryId][pvrname][globalIndex] }
...
{ ['PVRT'][bytesize][pvr data] }
```

ここで pvmstatus は以下の通り。

```
pvmstatus=PVMH_PS_PVRNAME
          | PVMH_PS_GLOBALINDEX
          | PVMH_PS_CHUNK_PVRT;
```

このインフォレベルでは

複数のテクスチャを一括して扱える。
texlist の生成ができる。
グローバルインデックスの付け替えができる。
テクスチャ変換時の履歴は見れない。
テクスチャの再変換はできない。

パレットが必要な場合は PVPL チャンクが付加される。pvmstatus には global Index の上位 6 ビットにパレットバンク ID が含まれることを示す PVMH_PS_BANKID フラグとパレットデータ PVPL チャンクが pvm に含まれることを示す PVMH_PS_CHUNK_PVPL フラグが設定される。

```
['PVMH'][bytesize]
[pvmstatus][entrycount]
{ [entryId][pvrname][bankId|global Index] }
...
{ ['PVPN'][bytesize][palette name]['PVPL'][bytesize][palette data] }
...
{ ['PVRT'][bytesize][pvr data] }
```

ここで pvmstatus は以下の通り。

```
pvmstatus=PVMH_PS_PVRNAME
          | PVMH_PS_GLOBALINDEX
          | PVMH_PS_BANKID
          | PVMH_PS_CHUNK_PVPN
          | PVMH_PS_CHUNK_PVPL
          | PVMH_PS_CHUNK_PVRT;
```

PVPL チャンクは任意の数格納可能。ただしハードウェアは複数のピクセルタイプを同時に使えないため RGB565、ARGB8888、ARGB1555、ARGB4444 のうちのどれか一種類だけのパレット群のみを格納する必要がある。

3.4 オペレーション

3.4.1 規約

ここで pvm 用ツールを作る場合を守るべきルールをまとめる。

次のチェックをパスできない場合規約違反の pvm ファイルとしてエラー処理する。Pvm ツール製作者は次のチェックをパスできるように pvm ファイルを生成しなければならない。

COMM チャンクは任意の位置に配置される。pvm ロードはこれを読み飛ばすことができる。

COMM チャンクを除いた状態で PVMH チャンクが必ず先頭にくる。これを検出できない場合 pvm ファイルでないものとしてエラーとする。

PVMH の entrycount は pvm ファイルに含まれるテクスチャの数を与える。IMGH チャンク、PVRT チャンクに含まれるテクスチャの数はこれと一致しなければならない。一致しない場合エラーとする。

pvm に関しては PVMConv もしくは Ninja2 データオプティマイザーを利用してください。

3.4.2 デフォルトインフォレベル

デフォルトで出力されるインフォレベルは次の通り。

```
['PVMH'][bytesize]
[pvmstatus][entrycount]
{ [entryId][pvrname]
  [categoryCode(2)][entryStatus(2)][globalIndex] }
```

...

```
{ ['PVRT'][bytesize][pvr data] }
```

...

ここで pvmstatus は以下の通り。

```
pvmstatus=PVMH_PS_PVRNAME
| PVMH_PS_CATEGORYCODE
| PVMH_PS_ENTRYINFO
| PVMH_PS_GLOBALINDEX
| PVMH_PS_CHUNK_MDLN
| PVMH_PS_CHUNK_PVMI
| PVMH_PS_CHUNK_PVRT;
```

パレット使用時は PVMH_PS_BANKID、PVMH_PS_CHUNK_PVPN、PVMH_PS_CHUNK_PVPL フラグが追加される。

このインフォレベルでは

複数のテクスチャを一括して扱える。

texlist の生成ができる。

グローバルインデックスの付け替えができる。

カテゴリーコードによるテクスチャのグループ分けができる。スモールVQの最適化に重要。

テクスチャの種類変換時のディザの状態、サイズ、変換オプションなどの情報が見れる。

pvm を変換したマシン上で指定されたテクスチャだけの再変換ができる。各テクスチャごとにオプション指定が可能。

pvm のテクスチャの順番を入れ替えたり複数の pvm をマージできる。またこの時参照される複数のモデルの texId を更新できる。

ユーザが編集したテクスチャの上書きを禁止できる。

パレットの取り込み、取り出しが可能。

3.4.3 インフォレベルの変更

インフォレベルのアップ(データの追加)は再コンバートによりされる。コンバートし直してください。インフォレベルを下げる(データの削除)は PVMConv で可能。また次の操作は動的に行うことができる。

グローバルインデックスの更新。
モデル texId の更新。
pvm から pvr ファイル群の展開。pvr ファイル群の取り込み。
pvm から元画像ファイルの展開。元画像の取り込み。
pvm 同士のマージ。pvr データの pvm エントリからの削除。
テクスチャエントリの順番の入れ替え。

3.4.4 テクスチャの順番の入れ替え

順番の入れ替えは PVMH チャンクのエントリデータのスワップのみによりされる。他のチャンクデータの順番の入れ替えはしなくても PVMH チャンクの状態変更で差し替えの表現が可能。もちろん全チャンクデータを差し替えることでも入れ替えは可能。

最初の状態のエントリ番号が entryId に格納されているので順番が変わっても他のチャンクデータのどのエントリと対応させればいいのかを得ることができる。この時のモデルの texId の更新は次の通り。

MDLN チャンクから関係するモデルファイルを得る。これが得られない場合エラーとなる。

PVMH チャンクからデータを取り出す。現在の順番と entryId を対応づけるテーブルを作成する。

```
entryList[entryId] = i; (ここで i は PVMH の実際のエントリ番号)
```

このテーブルの entryId の代わりにモデルの texId を入れた時の番号 (元は i) が新しい texId となるのでこれでモデルデータを置換する。

3.4.5 pvm のマージ

二つの pvm ファイルのマージをする場合は二つの pvm ファイルの各チャンク同士を連結し PVMH の連結した二つ目の entryId に一つ目の PVMH の entrycount を加算する。また二つの entrycount の合計を新しい pvm の entrycount に入れる。この状態では MDLN チャンクには二つのモデルネームが格納される。

3.4.6 PVMConv

PVMConv は pvm ファイルを操作する基本ツール。pvm のすべての機能を包含する。次に説明する pvmUtil ライブラリを元に pvm を操作/編集する。

3.4.7 pvmUtil

次の機能を持つ。

pvr ファイルの変換、Ninja リソースファイル.nre を参照し各テクスチャを変換する。

pvr 変換時に pvm の各チャンクデータを生成できる。

すべてのチャンクデータを pvm に書き込める。

pvm ファイルからチャンクデータを読み出せる。

二つの pvm をマージできる。

pvm から pvr を展開できる。

pvm から元画像を展開できる。

モデルの texId の更新ができる。

pvm の COMM チャンクをテキストで表示できる。

指定されたフィールド、チャンクを消去しインフォレベルを下げるができる。

pvm の中身の情報を表示できる。

pvm のモデルのパスを書き換えられる。

pvm の元画像のパスを書き換えられる。

4. マテリアルネーム（サーフェスネーム）仕様

4.1 マテリアルネーム（サーフェスネーム）概要

Ninja2 では各モデラーで共通に利用可能なマテリアルネームを使いモデルに対するフラグ、オプション設定をする。基本的には各モデラーごとに用意されたマテリアルネームエディタ（LightWave はサーフェスネームエディタ）でマテリアルを選択しダイアログからのボタンの ON/OFF で設定をする。直接文字列を名前に設定することも可能。またモデラーそのものの機能として FLAT（soft image の Faced 等）などの指定がされている場合マテリアルネームに設定されていなくてもフラグが変換時に設定される場合があるので注意が必要。マテリアルネームから設定できるのは次の設定。

マテリアル単位のポリゴンに対するオプション。フラット、両面、光源無視、フィルタの種類等。

パレットテクスチャ時のパレットバンク番号。

モデル単位のオプション。UVN(0-255)、UVH(0-1023)の選択、ユーザフラグ領域サイズ等。モデル単位の設定はモデルに使われているすべてのマテリアルのどれかに設定されると有効。トラブル防止のため複数のマテリアルを使っている場合複数のマテリアルに分散してモデル単位のオプションフラグを設定することは避ける。

4.2 マテリアルネーム一覧

ここでの仕様は各モデラーで共通。マテリアル単位で影響するものを表 1 にモデル単位で影響するものを表 2 にまとめる。

文字列	機能名	説明
Dxxx	Mipmap adjust 'D'	adjust 'D'パラメータを設定する。D に続けて数字が三つ並ぶと有効。D000 は無効。通常はデフォルトの D100 を利用のこと。25 刻みで 025 ~ 375 まだが有効。補足説明参照。
Axx	blending Alpha	A に続けて数字が二つ並ぶと有効。
Bxx	Palette Bank	B に続けて数字が二つ並ぶと有効。8bpp の場合 B00, B16, B32, B48 の 4 つ、4bpp の場合 B00 ~ B63 が有効。
An	Anisotropic Filter	Anisotropic Filter を ON する。Ps, Bi, TI と独立して使える。
Ps	Point Sampling Fliter	テクスチャを Point Sampling で描画する。
Bi	Bilinear Fliter	テクスチャを Bilinear で描画する。
TI	Trilinear Filter	テクスチャを Trilinear で描画する。
D	Double Side	両面ポリゴンフラグ。
E	Environment Mapping	環境マッピングフラグ。
En	Environment Mapping2	環境マッピングでモデルの中心からの放射状の法線を生成する。環境マッピングの映り込みが E の場合と比べ変化する。En のセットされたマテリアルに属する頂点の法線のみ放射状に再計算される。
F	Flat Shading	フラットシェーディングフラグ。
L or NI	Ignore Light	光源無視。
Ns	Ingore Specular	スペキュラ無視。
Na	Ignore Ambient	アンビエント無視。
G	Gouraud Shading	グローシェーディングフラグ。実際はモデラーで設定されたフラットシェーディングフラグをマテリアル単位で無効する。Softimage 等で利用。
Cu	U ClampBit ON	強制的に U 方向にクランプビットを立てる。
Cv	V ClampBit ON	強制的に V 方向にクランプビットを立てる。
Ua	Use Alpha	強制的に アルファビットを立てる。
2P	2 para modifier data	モディファイアボリュームに対しテクスチャ、マテリアルを二重に持ち交差判定で切り替えられる 2 パラモディファイアデータとして出力する。本マテリアルの関わるテクスチャが 1 枚の場合無効。

表 1 マテリアル単位で影響するもの

文字列	機能名	説明
H	Hiresolution UV	0-1023 の分解能で UV 値を出力する。
Es	Eval Skip	モデルに対するオプション。このモデルはモーションに関与しないことを示す。モーションデータにこのモデルのデータは出力されない。オブジェクト構造体に EvalSkip フラグが設定される。
Ss	Eval Shape Skip	モデルに対するオプション。このモデルは頂点モーションに関与しないことを示す。頂点モーションデータにこのモデルのデータは出力されない。オブジェクト構造体に EvalShapeSkip フラグが設定される。
Hd	Eval Hide	モデルをハイドし描画しない。
Bk	Eval Break	子供モデルのトレースを打ち切る。モーション実行時は nam のデータとずれてくるので注意が必要。ルートモデルに設定するとモデル階層全体をまとめてハイドできる。
Kt	Keep translation	モデルに対するオプション。モーション出力において translation が NULL として出力されないことを保証する。
Kr	Keep rotation	モデルに対するオプション。モーション出力において rotation が NULL として出力されないことを保証する。
Ks	Keep scale	モデルに対するオプション。モーション出力において scale が NULL として出力されないことを保証する。
Ec	Eval Clip	<p>EvalClip とはモデル単位で行われている画面上に表示されるかのクリップ判定処理を指定されたモデルの中心点と半径で行いこれが画面外の場合以降子供のモデルのクリップ判定を省略する機能。クリップ判定計算量を減らせる。</p> <p>外接球を発生させるための EvalClip 設定用モデルを作成これを EvalClip を有効にしたいモデルの子供として設定する。このモデルに任意の形状を設定しマテリアルを貼りつけこれにマテリアルネームエディタで Ec を設定する。コンバータは Ec を検出しこのモデルの外接球を計算し中心点と半径をその親のモデルに設定し EvalClip を有効にし EvalClip 設定用モデルを削除する。設定したいモデルが NULL モデルの場合ダミーのモデルデータが設定され中心点、半径が設定される。</p>
—	End	アンダーバー以降を評価しない。マテリアルに設定されたフラグ文字列の最後と実際のマテリアルネームの間に設定する。

表 2 モデル単位で影響するもの

4.3 補足説明

4.3.1 Es,Ss,Kt,Kr,Ks,Hd オプション

Es,Ss,Kt,Kr,Ks はモデルに対するオプション。モデルに利用される複数のマテリアルのどれかに設定されると有効。このフラグ群はmrsに出力される。Es,Ss,Kt,Kr,Ks,Hdを設定した基本型から求められたmrsを利用してモーション出力をした場合モーションデータの格納されるシーンファイルのモデルにEs,Ss,Kt,Kr,Ks,Hdが設定されていなくてもmrsの設定が利用できる。

4.3.2 'G'オプション

gouraud shading をすることを示す。これは設定されて flat shading フラグを無効化することで実現される。例えば softimage の標準機能でモデル全体にフラットの設定ができ (Faced) このモデルの一部を gouraud shading にしたい場合に使用する。

4.3.3 'Axx'オプション

Axx の xx にはそれぞれ SRC, DST の Field Value を設定する。

Instruction	Field Value	Values Returned
Zero	0	(0, 0, 0, 0)
One	1	(1, 1, 1, 1)
'Other' Colour	2	(O _R , O _G , O _B , O _A)
Inverse 'Other' Colour	3	(1 - O _R , 1 - O _G , 1 - O _B , 1 - O _A)
SRC Alpha	4	(S _A , S _A , S _A , S _A)
Inverse SRC Alpha	5	(1 - S _A , 1 - S _A , 1 - S _A , 1 - S _A)
DST Alpha	6	(D _A , D _A , D _A , D _A)
Inverse DST Alpha	7	(1 - D _A , 1 - D _A , 1 - D _A , 1 - D _A)

4.3.4 'Bxx'オプション

パレットバンク指定。B00 ~ B63 まで設定可能。ハードウェアは 1024 色をパレットに持てこれを 16 色単位に分け 64 バンクとする。4bpp の場合は 0 ~ 63 バンクを利用するが 8bpp の場合パレットは 256 色となるため番号の下位 4 ビットを無視して B00, 16, 32, 48 のみが有効となる。マテリアルネームからバンク番号が指定されると texlist ヘバンク番号が出力される。texlist では同一のテクスチャが複数回出てこないようにエントリをまとめるがバンク番号が指定された場合は同一テクスチャでもバンク番号が異なれば別エントリとして出力する。

4.3.5 'En'オプション

例えば平らな面に環境マッピングをしても法線が同じ場合環境マッピングに使われるピクセル値が同じとなり見栄えがよい。モデルのセンターと頂点を結んだ方向に法線を設定し見え方を改善する。あくまでも疑似環境マッピングに対する疑似法線生成であり見た目では見えそうな場合に利用する機能。モデルが完全な平面の場合モデルの外接矩形が幅を持たないと生成される法線は面の上ののってしまうためあまり効果はない。En を使ったマテリアルと使ってないマテリアルの両方に含まれる境界頂点は En が有効になるため境界のつなぎ目が多少おかしくなる場合がある。

4.3.6 'Dxxx'オプション

adjust 'D' パラメータに設定可能な数値は次の通り。

マテリアルネームエディタ設定値	設定される adjust 'D'
D025	0.25
D050	0.50
D075	0.75
D100	1.00
D125	1.25
D175	1.75
D200	2.00
D225	2.25
D250	2.50
D275	2.75
D300	3.00
D325	3.25
D350	3.50
D375	3.75

これ以外の数値は無効。D000 も無効。これはハードウェア仕様。

5. FlatBump 仕様

5.1 FlatBump 概要

ハードウェアのバンプ機能を利用するための仕組み。ハードウェアはテクスチャのピクセル値に法線情報を持つことでバンプマップを表示できる。ただし本ハードウェアは平らな連続する面において正面からテクスチャをプレーンで貼った場合を前提としている。平らでない丸みを帯びた面（曲面）に貼った場合ポリゴンの間の角度の影響で正しく描画されない。これはハードウェア仕様。

ハードウェアはテクスチャの貼った方向の情報が重要となる。この情報をマテリアルとして設定する。ここではハードウェア機能の枠の中でバンプマップを利用するための説明をする。

< 使用条件 >

原則テクスチャ貼り方をプレーン（XY 面貼り等）のみとする。

平面（flat）に対し利用のこと。

例えばシリンダカルで貼ることが可能だがこの場合方向によっては破綻する。ユーザ責任において利用可能。

光源をバンプマッピング面に垂直な位置にしない。この部分が破綻する。

テクスチャ二枚貼りに対応する。一枚をバンプテクスチャとすることでテクスチャとバンプが同時に使える。これは同一モデルを二回描くこと（2パス）で実現される。

テクスチャは一枚目に普通のもの二枚目にバンプマップを貼る。

5.2 FlatBump フォーマット

Chunk Model フォーマットにおいて FlatBump 用に次の定義をする。

NinjaCnk.h において

```
#define NJD_CM_BU      (NJD_MATOFF+8)      /* [CHead][6(Size)][dx(16)][dy(16)][dz(16)]  
                                              [ux(16)][uy(16)][uz(16)]          */
```

これはバンプマップ用のマテリアルとして貼った時の方向を与えるベクトルをポリゴン描画前に設定するチャンク。dx, dy, dz は貼る方向のベクトル、ux, uy, uz は上方向を示すベクトル。16ビット精度のため32767を1.0とする整数値に変換する。

アスキーフォーマットのために次のような定義をする。

NjDef.h において

```
#define CnkM_BU(_bits)      NJD_CM_BU  
#define _BuDir(_dx, _dy, _dz)  Pvn(_dx, _dy, _dz)  
#define _BuUp(_dx, _dy, _dz)  Pvn(_dx, _dy, _dz)
```

バンプマップ用のPlistが処理される前にCnkM_BUを呼びデータを設定する。チャンクサイズはショートサイズで常に6となる。Pvnマクロはポリゴンリスト用法線を出力するためのマクロでありフロートの入力値を16ビットの32767を最大値とする整数値に変換する。pvrロード時の情報でテクスチャの種類を識別しバンプとして処理する。

アスキーフォーマットにおける形式は次の通り。

```
CnkM_BU(0), 6,  
_BuDir(_dx, _dy, _dz),  
_BuUp(_dx, _dy, _dz)
```

CnkM_BU の括弧の中には常に 0 が入る。リザーブ領域。

(出力例)

```
CnkM_DA( FBS_SA|FBD_ISA), 4,  
MDiff( 255, 255, 255, 255),  
MAmbi(255, 255, 255, 255),  
CnkT_TID( FCL_U|FCL_V|FDA_100 ), _TID (FFM_BF, 0 ),  
CnkM_BU( 0), 6, <==設定  
_BuDir(1.000000f, 0.00000f, 0.00000f), <==設定  
_BuUp(0.000000f, 1.00000f, 0.00000f), <==設定  
CnkS_UVN( FST_IS ), 320, _NB( UFO_0, 13 ),  
StripL(8),  
234, Uvn ( 159, 0 ),  
...
```

6. Envelope 仕様

6.1 Envelope 概要

モーションを伴う一体整形モデル描画を実現する NinjaEnvelope 構造について説明する。Envelope の概念は各モデラーごとに存在し基本的な条件のみを利用する場合ほぼ同様の手法で実機での再現が可能。現在次のものに対応している。

Softimage の GlobalEnvelope
3D Studio MAX の physique
LightWave3D の Bone
MAYA

条件は次の通り。

『頂点が複数モデルの関節から影響を受けその影響度を合計 100% のウェイト値として持ちそのウェイト値のみを利用して一体整形モデルを描画する方式。』

これは関節のつながったモデル描画における基本であり公知の一般的な技術。NinjaEnvelope ではモデラーから取り出した頂点データ、ウェイト値を実機で高速に実行できる形式にコンバータで変換しこれを実行する。VN_NF 頂点チャンクを利用する。

次のようなオプティマイズをする。

ウェイト値を持つ頂点を各モデルに分配し頂点リスト化し従来の処理をほとんど変更することなく Envelope を描画できるようにしている。

処理の同じ頂点群をグループ化しメモリ連続の頂点リストを生成しキャッシュの処理効率を高めている。

EasyDraw のストアキュー対応のためにインデックス先頭が偶数番号から始まるように頂点番号生成を制御している。

ウェイト計算の必要のない頂点は従来の計算がされるようにし必要最小限の部分だけウェイト計算している。

共有頂点バッファに頂点を展開し複数モデルからの頂点書き込みその完了を待ってポリゴンを描画する。ポリゴン描画タイミングの自動検出している。

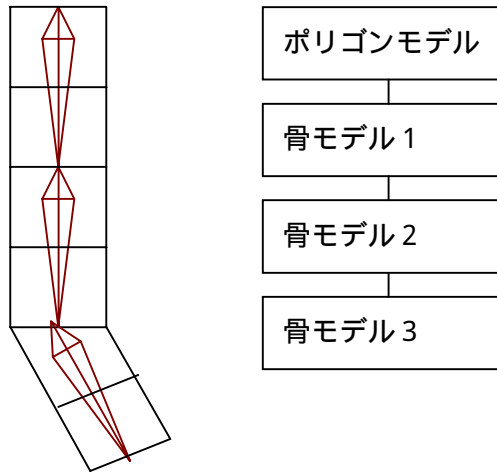
共有頂点バッファに格納される頂点が描画を完了したタイミングでそのエリアを開放しそのエリアを再利用することで描画にかかるメモリの効率化を実現している。この制御に合わせた頂点インデックスを生成している。

通常モデラーではポリゴンを持たないモデルを骨として組み込みポリゴンを持つモデルと組み合わせてモデル階層を構成する。ポリゴンを持たないモデルである骨はモーションに伴う変形に密接に関係する。顔モデルの内部に骨を仕込んだり、洋服やスカーフ、髪などに骨を仕込むことで動きによって変形を伴うキャラクターの表現が可能となる。

また CompactShape が利用される場合頂点グループ指定により頂点モーションをする部分に関するウェイト値は削除され部分的に Envelope1 が適用される。さらに頂点グループ指定されている頂点はソーティングされ Envelope1 用の頂点データの先頭に集められる。CompactShape 頂点モーションデータはこの先頭に集められた頂点データに対し実行される。

6.2 基本的な構造

基本的なモデル構造は次の通り。



この例ではポリゴンを持っているモデルは階層のトップのポリゴンモデルのみ。それに連なるポリゴンを持たない骨モデル 1 ~ 3 はポリゴンモデルの各頂点とウェイト値で関係を持つ。各頂点はモーションによる骨モデルの移動の影響をウェイト値分だけ受ける。複数の骨モデルからの結果をウェイト値の割合で合成した位置を頂点位置とすることでポリゴンモデルを変形し描画をする。また法線についてもウェイト値の割合で合成し求めている。

ポリゴンモデルと骨モデルの親子階層関係は任意。必ずしも上記のようにポリゴンモデルの下に骨モデルがくる構造は必要ない。骨モデルの兄弟にポリゴンモデルがくることも可能。この部分はモデラーの特性による。骨モデルとポリゴンモデルが階層構造を構成しモデルと頂点のウェイト値での関係が設定されていれば OK。

ポリゴンモデル自身も骨モデルと同じで自分の頂点や他のモデルの頂点に対し影響するウェイト値を持つことができる。

ある頂点が複数箇所からの影響を受けるウェイト値の合計は必ず 100% になる必要がある。100% になっていない部分は元モデルに残る % を割り振る事で 100% にしている。

6.3 頂点リストの再構成

エンベロープモデルでは通常一つの頂点は二つ以上のモデルからの影響を受ける。一つの頂点データに任意数のウェイト値を持たせるデータ構造は得策ではなくまた各モデルのモーションマトリックスを掛けた結果同士のウェイト値の割合による合成のために一つの頂点に対する処理を優先し複数のマトリックス結果を求める手順はマトリックス入れ替え分のキャッシュ効率が悪くなり良くない。そのため次の方法を採用する。ある頂点に注目しこれに影響するモデルが複数ある場合その頂点ウェイトとともに影響する側のモデルに頂点を分配する。頂点ウェイトが複数の場合各モデルに分配されるので複数の (ウェイト値を持った) 頂点になる。例えば三つの骨モデルから影響を受ける場合それぞれのモデルに頂点を持たせるためオリジナルの頂点は三つの頂点として扱われる。ここで分配時の重要なルールがある。

『骨モデルとポリゴンモデルの関係を決定付ける初期位置 (基本姿勢) におけるグローバルの頂点座標値を各モデルのローカル座標系から見たローカル座標値に変換し分配する。』

頂点は例えば三つのモデルに分配された場合それぞれのローカル座標系値に変換されるため見た目上同じ値の頂点にはならない。描画時にはインデックス番号を元にライブラリ実行中の頂点バッファ同一メモリアreaに各ウェイト値を掛けた値を累積加算することで結果を一つにまとめ最終的な頂点を求める。初期位置はとても重要。骨モデルとポリゴン頂点の相対位置を決め座標を決める。この初期位置を正しく与えることでモデラー上での変形を正しく再現することができる。その結果次のようなことが起きる。

他のモデルに 100% 支配される場合オリジナルモデルの頂点リストから消える。

ウェイト値を持たない頂点は 100% 自分自身に支配されると考えることができる。

骨モデルに頂点リストが発生する。100%依存ものはウェイト値を持たない頂点リスト、100%未満のウェイト値を持つものはウェイト値付きの頂点リストになる。チャンクフォーマットで各グループを複数チャンクで表現する。これらのデータは描画時にオリジナルモデルの頂点バッファに頂点を書き戻すインデックス情報を持つことでポリゴン描画を実現している。

ただし3D Studio MAXでは内部ですでにローカル座標値を持つため基本姿勢から求めることなくこのローカル値を使っている。

6.4 頂点リストグループ

処理高速化のために頂点を4つのグループに分ける。

Native	自分自身もしくは他のモデルに100%支配される（ウェイト値100%）頂点グループを与える。
StartWeight	描画時に頂点バッファに最初にかかれる分配された頂点グループを与える。
MiddleWeight	描画時に最初でも最後でもなく描画される分配された頂点グループを与える。
EndWeight	描画時に最後に書かれる分配された頂点グループを与える。

頂点はウェイト計算ありとなし（Native グループ）の二つのグループに分けられる。ウェイト計算ありの場合メモリへの書き込みや後処理タイミングを明確にするためにさらに三つのグループ（Start, Middle, End）に分ける。

Native グループはエンベロープモデルでない従来の処理のため高速。さらに処理を最適化するためコンバータは各グループの頂点を描画時に書き戻す領域がオリジナルモデル上でメモリ連続になるように順番を入れ替えている。

StartWeight グループでは累積の最初のためメモリアreaに上書きする。MiddleWeight グループではすでに書き込まれている値に新しい値を累積加算する。EndWeight グループでは累積加算後その頂点に対する計算が完了したことを与え正規化などデータ完成のタイミングを作る。

ここで次の点に注意する必要がある。

MiddleWeight と EndWeight グループの頂点は必ず StartWeight で書き込まれたメモリアreaへの累積加算処理。つまり Native グループと StartWeight グループの頂点の合計が全頂点数となる。

オリジナルモデル頂点リストにおいて各モデルに分配され散っていく頂点を描画時に効率よく書き戻すため頂点リストの頂点の順番を連続するように入れ替えグループ化する。ウェイト値ありなし、さらにウェイト値ありの場合影響を受けるモデル単位に頂点を連続させグループ化する。この結果 Native グループと各影響するモデルごとの複数の StartWeight グループがオリジナルモデル頂点リストに配列される。

オリジナル頂点リストへの書き戻しのためのインデックスがブロック先頭のオフセットもしくはウェイト値とともに与えられるインデックスとして StartWeight, MiddleWeight, EndWeight グループに格納される。

6.5 共有頂点バッファ

エンベロープモデルにおいて各モデルに分配された頂点の計算が完全に終わるのは必ずしもオリジナルモデル通過時ではない。それより後に計算されるモデルからのウェイトを受ける場合がある。この場合オリジナルモデルでのポリゴン描画はできない。そのため次の方法を採用している。

コンバータレベルでオリジナルモデルのポリゴン描画に関わる頂点計算が完了していないことを検出し完了していない場合未描画ポリゴンリストテーブル登録コマンドを発行しポリゴンリストを登録する。

コンバータレベルで未描画ポリゴンリストに登録されているポリゴンリストに関わる頂点計算が完了したタイミングを検出しこの未描画ポリゴンリストを実行するためのコマンドを他のモデルのポリゴンリストに書き込む。描画時にはこのコマンド発行とともにポリゴンリスト描画が実行される。

この機構によりポリゴンリスト描画タイミングを変えることができる。しかしここでもう一つ重要なことはポリゴン描画完了時まで複数モデルの頂点計算過程を保存する必要があることである。

そのため従来の頂点バッファに対し未描画部分の頂点がある場合この部分をつぶさないようにオフセットを与えた領域に次のモデルの頂点バッファを作り処理を継続する。複数のモデル間の頂点計算は共有頂点バッファ上で並列に処理される。モデルの違いは頂点インデックスのオフセットだけで表現される。

ポリゴンリスト描画が終わった時点で頂点バッファ領域を開放する。次のモデル描画では開放された領域を再利用する。オフセットによりポリゴンリストの頂点番号が影響を受け変化する。この方法により必要なメモリエリアを最小にし効率のよいエンベロープ処理を実現している。

6.6 Envelope フォーマット

エンベロープ用のウェイト値を持つ頂点定義に NJD_CV_VN_NF チャンクを使用する。

NinjaCnk.h において

```
#define NJD_CV_VN_NF      (NJD_VERTOFF+12)    /* x, y, z, nx, ny, nz, NinjaFlags32,... */
```

NinjaFlag 領域に頂点インデックスとウェイト値を格納する。

ウェイト計算における StartWeight, MiddleWeight, EndWeight の三つのサブグループを示すフラグをチャンクバーテックスの headbits に定義する。

```
#define NJD_FW_SHIFT      8
#define NJD_FW_START      (0<<NJD_FW_SHIFT)   /* Start */
#define NJD_FW_MIDDLE     (1<<NJD_FW_SHIFT)   /* Middle */
#define NJD_FW_END        (2<<NJD_FW_SHIFT)   /* End */
```

頂点バッファをクリアするタイミングを制御するためのフラグを同様にチャンクバーテックスの headbits に定義する。

```
#define NJD_FV_SHIFT      8
#define NJD_FV_CONT       (0x80<<NJD_FW_SHIFT)/* vertex calculation continue */
```

このフラグは頂点バッファのクリアをしないことを示す。エンベロープの頂点計算が継続中の場合絶えずフラグを ON しておく必要がある。またユーザが自力で親子の頂点をつなぐポリゴンを作る場合には親の頂点計算結果を残すためこのフラグを設定する必要がある。

PList に格納される未描画ポリゴンリスト登録コマンドと未描画ポリゴンリスト実行コマンドを Chunk Bits として定義する。

```
#define NJD_CB_CP          (NJD_BITSOFF+3)
#define NJD_CB_DP          (NJD_BITSOFF+4)
```

NJD_CB_CP は CashPolygon を意味しライブラリの持つ未描画ポリゴンリストテーブルにカレントのポリゴンリストを登録 I d とともに登録する。

NJD_CB_DP は DrawPolygon を意味し登録 I d から呼び出されるポリゴンリストを描画する。

NjDef.h において

```
/* Flag Weight Status (NF only) */
#define FW_START          NJD_FW_START
```



```
#define FW_MIDDLE    NJD_FW_MIDDLE
#define FW_END       NJD_FW_END
```

```
/* Flag Vertex */
```

```
#define FV_CONT      NJD_FV_CONT
```

```
#define CnkV_VN_NF(_f, _s)(_CkV(_f,_s)|NJD_CV_VN_NF)
/* x, y, z, nx, ny, nz, NFlags32, ... */
```

_f に FW_START, FW_MIDDLE, FW_END, FV_CONT が設定される。_s はロングワードでのチャンクサイズ。

```
#define _WPCa2(_p)      ((unsigned long)((_p)*65535.0F)/100.0F+0.5F))
#define NFlagsW2(_idx, _wp) (((_WPCa2(_wp) & 0xffff)<<16)|((_idx) & 0xffff))
```

共有頂点バッファ上でのエントリ番号とウェイト値を与える。32ビットのフラグ領域に16ビット精度で設定。

```
#define CnkB_CP(_bits)  ((_bits)<<8)|NJD_CB_CP
#define CnkB_DP(_bits)  ((_bits)<<8)|NJD_CB_DP
```

_bits には登録IDが入る。

< ウェイト値を持つ頂点リストの例 >

```
VLIST    vertex_same_heavywalk_body_3_kosi[]
```

```
START
```

CnkV_VN(FV_CONT, 7),	FV_CONTが頂点バッファのクリアをしないことを示す。
OffnbIdx(96, 1),	96は頂点インデックスのオフセット。
VERT(0x40b52c18, 0xbea8f5e8, 0x3bf782fa),	Nativeグループの頂点がこの例では一頂点のみ。
NORM(0x3f7ffff6, 0x3a8dd040, 0xb89018fa),	
CnkV_VN_NF(FV_CONT FW_END, 253),	頂点バッファクリアなしでEndWeightグループの頂点データ。
OffnbIdx(0, 36),	オフセットは0。
VERT(0x40ac2a72, 0xbea7c033, 0x3f9073dd),	
NORM(0x3f6b3ac5, 0x3b2f5a60, 0x3eca0400),	
NFlagsW(156, 99.793091),	頂点バッファの156番目に99.7%のウェイト値で書き込む。
VERT(0x4092877e, 0xbea97517, 0x4024c9c6),	
NORM(0x3f32fbfc, 0x3baf3c8, 0x3f3706cd),	
NFlagsW(157, 94.414238),	
VERT(0x405862d2, 0xc05ab548, 0x3a3f44d0),	
NORM(0x3e73aa6a, 0xbf78a58e, 0xb9fa8aa0),	
NFlagsW(158, 79.658073),	
...	
VERT(0x3eff389c, 0xc05467c4, 0x38e1df14),	
NORM(0xbdc142e6, 0xbf7edb8c, 0xb7aa0898),	
NFlagsW(211, 51.115231),	

```

VERT( 0x3e906948, 0x3fff9549, 0xc011885e ),
NORM( 0xbd8d0601, 0x3f3095e2, 0xbf3881c7 ),
NFlagsW( 212, 50.432468 ),
CnkEnd()

```

END

StartWeight、MiddleWeight グループも EndWeight グループと同様な仕様で記述される。

< ポリゴンリストを未描画ポリゴンリストに登録する例 >

```
PLIST      strip_same_heavywalk_body_3_body_3[]
```

```
START
```

```

CnkB_CP( 0 ), /* body_3 */
CnkM_DA( FBS_SA|FBD_ISA ), 4,
MDiff( 255, 255, 255, 255 ),
MAmbi( 255, 255, 255, 255 ),
CnkT_TID( FCL_U|FCL_V|FDA_100 ), _TID( FFM_BF, 0 ),
CnkS_UVN( FST_IS|FST_UA ), 532, _NB( UFO_0, 21 ),
StripL(3),
2,      Uvn( 256, 167 ),
3,      Uvn( 252, 167 ),
4,      Uvn( 252, 164 ),
StripL(5),
4,      Uvn( 252, 164 ),
5,      Uvn( 252, 163 ),
2,      Uvn( 256, 167 ),
6,      Uvn( 252, 164 ),
7,      Uvn( 252, 167 ),
...
CnkT_TID( FCL_U|FCL_V|FDA_100 ), _TID( FFM_BF, 7 ),
CnkS_UVN( FST_IS ), 216, _NB( UFO_0, 10 ),
StripL(4),
29,     Uvn( 185, 31 ),
60,     Uvn( 209, 39 ),
28,     Uvn( 201, 47 ),
27,     Uvn( 204, 67 ),
...
CnkEnd()

```

登録 I d 0 番で未描画ポリゴンリストテーブルに設定。

END

< 未描画ポリゴンリストに登録されたポリゴンリストの描画例 >

```
PLIST      strip_same_heavywalk_body_3_kosi[]  
START  
          CnkB_DP( 0 ), /* body_3 */  
          CnkEnd()  
END
```

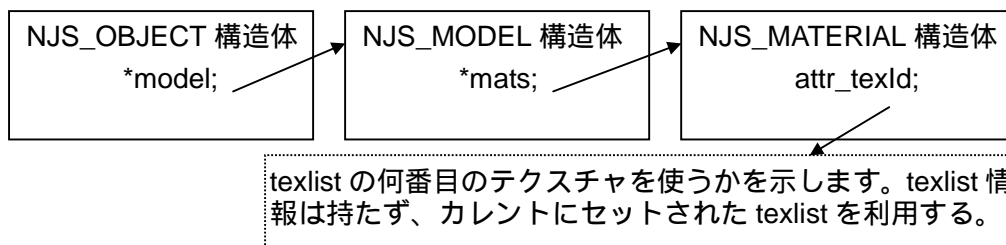
本来ポリゴンリストを持たないモデルで頂点計算が完了したことをコンバータレベルで検出し未描画ポリゴンリストの登録 I d 0 番のポリゴンリストの描画を実行する。

7. texlist 仕様

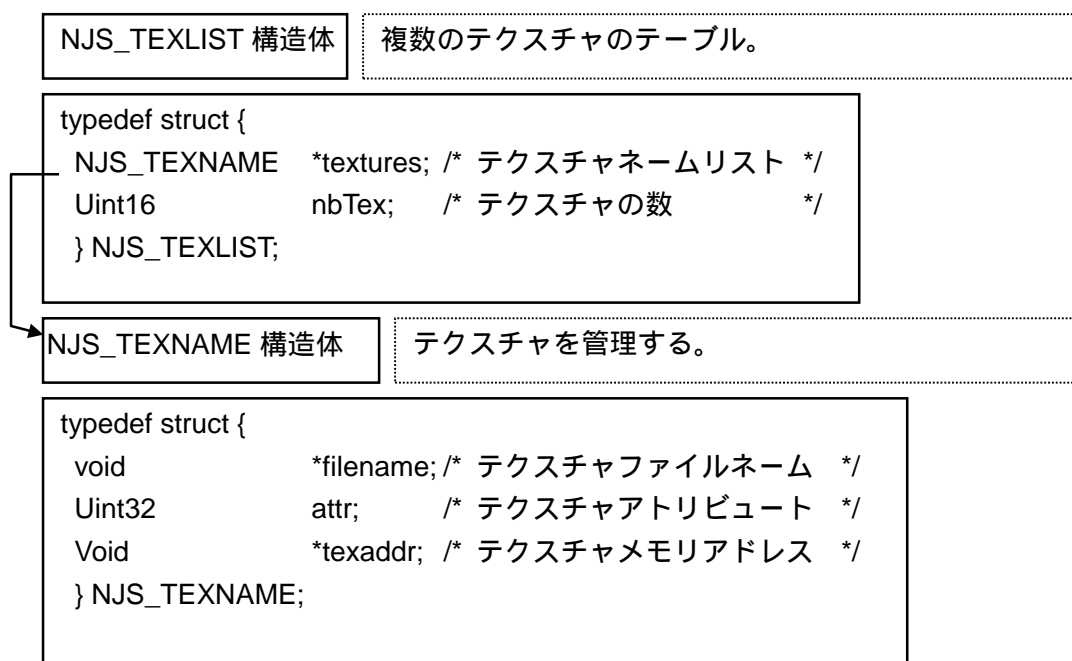
7.1 texlist 構造

7.2.1 構造体図

ObjectTree



Texlist



texlist

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

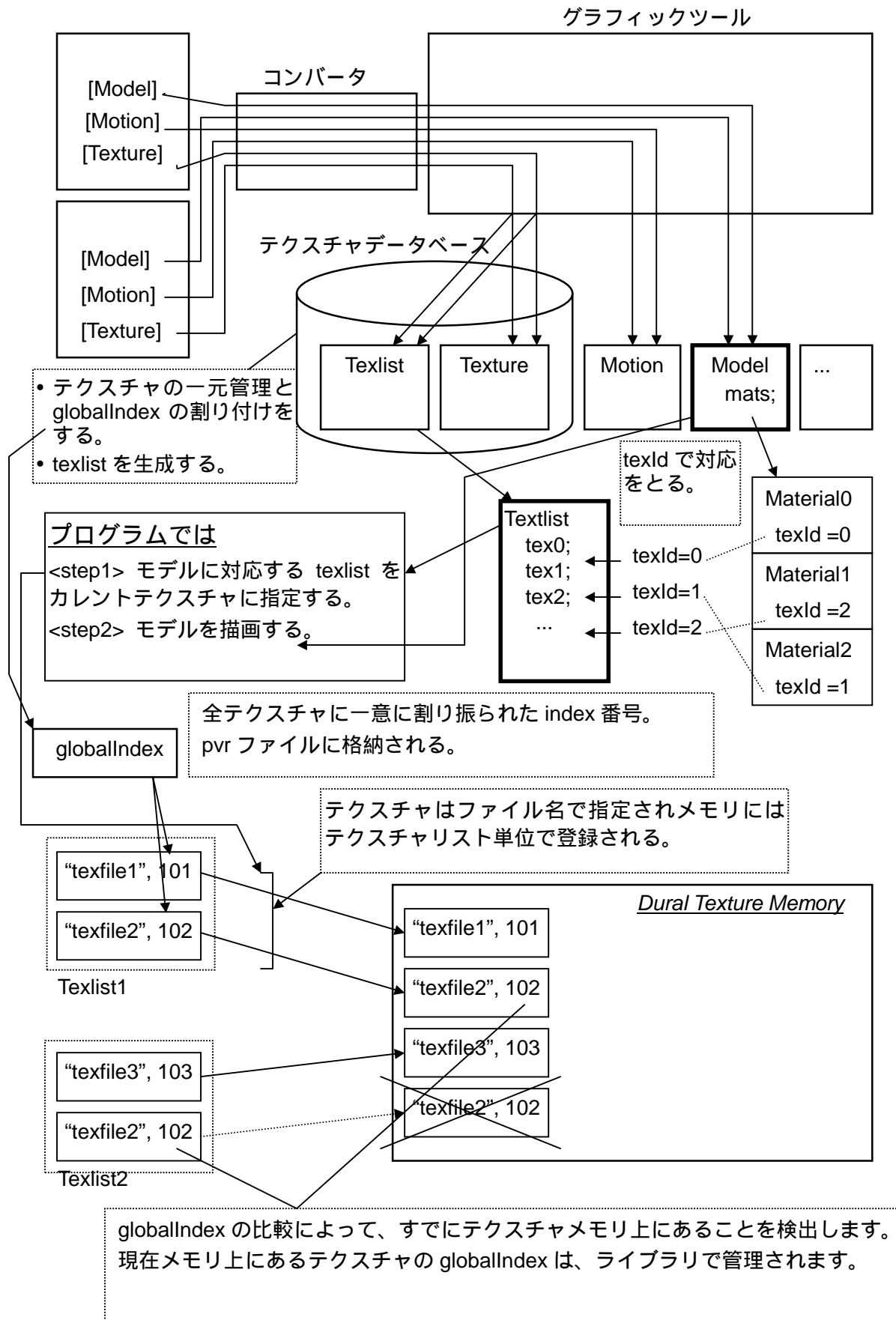
globalIndex

- ・ 全テクスチャに対し一意な番号を割り振る。
- ・ 番号の比較によりテクスチャ重複をチェックする。
- ・ globalIndex は pvr ファイルに格納される。

texaddr

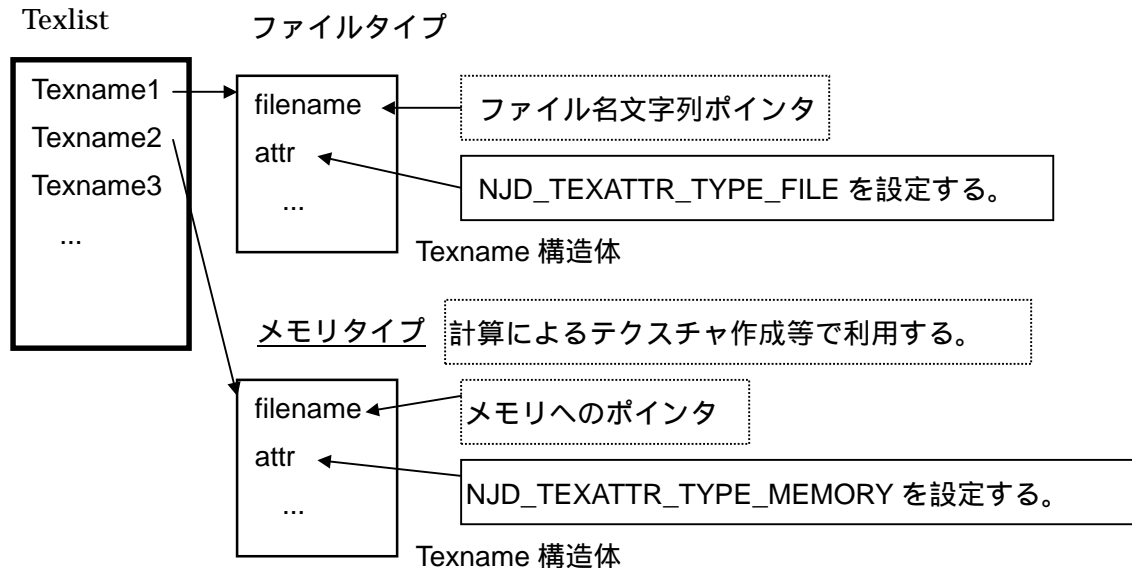
- ・ ライブラリ関数を使用して、カレントテクスチャを texlist 単位でセットする。この時テクスチャメモリに登録されたアドレスを格納する。

7.1.2 テクスチャ処理の概要



7.1.3 メモリ形式テクスチャ、テクスチャキャッシュの概要

ここでは概要のみを示す。詳細はライブラリ仕様書を参照のこと。



attr に **NJD_TEXATTR_CASHE** フラグをセットするとテクスチャキャッシュのみへのセットを指定する。**NJD_TEXATTR_BOTH** をセットすると **texlist** 単位でのテクスチャメモリを登録する場合にキャッシュにテクスチャがあれば自動的にキャッシュからテクスチャメモリへ登録する。

7.2 マクロ及び構造体定義

```
typedef struct {  
    void          *filename; /* texture filename strings */  
    Uint32        attr; /* texture attribute */  
    Uint32        texaddr; /* texture memory list address */  
} NJS_TEXNAME;
```

NJS_TEXNAME構造体は**texlist**上の各テクスチャのエントリを定義する。**filename**には拡張子が省略されたpvrファイル名を設定する。**attr**にはテクスチャに関するフラグ、**texaddr**はライブラリが使用するワークポインタに利用される。

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

texturesには**NJS_TEXNAME**構造体の配列の先頭ポインタ、**nbTexture**には配列の長さが格納される。

7.3 texlist のパレット対応について

モデラーで作成したモデルのテクスチャをパレットテクスチャに入れ替える手順について説明する。
説明に必要なアスキーマクロを示す。

```
#define TEXN(_texname)      {(_texname), 0x0, 0}  
#define TEXN3(_texname, _attr, _texaddr) {(_texname), _attr, _texaddr}
```

通常ファイル出力時は TEXN マクロが使われる。ユーザがプログラム用に第三引数をアスキーフォーマット上で使いたい場合オプションが指定することで場合に TEXN3 出力がされる。

```
#define PTEXN(_tname, _cnt, _bnk) {(_tname), _SET_CNT(_cnt), _SET_BNK(_bnk)}  
#define PTEXN5(_tname, _attr, _texaddr, _cnt, _bnk)          ¥  
    {(_tname), _SET_CNT(_cnt)|(_attr), _SET_BNK2(_bnk, _texaddr)}
```

PTEXN、PTEXN5 はパレット用マクロ。マテリアルネームからバンク番号が指定された場合そのエントリの出力は TEXN から PTEXN に自動で切り替わる。TEXN3 の場合は PTEXN5 になる。

NJS_TEXNAME構造体のattrにはライブラリが利用するフラグ群がセットされる。パレット用に次のフラグが定義される。

```
<NinjaDef.h>  
#define NJD_TEXATTR_TEXCONTINUE          BIT_16
```

このフラグはパレットテクスチャにおいて同一テクスチャでバンク番号のみが異なる連続するエントリで一つ前のテクスチャと同じであることを示しカレントのテクスチャファイルのオープンを省略することを可能とする。コンバータで自動設定される。

texaddrは通常ライブラリ用のワーク構造体ポインタとユーザ指定のglobalIndex指定に使われる。パレットバンク指定時にはこの上位6ビットにバンク番号0～63を設定する。globalIndex指定はメモリタイプのテクスチャの場合のみ使われますがメモリタイプでかつパレットタイプでは上位6ビットがバンク番号指定になるため26ビット精度の値までしか使えないので注意が必要。エントリのテクスチャがパレットかそうでないかはpvrファイルの中のカテゴリコードで検出される。

パレットは 1024 エントリから構成される。各エントリはバンク番号で分割して管理される。8bpp の場合は 256 色で 4 バンク、4bpp の場合は 16 色で 6 4 バンクになる。バンクにまたがる 16 色、256 色を利用したインデックス画像は生成できない。パレットのカラーは次の 4 種類がある。

RGB565
ARGB1555
ARGB4444
ARGB8888

ARGB8888 は 32 ビットカラーでありその他は 16 ビットカラー。32 ビットカラーは本ハードウェアではパレットでのみ使うことができる。四つのカラータイプは混在して同時に使えない。

Ninja ではパレットテクスチャを利用するために texlist にバンク番号 (texaddr の最上位 6 ビットにセット) とテクスチャ連続フラグ (新規フラグを定義したフラグを attr にセット) を格納する。

バンク番号の指定はマテリアルネームからする。文字列 B00 ~ B63 で指定を使用する。例えば B01 の代わりに B1 は使えない。必ず 0 をつけて B01 として設定する。この番号は 4bpp の場合のバンク番号だが 8bpp の場合はこの番号の下位 4 ビットを無視した値が有効となる。つまり 8bpp において B00、B16、B32、B48 だけが意味を持つ。

デザイナーはモデラー上で 16 ビットカラーでテクスチャを作成しモデル作成する。その時後でパレットに差し替える予定のテクスチャにバンク番号を指定しモデルをコンバートする。

texlist 出力では同じテクスチャは一つのエントリにまとめられるがバンク番号が指定された場合同じテクスチャでもバンク番号が異なれば別のエントリになる。コンバートにより 16 ビットテクスチャが貼られる。ビューアでそのままテクスチャ付きのモデルを見ることができる。その後 8 ビットペインタでテクスチャのパレット化をし pvr ファイルを差し替える。

パレットテクスチャ出力は二つのファイルから構成される。

pvr ファイル：インデックス画像

pvp ファイル：パレットデータ

パレット pvr は 16 ビットカラーテクスチャ同様にハードウェアが期待するバイトイメージ。インデックス画像は正方形であり twiddled。ミップマップが利用できるが 16 ビットカラーテクスチャでは一つ上のミップマップのピクセルを複数参照による解像度変換をしていたがパレットの場合もっとも上位のパレットテクスチャ画像に対するパレットの色をそのまま利用するため隣り合うピクセル間の色の変化がきつくなる可能性がある。

pvp ファイルはピクセルのタイプ、バンク番号、エントリオフセット、エントリカウントをヘッダ情報として持つ PVPL チャンクを格納する。ユーザカスタマイズにより複数の PVPL チャンクを格納することで一括したパレットデータの書き込みが可能。

<パレット出力の例>

TEXTURENAME textures_MODEL[]

START

```
PTEXN( "texturePAL1", 0, 20 ),   バンク番号が指定された。パレット用マクロを使用。
PTEXN( "texturePAL1", 1, 21 ),   バンク番号が指定された。パレット用マクロを使用。
TEXN( "texture1" ),
PTEXN( "texturePAL2", 0, 63 ),   バンク番号が指定された。パレット用マクロを使用。
TEXN( "texture2" ),
```

END

TEXTURELIST texlist_MODEL

START

```
TextureList textures_MODEL,
TextureNum 5,
```

END

PTEXN の第二引数の 1 は一つ前のテクスチャと同じテクスチャが指定されている場合に利用する。このフラグが 1 の場合は二つ目のテクスチャ読み込み(ここでは 2 ライン目の texturePAL1 のロード)を省略し一つ前のテクスチャの情報を利用する。パレットではテクスチャが同じで異なるパレットバンクを使う可能性がありこれに効率よく対応するためにこのフラグを活用する。同一テクスチャ名はコンバートにより連続するように出力される。またこのフラグがセットされていなくても global index のチェックで texlist 上の同じファイルが二重読み込みされることはない。

パレットテクスチャデータの生成の手順を示す。モデラーで直接パレットテクスチャを貼ることはしない。データ作成後に pvr ファイルをパレットデータに置き換えることによりパレットを利用する。

<step1>

通常の手順でモデルにテクスチャを貼る。この時貼り込むテクスチャはパレット化する元画像となる 16 ビットカラーのテクスチャ。

<step2>

マテリアルネームからバンク番号を設定する。

4bpp (16 色カラー) の場合

B00 ~ B63 をマテリアルネームに設定する。

8bpp (256 色カラー) の場合

B00、B16、B32、B48 の 4 つが有効。これをマテリアルネームに設定する。これ以外のバンク番号は丸められる。B01 ~ B15 は B0 に B17 ~ B31 は B16 に B33 ~ B47 は B32 に B49 ~ B63 は B48 として扱われる。

4bpp と 8bpp のどちらも同じバンク番号表記 0 ~ 63 を使うがカテゴリーコードに格納されるテクスチャのタイプにより意味が異なることに注意。

<step3>

コンバータで texlist を出力する。バンク番号が指定されたテクスチャに関してはこのバンク番号が texlist に格納される。同一テクスチャでもマテリアルに設定されたバンク番号が異なる場合別のエントリとして texlist に登録される。マテリアルネームからバンク番号を指定しなかったテクスチャに関しては従来の通りの動作をする。全テクスチャが 16 ビットカラーとしてコンバートされ pvr ファイルが出力される。

<step4>

モデルに貼った元画像テクスチャを 8 ビットペインタでロードしパレットタイプへ変換、修正する。そのデータを 8 ビットペインタから pvr ファイルと pvp ファイルで出力する。pvp ファイル出力時にバンク番号 0 ~ 63 を指定する。このバンク番号はモデル出力時に設定したバンク番号と一致する必要がある。

<step5>

モデルコンバート時に生成された pvr データ群のパレットにしたい部分のテクスチャファイルを上書きする。

pvp をパレットにロードし差し替えたテクスチャ及びモデルを描画することでパレットテクスチャが利用できる。

8. チャンクモデル (Chunk Model) 仕様

8.1 削除されるチャンク

Ninja2 では Dreamcast の性能、チャンクの利用実績を考慮しチャンクを整理する。NormalDraw 関数はサポートされない。これに伴う NormalDraw 関数で利用するチャンクが削除される。NormalDraw がなくなったことでスポットライトもなくなった。またスペキュラーの exponent は値はコンバータにより出力されるがライブラリでは利用されない。

Ninja2 で削除されるチャンクのリストは次の通り。

< 削除される Chunk Vertex >

```
#define NJD_CV_SH      (NJD_VERTOFF+0) /* x,y,z,1.0F, ... */
#define NJD_CV_VN_SH   (NJD_VERTOFF+1) /* x,y,z,1.0F,nx,ny,nz,0.0F, ... */
#define NJD_CV_S5      (NJD_VERTOFF+6) /* x,y,z,D565|S565, ... */
#define NJD_CV_S4      (NJD_VERTOFF+7) /* x,y,z,D4444|S565, ... */
#define NJD_CV_IN      (NJD_VERTOFF+8) /* x,y,z,D16|S16, ... */
#define NJD_CV_VN_S5   (NJD_VERTOFF+13) /* x,y,z,nx,ny,nz,D565|S565, ... */
#define NJD_CV_VN_S4   (NJD_VERTOFF+14) /* x,y,z,nx,ny,nz,D4444|S565, ... */
#define NJD_CV_VN_IN   (NJD_VERTOFF+15) /* x,y,z,nx,ny,nz,D16|S16, ... */
#define NJD_CV_VNX     (NJD_VERTOFF+16) /* x,y,z,nxyz32, ... */
#define NJD_CV_VNX_D8  (NJD_VERTOFF+17) /* x,y,z,nxyz32,D8888, ... */
#define NJD_CV_VNX_UF  (NJD_VERTOFF+18) /* x,y,z,nxyz32,UserFlags32, ... */
```

< 削除される Chunk Strip >

```
#define NJD_CS_VN      (NJD_STRIPPOFF+3) /* <Format3> */
#define NJD_CS_UVN_VN  (NJD_STRIPPOFF+4) /* <Format4> */
#define NJD_CS_UVH_VN  (NJD_STRIPPOFF+5) /* <Format4> */

#define NJD_CS_D8      (NJD_STRIPPOFF+6) /* <Format5> */
#define NJD_CS_UVN_D8  (NJD_STRIPPOFF+7) /* <Format6> */
#define NJD_CS_UVH_D8  (NJD_STRIPPOFF+8) /* <Format6> */
```

NormalDraw がなくなったためポリゴン側の頂点法線、頂点カラーはサポートされなくなった。ただし Ninja2 ライブラリインクルードファイル NjDef.h において削除されたチャンクの番号を欠番として保持する目的で上記の定義は残される。

8.2 利用可能なチャンク

< Chunk Vertex > (ただし NF のウェイト値は 16 ビット精度)

```
#define NJD_CV      (NJD_VERTOFF+2) /* x,y,z, ... */
#define NJD_CV_D8   (NJD_VERTOFF+3) /* x,y,z,D8888, ... */
#define NJD_CV_UF   (NJD_VERTOFF+4) /* x,y,z,UserFlags32, ... */
#define NJD_CV_NF   (NJD_VERTOFF+5) /* x,y,z,NinjaFlags32, ... */

#define NJD_CV_VN    (NJD_VERTOFF+9) /* x,y,z,nx,ny,nz, ... */
#define NJD_CV_VN_D8 (NJD_VERTOFF+10) /* x,y,z,nx,ny,nz,D8888, ... */
```

```
#define NJD_CV_VN_UF (NJD_VERTOFF+11) /* x,y,z,nx,ny,nz,UserFlags32,... */
#define NJD_CV_VN_NF (NJD_VERTOFF+12) /* x,y,z,nx,ny,nz,NinjaFlags32,... */
```

<新規追加される Chunk Vertex>

コンバータから直接出力は現在しない。Ninja2 ライブラリのダイレクトフォーマットで対応。

```
#define NJD_CV_D8_S8 (NJD_VERTOFF+19) /* x,y,z,D8888,S8888,... for Ninja2 */
```

頂点カラー付きエンベロープ。現在未対応。リザーブ。

```
#define NJD_CV_NF_D8 (NJD_VERTOFF+20) /* x,y,z,NinjaFlags32,D8888,... for Ninja2 */
```

<CompactShape で追加される Chunk Vertex>

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

<Chunk Null>

```
#define NJD_CN (NJD_NULLOFF+0) /* NULL chunk for boundary control */
```

<Chunk End>

```
#define NJD_CE (NJD_ENDOFF+0)
```

<Chunk Bits>

```
#define NJD_CB_BA (NJD_BITSOFF+0)
#define NJD_CB_DA (NJD_BITSOFF+1)
#define NJD_CB_EXP (NJD_BITSOFF+2)
#define NJD_CB_CP (NJD_BITSOFF+3)
#define NJD_CB_DP (NJD_BITSOFF+4)
```

<Chunk Tiny>

```
#define NJD_CT_TID (NJD_TINYOFF+0)
#define NJD_CT_TID2 (NJD_TINYOFF+1)
```

<Chunk Material>

```
#define NJD_CM_D (NJD_MATOFF+1) /* [CHead][4(Size)][ARGB] */
#define NJD_CM_A (NJD_MATOFF+2) /* [CHead][4(Size)][NRGB] N: NOOP(255) */
#define NJD_CM_DA (NJD_MATOFF+3) /* [CHead][8(Size)][ARGB][NRGB] */
#define NJD_CM_S (NJD_MATOFF+4) /* [CHead][4(Size)][ERGB] E: Exponent */
#define NJD_CM_DS (NJD_MATOFF+5) /* [CHead][8(Size)][ARGB][ERGB] */
#define NJD_CM_AS (NJD_MATOFF+6) /* [CHead][8(Size)][NRGB][ERGB] */
#define NJD_CM_DAS (NJD_MATOFF+7) /* [CHead][12(Size)][ARGB][NRGB][ERGB] */
```

```
/* Bump */
```

```
#define NJD_CM_BU (NJD_MATOFF+8) /* [CHead][12(Size)][dx(16)][dy(16)][dz(16)]
                                   [ux(16)][uy(16)][uz(16)] */
```

```
/* 2para modifier */
```

```
#define NJD_CM_D2 (NJD_MATOFF+9) /* [CHead][4(Size)][ARGB] */
#define NJD_CM_A2 (NJD_MATOFF+10) /* [CHead][4(Size)][NRGB] N: NOOP(255) */
#define NJD_CM_DA2 (NJD_MATOFF+11) /* [CHead][8(Size)][ARGB][NRGB] */
#define NJD_CM_S2 (NJD_MATOFF+12) /* [CHead][4(Size)][ERGB] E: Exponent */
```

```

#define NJD_CM_DS2 (NJD_MATOFF+13) /* [CHead][8(Size)][ARGB][ERGB] */
#define NJD_CM_AS2 (NJD_MATOFF+14) /* [CHead][8(Size)][NRGB][ERGB] */
#define NJD_CM_DAS2 (NJD_MATOFF+15) /* [CHead][12(Size)][ARGB][NRGB][ERGB] */

```

< Chunk Strip >

```

#define NJD_CS (NJD_STRIP0FF+0) /* <Format1> */
#define NJD_CS_UVN (NJD_STRIP0FF+1) /* <Format2> */
#define NJD_CS_UVH (NJD_STRIP0FF+2) /* <Format2> */

```

/* 2para modifier */

```

#define NJD_CS_2 (NJD_STRIP0FF+9) /* <Format1> */
#define NJD_CS_UVN2 (NJD_STRIP0FF+10) /* <Format7> */
#define NJD_CS_UVH2 (NJD_STRIP0FF+11) /* <Format7> */

```

8.3 チャンクモデルの特徴

Chunk Model の特徴は次の通り。

性能優先のためすべての描画はトライアングルストリップを基本とする。三角形、四角形、N角形を直接描画できない。

データは、頂点リスト vlist とポリゴンリスト plist からなる。vlist、plist 上に IFF チャンク形式でデータを配置してメモリ領域を一本化し、描画実行中にキャッシュを壊さないようにしている。

ただし vlist のチャンクサイズはロングワード (Sint32) 数、plist のチャンクサイズはワード (Sint16) 数で表現される。

頂点側に頂点カラーを持つ。単一頂点が複数の頂点カラーを持つ場合コンバータがこれを複製することでデータを生成する。複数の法線を持つ場合も同様。

ポリゴン側 (最大 16bit × 3) と頂点側 (32bit × 1) にユーザフラグ領域を持つことができる。

マテリアルは plist に格納され、以前に設定したマテリアルの変更部分 (差分) のみの設定だけを更新する。

コンバータ出力時にマテリアルを削除できる。同じモデルの描画 (例えば木) などすべてのマテリアルをなくし、ユーザが外で設定することでデータを最適化することができる。

コリジョン用データ Chunk Volume の出力が用意される。三角形、四角形、トライアングルストリップで形状を出力できる。マテリアル情報は持たない。ユーザフラグ領域を持つことができる。現在この領域にマテリアルカラーを出力できる。ライブラリは Chunk Volume をポリゴンとして描画しない。

独立三角形の Chunk Volume (volume3) は、モディファイアボリュームにも利用する。

コンバータで volume34 を指定すると、面間角度が 0.1 度の三角形同士を接続し四角形を作る。3D Studio MAX は三角形データしか出力できないがこの場合でも四角形コリジョンが生成できる。

二種類の UV 値表現を持つ。0-255 による UVN、分解能を高めた 0-1023 による UVH。UVN であると 256 を超えるテクスチャでは、1 ピクセル単位で指定できない。UVH はハイレゾモード、1024x1024 のテクスチャで、1 ピクセル単位での指定が可能。ただし UVH は UVN よりも分解能を上げた分だけ、テクスチャのリピートの表現回数が減少する (UVN が 128 回に対し UVH では 32 回)。コンバートオプションによりモデルツリー全体に対しまたマテリアルネームにより各モデル単位で UVN、UVH を切り替えることができる。マテリアルネームからの指定では単一のモデルに使われる複数のマテリアルのどれか一つに設定することでモデル全体に対する UV 表現が変更される。デフォルトは UVN。

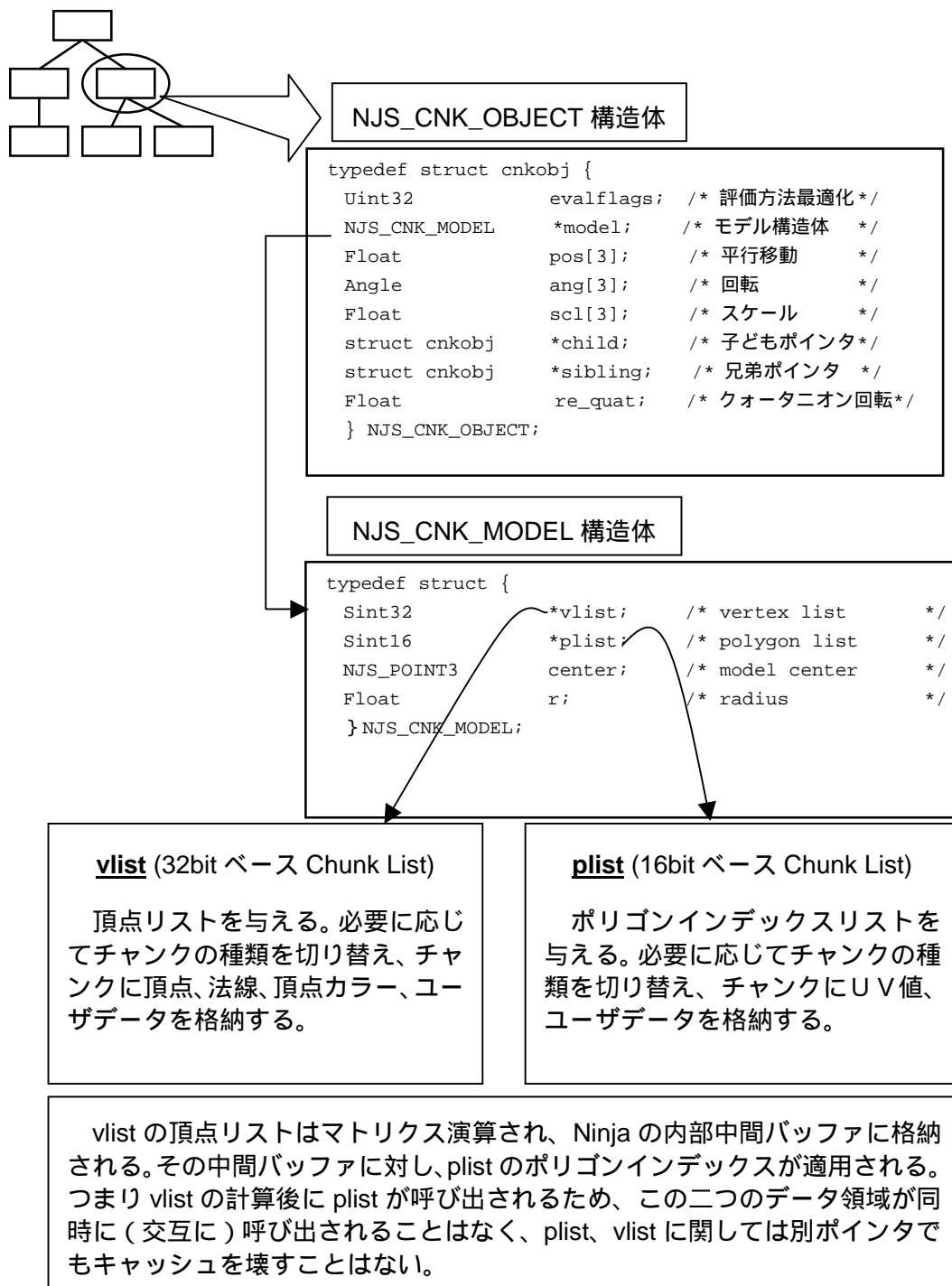
モデラーで作成された一体成形モデルを再現するエンベロープをサポートする。Ninja のエンベロープはウェイトの数だけ頂点を複製しそれぞれバラバラに計算した結果を中間バッファ上でウェイト値を乗算して合成することを特徴とする。

CompactShape 実行に必要なエンベロープとの共存をデータ表現できる。頂点グループに指定された頂点のマルチウェイト値を無視し部分的にエンベロープ 1 (= 100%ウェイト) を適用しこれを表現できる。頂点は最適化のために任意の順番に組み替えられる。

8.4 チャンクモデル構造体

8.4.1 NJS_CNK_OBJECT 構造体図

Chunk Object Tree



8.4.2 NJD_CNK_OBJECT 構造体

```
typedef float Float          /* 浮動小数点演算型          */
typedef Sint32 Angle        /* 回転角度                */
```

アングルは 0x0000 ~ 0xFFFF が 0 ~ 360 度。

```
typedef struct {
    Float      x;          /* X 値                    */
    Float      y;          /* Y 値                    */
    Float      z;          /* Z 値                    */
} NJS_POINT3, NJS_VECTOR;
```

```
typedef struct {
    Sint32      *vlist;    /* 頂点チャンクリスト      */
    Sint16      *plist;    /* ポリゴンチャンクリスト  */
    NJS_POINT3  center;    /* モデルの中心            */
    Float        r;        /* モデルの半径            */
} NJS_CNK_MODEL;
```

vlist には頂点リストのデータを Sint32 配列上に IFF チャンク形式で並べる。plist にはポリゴンインデックスリストのデータを、Sint16 配列上に IFF チャンク形式で並べる。center はモデルの外接球の中心位置であり、r はその半径。vlist のチャンクサイズはロングワード (Sint32) 数、plist のチャンクサイズはワード (Sint16) 数で表現される。

```
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。

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。

8.4.3 evalflags の説明

```
#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。移動成分のマトリッ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。回転成分のマトリッ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。スケールのマトリッ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ラー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	カレントのマトリックスを保存する。
NJD_EVAL_ROTATE_SET	保存されたマトリッ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>

8.4.4 evalflags のユーザフラグ領域 (上位 8 ビット)

evalflags は Sint32 のフィールドを持つ。現在 BIT_0 から BIT_13 まで現在利用しているが必要であれば BIT_14 以降フラグを追加していく。ここで上位 8 ビットをユーザエリアとして開放する。BIT_31 から BIT_24 までは今後の拡張でも利用しないものとする。ユーザはこの 8 ビットに対しユーザ目的のためのフラグを割り当ててもよい。

8.5 チャンクの種類

チャンクの名前	略号	サイズ	説明
<u>C</u> hunk <u>N</u> ULL	CN	16bit	ロングワードアライメントあわせ。Ninja2 用のコンバータからは出力されない。
<u>C</u> hunk <u>E</u> nd	CE	16bit	チャンクデータリストの終わりを示す。
<u>C</u> hunk <u>B</u> its	CB	16bit	Blending Alpha などのフラグを設定する。
<u>C</u> hunk <u>T</u> iny	CT	32bit	TexId などのフラグと単一の値を設定する。
<u>C</u> hunk <u>M</u> aterial	CM	可変	Diffuse, Specular, Exponent, Ambient を設定する。
<u>C</u> hunk <u>V</u> ertex	CV	可変	頂点リストを与える。
<u>C</u> hunk <u>V</u> olume	CO	可変	コリジョン、モディファイアボリューム用データを与える。
<u>C</u> hunk <u>S</u> trip	CS	可変	ストリップデータを与える。

基本構造をベースに、目的に合わせて構造を簡略化した Chunk(Bits, Tiny 等)を定義している。頂点用の Chunk Vertex は Chunk Model 構造体の vlist に格納される。それ以外のチャンクは plist に格納される。これらは NjChunk.h に定義される。

8.6 Chunk の構造

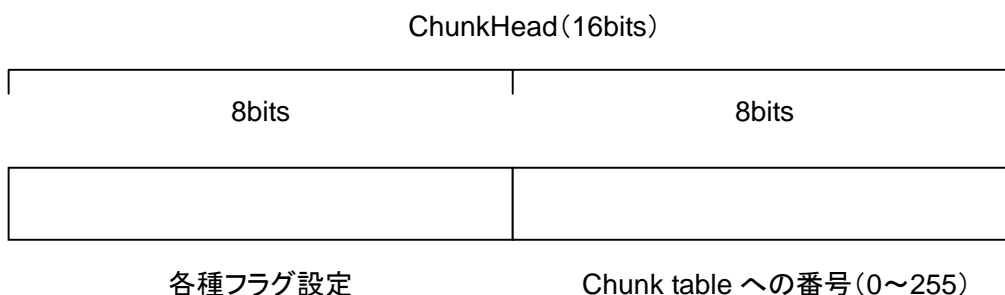
チャンクの基本構造は次の通り。

Chunk Vertex の場合

[headbits(15-8) | ChunkHead(7-0)][longsize(15-0)][data]

Chunk Vertex 以外の場合

[headbits(15-8) | ChunkHead(7-0)][shortsize(15-0)][data]



ChunkHead は、そのチャンクを処理する関数 table のエントリ番号を与える。ライブラリはこの番号から関数を選択し実行する。チャンクごとに処理関数を分けることにより、描画ルーチンを単純化し、高速化ルーチンを実現する。テーブルサイズは最大 256 個とするため上位 8bit が余る。この部分 (headbits と呼ぶ) 8bit へチャンクの種類により目的に合わせてアトリビュートフラグの一部を格納しデータサイズの効率を高める。そのためチャンクテーブルの関数エントリ番号を得るには上位 8bit をマスクする必要がある。

shortsize、longsize は、次のチャンクの先頭までのオフセットを与える。通常 IFF 形式では次のチャンクまでのデータオフセットをバイトで与えるが対象とする plist が Sint16 の配列、vlist が Sint32 の配列であることからそれぞれ shortsize (2 バイト) 単位、longsize (4 バイト) 単位でオフセットを表現する。

チャンク形式により例えばユーザがマテリアルだけを書き換えたい場合マテリアルのチャンクが見つかるまで shortsize を利用してデータをスキップしマテリアルを発見したところで値を書き換えるようなことが可能となる。

8.7 Chunk NULL

ChunkName : 'NJD_CN'

(Chunk NULL)

概要 :

plist においてロングワードアライメント調整のためにチャンクとチャンクの間に挿入される。
Ninja2 では無くても良い。リザーブ。ChunkHead (16bit) のみのチャンク。

形式 :

[ChunkHead(15-0)]

ChunkHead:

NJD_CN

説明 :

#define NJD_CN (NJD_NULLOFF+0)

8.8 Chunk End

ChunkName : 'NJD_CE'

(Chunk End)

概要 :

plist、vlist のチャンクリストの最後を与える。ChunkHead (16bit) のみのチャンク。vlist の場合は、ChunkHead (32bit) として扱われる。実際の値は最上位ビットに 1 が立つことにより終了が検出される。

形式 :

plist の場合 [ChunkHead(15-0)] (16 bits chunk)

vlist の場合 [ChunkHead(31-0)] (32 bits chunk)

ChunkHead:

NJD_CE

説明 :

```
#define NJD_CE (NJD_ENDOFF+0)
```

リストの完了を検出する。

8.9 Chunk Bits

Chunk Bits はデータのフラグを書き換えるために利用する。

[headbits(15-8)|ChunkHead(7-0)](16 bits chunk)

上位 8bit にフラグを格納し下位 8bit でチャンクの番号を与える。ChunkHead (16bit) のみのチャンク。次に実際の Chunk Bits について説明する。

ChunkName : 'NJD_CB_BA'
(Chunk Bits Blending Alpha)

概要：
plist において Blending Alpha 及びトライリニアフィルタ制御に必要な SRC select , DST select を設定する。

形式：
[headbits(15-8)|ChunkHead(7-0)]
headbits:
15 = SRC Select(1bit)
14 = DST Select(1bit)
13-11 = SRC Alpha Instruction(3bit)
10- 8 = DST Alpha Instruction(3bit)
ChunkHead:
NJD_CB_BA

説明：
#define NJD_CB_BA (NJD_BITS0FF+0)

Blending Alpha は 2 種類の方法で設定される。Chunk Material (後述) の headbits として、diffuse、specular、ambient の設定とともに Blending Alpha を設定できる。マテリアルを変更せず Blending Alpha のみ設定する場合に NJD_CB_BA を利用する。またトライリニアフィルタを利用したポリゴン描画において描画にアキュムレータバッファを利用するため SRC Select、DST Select の制御、Blending Alpha の制御が必要となる。
SRC Select、DST Select はアキュムレーションバッファを利用するトライリニアフィルタ描画に利用される。ハードウェアの制御手順に合わせて SRC Select、DST Select が設定される。ユーザが直接設定する必要はなくまたした場合のハードウェアの誤動作はユーザ責任となる。

```
/* Src Select */
#define NJD_FBS_SEL          (BIT_15)

/* Dst Select */
#define NJD_FBD_SEL          (BIT_14)
```

ブレンディング機能では 2 つの ARGB 値、SRC と DST が下記のように合成されその結果は DST に書き戻される。

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

ここで、BlendFunction(Instruction)には SRC/DST カラーとともに 3 ビットの Instruction が入力される。そしてそれぞれの ARGB に対して 4 つの 値によって重み付けされた係数値を戻す。

Instruction	Field Value	Values Returned
Zero	0	(0, 0, 0, 0)
One	1	(1, 1, 1, 1)
'Other' Colour	2	(O _R , O _G , O _B , O _A)

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

‘Other Colour’と‘Inverse Other Colour’は、SRC インストラクションに指定された時には DST のカラーが使用され、DST インストラクションに指定された時には SRC のカラーが使用されることを意味する。

略号の意味は次の通り。

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)
```

ChunkName : 'NJD_CB_DA'

(Chunk Bits 'D' Adjust)

概要 :

plist において Mipmap 'D' adjust 値を設定する。

形式 :

```
[headbits(11-8)|ChunkHead(7-0)]
headbits:
    11- 8 = Mipmap 'D' adjust(4)
ChunkHead:
    NJD_CB_DA
```

説明 :

```
#define NJD_CB_DA                (NJD_BITSOFF+1)
```

Mipmap の切り替えの深さを調整する。デフォルトは 1.00。これは頻繁に切り替えるものではなく、mipmap の切り替えが目立つ場合にこれを抑制する目的で利用する。

Flag 'D' Adjust :

```
#define NJD_FDA_SHIFT            8
#define NJD_FDA_025              (1<<NJD_FDA_SHIFT)      /* 0.25 */
#define NJD_FDA_050              (2<<NJD_FDA_SHIFT)      /* 0.50 */
#define NJD_FDA_075              (3<<NJD_FDA_SHIFT)      /* 0.75 */
#define NJD_FDA_100              (4<<NJD_FDA_SHIFT)      /* 1.00 */
#define NJD_FDA_125              (5<<NJD_FDA_SHIFT)      /* 1.25 */
#define NJD_FDA_150              (6<<NJD_FDA_SHIFT)      /* 1.50 */
#define NJD_FDA_175              (7<<NJD_FDA_SHIFT)      /* 1.75 */
#define NJD_FDA_200              (8<<NJD_FDA_SHIFT)      /* 2.00 */
#define NJD_FDA_225              (9<<NJD_FDA_SHIFT)      /* 2.25 */
#define NJD_FDA_250              (10<<NJD_FDA_SHIFT)     /* 2.25 */
#define NJD_FDA_275              (11<<NJD_FDA_SHIFT)     /* 2.25 */
#define NJD_FDA_300              (12<<NJD_FDA_SHIFT)     /* 3.00 */
#define NJD_FDA_325              (13<<NJD_FDA_SHIFT)     /* 3.25 */
#define NJD_FDA_350              (14<<NJD_FDA_SHIFT)     /* 3.50 */
#define NJD_FDA_375              (15<<NJD_FDA_SHIFT)     /* 3.75 */

#define NJD_FDA_MASK              (0xf<<NJD_FDA_SHIFT)
```

ChunkName : 'NJD_CB_EXP'

(Chunk Bits Exponent)

概要 :

plist においてスペキュラの Exponent を設定する。0 ~ 16 の間が有効。Ninja1 の NormalDraw で利用されていたが Ninja2 では NormalDraw がないため Exponent は無効。リザーブ。

形式 :

```
[headbits(12-8)|ChunkHead(7-0)]
  headbits:
    12- 8 = Exponent(5) range:0-16
  ChunkHead:
    NJD_CB_EXP
```

説明 :

```
#define NJD_CB_EXP (NJD_BITS0FF+2)
```

exponent は 2 種類の方法で設定される。Chunk Material (後述) の中で specular を設定する場合は、Chunk Material の specular 成分の上位 8bit で exponent を設定する(ERGB8888 となる)。その前に設定した specular 値が変化せず exponent のみが増加した場合に限り、NJD_CB_EXP が使用される。

Flag EXPonent(range 0-16) :

```
#define NJD_FEXP_SHIFT      8
#define NJD_FEXP_00         (0<<NJD_FEXP_SHIFT)      /* 0.0 */
#define NJD_FEXP_01         (1<<NJD_FEXP_SHIFT)      /* 1.0 */
#define NJD_FEXP_02         (2<<NJD_FEXP_SHIFT)      /* 2.0 */
#define NJD_FEXP_03         (3<<NJD_FEXP_SHIFT)      /* 3.0 */
#define NJD_FEXP_04         (4<<NJD_FEXP_SHIFT)      /* 4.0 */
#define NJD_FEXP_05         (5<<NJD_FEXP_SHIFT)      /* 5.0 */
#define NJD_FEXP_06         (6<<NJD_FEXP_SHIFT)      /* 6.0 */
#define NJD_FEXP_07         (7<<NJD_FEXP_SHIFT)      /* 7.0 */
#define NJD_FEXP_08         (8<<NJD_FEXP_SHIFT)      /* 8.0 */
#define NJD_FEXP_09         (9<<NJD_FEXP_SHIFT)      /* 9.0 */
#define NJD_FEXP_10         (10<<NJD_FEXP_SHIFT)     /* 10.0 */
#define NJD_FEXP_11         (11<<NJD_FEXP_SHIFT)     /* 11.0 */
#define NJD_FEXP_12         (12<<NJD_FEXP_SHIFT)     /* 12.0 */
#define NJD_FEXP_13         (13<<NJD_FEXP_SHIFT)     /* 13.0 */
#define NJD_FEXP_14         (14<<NJD_FEXP_SHIFT)     /* 14.0 */
#define NJD_FEXP_15         (15<<NJD_FEXP_SHIFT)     /* 15.0 */
#define NJD_FEXP_16         (16<<NJD_FEXP_SHIFT)     /* 16.0 */

#define NJD_FEXP_MASK        (0x1f<<NJD_FEXP_SHIFT)
```


ChunkName : 'NJD_CB_CP'

(Chunk Bits Cash Polygon)

概要：

このチャンクの次から最後までポリゴンリストを未描画ポリゴンテーブルに登録 I d 番号とともに登録する。

形式：

[headbits(12-8)|ChunkHead(7-0)]

headbits:

12- 8 = Id

ChunkHead:

NJD_CB_CP

説明：

#define NJD_CB_CP (NJD_BITSOFF+3)

エンベロープモデル描画においてオリジナルモデルのポリゴンリストのための頂点計算が完了していない場合後のモデルによる頂点計算完了までの間ポリゴン描画を待機させる。他のポリゴンリスト上で NJD_CB_DP コマンドで登録 I d を指定することで待機していたポリゴンリスト描画を実行できる。

ChunkName : 'NJD_CB_DP'

(Chunk Bits Draw Polygon)

概要：

このチャンクの次から最後までポリゴンリストを未描画ポリゴンテーブルに登録 I d 番号とともに登録する。

形式：

[headbits(12-8)|ChunkHead(7-0)]

headbits:

12- 8 = Id

ChunkHead:

NJD_CB_DP

説明：

#define NJD_CB_DP (NJD_BITSOFF+4)

エンベロープモデル描画において未描画ポリゴンリストテーブルに登録されている未描画ポリゴンリストの描画実行を発行する。未描画ポリゴンリストのための頂点計算が完了したタイミングはコンバータレベルで検出しデータに埋め込まれる。未描画ポリゴンリストテーブルへの登録は NJD_CB_CP コマンドに登録 I d とともに発行することでされる。

8.10 Chunk Tiny

Chunk Tiny は、フラグと一つの値を設定するために利用する。これに対応するものとして TexId がある。形式は ChunkHead が 16bit と 16bit の定数の合計 32bit サイズ固定となる。

$[\text{headbits}(15-8)|\text{ChunkHead}(7-0)][\text{value}(15-0)]$ (32 bits chunk)

現在定義されている Chunk Tiny には TexId 及びテクスチャ関係のアトリビュートフラグを設定する NJD_CT_TID、2 パラメータモディファイアに利用される二つ目のテクスチャを指定する NJD_CT_TID2 が定義されている。

ChunkName : 'NJD_CT_TID', 'NJD_CT_TID2'

(Chunk Tiny TexId and Chunk Tiny TexId2)

概要 :

plist において TexList のエントリ番号である TexId を設定する。NJD_CT_TID2 は 2 パラモディファイアデータにおいて第 2 のテクスチャを指定するために利用する。NJD_CT_TID2 はテクスチャ指定に利用されるため headbits は無視される。同時に使われる NJD_CT_TID の headbits が利用される。

形式 :

[headbits(15-8)|ChunkHead(7-0)][texbits(15-13)|TexId(12-0)]

headbits:

15-14 = FlipUV(2)

13-12 = ClampUV(2)

11- 8 = Mipmap 'D' adjust(4)

ChunkHead:

NJD_CT_TID, NJD_CT_TID2

texbits:

15-14 = Filter Mode(2)

13 = Super Sample(1)

TexId:

0 ~ 8191 (TexId Max = 8191)

説明 :

```
#define NJD_CT_TID (NJD_TINYOFF+0)
```

```
#define NJD_CT_TID2 (NJD_TINYOFF+1)
```

テクスチャ関連のアトリビュートフラグと TexId を設定する。テクスチャの切り替えタイミングでそれに関係するフラグを設定することにより効率よくテクスチャの切り替えを行える。TexId に割り振られたビット数は 13bit のため TexId の最大値は 8191 となる。NJD_CT_TID2 の headbits のフラグは無効。同時に使われる NJD_CT_TID のフラグが利用される。

Flag FLip (headbits) :

```
#define NJD_FFL_U (BIT_15)
```

```
#define NJD_FFL_V (BIT_14)
```

UV 値のフリップを制御する。

Flag CLamp (headbits) :

```
#define NJD_FCL_U (BIT_13)
```

```
#define NJD_FCL_V (BIT_12)
```

UV 値のクランプを制御する。

Flag Filter Mode (texbits) :

PS : Point Sampling

BF : Bilinear Filter (default)

TF : Trilinear Filter

```
#define NJD_FFM_SHIFT      14
#define NJD_FFM_PS        (0<<NJD_FFM_SHIFT)
#define NJD_FFM_BF        (1<<NJD_FFM_SHIFT)
#define NJD_FFM_TF        (2<<NJD_FFM_SHIFT)

#define NJD_FFM_MASK      (0x3<<NJD_FFM_SHIFT)
```

フィルタリングモードを制御する。

Flag Super Sample (texbits) :

```
#define NJD_FSS            (BIT_13)
```

スーパーサンプリングを制御する。

```
#define NJD_TID_MASK      (~(NJD_FSS|NJD_FFM_MASK))
```

8.11 Chunk Material

Chunk Material は、diffuse、specular、ambient を設定する。その前に設定されている値から変化があった差分だけ設定される。Chunk Strip(後述)の headbits で specular 無視のフラグ (NJD_FST_IL) が設定されている場合、specular の設定が省略される。ambient 無視のフラグ (NJD_FST_IA) が設定されていると ambient の設定が省略される。headbits (上位 8bit) に Blending Alpha を設定する。この部分は NJD_CB_BA と等価である。Blending Alpha の詳細については NJD_CB_BA を参照のこと。specular の最上位 8bit は exponent 設定に使用される。この部分は NJD_CB_EXP と機能的には等価である。NJD_CB_EXP では、0~16 までのフラグビット (NJD_FEXP_*) を用意しこれに設定していたが、Chunk Material の specular では 0~16 の値を直接設定する。2 パラメータモディファイアボリュームデータの二つ目のマテリアル設定用に NJS_CM_*2 が定義される。2 パラメータモディファイアデータでは 2 組のマテリアルとテクスチャを持つ必要があり NJS_CM_*2 を利用して二つ目のマテリアルの値をライブラリに設定する。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。またバンプ用に CnKM_BU が定義される。

```
[headbits(15-8)|ChunkHead(7-0)][shortsize(15-0)][Data]
headbits:
    15    = SRC Select(1bit)
    14    = DST Select(1bit)
    13-11 = SRC Alpha Instruction(3)
    10- 8 = DST Alpha Instruction(3)
ChunkHead:
    NJD_CM_D、NJD_CM_A、NJD_CM_DA、NJD_CM_S、
    NJD_CM_DS、NJD_CM_AS、NJD_CM_DAS
    NJD_CM_D2、NJD_CM_A2、NJD_CM_DA2、NJD_CM_S2、
    NJD_CM_DS2、NJD_CM_AS2、NJD_CM_DAS2
Data:
    Diffulse = ARGB8888
    Specular = ERGB8888 ( E : exponent 0 ~ 16 )
    Ambient = NRGB8888 ( N : NOOP )
```

ChunkName : 'NJD_CM_D', 'NJD_CM_D2'

(Chunk Material Diffuse and Chunk Material Diffuse 2)

概要 :

plist において diffuse の値を設定する。diffuse の上位 8bit には、値が格納される。
NJD_CM_D2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits には
NJD_CM_D, NJD_CM_D2 とともに同じ値が設定される。これはライブラリの動作を単純化するための
処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][2(shortsize)][ARGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_D, NJD_CM_D2

ARGB:

ARGB8888

説明 :

#define NJD_CM_D (NJD_MATOFF+1)

#define NJD_CM_D2 (NJD_MATOFF+9)

Blending Alpha、diffuse のみを設定する。specular、ambient については現在の設定値のまま
とする。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現
されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_A', 'NJD_CM_A2'

(Chunk Material Ambient and Chunk Material Ambient 2)

概要 :

plist において ambient の値を設定する。ambient の上位 8bit はダミー (NOOP) として 255 が格納される。NJD_CM_A2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits は NJD_CM_A, NJD_CM_A2 とともに同じ値が設定される。これはライブラリの動作を単純化するための処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][2(shortsize)][NRGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_A, NJD_CM_A2

NRGB:

NRGB8888 (N : NOOP 255)

説明 :

#define NJD_CM_A (NJD_MATOFF+2)

#define NJD_CM_A2 (NJD_MATOFF+10)

Blending Alpha、ambient のみを設定する。diffuse、specular については現在の設定値のままとする。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_DA', 'NJD_CM_DA2'

(Chunk Material Diffuse and Ambient and Chunk Material Diffuse and Ambient 2)

概要 :

plist において diffuse、ambient の値を設定する。diffuse の上位 8bit には 値が設定される。ambient の上位 8bit はダミー(NOOP)として 255 が格納される。NJD_CM_DA2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits には NJD_CM_DA, NJD_CM_DA2 とともに同じ値が設定される。これはライブラリの動作を単純化するための処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][4(shortsize)][ARGB][NRGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_DA, NJD_CM_DA2

ARGB:

ARGB8888

NRGB:

NRGB8888 (N : NOOP 255)

説明 :

#define NJD_CM_DA (NJD_MATOFF+3)

#define NJD_CM_DA2 (NJD_MATOFF+11)

Blending Alpha、diffuse、ambient を設定する。specular については現在の設定値のままとする。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_S', 'NJD_CM_S2'

(Chunk Material Specular and Chunk Material Specular 2)

概要 :

plist において specular の値を設定する。specular の上位 8bit には exponent (0 ~ 16) が設定される。NJD_CM_S2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits には NJD_CM_S, NJD_CM_S2 とともに同じ値が設定される。これはライブラリの動作を単純化するための処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][2(shortsize)][ERGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_S, NJD_CM_S2

ERGB:

ERGB8888(E : Exponent 0 ~ 16)

説明 :

#define NJD_CM_S (NJD_MATOFF+4)

#define NJD_CM_S2 (NJD_MATOFF+12)

Blending Alpha、specular のみを設定する。diffuse、ambient については現在の設定値のままとする。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_DS', 'NJD_CM_DS2'

(Chunk Material Diffuse and Specular and Chunk Material Diffuse and Specular 2)

概要 :

plist において diffuse、specular の値を設定する。diffuse の上位 8bit には 値が設定される。specular の上位 8bit には exponent (0 ~ 16) が設定される。NJD_CM_DS2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits には NJD_CM_DS, NJD_CM_DS2 とともに同じ値が設定される。これはライブラリの動作を単純化するための処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][4(shortsize)][ARGB][ERGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_DS, NJD_CM_DS2

ARGB:

ARGB8888

ERGB:

ERGB8888(E : Exponent 0 ~ 16)

説明 :

#define NJD_CM_DS (NJD_MATOFF+5)

#define NJD_CM_DS2 (NJD_MATOFF+13)

Blending Alpha、diffuse、specular を設定する。ambient については現在の設定値のままとする。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_AS', 'NJD_CM_AS2'

(Chunk Material Ambient and Specular and Chunk Material Ambient and Specular 2)

概要 :

plist において diffuse、specular の値を設定する。diffuse の上位 8bit には 値が設定される。specular の上位 8bit には exponent (0 ~ 16) が設定される。NJD_CM_AS2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits には NJD_CM_AS, NJD_CM_AS2 とともに同じ値が設定される。これはライブラリの動作を単純化するための処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][4(shortsize)][NRGB][ERGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_AS, NJD_CM_AS2

NRGB:

NRGB8888(N : NOOP 255)

ERGB:

ERGB8888(E : Exponent 0 ~ 16)

説明 :

#define NJD_CM_AS (NJD_MATOFF+6)

#define NJD_CM_AS2 (NJD_MATOFF+14)

Blending Alpha、ambient、specular を設定する。diffuse については現在の設定値のままとする。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_DAS', 'NJD_CM_DAS2'

(Chunk Material Diffuse, Ambient and Specular and Chunk Material Diffuse, Ambient and Specular 2)

概要 :

plist において diffuse、specular、ambient の値を設定する。diffuse の上位 8bit には 値が設定される。specular の上位 8bit には exponent (0 ~ 16) が設定される。NJD_CM_DAS2 は 2 パラメータモディファイアボリュームデータ用に定義される。headbits には NJD_CM_DAS, NJD_CM_DAS2 とともに同じ値が設定される。これはライブラリの動作を単純化するための処理。

形式 :

[headbits(15-8)|ChunkHead(7-0)][6(shortsize)][ARGB][NRGB][ERGB]

headbits:

15 = SRC Select(1bit)

14 = DST Select(1bit)

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_DAS, NJD_CM_DAS2

ARGB:

ARGB8888

NRGB:

NRGB8888(N : NOOP 255)

ERGB:

ERGB8888(E : Exponent 0 ~ 16)

説明 :

#define NJD_CM_DAS (NJD_MATOFF+7)

#define NJD_CM_DAS2 (NJD_MATOFF+15)

Blending Alpha、diffuse、ambient、specular を設定する。SRC Select, DST Select はトライリニアフィルタで利用される。2 パス、3 パスで実現されるトライリニアフィルタ描画におけるアキュムレーションバッファの制御をする。

ChunkName : 'NJD_CM_BU'

(Chunk Material BUmp)

概要 :

バンプマップテクスチャの貼られた方向を定義する。headbits には 0 が設定される。

形式 :

[headbits(15-8)|ChunkHead(7-0)][6(shortsize)][dx][dy][dz][ux][uy][uz]

headbits:

現在未使用。

ChunkHead:

NJD_CM_BU

[dx][dy][dz]はテクスチャの貼った方向。[ux][uy][uz]は上方向を示すベクトル。

説明 :

```
#define NJD_CM_BU (NJD_MATOFF+8)
```

バンプマップテクスチャの貼った方向を与える。この方向を元にハードウェアはバンプマッピングの計算をする。

8.12 Chunk Vertex

Chunk Vertex はモデルの頂点リストを与える。頂点リストにユーザの目的とするデータを格納するために頂点リストもチャンク形式を利用する。チャンクの種類を切り替えることで頂点、頂点法線、頂点カラー、ユーザフラグ等のデータをケースバイケースで頂点リストに持たせることができる。ChunkHead の上位 8bit の部分 (headbits) はエンベロープの制御、CompactShape の頂点モーションの中で利用されている。

モデルに複数のチャンクタイプを同時に使用することはライブラリ側の処理を複雑にするため現在のところ対応しない。一つのモデルに対しては一種類の Chunk Vertex を利用する。Sint16 ベースでヘッダは構成されるが格納する配列が Sint32 のため Sint16 のデータ二つを Sint32 に合成して配置する。

NinjaFlags32 を利用する NF はコンバータがエンベロープを出力する時に利用する。ユーザ指定での出力はできない。エンベロープのウェイトの精度は Ninja2 より 8 ビットから 16 ビット精度の変更された。

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
```

ChunkHead:

- < 標準的な形式 (頂点法線なし) >
- NJD_CV、NJD_CV_D8、NJD_CV_UF、NJD_CV_NF
- < 標準的な形式 (頂点法線あり) >
- NJD_CV_VN、NJD_CV_VN_D8、NJD_CV_VN_UF、NJD_CV_VN_NF
- < ライブラリダイレクト形式で利用されるチャンク >
- NJD_CV_D8_S8
- ライブラリの仕様書参照のこと。
- < 頂点カラー付き NF チャンク >
- NJD_CV_NF_D8
- 現在リザーブ。
- < CompactShape で追加されたチャンク >
- NJD_VN、NJD_VN_D8、NJD_D8
- Ninja2 CompactShape 仕様書参照のこと。

略号の意味は以下のとおり。

VN : use vertex normal
D8 : Diffuse ARGB8888
S8 : Specular ERGB8888
NF : NinjaFlags32 for envelope
UF : UserFlags32

IndexOffset はライブラリの頂点中間バッファの使用開始位置オフセットを与える。例えばオフセット 0 である親ノードの頂点を計算しその子供が親の頂点数と等しいオフセットを指定してその位置から頂点計算結果を格納したとすると親の頂点計算結果は上書きされずに中間バッファに残る。この状態で親の頂点インデックスまでさかのぼる番号を指定することにより親と子の頂点を結んだポリゴンを表現することができる。

nbIndices は、チャンクに格納される頂点数を与える。

CompactShape 用に追加された NJD_VN、NJD_VN_D8、NJD_D8 は頂点モーションの表現力を高めるために追加された。法線のみ、法線頂点カラー、頂点カラーのみの頂点モーションをライブラリはサポートするが現時点で頂点カラーモーションのデータ作成ができかつ出力できるのは LightWave のみ。頂点モーションを構成するモデルに頂点カラーを順次埋め込むことで利用可能。

エンベロープ及び CompactShape における headbits の使用方法是次の通り。

```
/* Flag Weight status (NF only) */
#define NJD_FW_SHIFT      8
#define NJD_FW_START      (0<<NJD_FW_SHIFT)    /* Start */
#define NJD_FW_MIDDLE     (1<<NJD_FW_SHIFT)    /* Middle */
```

```
#define NJD_FW_END      (2<<NJD_FW_SHIFT)      /* End      */
```

```
#define NJD_FW_MASK     (0x2<<NJD_FW_SHIFT)
```

エンベロープで分割された頂点の種類を示す。START, MIDDLE, END においてそれぞれ段階に必要な頂点の計算処理がされる。NF のみで有効。

```
/* Flag Vertex */
```

```
#define NJD_FV_SHIFT     8
```

```
#define NJD_FV_CONT      (0x80<<NJD_FV_SHIFT)    /* vertex calculation continue */
```

エンベロープでは自分より後にトレースされるモデルから影響を受けている場合その後ろのモデル頂点計算が終わらないと描画できない。コンバータはこの計算の完了を検出し完了時にポリゴン描画を実行する。すべてのポリゴン描画が終わらない状態では未描画の頂点の計算結果をリセットせず残しておく必要がありこのフラグはその状態を示す。

```
/* Flag Compact Shape */
```

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

頂点チャンクを CompactShape 用に使用していることを示す。CompactShape 用に使用している場合 [IndexOffset(31-16)|nbIndices(15-0)]を省略しているので注意が必要。

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

詳しくはNinja2 CompactShape 仕様書参照のこと。

ChunkName : 'NJD_CV'

(Chunk Vertex)

概要 :

vlist において頂点リストを定義する。頂点法線なし。

形式 :

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
  headbits:
    NJD_FV_SHAPE(CompactShape 使用時)
  ChunkHead:
    NJD_CV
  longsize:
    次のチャンクまでのロングワード(4バイト)オフセット。
  IndexOffset:
    頂点中間バッファの開始位置。
  nbIndices:
    頂点の数。
  Data:
    x,y,z, ...
```

説明 :

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

頂点リストを与える。頂点法線なし。

ChunkName : 'NJD_CV_D8'
(Chunk Vertex Diffuse ARGB8888)

概要 :

vlist において頂点リストを定義する。頂点法線なし。頂点カラーあり。

形式 :

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
  Headbits:
    NJD_FV_SHAPE(CompactShape 使用時)
  ChunkHead:
    NJD_CV_D8
  longsize:
    次のチャンクまでのロングワード(4バイト)オフセット。
  IndexOffset:
    頂点中間バッファの開始位置。
  nbIndices:
    頂点の数。
  Data:
    x,y,z,D8888,...
```

説明 :

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

頂点リストを与える。頂点法線なし。頂点カラーあり。頂点カラーは Sint32 にパックされる。

ChunkName : 'NJD_CV_UF'

(Chunk Vertex UserFlag)

概要 :

vlist において頂点リストを定義する。頂点法線なし。ユーザフラグ領域を 32bit 持つ。

形式 :

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
  Headbits:
    0 固定。
  ChunkHead:
    NJD_CV_UF
  longsize:
    次のチャンクまでのロングワード ( 4 バイト ) オフセット。
  IndexOffset:
    頂点中間バッファの開始位置を与える。
  nbIndices:
    頂点の数。
  Data:
    x,y,z,UserFlags32, ...
```

説明 :

```
#define NJD_CV_UF (NJD_VERTOFF+4)
```

頂点リストを与える。頂点法線なし。ユーザフラグ領域を 32bit 持つ。現在この領域に頂点カラーを出力できる。

ChunkName : 'NJD_CV_NF'
(Chunk Vertex NinjaFlags32)

概要：

vlist においてエンベロープ頂点リストを定義する。頂点法線なし。デフォルトで法線ありのエンベロープデータを使用するのでこちらはあまり使われない。ライブラリは現在未対応。リザーブ。

形式：

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
Headbits:
    NJD_FW_START, NJD_FW_MIDDLE, NJD_FW_END, NJD_FV_CONT
ChunkHead:
    NJD_CV_NF
longsize:
    次のチャンクまでのロングワード ( 4 バイト ) オフセット。
IndexOffset:
    頂点中間バッファの開始位置。
nbIndices:
    頂点の数。
Data:
    x,y,z,NinjaFlags32, ...
```

説明：

```
#define NJD_CV_NF (NJD_VERTOFF+5)
```

エンベロープ用頂点リストを与える。頂点法線なし。headbits に設定されるフラグは次の 4 つ。

```
#define NJD_FW_START (0<<NJD_FW_SHIFT) /* Start */
#define NJD_FW_MIDDLE (1<<NJD_FW_SHIFT) /* Middle */
#define NJD_FW_END (2<<NJD_FW_SHIFT) /* End */
#define NJD_FV_CONT (0x80<<NJD_FV_SHIFT) /* vertex calculation continue */
```

マルチウェイトのエンベロープ頂点データにおいてマルチウェイト分分割された頂点データの頂点バッファへの最初の書き込みデータを NJD_FW_START、最後の書き込みを NJD_FW_END、最初でも最後でもない書き込みを NJD_FW_MIDDLE で表現する。NJD_START、NJD_FW_END は任意の頂点に対し各一つずつ存在し NJD_FW_MIDDLE は任意の数存在する。エンベロープでは自分より後にトレースされるモデルから影響を受けている場合その後ろのモデル頂点計算が終わらないと描画できない。コンバータはこの計算の完了を検出し完了時にポリゴン描画を実行する。すべてのポリゴン描画が終わらない状態では未描画の頂点の計算結果をリセットせず残しておく必要があり NJD_FW_CONT は頂点バッファをリセットしないことを示す。

フラグの上位 16bit にウェイト値を 16 ビット精度で格納する。下位 16bit に共有頂点バッファ上の頂点インデックス番号が入る。IndexOffset には頂点インデックス番号のベース番号が入る。つまり実際の頂点インデックス番号は IndexOffset+頂点インデックス番号で表される。また Native グループは通常の NJD_CV_VN を使うため NF 部分を持たないため IndexOffset のみで共有頂点バッファ上の領域を指定する。

ChunkName : 'NJD_CV_VN'
(Chunk Vertex VertexNormal)

概要 :

vlist において頂点リストを定義する。頂点法線あり。

形式 :

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]  
[IndexOffset(31-16)|nbIndices(15-0)][Data]  
  Headbits:  
    NJD_FV_SHAPE(CompactShape 使用時)  
  ChunkHead:  
    NJD_CV_VN  
  longsize:  
    次のチャンクまでのロングワード(4バイト)オフセット。  
  IndexOffset:  
    頂点中間バッファの開始位置。  
  nbIndices:  
    頂点の数。  
  Data:  
    x,y,z,nx,ny,nz, ...
```

説明 :

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

頂点リストを与える。頂点法線あり。もっとも標準的な頂点リストであり頻繁に使用される。

ChunkName : 'NJD_CV_VN_D8'

(Chunk Vertex VertexNormal and Diffuse ARGB8888)

概要 :

vlist において頂点リストを定義する。頂点法線あり。頂点カラーあり。

形式 :

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
  Headbits:
    NJD_FV_SHAPE(CompactShape 使用時)
  ChunkHead:
    NJD_CV_VN_D8
  longsize:
    次のチャンクまでのロングワード(4バイト)オフセット。
  IndexOffset:
    頂点中間バッファの開始位置。
  nbIndices:
    頂点の数。
  Data:
    x,y,z,nx,ny,nz,D8888,...
```

説明 :

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

頂点リストを与える。頂点法線あり。頂点カラーあり。頂点カラーは Sint32 にパックされる。

ChunkName : 'NJD_CV_VN_UF'

(Chunk Vertex VertexNormal and UserFlags32)

概要 :

vlist において頂点リストを定義する。頂点法線あり。ユーザフラグ領域を 32bit 持つ。

形式 :

```
[Headbits(31-24)][ChunkHead(23-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
  Headbits:
    0 固定。
  ChunkHead:
    NJD_CV_VN_UF
  longsize:
    次のチャンクまでのロングワード ( 4 バイト ) オフセット。
  IndexOffset:
    頂点中間バッファの開始位置。
  nbIndices:
    頂点の数。
  Data:
    x,y,z,nx,ny,nz,UserFlags32,...
```

説明 :

```
#define NJD_CV_VN_UF (NJD_VERTOFF+11)
```

頂点リストを与える。頂点法線あり。ユーザフラグ領域を 32bit 持つ。現在この領域に頂点カラーを出力できる。

ChunkName : 'NJD_CV_VN_NF'

(Chunk Vertex VertexNormal and NinjaFlags32)

概要 :

vlist においてエンベロープ用頂点リストを定義する。頂点法線あり。エンベロープ用頂点リストの基本となるチャンク。

形式 :

```
[Headbits(31-24)][ChunkHead(31-16)|longsize(15-0)]
[IndexOffset(31-16)|nbIndices(15-0)][Data]
Headbits:
    NJD_FW_START, NJD_FW_MIDDLE, NJD_FW_END, NJD_FV_CONT
ChunkHead:
    NJD_CV_VN_NF
longsize:
    次のチャンクまでのロングワード ( 4 バイト ) オフセット。
IndexOffset:
    頂点中間バッファの開始位置。
nbIndices:
    頂点の数。
Data:
    x,y,z,nx,ny,nz,NinjaFlags32,...
```

説明 :

```
#define NJD_CV_VN_NF (NJD_VERTOFF+12)
```

エンベロープ用頂点リストを与える。頂点法線あり。headbits に設定されるフラグは次の 4 つ。

```
#define NJD_FW_START (0<<NJD_FW_SHIFT) /* Start */
#define NJD_FW_MIDDLE (1<<NJD_FW_SHIFT) /* Middle */
#define NJD_FW_END (2<<NJD_FW_SHIFT) /* End */
#define NJD_FV_CONT (0x80<<NJD_FV_SHIFT) /* vertex calculation continue */
```

マルチウェイトのエンベロープ頂点データにおいてマルチウェイト分分割された頂点データの頂点バッファへの最初の書き込みデータを NJD_FW_START、最後の書き込みを NJD_FW_END、最初でも最後でもない書き込みを NJD_FW_MIDDLE で表現する。NJD_START、NJD_FW_END は任意の頂点に対し各一つずつ存在し NJD_FW_MIDDLE は任意の数存在する。エンベロープでは自分より後にトレースされるモデルから影響を受けている場合その後ろのモデル頂点計算が終わらないと描画できない。コンバータはこの計算の完了を検出し完了時にポリゴン描画を実行する。すべてのポリゴン描画が終わらない状態では未描画の頂点の計算結果をリセットせず残しておく必要があり NJD_FW_CONT は頂点バッファをリセットしないことを示す。

フラグの上位 16bit にウェイト値を 16 ビット精度で格納する。下位 16bit に共有頂点バッファ上の頂点インデックス番号が入る。IndexOffset には頂点インデックス番号のベース番号が入る。つまり実際の頂点インデックス番号は IndexOffset+頂点インデックス番号で表される。また Native グループは通常の NJD_CV_VN を使うため NF 部分を持たないため IndexOffset のみで共有頂点バッファ上の領域を指定する。

8.13 Chunk Volume

Chunk Volume は、コリジョン、モディファイアボリューム用に用意される。直接ライブラリで描画に使用されずマテリアルデータは持たない。次の三つがある。

```
#define NJD_CO_P3      (NJD_VOLOFF+0)
#define NJD_CO_P4      (NJD_VOLOFF+1)
#define NJD_CO_ST      (NJD_VOLOFF+2)
```

NJD_CO_P3 は独立三角形データ、NJD_CO_P4 独立三角形及び独立四角形から構成される。3D Studio MAX ではすべてデータが三角形データとして出力されるがコンバータはオプションで面間角度が 0.1 度以下の面同士を接続して四角形を復元し独立四角形として出力できる。この機能により 3D Studio MAX においても四角形コリジョンを生成可能。

また元データが三角形/四角形/N角形混在データの場合にコンバータオプションですべてを三角形に分割し再び独立三角形、四角形に再構成することができる。この場合データには N 角形がなくなり plist には NJD_CO_P3 と NJD_CO_P4 が並ぶ。

NJD_CO_ST は、トライアングルストリップによる Chunk Volume である。Chunk Volume は Chunk Strip(後述)のものと等価なポリゴンに対するユーザフラグ領域 (16, 32, 48bit) を持つことができる。またモデラーで設定されたマテリアルカラーを各ポリゴンのユーザフラグ領域に出力することができる。

NJD_CO_P3 の独立三角形 Chunk Volume はモディファイアボリュームデータにそのまま利用できる (ただし 3D 空間的に閉じた独立三角形のみのボリュームにする必要があるので注意が必要。ハード仕様)。ChunkHead の上位 8bit の部分 (headbits) は、Chunk Volume に関しては未使用。

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14)|nbPolygon(13-0)][Data]
```

ユーザフラグ領域は 16bit 単位 (plist が Sint16 の一次元配列のため) で扱われ、16、32、48bit の三つが選べる。この領域サイズの指定はポリゴン数を与える nbPolygon の最上位 2bit に割り振られた UserOffset で指定される。

```
UserFlags_Offset:
#define NJD_UFO_SHIFT  14
#define NJD_UFO_0      (0<<NJD_UFO_SHIFT)
#define NJD_UFO_1      (1<<NJD_UFO_SHIFT)
#define NJD_UFO_2      (2<<NJD_UFO_SHIFT)
#define NJD_UFO_3      (3<<NJD_UFO_SHIFT)
#define NJD_UFO_MASK    (0x3<<NJD_UFO_SHIFT) /* 0 - 3 */
```

UserOffset=NJD_UFO_0 の時 UserFlags サイズは 0bit。

UserOffset=NJD_UFO_1 の時 UserFlags サイズは 16bit。

UserOffset=NJD_UFO_2 の時 UserFlags サイズは 32bit。

UserOffset=NJD_UFO_3 の時 UserFlags サイズは 48bit。

ChunkName : 'NJD_CO_P3'

(Chunk volume Polygon)

概要 :

plist において volume 用ポリゴンリストを定義する。直接描画には使用されずコリジョン、モディファイアボリュームに使用される。マテリアルは持たないがユーザフラグ領域にポリゴンカラーを出力できる。

形式 :

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14)|nbPolygon(13-0)][Data]
ChunkHead:
    NJD_CO_P3
shortsize:
    次のチャンクまでのショート ( 2 バイト ) オフセット。
UserOffset:
    ユーザフラグ領域のサイズ。
nbPolygon:
    ポリゴン数。
Data:
    index0, index1, index2, UserflagPoly0(*N),
    index3, index4, index5, UserflagPoly1(*N), ...
```

説明 :

```
#define NJD_CO_P3 (NJD_VOLOFF+0)
```

コリジョン、モディファイアボリューム用の独立三角形データ。マテリアルは持たないがユーザフラグ領域にポリゴンカラーを出力できる。このポリゴンカラーはマテリアルで設定されたもの。ポリゴンインデックスの後ろにユーザフラグ領域を持つ。このサイズは UserOffset の値 (NJD_UF0_0 ではなし、NJD_UF0_1 では 16bit、NJD_UF0_2 では 32bit、NJD_UF0_3 では 48bit) で決まる。ライブラリはユーザフラグ領域に関して何も処理せずスキップする。元データが三角形/四角形/ N 角形の混在する場合でもコンバータオプションですべてを独立三角形に分割し NJD_CO_P3 に出力することができる。

ChunkName : 'NJD_CO_P4'

(Chunk volume Polygon4)

概要 :

plist において volume 用ポリゴンリストを定義する。直接描画には使われずコリジョンに使用する。マテリアルは持たないがユーザフラグ領域にポリゴンカラーを出力できる。

形式 :

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14)|nbPolygon(13-0)][Data]
ChunkHead:
    NJD_CO_P4
shortsize:
    次のチャンクまでのショート ( 2 バイト ) オフセット。
UserOffset:
    ユーザフラグ領域のサイズ。
nbPolygon:
    ポリゴン数。
Data:
    index0, index1, index2, index3, UserflagPoly0(*N),
    index4, index5, index6, index7, UserflagPoly1(*N), ...
```

説明 :

```
#define NJD_CO_P4 (NJD_VOLOFF+1)
```

コリジョン用の独立四角形データ。マテリアルは持たないがユーザフラグ領域にポリゴンカラーを出力できる。このポリゴンカラーはマテリアルで設定されたもの。ポリゴンインデックスの後ろにユーザフラグ領域を持つ。このサイズは UserOffset の値 (NJD_UF0_0 ではなし、NJD_UF0_1 では 16bit、NJD_UF0_2 では 32bit、NJD_UF0_3 では 48bit) で決まる。ライブラリはユーザフラグ領域に関して何も処理せずスキップする。3D Studio MAX の場合すべてのデータが独立三角形で出力されるがコンバータオプションで面間角度が 0.1 度以下の場合二つの三角形を連結して四角形を再現しこれにより独立四角形データを生成できる。

ChunkName : 'NJD_CO_ST'
(Chunk volume Triangle Strip)

概要 :

plist において volume 用ポリゴンリストを定義する。直接描画には使用されずコリジョンに使用される。マテリアルは持たないがユーザフラグ領域にポリゴンカラーを出力できる。

形式 :

```
[ChunkHead(15-0)][shortsize(15-0)]
  [UserOffset(15-14)|nbPolygon(13-0)][Data]
ChunkHead:
  NJD_CO_ST
shortsize:
  次のチャンクまでのショート ( 2 バイト ) オフセット。
UserOffset:
  ユーザフラグ領域のサイズ。
nbPolygon:
  ポリゴン数。
Data:
  [flag(15)|len(14-0), i0, i1, i2, Userflag2(*N), i3, Userflag3(*N), ...]
```

説明 :

```
#define NJD_CO_ST (NJD_VOLOFF+2)
```

コリジョンとして使用される。NJD_CO_P3 及び NJD_CO_P4 よりもコリジョンデータサイズを小さくすることができる。ただしストリップでは三角形が連結される方向はより効率良い方向に進むため必ずしもユーザの意図した方向へは進まないで注意が必要。flag はストリップ先頭の三角形の回転方向(右回り/左回り)を与える。ChunkModel は、符号により右回り/左回りを切り替える。マイナス値の場合は右回り。len はストリップに含まれる頂点数の数。i?はポリゴン頂点インデックスを意味する。マテリアルは持たないがユーザフラグ領域にポリゴンカラーを出力できる。このポリゴンカラーはマテリアルで設定されたもの。ポリゴン頂点インデックスの後ろにユーザフラグ領域を持つ。このサイズはUserOffset の値(NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit)で決まる。ライブラリはユーザフラグ領域に関して何も処理せずスキップする。元データが三角形/四角形/N角形の混在する場合でもコンバータは自動的にすべてを独立三角形に分割しトライアングルストリップを作成 NJD_CO_ST に出力する。

8.14 Chunk Strip

Chunk Strip は Chunk Vertex の頂点リストからライブラリ内に展開された頂点中間バッファの頂点エントリ番号指定からなるトライアングルストリップデータ。ポリゴン単位のユーザフラグ領域を持つことができる。ユーザフラグ領域にはマテリアルから設定されたポリゴンカラーを出力できる。

Ninja1 ではポリゴン側に頂点カラー、法線を出力できたが Ninja2 ではこれをサポートしない。Ninja2 ではコンバータオプションで頂点分割を指定することでこれに対応する。頂点がポリゴン境界で複数の法線もしくは頂点カラーを持つ場合コンバータオプションの Divide Edge Vertex を呼び出すことで頂点複製されたデータが出力できる。ただしその数が多い場合頂点数は 1.5 倍を超える場合もあるので注意が必要。また頂点が分割されることでトライアングルストリップの連結率が下がることによる頂点増加も合わせて起きるので注意が必要。

ChunkHead の上位 8bit の部分 (headbits) にはマテリアルから設定されるアトリビュートフラグ (ChunkFlags) が設定される。

略号の意味は次の通り。

IL : Ignore light
IS : Ignore specular
IA : Ignore ambient
UA : Use alpha
DB : Double side
FL : Flat shading
ENV : Environment mapping

Flag Strip :

```
#define NJD_FST_SHIFT      8
#define NJD_FST_IL         (0x01<<NJD_FST_SHIFT)
#define NJD_FST_IS         (0x02<<NJD_FST_SHIFT)
#define NJD_FST_IA         (0x04<<NJD_FST_SHIFT)
#define NJD_FST_UA         (0x08<<NJD_FST_SHIFT)
#define NJD_FST_DB         (0x10<<NJD_FST_SHIFT)
#define NJD_FST_FL         (0x20<<NJD_FST_SHIFT)
#define NJD_FST_ENV        (0x40<<NJD_FST_SHIFT)
#define NJD_FST_MASK       (0xFF<<NJD_FST_SHIFT)
```

次に Chunk Strip のフォーマットを示す。ポリゴン単位の UserFlags が index2 以降に一つずつ挿入されていることに注意。これは三点目で始めて一つ目の三角形が生成されることによる。四点目以降は一点ごとに三角形が一つ生成されるため一頂点ごとに頂点の後ろにユーザフラグが配置される。

テクスチャなしの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),      index0(15-0),
                           index1(15-0),
                           index2, UserFlag2(*N), ...]
```

テクスチャありの場合

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)]
[flag(15)|len(14-0),      index0(15-0), U0(15-0), V0(15-0),
                           index1, U1, V1,
                           index2, U2, V2, UserFlag2(*N), ... ]
```

Ninja2 は二種類の UV 値表現を持つ。一方は 8 ビット(0-255)による UVN、もう一方は分解能を高めた 16 ビット(0-1023)による UVH。UVN ではピクセル数 256 を超える幅もしくは高さを持つテクスチャの 1 ピクセル単位での指定ができない。UVH はハイレゾモードを意味し 1024x1024 のテクスチャで 1 ピクセル単位での指定が可能。ただし UVH は UVN よりも分解能を上げた分だけテクスチャのリPEATの表現回数が減少する (UVN が 128 回に対し UVH では 32 回)。コンバートオプションによりモデルツリー全体に対しまたマテリアルネームにより各モデル単位で UVN、UVH を切り替えることができる。

マテリアルネームからの指定では単一のモデルに使われる複数のマテリアルのどれか一つに設定することでモデルに対する UV 表現が変更される。デフォルトは UVN。

UVN : Normal type Uv (0-255)

UVH : Hiresolution type Uv (0-1023)

Chunk Volume と同様に Chunk Strip でもユーザフラグ領域は nbStrip の最上位 2bit に割り付けられた UserOffset で与えられる。16bit 単位 (plist が Sint16 の一次元配列のため) で扱われ、16、32、48bit の三つが選べる。

UserFlags Offset:

```
#define NJD_UFO_SHIFT          14
#define NJD_UFO_0              (0<<NJD_UFO_SHIFT)
#define NJD_UFO_1              (1<<NJD_UFO_SHIFT)
#define NJD_UFO_2              (2<<NJD_UFO_SHIFT)
#define NJD_UFO_3              (3<<NJD_UFO_SHIFT)
#define NJD_UFO_MASK           (0x3<<NJD_UFO_SHIFT) /* 0 - 3 */
```

UserOffset=NJD_UFO_0 の時 UserFlags サイズは 0bit。

UserOffset=NJD_UFO_1 の時 UserFlags サイズは 16bit。

UserOffset=NJD_UFO_2 の時 UserFlags サイズは 32bit。

UserOffset=NJD_UFO_3 の時 UserFlags サイズは 48bit。

ChunkName : 'NJD_CS', 'NJD_CS_2'

(Chunk Strip and Chunk Strip 2)

概要 :

plist においてポリゴンリストを定義する。テクスチャなし。NJD_CS_2 は 2 パラメータモデル
ファイアデータにおいて使用する。UV 値を持たないため NJD_CS と形式は同じ。

形式 :

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
  ChunkFlags:
    NJD_FST_IL(光源無視)、NJD_FST_IS(スペキュラ無視)、
    NJD_FST_IA(アンビエント無視)、NJD_FST_UA(Use Alpha)、
    NJD_FST_DB(両面)、NJD_FST_FL(フラットシェーディング)、
    NJD_FST_ENV(環境マッピング)。
  ChunkHead:
    NJD_CS
  shortsize:
    次のチャンクまでのショート(2バイト)オフセット。
  UserOffset:
    ユーザフラグ領域のサイズ。
  nbStrip:
    ストリップの数。
  Data:
    [flag(15)|len(14-0), index0(15-0),
      index1(15-0),
      index2, UserFlag2(*N), ...]
```

説明 :

```
#define NJD_CS                (NJD_STRIPOFF+0)
#define NJD_CS_2              (NJD_STRIPOFF+9)
```

flag はストリップ先頭の三角形の回転方向(右回り/左回り)を与える。マイナス値の場合は右回り。len はストリップに含まれる頂点数の数。index?はポリゴン頂点インデックスを意味する。ポリゴン頂点インデックスの後ろにユーザフラグ領域を持つ。このサイズは UserOffset の値(NJD_UF0_0 ではなく、NJD_UF0_1 では 16bit、NJD_UF0_2 では 32bit、NJD_UF0_3 では 48bit)で決まる。ライブラリはユーザフラグ領域を何も処理せずスキップする。三角形/四角形/N角形を自動的に独立三角形に分割しストリップに変換して出力する。

ChunkName : 'NJD_CS_UVN', 'NJD_CS_UVN2'

(Chunk Strip UVN and Chunk Strip UVN 2)

概要 :

plist においてポリゴンリストを定義する。テクスチャあり。UV 値は UVN(0-255) で与えられる。
NJD_CS_UVN2 は 2 パラメータモディファイアデータにおいて使用する。UV 値を 2 重に持つため
NJD_CS_UVN と形式が異なる。

形式 :

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
  ChunkFlags:
    NJD_FST_IL(光源無視)、NJD_FST_IS(スペキュラ無視)、
    NJD_FST_IA(アンビエント無視)、NJD_FST_UA(Use Alpha)、
    NJD_FST_DB(両面)、NJD_FST_FL(フラットシェーディング)、
    NJD_FST_ENV(環境マッピング)。
  ChunkHead:
    NJD_CS_UVN
  shortsize:
    次のチャンクまでのショート(2バイト)オフセット。
  UserOffset:
    ユーザフラグ領域のサイズ。
  nbStrip:
    ストリップの数。
  Data:
    (NJD_CS_UVN の場合)
    [flag(15)|len(14-0), index0(15-0), U0(15-0), V0(15-0),
      index1, U1, V1,
      index2, U2, V2, UserFlag2(*N), ... ]
    (NJD_CS_UVN2 の場合)
    [flag(15)|len(14-0), index0(15-0), U0(15-0), V0(15-0),
      index1, U1a, V1a, U1b, V1b,
      index2, U2a, V2a, U2b, V2b, UserFlag2(*N), ... ]
```

ここでは U?a, V?a は一つ目のテクスチャの UV 値、U?b, V?b は二つ目のテクスチャの UV 値を意味する。

説明 :

```
#define NJD_CS_UVN                (NJD_STRIPOFF+1)
#define NJD_CS_UVN2              (NJD_STRIPOFF+10)
```

UV 値には UVN (0-255) が使用される。flag はストリップ先頭の三角形の回転方向(右回り/左回り)を与える。マイナス値の場合は右回り。len はストリップに含まれる頂点数の数。index? はポリゴン頂点インデックスを意味する。ポリゴン頂点インデックスの後ろにユーザフラグ領域を持つ。このサイズは UserOffset の値(NJD_UFO_0 ではなし、NJD_UFO_1 では 16bit、NJD_UFO_2 では 32bit、NJD_UFO_3 では 48bit)で決まる。ライブラリはユーザフラグ領域を何も処理せずスキップする。三角形/四角形/N角形を自動的に独立三角形に分割しストリップに変換して出力する。

ChunkName : 'NJD_CS_UVH', 'NJD_CS_UVH2'

(Chunk Strip UVH and Chunk Strip UVH 2)

概要 :

plist においてポリゴンリストを定義する。テクスチャあり。UV 値は UVH(0-1023)で与えられる。NJD_CS_UVH2 は 2 パラメータモディファイアデータにおいて使用する。UV 値を 2 重に持つため NJD_CS_UVH と形式が異なる。

形式 :

```
[ChunkFlags(15-8)|ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14)|nbStrip(13-0)][Data]
  ChunkFlags:
    NJD_FST_IL(光源無視)、NJD_FST_IS(スペキュラ無視)、
    NJD_FST_IA(アンビエント無視)、NJD_FST_UA(Use Alpha)、
    NJD_FST_DB(両面)、NJD_FST_FL(フラットシェーディング)、
    NJD_FST_ENV(環境マッピング)。
  ChunkHead:
    NJD_CS_UVH
  shortsize:
    次のチャンクまでのショート(2バイト)オフセット。
  UserOffset:
    ユーザフラグ領域のサイズ。
  nbStrip:
    ストリップの数。
  Data:
    (NJD_CS_UVH の場合)
    [flag(15)|len(14-0), index0(15-0), U0(15-0), V0(15-0),
      index1, U1, V1,
      index2, U2, V2, UserFlag2(*N), ... ]
    (NJD_CS_UVH2 の場合)
    [flag(15)|len(14-0), index0(15-0), U0(15-0), V0(15-0),
      index1, U1a, V1a, U1b, V1b,
      index2, U2a, V2a, U2b, V2b, UserFlag2(*N), ... ]
```

ここでは U?a, V?a は一つ目のテクスチャの UV 値、U?b, V?b は二つ目のテクスチャの UV 値を意味する。

説明 :

```
#define NJD_CS_UVH (NJD_STRIPOFF+2)
```

UV 値には UVH(0-1023)が使用される。UVN に比べリピートの上限值が低くなっている(32 回)ことに注意。flag はストリップ先頭の三角形の回転方向(右回り/左回り)を与える。マイナス値の場合は右回り。len はストリップに含まれる頂点数の数。index?はポリゴン頂点インデックスを意味する。ポリゴン頂点インデックスの後ろにユーザフラグ領域を持つ。このサイズは UserOffset の値(NJD_UF0_0 ではなし、NJD_UF0_1 では 16bit、NJD_UF0_2 では 32bit、NJD_UF0_3 では 48bit)で決まる。ライブラリはユーザフラグ領域を何も処理せずスキップする。三角形/四角形/N 角形を自動的に独立三角形に分割しストリップに変換して出力する。

9. Ninja2 フォーマット

9.1 拡張子一覧

Ninja2 ファイルフォーマットの拡張子を説明する。Ninja2 はアスキーフォーマットとバイナリフォーマットをサポートする。アスキーフォーマット、バイナリフォーマットはそれぞれに一対一対応する。拡張子のベース nj という文字列でありこれに中身を意味する 1 文字を付加する。アスキーでは nj の j を a に置き換えて表現する。

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
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

(注意事項)

Ninja2 では SimpleShape は使用できない。Ninja2 用頂点モーションデータ nas ファイルは無条件に CompactShape ファイルを意味する。

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。

ShapeList とは頂点モーションに必要な頂点群を同一キーフレームを持つキーフレームグループ単位でくりこれをリスト化したものである。頂点モーションデータからリストの Id (ShapeId) を指定することで頂点を指定できる。

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

Chunk モデルバイナリデータ出力ファイルである njd, nj において Ninja1 のバイナリチャンク名と異なるチャンク名を使用する。これは同じ Chunk フォーマットであるが Ninja1 と Ninja2 で仕様が変更されたため同

一扱いをしないための処置である。アスキーフォーマットに関してはヘッダなどで内容確認が容易なため特別な処置はしない。

ユーザはコンバート時のオプション指定でアスキーフォーマット、バイナリフォーマットのどちらでも得ることができる。

二種類以上のデータを格納した場合.nj(.nja)が使われる。現在のモデルコンバータはすべて.njaを出力する。これにはtexlist(.nat)とmodel(.nad)が含まれている。

Ninjaにおけるシーンファイルはシーンを構成するモデル、カメラ、ライト、モーションを括るためのファイル名リスト。NinjaViewerでのシーン表示のため単位にもなる。アスキー出力のみ。

リソースファイル.nreは汎用のリソース格納ファイル。libnreライブラリにより管理されNinja2グラフィックツールで必要とされるリソースの保存/読み出し/利用を一括管理する。現在テクスチャのコンバータオプション保存に適用されている。

アスキーフォーマットはアスキーローダによる読み込みもしくはNjDef.hをインクルードしてコンパイルすることで利用が可能。

バイナリーフォーマットはすべてIFFチャンク形式で格納される。各種データを同一のファイルにまとめて格納することが可能。バイナリローダを使うことでコンパイルなしでユーザメモリ領域へのデータ格納が可能。

アスキーローダ、バイナリローダはソースを開示している。

pvpファイルはパレットデータを格納する。

pvmファイルは複数のテクスチャを一括して扱うためのファイルフォーマット。パレットテクスチャ、pvpファイルも包含可能。

10. アスキーフォーマット

10.1 アスキーフォーマット概要

モデルのアスキー出力時のフォーマットに関して説明する。Ninjaのアスキー出力データは基本的にC言語ソースでありコンパイルすることでプログラムに組み込み実行できる。C言語ソース出力ではあるがC言語をそのまま出力すると改行、オフセットなど自由度が高すぎ一定のフォーマットにし難い。そのためすべての出力データにマクロを定義しこのマクロを通してデータを配置することで一定のフォーマットを構成する。マクロの文字列を利用することでアスキーローダの作成が容易となり、またマクロの内容変更によりデータの加工も可能となる。文字列置換も容易である。

アスキーフォーマットに関わるマクロはライブラリに定義される NjDef.h に定義される。

アスキーフォーマットに配置されるデータは extern 宣言なしで利用するためには他の構造体からのポインタ参照される前に配置される必要がある。そのため例えばモデルデータは下位のデータから PList, VList, Model, Object の順に配置される。またモデル階層は子供優先で左からトレースした場合の一番最後のモデルから順に配置される。つまりルートモデルの NJS_CNK_OBJECT 構造体はファイルの最後に定義される。

ユーザの利便性を考慮しルートモデルの NJS_CNK_OBJECT 構造体名をファイルの先頭にコメントとして定義する。またデータの情報としてモデルの数 n、PushPop の深さ d、エンベロープにおける最大頂点バッファエントリ番号 v を出力する。texlist がある場合これの変数名も合わせてコメント出力する。情報として texlist のエントリ数 n が出力される。

次にアスキー出力 (nja ファイル) における先頭の例を示す。一行目のヘッダに関しては後述する。

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */

/* ROOT OBJECT : object_same_heavywalk_body_3_body_3 n(53) d(14) v(268) */
/* TEXTLIST      : texlist_same_heavywalk_body_3 n(9) */
```

またバイナリ出力ではデータの配置はアスキーデータの逆順となる。つまりルートモデルの NJS_CNK_OBJECT 構造体がデータの先頭になり各モデルデータは左から子供優先でトレースした順番に格納される。ここで注意することはメモリ上の配置の違いからキャッシュのヒット率が微妙に変わりバイナリデータとアスキーデータの実行性能に差がある可能性があることである。簡単なテストではほとんど変わらなかった。しかし実使用において性能に違いがある可能性は考慮する必要がある。

アスキーフォーマットの各データはアスキーチャンク形式で表現される。これはアスキーレベルで処理の単位を明確にし読み取りの単位を示すものである。define の中身は空である。

NJS_CNK_OBJECT 出力におけるアスキーチャンクの定義の例を次に示す。

```
<NjDef.h>
#define CNKOBJECT_START
#define CNKOBJECT_END
```

```
<アスキーチャンク構造>
CNKOBJECT_START
```

```
    <オブジェクトに関わるデータ>
    <オブジェクトデータ>
```

```
CNKOBJECT_END
```

アスキーチャンクの内容はモデル、モーション、カメラ、ライト、CellSprite 等すべての Ninja2 アスキーデータ出力に適用される。

アスキーデータ出力においてファイルの最後にアスキーチャンク DEFAULT を持つ。これは一定の名前でサンプルプログラムに組み込む場合などに利用する。複数のアスキーファイルがインクルードされたサンプルプログラムにおいて一番最初にインクルードされたファイル名がデフォルト変数名に define される。

```
<NjDef.h>
#define DEFAULT_START
#define DEFAULT_END
```

<アスキーチャンク DEFAULT の例>

```
DEFAULT_START

#ifdef DEFAULT_OBJECT_NAME
#define DEFAULT_OBJECT_NAME object_same_heavywalk_body_3_body_3
#endif

#ifdef DEFAULT_TEXLIST_NAME
#define DEFAULT_TEXLIST_NAME texlist_same_heavywalk_body_3
#endif

DEFAULT_END
```

ここでは nja ファイルに定義される NJS_CNK_OBJECT 構造体、NJS_TEXLIST 構造体の変数名が固定変数名 DEFAULT_OBJECT_NAME, DEFAULT_TEXLIST_NAME に define されている。

アスキーチャンク DEFAULT は使用頻度は低いが定義があっても問題ないので出力される。

次にファイルの一行目に定義されるヘッダ情報について説明する。

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)

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (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。

モデルに関わるのは次の 4 つ。

NJA	.nja	複数の種類のアスキーチャンクを格納する場合に利用する。デフォルトで texlist とモデルデータを格納する出力に利用される。またユーザが意図的に他のアスキーチャンクをファイルに取り込んだ場合も nja ファイルとしては有効である。
NAD	.nad	アスキーチャンクでモデルデータのみを格納する場合に利用する。
NAT	.nat	アスキーチャンクで texlist データのみを格納する場合に利用する。
NSC	.nsc	シーンを構成するモデル、モーション、mrs ファイルなどのファイル名を列記する。ビューアなどに一括してファイルを指定する場合に利用する。

モーションその他に関しては Ninja2Motion 仕様書参照のこと。

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。
"Ninja2AsciiTexture"	テクスチャ (texlist) 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

頂点データのアスキー出力において C 言語仕様上 Sint32 の配列にデータを配置する都合上 float 値をそのままアスキー出力することができない。そのため float 値はすべて 16 進数で出力される。float 値を型変換なく取り出すには float f に対し *((long *)&f) として取り出すことで可能となる。

(頂点の 16 進数出力の例)

```
CnkV_VN(0x0, 337),
OffnbIdx(0, 56),
VERT( 0x4011498c, 0x3e64a3c0, 0x3fb945ce ),
NORM( 0x3f070018, 0x3f0b4166, 0x3eb1ab0e ),
VERT( 0x401f048a, 0xbe932030, 0x3f9117c4 ),
NORM( 0x3f538bc0, 0xbc2cacd2, 0xbddb6a1f ),
VERT( 0x40038f66, 0xbead2a60, 0x3fdfa11a ),
NORM( 0x3d5852f0, 0xbc8af570, 0x3f54dc37 ),
...
```

VERT は頂点データ、NORM は頂点法線データを意味する。コンバータはオプションでデータの後ろに実際の float 値をコメントとして出力できる。

10.2 Chunk Model アスキーフォーマットのマクロ定義

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利用される。

10.2.1 汎用マクロ

START	データの開始を意味する。
END	データの終了を意味する。
Uvn(_u, _v)	8 ビット (256) 精度の UV 値データを出力する。
Uvh(_u, _v)	10 ビット (1024) 精度の UV 値データを出力する。
_NJANG(_x, _y, _z)	Angle データを出力する。
_ARGB(_a, _r, _g, _b)	ARGB データを出力する。

```

/*-----*/
/* Start/End */
/*-----*/

#define START      ={
#define END        };

/*----*/
/* Uv */
/*----*/

#define Uvn( _u, _v)      (_u), (_v)
#define Uvh( _u, _v)      (_u), (_v)

/*-----*/
/* Angle */
/*-----*/

#define _NJAVAL          (65536.0F/360.0000F)
#define _NJANG1(_a)      (long)(_NJAVAL*(_a))
#define _NJANG( _x, _y, _z)          ¥
    {(long)(_NJAVAL*(_x)), (long)(_NJAVAL*(_y)), (long)(_NJAVAL*(_z))}

/*-----*/
/* ARGB */
/*-----*/

#define _ARGB( _a, _r, _g, _b)          ¥
    ( (unsigned long)((_a)<<24)|(unsigned long)((_r)<<16)  ¥
    | (unsigned long)((_g)<<8)|(_b) )
#define ARGB( _a, _r, _g, _b) _ARGB( _a, _r, _g, _b)
#define _NJARGB( _a, _r, _g, _b) _ARGB( _a, _r, _g, _b)

```

DEFAULT_START	DEFAULT アスキーチャンクの始まりを示す。
DEFAULT_END	DEFAULT アスキーチャンクの終わりを示す。

```

/*-----*/
/* DEFAULT */

```

```
/*-----*/
```

```
#define DEFAULT_START
```

```
#define DEFAULT_END
```

10.2.2 texlist マクロ

モデル用、CellSprite において利用される。テクスチャ参照の基本構造でありパレットテクスチャにおけるバンク I d も格納される。モデル、CellSpite は texlist のエントリ I d である texId を通してテクスチャを指定する。

TEXN(_texname)	NJS_TEXNAME 構造体データを出力する。
TEXN3(_texname, _attr, _texaddr)	NJS_TEXNAME 構造体データを出力する。アスキー出力に _attr, _texaddr の引数を出力したい場合に利用する。
TEXTURE_START	texlist アスキーチャンクの始まりを示す。
TEXTURE_END	texlist アスキーチャンクの終わりを示す。
TEXTURENAME	NJS_TEXNAME 構造体配列データを示す。
TEXTURELIST	NJS_TEXLIST 構造体データを示す。
TextureList	NJS_TEXNAME 構造体配列データポインタを示す。
TextureNum	NJS_TEXNAME 構造体配列データの長さを示す。
PTEXN(_tname, _cnt, _bnk)	パレット時の NJS_TEXNAME 構造体データを出力する。マテリアルネームから設定されたパレットのバンク番号が出力される。
PTEXN5(_tname, _attr, _texaddr, _cnt, _bnk)	パレット時の NJS_TEXNAME 構造体データを出力する。マテリアルネームから設定されたパレットのバンク番号が出力される。アスキー出力に _attr, _texaddr の引数を出力したい場合に利用する。

```
/*-----*/
```

```
/* TexList */
```

```
/*-----*/
```

```
#define TEXN(_texname)      {(_texname), 0x0, 0}
```

```
#define TEXN3(_texname, _attr, _texaddr) {(_texname), _attr, _texaddr}
```

```
#define TEXTURE_START
```

```
#define TEXTURENAME        NJS_TEXNAME
```

```
#define TEXTURELIST        NJS_TEXLIST
```

```
#define TEXTURE_END
```

```
#define TextureList
```

```
#define TextureNum
```

```
/* palette */
```

```
#define _SET_CNT(_f) (_f ? NJD_TEXATTR_TEXCONTINUE : 0)
```

```
#define _SET_BNK(_b) (((_b) << NJD_TEXBANK_SHIFT) & NJD_TEXBANK_MASK)
```

```
#define _SET_BNK2(_b, _g) (_SET_BNK(_b)|((_g) & ~NJD_TEXBANK_MASK))
```

```
#define PTEXN(_tname, _cnt, _bnk) {(_tname), _SET_CNT(_cnt), _SET_BNK(_bnk)}
```

```
#define PTEXN5(_tname, _attr, _texaddr, _cnt, _bnk)      ¥
```

```
{(_tname), _SET_CNT(_cnt)|(_attr), _SET_BNK2(_bnk, _texaddr)}
```


10.2.3 インスタンスマクロ

Ninja ではインスタンスを C 言語のポインタレベルで実現している。任意のモデルから参照できるようにファイルの先頭で変数名の extern 宣言をする。モデル単位の場合はインスタンスする NJS_CNK_MODEL 構造体名をモデル階層単位の場合はインスタンスする NJS_CNK_OBJECT 構造体名を extern 宣言する。

INSTANCE_PROTOTYPE_START	インスタンス用モデルの extern 宣言をくくる。インスタンス用アスキーチャンクの始まりを示す。
INSTANCE_PROTOTYPE_END	インスタンス用モデルの extern 宣言をくくる。インスタンス用アスキーチャンクの終わりを示す。

```
/*-----*/
/* instance */
/*-----*/

#define INSTANCE_PROTOTYPE_START
#define INSTANCE_PROTOTYPE_END
```

10.2.4 EvalFlags マクロ

モデル階層描画時の処理の最適化及び属性情報を持つ。NJS_CNK_OBJECT 構造体で利用される。

FEV_UT	NJD_EVAL_UNIT_POS をアスキーフォーマット上で出力する。
FEV_UR	NJD_EVAL_UNIT_ANG をアスキーフォーマット上で出力する。
FEV_US	NJD_EVAL_UNIT_SCL をアスキーフォーマット上で出力する。
FEV_HD	NJD_EVAL_HIDE をアスキーフォーマット上で出力する。
FEV_BR	NJD_EVAL_BREAK をアスキーフォーマット上で出力する。
FEV_ZXY	NJD_EVAL_ZXY_ANG をアスキーフォーマット上で出力する。
FEV_SK	NJD_EVAL_SKIP をアスキーフォーマット上で出力する。
FEV_SSK	NJD_EVAL_SHAPE_SKIP をアスキーフォーマット上で出力する。
FEV_CL	NJD_EVAL_CLIP をアスキーフォーマット上で出力する。
FEV_MD	NJD_EVAL_MODIFIER をアスキーフォーマット上で出力する。
FEV_QU	NJD_EVAL_QUATERNION をアスキーフォーマット上で出力する。
FEV_EN	NJD_EVAL_ENVELOPE をアスキーフォーマット上で出力する。

```
/*-----*/
/* Chunk Bits */
/*-----*/

/** EvalFlags */

#define FEV_UT    NJD_EVAL_UNIT_POS
#define FEV_UR    NJD_EVAL_UNIT_ANG
#define FEV_US    NJD_EVAL_UNIT_SCL
#define FEV_HD    NJD_EVAL_HIDE
#define FEV_BR    NJD_EVAL_BREAK
#define FEV_ZXY   NJD_EVAL_ZXY_ANG
#define FEV_SK    NJD_EVAL_SKIP
#define FEV_SSK   NJD_EVAL_SHAPE_SKIP
#define FEV_CL    NJD_EVAL_CLIP
```

```
#define FEV_MD      NJD_EVAL_MODIFIER
#define FEV_QU      NJD_EVAL_QUATERNION
#define FEV_EN      NJD_EVAL_ENVELOPE
```

10.2.5 Blending Alpha マクロ

ピクセル書き込み時の Src データ, Dst データの Blending Alpha の制御をする。Chunk Material、Chunk Bits で利用される。

FBS_ZER	NJD_FBS_ZER をアスキーフォーマット上で出力する。
FBS_ONE	NJD_FBS_ONE をアスキーフォーマット上で出力する。
FBS_OC	NJD_FBS_OC をアスキーフォーマット上で出力する。
FBS_IOC	NJD_FBS_IOC をアスキーフォーマット上で出力する。
FBS_SA	NJD_FBS_SA をアスキーフォーマット上で出力する。
FBS_ISA	NJD_FBS_ISA をアスキーフォーマット上で出力する。
FBS_DA	NJD_FBS_DA をアスキーフォーマット上で出力する。
FBS_IDA	NJD_FBS_IDA をアスキーフォーマット上で出力する。
FBS_SEL	NJD_FBS_SEL をアスキーフォーマット上で出力する。アキュームレーションバッファ制御フラグ。
FBD_ZER	NJD_FBD_ZER をアスキーフォーマット上で出力する。
FBD_ONE	NJD_FBD_ONE をアスキーフォーマット上で出力する。
FBD_OC	NJD_FBD_OC をアスキーフォーマット上で出力する。
FBD_IOC	NJD_FBD_IOC をアスキーフォーマット上で出力する。
FBD_SA	NJD_FBD_SA をアスキーフォーマット上で出力する。
FBD_ISA	NJD_FBD_ISA をアスキーフォーマット上で出力する。
FBD_DA	NJD_FBD_DA をアスキーフォーマット上で出力する。
FBD_IDA	NJD_FBD_IDA をアスキーフォーマット上で出力する。
FBD_SEL	NJD_FBD_SEL をアスキーフォーマット上で出力する。アキュームレーションバッファ制御フラグ。

```
/** Flags Blending SRC */
/* Flag Blending Src */
#define FBS_ZER      NJD_FBS_ZER
#define FBS_ONE      NJD_FBS_ONE
#define FBS_OC       NJD_FBS_OC
#define FBS_IOC      NJD_FBS_IOC
#define FBS_SA       NJD_FBS_SA
#define FBS_ISA      NJD_FBS_ISA
#define FBS_DA       NJD_FBS_DA
#define FBS_IDA      NJD_FBS_IDA

/* Flag Buffer Src SElect */
#define FBS_SEL      NJD_FBS_SEL

/* Flag Blending Dst */
#define FBD_ZER      NJD_FBD_ZER
#define FBD_ONE      NJD_FBD_ONE
#define FBD_OC       NJD_FBD_OC
#define FBD_IOC      NJD_FBD_IOC
#define FBD_SA       NJD_FBD_SA
```

```
#define FBD_ISA    NJD_FBD_ISA
#define FBD_DA     NJD_FBD_DA
#define FBD_IDA    NJD_FBD_IDA
```

```
/* Flag Buffer Dst SElect */
```

```
#define FBD_SEL    NJD_FBD_SEL
```

10.2.6 'D' Adjust マクロ

Z 値によるミップマップの切り替わり位置を調整する。デフォルト FDA_100。Chunk Material、Chunk Bits で設定される。

FDA_025	NJD_FDA_025 をアスキーフォーマット上で出力する。設定値 0.25 を意味する。
FDA_050	NJD_FDA_050 をアスキーフォーマット上で出力する。設定値 0.50 を意味する。
FDA_075	NJD_FDA_075 をアスキーフォーマット上で出力する。設定値 0.75 を意味する。
FDA_100	NJD_FDA_100 をアスキーフォーマット上で出力する。設定値 1.00 を意味する。
FDA_125	NJD_FDA_125 をアスキーフォーマット上で出力する。設定値 1.25 を意味する。
FDA_150	NJD_FDA_150 をアスキーフォーマット上で出力する。設定値 1.50 を意味する。
FDA_175	NJD_FDA_175 をアスキーフォーマット上で出力する。設定値 1.75 を意味する。
FDA_200	NJD_FDA_200 をアスキーフォーマット上で出力する。設定値 2.00 を意味する。
FDA_225	NJD_FDA_225 をアスキーフォーマット上で出力する。設定値 2.25 を意味する。
FDA_250	NJD_FDA_250 をアスキーフォーマット上で出力する。設定値 2.50 を意味する。
FDA_275	NJD_FDA_275 をアスキーフォーマット上で出力する。設定値 2.75 を意味する。
FDA_300	NJD_FDA_300 をアスキーフォーマット上で出力する。設定値 3.00 を意味する。
FDA_325	NJD_FDA_325 をアスキーフォーマット上で出力する。設定値 3.25 を意味する。
FDA_350	NJD_FDA_350 をアスキーフォーマット上で出力する。設定値 3.50 を意味する。
FDA_375	NJD_FDA_375 をアスキーフォーマット上で出力する。設定値 3.75 を意味する。

```
/** Flag 'D' Adjust **/
```

```
#define FDA_025    NJD_FDA_025
#define FDA_050    NJD_FDA_050
#define FDA_075    NJD_FDA_075
#define FDA_100    NJD_FDA_100
#define FDA_125    NJD_FDA_125
#define FDA_150    NJD_FDA_150
#define FDA_175    NJD_FDA_175
#define FDA_200    NJD_FDA_200
#define FDA_225    NJD_FDA_225
#define FDA_250    NJD_FDA_250
#define FDA_275    NJD_FDA_275
#define FDA_300    NJD_FDA_300
#define FDA_325    NJD_FDA_325
#define FDA_350    NJD_FDA_350
#define FDA_375    NJD_FDA_375
```

10.2.7 exponent マクロ

マテリアルのスペキュラーの反射の鋭さを定義する。小さい値ほどスペキュラーの輝きの輪が大きくなる。このパラメータは NormalDraw で有効であるが Ninja2 において NormalDraw が廃止されたため設定値はライブラリで使用されない。

FEXP_00	NJD_FEXP_00をアスキーフォーマット上で出力する。設定値0を意味する。
FEXP_01	NJD_FEXP_01をアスキーフォーマット上で出力する。設定値1を意味する。
FEXP_02	NJD_FEXP_02をアスキーフォーマット上で出力する。設定値2を意味する。
FEXP_03	NJD_FEXP_03をアスキーフォーマット上で出力する。設定値3を意味する。
FEXP_04	NJD_FEXP_04をアスキーフォーマット上で出力する。設定値4を意味する。
FEXP_05	NJD_FEXP_05をアスキーフォーマット上で出力する。設定値5を意味する。
FEXP_06	NJD_FEXP_06をアスキーフォーマット上で出力する。設定値6を意味する。
FEXP_07	NJD_FEXP_07をアスキーフォーマット上で出力する。設定値7を意味する。
FEXP_08	NJD_FEXP_08をアスキーフォーマット上で出力する。設定値8を意味する。
FEXP_09	NJD_FEXP_09をアスキーフォーマット上で出力する。設定値9を意味する。
FEXP_10	NJD_FEXP_10をアスキーフォーマット上で出力する。設定値10を意味する。
FEXP_11	NJD_FEXP_11をアスキーフォーマット上で出力する。設定値11を意味する。
FEXP_12	NJD_FEXP_12をアスキーフォーマット上で出力する。設定値12を意味する。
FEXP_13	NJD_FEXP_13をアスキーフォーマット上で出力する。設定値13を意味する。
FEXP_14	NJD_FEXP_14をアスキーフォーマット上で出力する。設定値14を意味する。
FEXP_15	NJD_FEXP_15をアスキーフォーマット上で出力する。設定値15を意味する。
FEXP_16	NJD_FEXP_16をアスキーフォーマット上で出力する。設定値16を意味する。

```
/** Flag EXponent range : 0-16 */
```

```
#define FEXP_00    NJD_FEXP_00
#define FEXP_01    NJD_FEXP_01
#define FEXP_02    NJD_FEXP_02
#define FEXP_03    NJD_FEXP_03
#define FEXP_04    NJD_FEXP_04
#define FEXP_05    NJD_FEXP_05
#define FEXP_06    NJD_FEXP_06
#define FEXP_07    NJD_FEXP_07
#define FEXP_08    NJD_FEXP_08
#define FEXP_09    NJD_FEXP_09
#define FEXP_10    NJD_FEXP_10
#define FEXP_11    NJD_FEXP_11
#define FEXP_12    NJD_FEXP_12
#define FEXP_13    NJD_FEXP_13
#define FEXP_14    NJD_FEXP_14
#define FEXP_15    NJD_FEXP_15
#define FEXP_16    NJD_FEXP_16
```

10.2.8 Chunk Null マクロ

CnkNull()	NJD_CN をアスキーチャンク上で出力する。オフセット調整用のダミーチャンクである CnkNull を意味する。
-----------	---

```
/* Chunk Null */
```

```
#define CnkNull()    (NJD_CN)
```

10.2.9 Chunk End マクロ

CnkEnd()	NJD_CE をアスキーチャンク上で出力する。チャンクデータの終了を示す。
----------	---------------------------------------

```
/* Chunk End */
```

```
#define CnkEnd()      (NJD_CE)
```

10.2.10 Chunk Bits マクロ

CnkB_BA(_bits)	NJD_CB_BA をアスキーフォーマット上で出力する。Blending Alpha および SRC Select, DST Select を設定する。
CnkB_DA(_bits)	NJD_CB_DA をアスキーフォーマット上で出力する。ミップマップの切り替わりの Z 値の調整。
CnkB_CP(_bits)	NJD_CB_CP をアスキーフォーマット上で出力する。エンベロープに関わるポリゴンリストでポリゴン描画に必要な全頂点の計算が完了していない場合これが完了するまでポリゴンリストの描画を保留するために利用する。ポリゴンリストを Id と共にテーブルに登録する。
CnkB_DP(_bits)	NJD_CB_DP をアスキーフォーマット上で出力する。保留されたポリゴンリストの描画を実行する。

```
/* Chunk Bits */
```

```
#define CnkB_BA( _bits)  (_bits)|NJD_CB_BA      /* Blending Alpha      */
```

```
#define CnkB_DA( _bits)  (_bits)|NJD_CB_DA      /* Mipmap 'D' adjust      */
```

```
#define CnkB_CP( _bits)  ((_bits)<<8)|NJD_CB_CP /* Cashe polygon list     */
```

```
#define CnkB_DP( _bits)  ((_bits)<<8)|NJD_CB_DP /* Draw polygon list      */
```

10.2.11 TexId マクロ (Chunk Tiny)

カレントのテクスチャの設定をする。

FFL_U	NJD_FFL_U をアスキーフォーマット上で出力する。U 値をフリップする。
FFL_V	NJD_FFL_V をアスキーフォーマット上で出力する。V 値をフリップする。
FCL_U	NJD_FCL_U をアスキーフォーマット上で出力する。U 値をクランプする。
FCL_V	NJD_FCL_V をアスキーフォーマット上で出力する。V 値をクランプする。
FFM_PS	NJD_FFM_PS をアスキーフォーマット上で出力する。ポイントサンプリングフィルタを使用する。
FFM_BF	NJD_FFM_BF をアスキーフォーマット上で出力する。バイリニアフィルタを利用する。
FFM_TFA	NJD_FFM_TF_A をアスキーフォーマット上で出力する。2 パストライリニアフィルタの A パスを設定する。
FFM_TFB	NJD_FFM_TF_B をアスキーフォーマット上で出力する。2 パストライリニアフィルタの B パスを設定する。
FSS	NJD_FSS をアスキーフォーマット上で出力する。アンアイソトロピックフィルタを設定する。ポイントサンプリング、バイリニア、トライリニアと独立に設定できる。

```
/*-----*/
```

```
/* Chunk Tiny */
```

```
/*-----*/
```

```
/** TexId **/
```

```
/* Flag FLip <headbits> */
```

```
#define FFL_U      NJD_FFL_U
```

```
#define FFL_V      NJD_FFL_V
```

```
/* Flag CLamp <headbits> */
```

```

#define FCL_U      NJD_FCL_U
#define FCL_V      NJD_FCL_V
/* Flag Filter Mode<texbits> */
#define FFM_PS     NJD_FFM_PS
#define FFM_BF     NJD_FFM_BF
#define FFM_TFA    NJD_FFM_TF_A
#define FFM_TFB    NJD_FFM_TF_B
/* Flag Super Sample<texbits> */
#define FSS        NJD_FSS

```

現在 Chunk Tiny には TexId 制御に関わるマクロのみが定義される。_TID マクロは CnkT_TID, CnkT_TID2 の引数として利用されるマクロ。

CnkT_TID(_bits)	NJD_CT_TIDをアスキーフォーマットで出力する。TexIdを設定する。
_TID(_bits,_tid)	NJD_CT_TID, NJD_CT_TID2の引数をアスキーフォーマットで出力する。
CnkT_TID2(_bits)	NJD_CT_TID2をアスキーフォーマットで出力する。2パラモディファイア時の二つ目のテクスチャのTexIdを設定する。

```

/* Chunk Tiny */
#define CnkT_TID( _bits)  (_bits)|NJD_CT_TID      /* Tiny TexId          */
#define _TID( _bits,_tid) (_bits)|(_tid)          /* TexId                */

/* Chunk Tiny for 2 parameter modifier data */
#define CnkT_TID2( _bits) (_bits)|NJD_CT_TID2    /* Tiny TexId          */

```

10.2.12 Chunk Material マクロ

MDiff(_a, _r, _g, _b)	マテリアルマクロの引数として利用される。Diffuse 成分を出力する。
MSpec(_a, _r, _g, _b)	マテリアルマクロの引数として利用される。Specular 成分を出力する。
MAmbi(_a, _r, _g, _b)	マテリアルマクロの引数として利用される。Ambient 成分を出力する。
AttrF(_f)	AttributeFlag 成分を出力する。現在未使用。リザーブ。

```

/*-----*/
/* Material */
/*-----*/
#define MDiff( _a, _r, _g, _b) ((_g) << 8)|(_b), ((_a) << 8)|(_r)
#define MSpec( _a, _r, _g, _b) ((_g) << 8)|(_b), ((_a) << 8)|(_r)
#define MAmbi( _a, _r, _g, _b) ((_g) << 8)|(_b), ((_a) << 8)|(_r)
#define AttrF( _f ) ((_f) & 0xffff), (((_f) >> 16) & 0xffff)

```

FlatBumpに関わるマクロ。

Pvn(_x, _y, _z)	Ninja1 においてポリゴン法線を 16 ビット精度で出力する。Ninja2 では直接利用されないが _BuDir, _BuUp で FlatBump 用に利用される。
_BuDir(_dx, _dy, _dz)	FlatBump の貼った方向を 16 ビット精度で出力する。

_BuUp(_dx, _dy, _dz)	FlatBump の貼った方向の上方向を 16 ビット精度で出力する。
----------------------	-------------------------------------

```

/*-----*/
/* PolygonVertexNormal */
/*-----*/
#define _PVN_MAX 32767.0F

#define FtoU(_a_) ((long)((_a_) * _PVN_MAX) & 0xffffU)

#ifdef USE_UNSIGNED_TYPE
#define Pvn( _x, _y, _z) FtoU(_x), FtoU(_y), FtoU(_z)
#else
#define Pvn( _x, _y, _z) (_x)*_PVN_MAX, (_y)*_PVN_MAX, (_z)*_PVN_MAX
#endif

/*-----*/
/* Bump */
/*-----*/
#define _BuDir(_dx, _dy, _dz) Pvn(_dx, _dy, _dz)
#define _BuUp(_dx, _dy, _dz) Pvn(_dx, _dy, _dz)

```

CnkM_D(_bits)	NJD_CM_D をアスキーフォーマット上で出力する。ディフューズ成分のみ出力。
CnkM_S(_bits)	NJD_CM_S をアスキーフォーマット上で出力する。スペキュラー成分のみ出力。
CnkM_DS(_bits)	NJD_CM_DS をアスキーフォーマット上で出力する。ディフューズ、スペキュラー成分出力。
CnkM_A(_bits)	NJD_CM_A をアスキーフォーマット上で出力する。アンビエント成分のみ出力。
CnkM_DA(_bits)	NJD_CM_DA をアスキーフォーマット上で出力する。ディフューズ、アンビエント成分出力。
CnkM_AS(_bits)	NJD_CM_AS をアスキーフォーマット上で出力する。アンビエント、スペキュラー成分出力。
CnkM_DAS(_bits)	NJD_CM_DAS をアスキーフォーマット上で出力する。ディフューズ、アンビエント、スペキュラー成分出力。
CnkM_D2(_bits)	NJD_CM_D2 をアスキーフォーマット上で出力する。ディフューズ成分のみ出力。2パラ用。
CnkM_S2(_bits)	NJD_CM_S2 をアスキーフォーマット上で出力する。スペキュラー成分のみ出力。2パラ用。
CnkM_DS2(_bits)	NJD_CM_DS2 をアスキーフォーマット上で出力する。ディフューズ、スペキュラー成分出力。2パラ用。
CnkM_A2(_bits)	NJD_CM_A2 をアスキーフォーマット上で出力する。アンビエント成分のみ出力。2パラ用。
CnkM_DA2(_bits)	NJD_CM_DS2 をアスキーフォーマット上で出力する。ディフューズ、アンビエント成分出力。2パラ用。
CnkM_AS2(_bits)	NJD_CM_AS2 をアスキーフォーマット上で出力する。アンビエント、スペキュラー成分出力。2パラ用。
CnkM_DAS2(_bits)	NJD_CM_DAS2 をアスキーフォーマット上で出力する。ディフューズ、アンビエント、スペキュラー成分出力。2パラ用。
CnkM_BU(_bits)	NJD_CM_BU をアスキーフォーマット上で出力する。バンプマップを貼った方向を示す成分を出力。

ポリゴン描画時のマテリアルの出力用マクロ。一つ前に設定された値との差分のみが設定される。各種差分を表現できるように種々のマクロが用意される。最後に 2 がつくマクロは 2 パラメータモディファイアボリューム描画における二つ目のマテリアルを意味する。

```

/* Chunk Material */
#define CnkM_D( _bits) (_bits)|NJD_CM_D /* Diffuse */

```

```

#define CnkM_S( _bits)    (_bits)|NJD_CM_S      /* Specular          */
#define CnkM_DS( _bits)   (_bits)|NJD_CM_DS     /* Diffuse, Specular */
#define CnkM_A( _bits)    (_bits)|NJD_CM_A      /* Ambient           */
#define CnkM_DA( _bits)   (_bits)|NJD_CM_DA     /* Diffuse, Ambient  */
#define CnkM_AS( _bits)   (_bits)|NJD_CM_AS     /* Ambient, Specular */
#define CnkM_DAS( _bits)  (_bits)|NJD_CM_DAS    /* Diff, Abmi, Spec  */

/* Chunk Material for 2 parameter modifier data */
#define CnkM_D2( _bits)   (_bits)|NJD_CM_D2     /* Diffuse          */
#define CnkM_S2( _bits)   (_bits)|NJD_CM_S2     /* Specular         */
#define CnkM_DS2( _bits)  (_bits)|NJD_CM_DS2    /* Diffuse, Specular */
#define CnkM_A2( _bits)   (_bits)|NJD_CM_A2     /* Ambient          */
#define CnkM_DA2( _bits)  (_bits)|NJD_CM_DA2    /* Diffuse, Ambient */
#define CnkM_AS2( _bits)  (_bits)|NJD_CM_AS2    /* Ambient, Specular */
#define CnkM_DAS2( _bits) (_bits)|NJD_CM_DAS2   /* Diff, Abmi, Spec */

/* Chunk Bump */
#define CnkM_BU(_bits)    NJD_CM_BU              /* bumpmap direction */

```

10.2.13 Chunk Vertex マクロ

Chunk Vertex の Headbits に設定される。頂点チャンクの頂点書き込み制御、頂点モーションデータ時の目印などに利用する。

FW_START	NJD_FW_START をアスキーフォーマット上で出力する。ウェイトありの頂点の最初の書き込みを示す。
FW_MIDDLE	NJD_FW_MIDDLE をアスキーフォーマット上で出力する。ウェイトありの頂点の最初と最後以外の書き込みを示す。
FW_END	NJD_FW_END をアスキーフォーマット上で出力する。ウェイトありの頂点の最後の書き込みを示す。
FV_CONT	NJD_FV_CONT をアスキーフォーマット上で出力する。頂点バッファのクリアをしないことを示す。
FV_SHAPE	NJD_FV_SHAPE をアスキーフォーマット上で出力する。頂点モーションデータチャンクであることを示す。

```

/*-----*/
/* Chunk Vertex */
/*-----*/
/** Flag WeightStatus (NF only) **/
#define FW_START    NJD_FW_START
#define FW_MIDDLE    NJD_FW_MIDDLE
#define FW_END      NJD_FW_END

/** Flag Vertex **/
#define FV_CONT      NJD_FV_CONT

/* Flag Compact Shape */
#define FV_SHAPE      NJD_FV_SHAPE

```


VERT(_x, _y, _z)	頂点データを出力する。
NORM(_x, _y, _z)	頂点法線データを出力する。
D8888(_a, _r, _g, _b)	Diffuse 成分 ARGB を出力する。
UFlags(_f)	ユーザフラグデータを出力する。
NFlagsW(_idx, _wp)	Ninja1 において 8 ビット(256)精度のエンベロップウェイトデータを出力する。
NFlagsW2(_idx, _wp)	Ninja2 において 16 ビット(1024)精度のエンベロップウェイトデータを出力する。

```
/*-----*/
```

```
/* Vertex */
```

```
/*-----*/
```

```
#define VERT( _x, _y, _z)  (_x), (_y), (_z)
```

```
#define NORM( _x, _y, _z)  (_x), (_y), (_z)
```

```
#define D8888( _a, _r, _g, _b)  _ARGB(_a, _r, _g, _b)
```

```
#define UFlags( _f)          (_f)
```

```
#define _WPCa( _p)           (((unsigned long)(((p)*255.0F)/100.0F+0.5F))
```

```
#define NFlagsW( _idx, _wp)  (((_WPCa(_wp) & 0xffff)<<16)|((_idx) & 0xffff))
```

```
#define _WPCa2( _p)          (((unsigned long)(((p)*65535.0F)/100.0F+0.5F))
```

```
#define NFlagsW2( _idx, _wp) (((_WPCa2(_wp) & 0xffff)<<16)|((_idx) & 0xffff))
```

CnkV(_f, _s)	NJD_CVをアスキーフォーマット上に出力する。法線無し頂点データ。
CnkV_D8(_f, _s)	NJD_CV_D8をアスキーフォーマット上に出力する。法線無し頂点カラーあり頂点データ。
CnkV_UF(_f, _s)	NJD_CV_UFをアスキーフォーマット上に出力する。法線無しユーザフラグ領域あり頂点データ。
CnkV_NF(_f, _s)	NJD_CV_NFをアスキーフォーマット上に出力する。法線無しエンベロップウェイトあり頂点データ。通常は使わない。法線ありを使う。
CnkV_VN(_f, _s)	NJD_CV_VNをアスキーフォーマット上に出力する。法線あり頂点データ。
CnkV_VN_D8(_f, _s)	NJD_CV_VN_D8をアスキーフォーマット上に出力する。法線あり頂点カラーあり頂点データ。
CnkV_VN_UF(_f, _s)	NJD_CV_VN_UFをアスキーフォーマット上に出力する。法線ありユーザフラグ領域あり頂点データ。
CnkV_VN_NF(_f, _s)	NJD_CV_VN_NFをアスキーフォーマット上に出力する。法線ありエンベロップウェイトありの頂点データ。
CnkV_D8_S8(_f, _s)	NJD_CV_D8_S8をアスキーフォーマット上に出力する。ディフューズ、スペキュラーの頂点カラーあり頂点データ。現在これに対応するのはライブラリのダイレクト形式のみ。リザーブ。
CnkV_NF_D8(_f, _s)	NJD_CV_NF_D8をアスキーフォーマット上に出力する。現在未対応。リザーブ。
CnkVN(_f, _s)	NJD_VNをアスキーフォーマット上に出力する。法線のみ。頂点モーション専用チャンク。
CnkVN_D8(_f, _s)	NJD_VN_D8をアスキーフォーマット上に出力する。法線と頂点カラー。頂点モーション専用チャンク。
CnkD8(_f, _s)	NJD_D8をアスキーフォーマット上に出力する。頂点カラーのみ。頂点モーション専用チャンク。
OffnbIdx(_off, _nb)	Chunk Vertexの引数を与える。頂点バッファ上のオフセットと頂点の数。頂点モーション時はこの引数が省略される。

```
/* Chunk Vertex */
```

```
#define _CkV(_f, _s)        (((_s)<<16)|(_f))
```

```
#define CnkV(_f, _s)        (_CkV(_f, _s)|NJD_CV)          /* x,y,z, ... */
```

```
#define CnkV_D8(_f, _s)     (_CkV(_f, _s)|NJD_CV_D8)       /* x,y,z,D8888,... */
```

```
#define CnkV_UF(_f, _s)     (_CkV(_f, _s)|NJD_CV_UF)       /* x,y,z,UserFlags32,... */
```

```
#define CnkV_NF(_f, _s)     (_CkV(_f, _s)|NJD_CV_NF)       /* x,y,z,NinjaFlags32,... */
```

```

#define CnkV_VN(_f, _s)  (_CkV(_f, _s)|NJD_CV_VN)      /* x,y,z, ... */
#define CnkV_VN_D8(_f, _s) (_CkV(_f, _s)|NJD_CV_VN_D8) /* x,y,z,nx,ny,nz,D8888,... */
#define CnkV_VN_UF(_f, _s) (_CkV(_f, _s)|NJD_CV_VN_UF) /* x,y,z,nx,ny,nz,UFlags32, */
#define CnkV_VN_NF(_f, _s) (_CkV(_f, _s)|NJD_CV_VN_NF) /* x,y,z,nx,ny,nz,NFlags32, */

#define CnkV_D8_S8(_f, _s) (_CkV(_f, _s)|NJD_CV_D8_S8) /* x,y,z,D8888,S8888,... for Ninja2 */
#define CnkV_NF_D8(_f, _s) (_CkV(_f, _s)|NJD_CV_NF_D8) /* x,y,z,NinjaFlags32,D8888,... for Ninja2 */

/* Chunk Vertex for CompactShape */

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

#define OffnbIdx( _off, _nb) ((_nb)<<16)|(_off)

```

10.2.14 Chunk Volume マクロ

コリジョンデータ、モディファイアボリューム（NJD_CO_P3のみ）データを出力する。ライブラリはこのモデルを描画できないが三角形、四角形、トライアングルストリップで形状が出力できる。

CnkO_P3(_bits)	NJD_CO_P3をアスキーフォーマットに出力する。コリジョン、モディファイアボリュームに利用。
CnkO_P4(_bits)	NJD_CO_P4をアスキーフォーマットに出力する。コリジョンデータに利用。0.1度以下の三角形を連結し四角形に自動変換。
CnkO_ST(_bits)	NJD_CO_STをアスキーフォーマットに出力する。コリジョンデータに利用。

```

/* Chunk volume */

#define CnkO_P3(_bits)    NJD_CO_P3 /* volume polygon 3 */
#define CnkO_P4(_bits)    NJD_CO_P4 /* volume polygon 4 */
#define CnkO_ST(_bits)    NJD_CO_ST /* volume strip */

```

10.2.15 Chunk Strip マクロ

ChunkStrip の Headbits に関わるマクロ。

FST_IL	NJD_FST_ILをアスキーフォーマット上で出力する。光源無視。
FST_IS	NJD_FST_ISをアスキーフォーマット上で出力する。スペキュラー無視。
FST_IA	NJD_FST_IAをアスキーフォーマット上で出力する。アンビエント無視。
FST_UA	NJD_FST_UAをアスキーフォーマット上で出力する。Use Alpha。
FST_DB	NJD_FST_DBをアスキーフォーマット上で出力する。両面ポリゴン。
FST_FL	NJD_FST_FLをアスキーフォーマット上で出力する。フラットポリゴン。
FST_ENV	NJD_FST_ENVをアスキーフォーマット上で出力する。疑似環境マッピング。
UFO_0	NJD_UFO_0をアスキーフォーマット上で出力する。ユーザフラグ領域がないことを示す。
UFO_1	NJD_UFO_1をアスキーフォーマット上で出力する。ユーザフラグ領域が2バイトx1あることを示す。
UFO_2	NJD_UFO_2をアスキーフォーマット上で出力する。ユーザフラグ領域が2バイトx2あることを示す。
UFO_3	NJD_UFO_3をアスキーフォーマット上で出力する。ユーザフラグ領域が2バイトx3あることを示す。

```

/*-----*/
/* Chunk Strip */
/*-----*/
/** Flag STrip **/
#define FST_IL   NJD_FST_IL
#define FST_IS   NJD_FST_IS
#define FST_IA   NJD_FST_IA
#define FST_UA   NJD_FST_UA
#define FST_DB   NJD_FST_DB
#define FST_FL   NJD_FST_FL
#define FST_ENV  NJD_FST_ENV

/** UserFlag Offset **/
#define UFO_0    NJD_UFO_0
#define UFO_1    NJD_UFO_1
#define UFO_2    NJD_UFO_2
#define UFO_3    NJD_UFO_3

```

StripL(_len)	左回りから始まるトライアングルストリップであることを示す。
StripR(_len)	右回りから始まるトライアングルストリップであることを示す。

```

/*-----*/
/* Strip */
/*-----*/
#define StripL( _len)      (_len)
#define StripR( _len)      (_len) * -1

```

Uf1(_ufo1)	ユーザフラグ領域を 2 バイト x1 出力する。
Uf2(_ufo1, _ufo2)	ユーザフラグ領域を 2 バイト x2 出力する。
Uf3(_ufo1, _ufo2, _ufo3)	ユーザフラグ領域を 2 バイト x3 出力する。

```

/*-----*/
/* Chunk */
/*-----*/
/* Chunk */

/* UserFlag Offset and Number of Strips */
#define _NB(_ufo, _ns)      (_ufo)|(_ns) /* UserFlag Offset and nbStrip */
#define Uf1( _ufo1 )        (_ufo1)
#define Uf2( _ufo1, _ufo2 )  (_ufo1), (_ufo2)
#define Uf3( _ufo1, _ufo2, _ufo3 ) (_ufo1), (_ufo2), (_ufo3)

```

CnkS(_bits)	NJD_CSをアスキーフォーマットに出力する。テキストチャなし。
CnkS_UVN(_bits)	NJD_CS_UVNをアスキーフォーマットに出力する。0-255のUV値出力。

CnkS_UVH(_bits)	NJD_CS_UVHをアスキーフォーマットに出力する。0-1023のUV値出力。
CnkS_2(_bits)	NJD_CS_2をアスキーフォーマットに出力する。
CnkS_UVN2(_bits)	NJD_CS_UVN2をアスキーフォーマットに出力する。0-255のUV値出力。2 パラモディファイア用と合わせてUV値を 2 セット持つ。
CnkS_UVH2(_bits)	NJD_CS_UVH2をアスキーフォーマットに出力する。0-1023のUV値出力。 2 パラモディファイア用と合わせてUV値を 2 セット持つ。

```

/* Chunk Strip */

#define CnkS(_bits)      (_bits)|NJD_CS
#define CnkS_UVN(_bits)  (_bits)|NJD_CS_UVN
#define CnkS_UVH(_bits)  (_bits)|NJD_CS_UVH

/* Chunk Strip for 2 parameter modifier data */

#define CnkS_2(_bits)      (_bits)|NJD_CS_2
#define CnkS_UVN2(_bits)  (_bits)|NJD_CS_UVN2
#define CnkS_UVH2(_bits)  (_bits)|NJD_CS_UVH2

```

10.2.16 NJS_CNK_MODEL 構造体マクロ

Center	モデル外接球の中心を与える。
Radius	モデル外接球の半径を与える。
CNKMODEL	NJS_CNK_MODEL 構造体データを示す。
CNKModel	NJS_CNK_MODEL 構造体データポインタを示す。
VList	頂点リストデータポインタを示す。
PList	ポリゴンリストデータポインタを示す。
VLIST	頂点リストデータを示す。
PLIST	ポリゴンリストデータを示す。

```

/*-----*/

/* Model */

/*-----*/

#define Center

#define Radius

#define CNKMODEL      NJS_CNK_MODEL
#define CNKModel

#define VList
#define PList
#define VLIST          Sint32
#define PLIST          Sint16

```

10.2.17 NJS_CNK_OBJECT 構造体マクロ

CNKOBJECT	NJS_CNK_OBJECT 構造体データを示す。
-----------	---------------------------

CNKOBJECT_START	オブジェクトアスキーチャンクの始まりを示す。
CNKOBJECT_END	オブジェクトアスキーチャンクの終わりを示す。
EvalFlags(_ef)	EvalFlag を出力する。
Child	child の NJS_CNK_OBJECT 構造体ポインタを示す。
Sibling	sibling の NJS_CNK_OBJECT 構造体ポインタを示す。
OPosition(_x, _y, _z)	移動成分を出力する。
OAangle(_x, _y, _z)	回転成分を出力する。
OScale(_x, _y, _z)	スケール成分を出力する。
OQuatlm(_x, _y, _z)	クォータニオン軸成分を出力する。
OQuatre(_c)	クォータニオン回転成分を出力する。

```

/*-----*/
/* Object */
/*-----*/

#define CNKOBJECT          NJS_CNK_OBJECT

#define CNKOBJECT_START

#define CNKOBJECT_END

#define EvalFlags( _ef)      (_ef)
#define Child                _OBJPOINTERTYPE
#define Sibling              _OBJPOINTERTYPE

#define OPosition(_x, _y, _z) (_x), (_y), (_z)
#define OAangle(_x, _y, _z)   _NJANG(_x, _y, _z)
#define OScale(_x, _y, _z)    (_x), (_y), (_z)

#define OQuatlm(_x, _y, _z)   (_x), (_y), (_z)
#define OQuatre(_c)          (_c)

```

10.3 モデルアスキーフォーマット出力例

ここでは主なアスキーフォーマットの例を説明する。長くなるためデータの一部を省略し示す。

10.3.1 エンベロープ nja ファイルの出力例

エンベロープモデルの描画のトレース順番は一番下のNJS_CNK_OBJECT構造体から上方向であることに注意。

```

/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */ ヘッダ情報。

/* ROOT OBJECT : object_same_heavywalk_body_3_body_3 n(53) d(14) v(268) */ objectデータ情報。
/* TEXTLIST      : texlist_same_heavywalk_body_3 n(9) */ texlistデータ情報。

```

本データにはOBJECT、TEXTLISTデータが格納されることとその変数名を示す。モデル数53、PushPopの最大の深さ14、頂点バッ

ファの最大インデックス268、texlistのエントリ数9。

TEXTURE_START texlistアスキーチャンク開始。

TEXTURENAME textures_same_heavywalk_body_3[]

START

TEXN("SAME_BETA1"), /* 0 32x32 565 TW_D MIP */ GlobalIndex0, テクスチャサイズ32x32, RGB565, Twi
ddled dma 形式、ミップマップあり。

TEXN("sameMARK"), /* 1 128x128 565 TW_D MIP */

TEXN("same_eye1"), /* 2 64x64 565 TW_D MIP */

TEXN("same_eye2"), /* 3 64x64 565 TW_D MIP */

TEXN("sameha1"), /* 4 128x128 4444 TW_D MIP */

TEXN("sameha3"), /* 5 128x128 4444 TW_D MIP */

TEXN("samehada"), /* 6 128x128 565 TW_D MIP */

TEXN("samehada2"), /* 7 128x128 565 TW_D MIP */

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

END

TEXTURELIST texlist_same_heavywalk_body_3 texlistデータ。

START

TextureList textures_same_heavywalk_body_3, テクスチャデータ配列先頭ポインタ。

TextureNum 9, エントリ数9。

END

TEXTURE_END texlistアスキーチャンク終了。

CNKOBJECT_START objectアスキーチャンク開始。

CNKOBJECT object_same_heavywalk_body_3_item2[]

START

EvalFlags (FEV_HD|FEV_BR), EvalflagsにHide, Breakが設定される。

CNKModel NULL,

OPosition (-4.515593F, 3.810135F, 0.000031F), 位置情報。

OAngle (87.491554F, -90.027992F, -62.573139F), 回転情報。

OScale (1.200000F, 1.200000F, 1.200000F), スケール。

Child NULL,

Sibling NULL,

OQuatRe (0.000000), クォータニオンの場合に利用する。第4パラメータ。

END

CNKOBJECT_END objectアスキーチャンク終了。

```

CNKOBJECT_START  objectアスキーチャンク開始

PLIST      strip_same_heavywalk_body_3_footr5_1[] ポリゴンリスト
START

    CnkM_DA( FBS_SA|FBD_ISA ), 4, マテリアル設定。
    MDiff( 255, 253, 223, 0 ), Diffuse設定。
    MAmbi( 255, 253, 223, 0 ), Ambient設定。
    CnkS( FST_IS ), 160, _NB( UFO_0, 17 ), specular無視、テクスチャ無し、ユーザフラグ無し、ストリップ数17。
    StripL(3), 6, 7, 33, 左回り開始。長さ3。
    ...
    CnkEnd() データ終了。

END

VLIST      vertex_same_heavywalk_body_3_footr5_1[] 頂点リスト
START

    CnkV_VN(0x0, 337), 法線あり頂点チャンク (CV_VN)
    OffnbIdx(0, 56), オフセット0、頂点数56。
    VERT( 0x4011498c, 0x3e64a3c0, 0x3fb945ce ), /* 2.270114, 0.223281, 1.447443 */
    NORM( 0x3f070018, 0x3f0b4166, 0x3eb1ab0e ), /* 0.527345, 0.543967, 0.347008 */
    頂点、法線をヘキサでunsigned longの配列に出力。コンバータオプションで実値のコメントが付けられる。
    ...
    CnkEnd() データ終了。

END

CNKMODEL    model_same_heavywalk_body_3_footr5_1[]
START
VList      vertex_same_heavywalk_body_3_footr5_1, 頂点リスト
PList      strip_same_heavywalk_body_3_footr5_1, ポリゴンリスト
Center      0.821357F, -0.048197F, 0.271545F, モデルの中心点。
Radius      2.431669F, モデルの中心点からの外接半径。
END

CNKOBJECT    object_same_heavywalk_body_3_footr5_1[]
START
EvalFlags ( FEV_UT|FEV_BR|FEV_EN ), 移動成分なし、Break, Envelopeに関わるモデル。
CNKModel    model_same_heavywalk_body_3_footr5_1,
OPosition ( 0.000000F, -0.000065F, 0.000092F ),
OAngle ( 0.000003F, 0.000035F, 0.013214F ),
OScale ( 1.000000F, 0.869565F, 0.769231F ),
Child      NULL,
Sibling     NULL,
OQuatRe ( 0.000000 ),

```

END

CNKOBJECT_END Objectアスキーチャンク終了。

CNKOBJECT_START

CNKOBJECT object_same_heavywalk_body_3_item4[]

START

EvalFlags (FEV_HD), NULLモデルには必ずHideが設定される。

CNKModel NULL,

OPosition (-3.018911F, 0.000720F, 0.000001F),

OAngle (85.338722F, -90.000015F, 85.338722F),

OScale (1.200000F, 1.380000F, 1.560000F),

Child object_same_heavywalk_body_3_footr5_1,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_same_heavywalk_body_3_foot_eff_l[]

START

EvalFlags (FEV_UR|FEV_US|FEV_HD), 回転成分無し、スケール成分無し、NULLモデルであるのでHide。

CNKModel NULL,

OPosition (3.020419F, 0.000000F, 0.000000F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_same_heavywalk_body_3_item4,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

...

CNKOBJECT_START

PLIST strip_same_heavywalk_body_3_kosi[]

START

CnkB_DP(0), /* body_3 */ テーブルのID 0 番に登録されたPLISTを描画する。

CnkEnd()

END

VLIST vertex_same_heavywalk_body_3_kosi[]

START

CnkV_VN(FV_CONT, 7), FV_CONTにより頂点バッファをクリアしないことを示す。

OffnbIdx(96, 1), 頂点バッファへの書き込みのオフセットを96に設定。

VERT(0x40b52c18, 0xbea8f5fe, 0x3bf782fa), /* 5.661633, -0.330002, 0.007553 */

NORM(0x3f7ffff6, 0x3a8dcc40, 0xb8901f10), /* 0.999999, 0.001082, -0.000069 */

ウェイト100%の場合はウェイト値を省略し従来の頂点として処理される。この場合該当する頂点は1つだけあった。

CnkV_VN_NF(FV_CONT|FW_END, 253), エンベロープEndWeightグループ。頂点バッファをクリアしない。

OffnbIdx(0, 36),

VERT(0x40ac2a71, 0xbea7c048, 0x3f9073dd), /* 5.380181, -0.327639, 1.128536 */

NORM(0x3f6b3ac5, 0x3b2f5890, 0x3eca0400), /* 0.918866, 0.002676, 0.394562 */

NFlagsW2(156, 99.793091),

...

CnkEnd()

END

CNKMODEL model_same_heavywalk_body_3_kosi[]

START

VList vertex_same_heavywalk_body_3_kosi,

PList strip_same_heavywalk_body_3_kosi,

Center 0.000000F, 0.000000F, 0.000000F,

Radius 0.000000F, エンベロープに関わるモデルはRadiusを0にする。

END

CNKOBJECT object_same_heavywalk_body_3_kosi[]

START

EvalFlags (FEV_UT|FEV_US|FEV_EN), 移動成分無し、スケール成分無し、エンベロープに関わるモデル。

CNKModel model_same_heavywalk_body_3_kosi,

OPosition (0.000000F, 0.000000F, 0.000000F),

OAngle (0.001969F, 0.012202F, -80.812241F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_same_heavywalk_body_3_kosi_eff,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_same_heavywalk_body_3_c_kosi[]

START

EvalFlags (FEV_UT|FEV_US|FEV_HD), 移動成分無し、スケール成分無し、NULLモデルであるのでHideが設定される。

CNKModel NULL,

OPosition (0.000000F, 0.000000F, 0.000000F),

OAngle (90.014565F, 90.019798F, 90.014565F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_same_heavywalk_body_3_kosi,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_same_heavywalk_body_3_item1[]

START

EvalFlags (FEV_HD|FEV_BR), NULLモデルでChildを持たないのでHideとBreakが設定される。

CNKModel NULL,

OPosition (0.000062F, -2.833841F, -4.551531F),

OAngle (-71.167282F, -0.000001F, 0.000001F),

OScale (1.200000F, 1.200000F, 1.200000F),

Child NULL,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

PLIST strip_same_heavywalk_body_3_mabuta_l[]

START

CnkM_DA(FBS_SA|FBD_ISA), 4, マテリアル設定。

MDiff(255, 255, 255, 255), Diffuseを設定。

MAmbi(255, 255, 255, 255), Ambientを設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_BF, 0), UVクランプ、D adjust 1.00、 バイリニアフィルタ、Tex
Id0を設定。

CnkS_UVN(FST_IS), 341, _NB(UFO_0, 16), specular無視, ユーザフラグ無し、ストリップ数16。

StripR(5),

228, Uvn(229, 43), 8ビット(256)精度でUV値を出力。

230, Uvn(174, 6),

234, Uvn(159, 0),

231, Uvn(103, 16),

235, Uvn(79, 6),

...

CnkEnd()

END

VLIST vertex_same_heavywalk_body_3_mabuta_l[]

START

CnkV_VN(FV_CONT, 241), 頂点バッファをクリアしない。

OffnbIdx(228, 40), 頂点バッファ上のオフセット228、頂点数40。

VERT(0x3f91e266, 0x3f7b1528, 0xbe7e4620), /* 1.139722, 0.980792, -0.248314 */

NORM(0x3f3d5a1a, 0x3f244358, 0xbe4fd38c), /* 0.739656, 0.641653, -0.202955 */

...

CnkEnd()

END

CNKMODEL model_same_heavywalk_body_3_mabuta_l[]

START

VList vertex_same_heavywalk_body_3_mabuta_l,

PList strip_same_heavywalk_body_3_mabuta_l,

Center -0.025927F, -0.000637F, 0.225470F,

Radius 1.745086F,

END

...

CNKOBJECT_START

VLIST vertex_same_heavywalk_body_3_jnt22_2_1[]

START

CnkV_VN(FV_CONT, 55), 頂点バッファをクリアしない。100%ウェイト頂点群。

OffnbIdx(0, 9),

VERT(0x408fb8de, 0x3e8b3ddb, 0x3f037454), /* 4.491317, 0.271956, 0.513494 */

NORM(0x3f014b77, 0x3f2e8558, 0x3f078104), /* 0.505058, 0.681722, 0.529312 */

...

CnkV_VN_NF(FV_CONT|FW_END, 169), 頂点バッファをクリアしない。EndWeightグループ。
OffnbIdx(0, 24), 頂点バッファ上のオフセット0、頂点数24。
VERT(0x3cb3ad00, 0x3effb6c7, 0xbf8b71c1), /* 0.021933, 0.499441, -1.089409 */
NORM(0x3d18857f, 0x3f44cfc7, 0xbf236ed2), /* 0.037237, 0.768795, -0.638410 */
NFlagsW2(108, 50.016739),

...

CnkEnd()

END

CNKMODEL model_same_heavywalk_body_3_jnt22_2_1[]

START

VList vertex_same_heavywalk_body_3_jnt22_2_1,

PList NULL,

Center 0.000000F, 0.000000F, 0.000000F,

Radius 0.000000F,

END

CNKOBJECT object_same_heavywalk_body_3_jnt22_2_1[]

START

EvalFlags (FEV_EN), エンベロープに関わるモデル。

CNKModel model_same_heavywalk_body_3_jnt22_2_1,

OPosition (3.187911F, 0.000000F, 0.000000F),

OAngle (0.000000F, 21.839224F, 7.829946F),

OScale (1.000000F, 1.200000F, 1.000000F),

Child object_same_heavywalk_body_3_eff22_1,

Sibling object_same_heavywalk_body_3_bound3_1,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

VLIST vertex_same_heavywalk_body_3_jnt22_1_1[]

START

CnkV_VN_NF(FV_CONT|FW_START, 169), 頂点バッファをクリアしない。StartWeightグループ。
OffnbIdx(108, 24), 頂点バッファ上のオフセット108、頂点数24。
VERT(0x404d6e14, 0x3effb6c6, 0xbf8b71c2), /* 3.209844, 0.499441, -1.089409 */
NORM(0x3d18857f, 0x3f44cfc7, 0xbf236ed2), /* 0.037237, 0.768795, -0.638410 */
NFlagsW2(0, 49.983261),

```

...
    CnkEnd()
END

CNKMODEL    model_same_heavywalk_body_3_jnt22_1_1[]
START
VList      vertex_same_heavywalk_body_3_jnt22_1_1,
PList      NULL,
Center      0.000000F, 0.000000F, 0.000000F,
Radius      0.000000F,
END

CNKOBJECT    object_same_heavywalk_body_3_jnt22_1_1[]
START
EvalFlags ( FEV_UT|FEV_US|FEV_EN ), 移動成分無し、スケール成分無し、エンベロープに関わるモデル。
CNKModel    model_same_heavywalk_body_3_jnt22_1_1,
OPosition   ( 0.000000F, 0.000000F, 0.000000F ),
OAngle      ( -180.000031F, -203.736557F, 25.930191F ),
OScale      ( 1.000000F, 1.000000F, 1.000000F ),
Child       object_same_heavywalk_body_3_jnt22_2_1,
Sibling     object_same_heavywalk_body_3_null119,
OQuatRe     ( 0.000000 ),
END

CNKOBJECT_END

...

CNKOBJECT_START

PLIST      strip_same_heavywalk_body_3_body_3[]
START
    CnkB_CP( 0 ), /* body_3 */ エンベロープに関わる頂点計算のすべてが現時点で完了していないので描画を保留し
    テーブルにID番号0で登録する。
    CnkM_DA( FBS_SA|FBD_ISA ), 4, マテリアルを設定する。
    MDiff( 255, 255, 255, 255 ), Diffuseを設定。
    MAmbi( 255, 255, 255, 255 ), Ambientを設定。
    CnkT_TID( FCL_U|FCL_V|FDA_100 ), _TID( FFM_BF, 0 ), UVクランプ、D adjust 1.00、バイリニアフィルタ、TexId0
    を設定。
    CnkS_UVN( FST_IS|FST_UA ), 532, _NB( UFO_0, 21 ), ユーザフラグ無し、ストリップ数21。
    StripL(3), 左回り開始、長さ3。

```

```

2,      Uvn( 256, 167 ),   8 ビット(256)精度でUV値を出力。
3,      Uvn( 252, 167 ),
4,      Uvn( 252, 164 ),
...
    CnkEnd()
END

CNKMODEL   model_same_heavywalk_body_3_body_3[]
START
VList      NULL,
PList      strip_same_heavywalk_body_3_body_3,
Center      0.000000F,  3.312187F,  0.537337F,
Radius      0.000000F, エンベロープに関わるモデルはradiusを0にする。
END

CNKOBJECT  object_same_heavywalk_body_3_body_3[]
START
EvalFlags ( FEV_US|FEV_EN ), スケール成分無し、エンベロープに関わるモデル。
CNKModel   model_same_heavywalk_body_3_body_3,
OPosition  ( 0.000000F, -0.978237F,  0.058765F ),
OAngle     ( -3.437747F,  0.000000F,  0.000000F ),
OScale     ( 1.000000F,  1.000000F,  1.000000F ),
Child      object_same_heavywalk_body_3_null,
Sibling     NULL,
OQuatRe    ( 0.000000 ),
END

CNKOBJECT_END

DEFAULT_START

#ifdef DEFAULT_OBJECT_NAME
#define DEFAULT_OBJECT_NAME object_same_heavywalk_body_3_body_3
#endif
#ifdef DEFAULT_TEXLIST_NAME
#define DEFAULT_TEXLIST_NAME texlist_same_heavywalk_body_3
#endif

```

本データが持つデータはOBJECTとTEXLISTでありコンパイルして実行する場合defineされた固定の名前DEFAULT_OBJECT_NAME、DEFAULT_TEXLIST_NAMEで変数が利用できる。複数のデータファイルがインクルードされる場合先頭にインクルードされたデータが固定の名前に設定される。

DEFAULT_END

10.3.2 テクスチャ 2 枚貼り nja ファイルの出力例

テクスチャ 2 枚を同時に 1 枚のポリゴンに貼る機能をハードウェアはもたない。そのためこれを実現するには一枚目のテクスチャでポリゴンを描画した上に半透明の二つ目のテクスチャを貼ったポリゴンを重ねる必要がある。

コンバータはモデラーデータで設定された 2 枚のテクスチャを検出しポリゴンの 2 重出力を自動で行う。

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */ ヘッダ情報。
```

```
/* ROOT OBJECT : object_2maihari_cube7_cube7 n(1) d(1) v(8) */ objectデータ情報。
```

```
/* TEXTLIST      : texlist_2maihari_cube7 n(2) */ texlistデータ情報。
```

本データにはOBJECT、TEXTLISTデータが格納されることとその変数名を示す。モデル数1、PushPopの最大の深さ1、頂点バッファの最大インデックス8、texlistのエントリ数2。

TEXTURE_START texlistアスキーチャンク開始。

```
TEXTURENAME textures_2maihari_cube7[]
```

```
START
```

```
    TEXN( "f40_number" ),      /* 0 64x64 565 TW_D MIP */ GlobalIndex 0, テクスチャサイズ64x64, RGB565, Twi  
ddled dma形式、ミップマップあり。
```

```
    TEXN( "vip_emblem" ),      /* 1 64x64 565 TW_D MIP */
```

```
END
```

2 枚貼りに利用される 2 つのテクスチャを格納。

```
TEXTURELIST texlist_2maihari_cube7 texlistデータ。
```

```
START
```

```
TextureList textures_2maihari_cube7,
```

```
TextureNum 2,
```

```
END
```

TEXTURE_END texlistアスキーチャンク終了。

```
CNKOBJECT_START
```

```
PLIST      strip_2maihari_cube7_cube7[] ポリゴンリスト
```

```
START
```

```
    CnkM_DAS( FBS_SA|FBD_ISA ), 6, マテリアル設定。
```

```

MDiff( 255, 178, 178, 178 ),    Diffuse設定。
MAmbi( 255, 127, 127, 127 ),    Ambient設定。
MSpec( 11, 255, 255, 255 ),    Specular設定。第一引数の11はexponent値。Ninja2では無効。
CnkS( 0x0 ), 17, _NB( UFO_0, 2 ), テクスチャ無し、ユーザフラグ無し、ストリップ数 2。
StripR(10), 7, 4, 5, 0, 1, 2, 3, 6, 7, 4,
StripL(4), 4, 0, 6, 2,
CnkT_TID( FCL_U|FCL_V|FDA_100 ), _TID( FFM_BF, 0 ), テクスチャ 2 枚貼り開始。UVクランプ、D adjust 1.00,
バイリニアフィルタ、texId 0。
CnkS_UVN( 0x0 ), 14, _NB( UFO_0, 1 ), ユーザフラグ無し、ストリップ数 1。
StripL(4),                                左回り開始、長さ 4。texId 0で一度目の描画。
1,      Uvn( 0, 256 ),                    8 ビット(256)精度でUV値を出力。
5,      Uvn( 256, 256 ),
3,      Uvn( 0, 0 ),
7,      Uvn( 256, 0 ),
CnkM_D( FBS_SA|FBD_ISA ), 2,              マテリアル設定。( 値に変更のあった ) Diffuseのみ。
MDiff( 113, 178, 178, 178 ),              Diffuse設定。第一引数の113が半透明(113/255)を意味する。
CnkT_TID( FCL_U|FCL_V|FDA_100 ), _TID( FFM_BF, 1 ), UVクランプ、D adjust 1.00, バイリニアフィルタ、texI
d 1。
CnkS_UVN( FST_UA ), 14, _NB( UFO_0, 1 ), 半透明、ユーザフラグ無し、ストリップ数 1。
StripL(4),                                左回り、長さ 4。texId 1で二度目の描画。
1,      Uvn( 0, 256 ),
5,      Uvn( 256, 256 ),
3,      Uvn( 0, 0 ),
7,      Uvn( 256, 0 ),
CnkEnd()
END

VLIST      vertex_2mai hari_cube7_cube7[]
START
CnkV_VN(0x0, 49),
OffnbIdx(0, 8),
VERT( 0xc0a00000, 0xc0a00000, 0xc0a00000 ),
NORM( 0xbf13cd3a, 0xbf13cd3a, 0xbf13cd3a ),
...
CnkEnd()
END

CNKMODEL   model_2mai hari_cube7_cube7[]
START
VList      vertex_2mai hari_cube7_cube7,
PList      strip_2mai hari_cube7_cube7,

```



```
Center      0.000000F,  0.000000F,  0.000000F,
Radius      8.660254F,
END
```

```
CNKOBJECT  object_2maihari_cube7_cube7[]
START
EvalFlags ( FEV_UT|FEV_UR|FEV_US|FEV_BR|FEV_EN ),
CNKModel   model_2maihari_cube7_cube7,
OPosition  ( 0.000000F,  0.000000F,  0.000000F ),
OAngle     ( 0.000000F,  0.000000F,  0.000000F ),
OScale     ( 1.000000F,  1.000000F,  1.000000F ),
Child      NULL,
Sibling    NULL,
OQuatRe    ( 0.000000 ),
END
```

```
CNKOBJECT_END
```

```
DEFAULT_START
```

```
...
```

```
DEFAULT_END
```

10.3.3 トライリニアフィルタ nja ファイルの出力例

トライリニアフィルタはアキュムレーションバッファを利用し2段のミップマップテクスチャをブレンドしその結果をポリゴンに貼る。そのため非半透明の場合2パス(2度書き)、半透明の3パス(3度書き)の動作が必要となる。Ninja2ではこれをデータ上に表現する。つまりデータを2パスなら2度、3パスなら3度列記することでトライリニアフィルタの結果を得る。トライリニアフィルタを使いテクスチャの2枚貼りをするためには5パス必要となる。トライリニアフィルタは処理が重くデータ全体にかけるのは不向きである。ワンポイントでの使用を推奨。

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */ ヘッダ情報。
```

```
/* ROOT OBJECT : object_cube_cube2_cube2 n(1) d(1) v(8) */ objectデータ情報。
```

```
/* TEXTLIST      : texlist_cube_cube2 n(4) */ texlistデータ情報。
```

本データにはOBJECT、TEXTLISTデータが格納されることとその変数名を示す。モデル数1、PushPopの最大の深さ1、頂点バッファの最大インデックス8、texlistのエントリ数4。

```
TEXTURE_START texlistアスキーチャンク開始。
```

```
TEXTURENAME textures_cube_cube2[]
```

START

TEXN("f40_number"), /* 0 64x64 565 TW_D MIP */ GlobalIndex 0, テクスチャサイズ64x64, RGB565, twiddled dma 形式、ミップマップあり。

TEXN("f4e_wheel"), /* 1 128x128 565 TW_D MIP */

TEXN("fer_emblem_sy"), /* 2 64x64 4444 TW_D MIP */

TEXN("ref_sky_ts4"), /* 3 128x128 565 TW_D MIP */

END

TEXTURELIST texlist_cube_cube2 texlistデータ。

START

TextureList textures_cube_cube2, テクスチャデータ配列先頭ポインタ。

TextureNum 4, エントリ数4。

END

TEXTURE_END texlistアスキーチャンク終了。

CNKOBJECT_START

PLIST strip_cube_cube2_cube2[] ポリゴンリスト

START

トライリニアフィルタ非半透明ポリゴン例 =====

CnkM_DAS(FBS_ONE|FBD_ZER), 6, 非半透明の場合のトライリニアフィルタAパス処理開始。トライリニア用にBlending AlphaにNJD_FBS_ONE, NJD_FBD_ZERを設定。

MDiff(255, 178, 178, 178), Diffuseを設定。

MAmbi(255, 127, 127, 127), Ambientを設定。

MSpec(11, 255, 255, 255), Specularを設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFA, 0), UVクロップ、D adjust 1.00, トライリニアフィルタAパス、texId 0。

/* opaque A */

CnkS_UVN(0x0), 53, _NB(UFO_0, 4), ユーザフラグ無し、ストリップ数4。

StripL(6), 左回り開始、長さ6。

3, Uvn(0, 0),

1, Uvn(0, 256),

5, Uvn(256, 256),

0, Uvn(0, 256),

4, Uvn(256, 256),

6, Uvn(256, 0),

StripL(3), 左回り開始、長さ3。

5, Uvn(256, 256),

7, Uvn(256, 0),

3, Uvn(0, 0),

StripL(3), 左回り開始、長さ 3。

0, Uvn(0, 256),
2, Uvn(0, 0),
6, Uvn(256, 0),

StripL(4), 左回り開始、長さ 4。

2, Uvn(0, 256),
3, Uvn(0, 256),
6, Uvn(256, 256),
7, Uvn(256, 256),

CnkB_BA(FBS_ONE|FBD_ONE), /* opaque B */ 非半透明の場合のトライリニアフィルタ B パス処理開始。トライリニア用に Blending Alpha の NJD_FBS_ONE, NJD_FBD_ONE を設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFB, 0), UV クロップ、D adjust 1.00, トライリニアフィルタ B パス、texId 0。

CnkS_UVN(FST_UA), 53, _NB(UFO_0, 4), 半透明、ユーザフラグ無し、ストリップ数 4。

StripL(6), 左回り開始、長さ 6。A パスで描画したデータの再描画。

3, Uvn(0, 0),
1, Uvn(0, 256),
5, Uvn(256, 256),
0, Uvn(0, 256),
4, Uvn(256, 256),
6, Uvn(256, 0),

StripL(3),

5, Uvn(256, 256),
7, Uvn(256, 0),
3, Uvn(0, 0),

StripL(3),

0, Uvn(0, 256),
2, Uvn(0, 0),
6, Uvn(256, 0),

StripL(4),

2, Uvn(0, 256),
3, Uvn(0, 256),
6, Uvn(256, 256),
7, Uvn(256, 256),

トライリニアフィルタ透明ポリゴン例 =====

CnkM_D(FBS_ONE|FBD_ZER|FBD_SEL), 2, 半透明の場合のトライリニアフィルタ A パス処理開始。トライリニアフィルタ用に Blending Alpha の NJD_FBS_ONE, NJD_FBD_ZER 及びアキュムレーションバッファ用の NJD_FBD_SEL (書き込み先をアキュムレーションバッファに指定) を設定。USE_ALPHA(NJD_FST_UA) は常にオンにする必要がある。

MDiff(160, 100, 197, 54), 値に変更のあった Diffuse のみ設定。第一引数の 160 が半透明(160/255)を意味する。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFA, 3), UV クランプ、D adjust 1.00、トライリニアフィルタ A パス設定、texId 3。

```
/* translucent A */
```

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数1。アキュムレーションバッファへの書き込み開始。

```
StripL(4), 左回り開始、長さ4。
```

```
0, Uvn( 0, 256 ),
1, Uvn( 256, 256 ),
2, Uvn( 0, 0 ),
3, Uvn( 256, 0 ),
```

CnkB_BA(FBS_ONE|FBD_ONE|FBD_SEL), /* translucent B */ 半透明の場合のトライリニアフィルタBパスの処理開始。トライリニアフィルタ用にBlending AlphaのNJD_FBS_ONE, NJD_FBD_ONE及びアキュムレーションバッファ用NJD_FBD_SEL (書き込み先をアキュムレーションバッファに指定)を設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFB, 3), UVクランプ、D adjust 1.00、トライリニアフィルタBパス設定、texId 3。

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数1。アキュムレーションバッファへの書き込み開始。

```
StripL(4), 左回り開始、長さ4。Aパスで描画したデータの再描画。
```

```
0, Uvn( 0, 256 ),
1, Uvn( 256, 256 ),
2, Uvn( 0, 0 ),
3, Uvn( 256, 0 ),
```

CnkB_BA(FBS_SA|FBD_ISA|FBS_SEL), /* translucent end */ アキュムレーションバッファに格納されるデータの書き込み開始。モデラーから指定されたBlending Alpha値NJD_FBS_SA, NJD_FBD_ISA及びアキュムレーションバッファ用NJD_FBS_SEL (ソース元をアキュムレーションバッファに指定)を設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFB, 3), UVクランプ、D adjust 1.00、トライリニアフィルタBパス、texId 3。

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数1。アキュムレーションバッファデータのフレームバッファへの書き込み開始。

```
StripL(4), アキュムレーションバッファ内のデータの書き出し。
```

```
0, Uvn( 0, 256 ),
1, Uvn( 256, 256 ),
2, Uvn( 0, 0 ),
3, Uvn( 256, 0 ),
```

トライリニアフィルタテクスチャ二枚貼例 =====

CnkM_D(FBS_ONE|FBD_ZER), 2, 非半透明の場合のトライリニアフィルタAパス処理開始。トライリニアフィルタ用にBlending AlphaにNJD_FBS_ONE, NJD_FBD_ZERを設定。

MDiff(255, 21, 51, 236), 変更があったDiffuseのみの設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFA, 1), UVクロップ、D adjust 1.00、トライリニアフィルタAパス、texId 0。

```
/* opaque A */
```

CnkS_UVN(0x0), 14, _NB(UFO_0, 1), ユーザフラグ無し、ストリップ数1。

```
StripL(4), 左回り開始、長さ4。一枚目テクスチャでAパス(二枚貼り1回目描画)
```

```

5,      Uvn( 256, 256 ),
4,      Uvn( 0, 256 ),
7,      Uvn( 256, 0 ),
6,      Uvn( 0, 0 ),

```

CnkB_BA(FBS_ONE|FBD_ONE), /* opaque B */ 非半透明の場合のトライリニアフィルタ B パス処理開始。トライリニアフィルタ用にBlending AlphaのNJD_FBS_ONE, NJD_FBD_ONEを設定。B パスにはUSE ALPHA (NJD_FST_UA) の設定が必要。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFB, 1), UVクロップ、D adjust 1.00、トライリニアフィルタ B パス、texId 1。

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数 1。

StripL(4), 左回り開始、長さ 4。1 枚目テクスチャで B パス (二枚貼り 2 回目描画)

```

5,      Uvn( 256, 256 ),
4,      Uvn( 0, 256 ),
7,      Uvn( 256, 0 ),
6,      Uvn( 0, 0 ),

```

CnkM_D(FBS_ONE|FBD_ZER|FBD_SEL), 2, 2 枚目テクスチャによる半透明のトライリニアフィルタ A パス処理開始。トライリニアフィルタ用にBlending AlphaのNJD_FBS_ONE, NJD_FBD_ZER及びアキュムレーションバッファ用のNJD_FBD_SEL (書き込み先をアキュムレーションバッファに指定)を設定。

MDiff(129, 21, 51, 236), 値に変更のあったDiffuse値のみ設定。第一引数129が半透明(129/255)を意味する。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFA, 2), UVクランプ、D adjust 1.00、トライリニアフィルタ A パス設定、texId 2。

/* translucent A */

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数 1。

StripL(4), 左回り開始、長さ 4。2 枚目テクスチャで A パス (二枚貼り 3 回目の描画)

```

5,      Uvn( 256, 256 ),
4,      Uvn( 0, 256 ),
7,      Uvn( 256, 0 ),
6,      Uvn( 0, 0 ),

```

CnkB_BA(FBS_ONE|FBD_ONE|FBD_SEL), /* translucent B */ 2 枚目テクスチャによる半透明トライリニアフィルタ B パス処理開始。トライリニアフィルタ用にBlending AlphaのNJD_FBS_ONE, NJD_FBS_ONE及びアキュムレーションバッファ用NJD_FBD_SEL (書き込み先をアキュムレーションバッファに指定)を設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFB, 2), UVクランプ、D adjust 1.00、トライリニアフィルタ B パス、texId 2。

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数 1。

StripL(4), 左回り開始、長さ 4。2 枚目テクスチャで B パス (二枚貼り 4 回目の描画)

```

5,      Uvn( 256, 256 ),
4,      Uvn( 0, 256 ),
7,      Uvn( 256, 0 ),
6,      Uvn( 0, 0 ),

```

CnkB_BA(FBS_SA|FBD_ISA|FBS_SEL), /* translucent end */ 2 枚目テクスチャによるアキュムレーションバッファに格納されるデータの書き込み開始。モデラーから指定されたBlending Alpha値NJD_FBS_SA, NJD_FBD_ISA及びアキュムレーションバッファ用NJD_FBS_SEL (ソース元をアキュムレーションバッファに指定)を設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_TFB, 2), UVクランプ、D adjust 1.00、トライリニアフィルタB
パス、texId 2。

CnkS_UVN(FST_UA), 14, _NB(UFO_0, 1), 半透明、ユーザフラグ無し、ストリップ数 1。アキュムレーション
バッファデータのフレームバッファへの書き込み開始。

StripL(4), 2 枚目テクスチャアキュムレーションバッファ内のデータの書き出し (二枚貼り 5 回目の描画)
5, Uvn(256, 256),
4, Uvn(0, 256),
7, Uvn(256, 0),
6, Uvn(0, 0),

CnkB_BA(FBS_SA|FBD_ISA), /* reset src select */ アキュムレーションバッファに関わるNJD_FBS_SEL, NJ
D_FBD_SELのリセット。ハードの誤動作を防ぐためにトライリニアフィルタを使った場合PListの最後に必ずリセットが必要。

CnkEnd()

END

VLIST vertex_cube_cube2_cube2[] 頂点リスト

START

CnkV_VN(0x0, 49),
OffnbIdx(0, 8), 頂点バッファ上のオフセット 0、頂点数 8。
VERT(0xc0a00000, 0xc0a00000, 0xc0a00001),
NORM(0xbf13cd3a, 0xbf13cd3b, 0xbf13cd3b),
...
CnkEnd()

END

CNKMODEL model_cube_cube2_cube2[] モデルデータ。

START

VList vertex_cube_cube2_cube2,
PList strip_cube_cube2_cube2,
Center 0.000000F, 0.000000F, 0.000000F,
Radius 8.660254F,
END

CNKOBJECT object_cube_cube2_cube2[] オブジェクトデータ。

START

EvalFlags (FEV_UT|FEV_US|FEV_BR), 移動成分無し、スケール成分無し、Childを持たないのでBreak。

CNKModel model_cube_cube2_cube2,
OPosition (0.000000F, 0.000000F, 0.000000F),
OAngle (0.000000F, 497.900391F, 0.000000F),
OScale (1.000000F, 1.000000F, 1.000000F),
Child NULL,
Sibling NULL,

```
OQuatRe    ( 0.000000 ),
END
```

```
CNKOBJECT_END
```

```
DEFAULT_START
```

```
...
```

```
DEFAULT_END
```

10.3.4 モデル単位のインスタンス nja ファイルの出力例

一つのモデルをインスタンス元とし複数のインスタンスモデルを組み合わせる親子階層を構成した例。テクスチャ無し。

< 出力例モデル階層構造概要 >

0	0	"null3"	ルートNULLモデル。
1	1	"null1"	NULLモデル。
2	2	"cube1"	インスタンス元データ。
3	2	"inst1"	cube1インスタンスデータ。
4	3	"inst2"	cube1インスタンスデータ。
5	1	"null2"	NULLモデル。
6	2	"inst3"	cube1インスタンスデータ。
7	2	"inst4"	cube1インスタンスデータ。
8	3	"inst5"	cube1インスタンスデータ。
9	3	"inst6"	cube1インスタンスデータ。
10	4	"inst7"	cube1インスタンスデータ。

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */ ヘッダ情報。
```

```
/* ROOT OBJECT : object_null13_null13 n(11) d(5) v(8) */ objectデータ情報。
```

本データにはOBJECTが格納されることとその変数名を示す。モデル数11、PushPopの最大の深さ5、頂点バッファの最大インデックス8。

```
/* INSTANCE OBJECT COUNT 7 */ インスタンスモデルが7つ含まれることを示す。
```

```
INSTANCE_PROTOTYPE_START インスタンスアスキーチャンク開始。
```

```
extern CNKMODEL model_null13_cube1[]; 他のNJS_CNK_OBJECT構造体から参照できるように先頭でextern宣言をする。
```

```
INSTANCE_PROTOTYPE_END インスタンスアスキーチャンク終了。
```

CNKOBJECT_START オブジェクトアスキーチャンク開始。

CNKOBJECT object_null13_inst7[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */

インスタンスモデルのポインタを参照していることを示す。

OPosition (0.283688F, 0.170214F, -6.950356F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child NULL,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END オブジェクトアスキーチャンク終了。

CNKOBJECT_START

CNKOBJECT object_null13_inst6[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */

インスタンスモデルのポインタを参照していることを示す。

OPosition (6.127660F, 5.758864F, 2.780142F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_null13_inst7,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_null13_inst5[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */

インスタンスモデルのポインタを参照していることを示す。

OPosition (8.381907F, 14.388945F, -0.179350F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),


```

Child      NULL,
Sibling    object_null13_inst6,
OQuatRe    ( 0.000000 ),
END

```

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_null13_inst4[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */

インスタンスモデルのポインタを参照していることを示す。

OPosition (-6.411347F, -3.460993F, 0.000000F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_null13_inst5,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_null13_inst3[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */

インスタンスモデルのポインタを参照していることを示す。

OPosition (0.000000F, -5.702127F, -7.489362F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child NULL,

Sibling object_null13_inst4,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_null13_inst2[]

START

EvalFlags (FEV_UT|FEV_UR|FEV_US|FEV_HD),

CNKModel NULL,

OPosition (0.000000F, 0.000000F, 0.000000F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_null13_inst3,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_null13_inst2[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */ インスタンスモデルのポインタを参照していることを示す。

OPosition (-14.921986F, -1.900709F, 0.000000F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child NULL,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

CNKOBJECT_START

CNKOBJECT object_null13_inst1[] /* INSTANCE_OBJECT */ インスタンスであることを示す。

START

EvalFlags (FEV_UR|FEV_US|FEV_BR),

CNKModel model_null13_cube1, /* instance */ インスタンスモデルのポインタを参照していることを示す。

OPosition (4.170213F, 4.425532F, 0.000000F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child object_null13_inst2,

Sibling NULL,

```
OQuatRe    ( 0.000000 ),  
END
```

```
CNKOBJECT_END
```

```
CNKOBJECT_START
```

```
PLIST      strip_null13_cube1[]   ポリゴンリスト
```

```
START
```

```
    CnkM_DAS( FBS_SA|FBD_ISA ), 6,   マテリアルの設定。  
    MDiff( 255, 105, 122, 255 ),    diffuseの設定。  
    MAmbi( 255, 92, 63, 221 ),      ambientの設定。  
    MSpec( 11, 255, 255, 255 ),     specularの設定。  
    CnkS( 0x0 ), 22, _NB( UFO_0, 3 ), テクスチャ無し、ユーザフラグ無し、ストリップ数3。  
    StripL(10), 7, 4, 6, 0, 2, 1, 3, 5, 7, 4,   左回り開始、長さ10。  
    StripL(4), 1, 0, 5, 4,                      左回り開始、長さ4。  
    StripL(4), 7, 6, 3, 2,                      左回り開始、長さ4。  
    CnkEnd()
```

```
END
```

```
VLIST      vertex_null13_cube1[]   頂点リスト
```

```
START
```

```
    CnkV_VN(0x0, 49),  
    OffnbIdx(0, 8),  
    VERT( 0xbf000000, 0xbf000000, 0xbf000000 ),  
    NORM( 0xbf13cd3a, 0xbf13cd3a, 0xbf13cd3a ),  
    ...  
    CnkEnd()
```

```
END
```

```
CNKMODEL   model_null13_cube1[]
```

```
START
```

```
VList      vertex_null13_cube1,
```

```
PList      strip_null13_cube1,
```

```
Center      0.000000F, 0.000000F, 0.000000F,
```

```
Radius      0.866025F,
```

```
END
```

```
CNKOBJECT  object_null13_cube1[]   インスタンス元のオリジナルデータ。
```

```
START
```

```
EvalFlags ( FEV_UR|FEV_US|FEV_BR ),
```

```

CNKModel    model_null13_cube1, インスタンス元のオリジナルモデルポインタ。
OPosition   ( -3.943262F, -2.297872F,  0.000000F ),
OAngle      (  0.000000F,  0.000000F,  0.000000F ),
OScale      (  1.000000F,  1.000000F,  1.000000F ),
Child       NULL,
Sibling     object_null13_inst1,
OQuatRe     ( 0.000000 ),
END

```

CNKOBJECT_END

CNKOBJECT_START

```

CNKOBJECT   object_null13_null1[]
START
EvalFlags   ( FEV_UT|FEV_UR|FEV_US|FEV_HD ),
CNKModel    NULL,
OPosition   (  0.000000F,  0.000000F,  0.000000F ),
OAngle      (  0.000000F,  0.000000F,  0.000000F ),
OScale      (  1.000000F,  1.000000F,  1.000000F ),
Child       object_null13_cube1,
Sibling     object_null13_null2,
OQuatRe     ( 0.000000 ),
END

```

CNKOBJECT_END

CNKOBJECT_START

```

CNKOBJECT   object_null13_null3[]
START
EvalFlags   ( FEV_UT|FEV_UR|FEV_US|FEV_HD ),
CNKModel    NULL,
OPosition   (  0.000000F,  0.000000F,  0.000000F ),
OAngle      (  0.000000F,  0.000000F,  0.000000F ),
OScale      (  1.000000F,  1.000000F,  1.000000F ),
Child       object_null13_null1,
Sibling     NULL,
OQuatRe     ( 0.000000 ),
END

```

CNKOBJECT_END

DEFAULT_START

...

DEFAULT_END

10.3.5 モデル階層のインスタンス nja ファイルの出力例

モデル階層の一部を他のモデルのchildに設定する。複数のモデルを一括してインスタンスできる。Softimageのみで利用可能。
モデル階層をインスタンスした例。テクスチャ無し。

< 出力例モデル階層構造概要 >

0	0	"cube1"	モデルデータ。
1	1	"cube2"	モデルデータ。
2	2	"inst1"	sphere2をインスタンスとして持つ。sphere2のsiblingのsphere1もインスタンスされる。
3	1	"null1"	NULLモデルデータ。
4	2	"sphere2"	インスタンス元データのトップ。
5	2	"sphere1"	インスタンス元データの階層に含まれるデータ。

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */
```

```
/* ROOT OBJECT : object_nullmodel_cube1_cube1 n(6) d(3) v(58) */ objectデータ情報。
```

本データにはOBJECTデータが格納されることとその変数名を示す。モデル数3、PushPopの最大の深さ3、頂点バッファの最大インデックス58。

```
/* INSTANCE OBJECT COUNT 1 */ インスタンスオブジェクトが1つであることを示す。
```

```
INSTANCE_PROTOTYPE_START インスタンスアスキーチャンク開始。
```

extern CNKOBJECT object_nullmodel_cube1_sphere2[]; インスタンス用データを他のNJS_CNK_OBJECT構造体から参照できるように先頭でextern宣言をする。

```
INSTANCE_PROTOTYPE_END インスタンスアスキーチャンク終了。
```

```
CNKOBJECT_START オブジェクトアスキーチャンク開始。
```

```
PLIST      strip_nullmodel_cube1_sphere1[]
```

```
START
```

```
    CnkM_DA( FBS_SA|FBD_ISA ), 4, マテリアル設定。
```

```
    MDiff( 255, 255, 255, 255 ), diffuseの設定。
```

```

MAmbi( 0, 0, 0, 0 ),          ambientの設定。
CnkS( FST_IL|FST_IS ), 149, _NB( UFO_0, 12 ), テクスチャ無し、光源、specular無視、ストリップ数12。
StripR(12), 7, 13, 6, 12, 5, 11, 4, 10, 3, 9, 右回り開始、長さ12。
    2, 0,
    ...
CnkEnd()

```

END

```

VLIST      vertex_nullmodel_cube1_sphere1[] 頂点リスト
START
    CnkV_VN(0x0, 349),
    OffnbIdx(0, 58),
    VERT( 0x00000000, 0xc0a00000, 0x00000000 ),
    NORM( 0xb23e0fcd, 0xbf800000, 0xb10e8bda ),
    ...
CnkEnd()

```

END

```

CNKMODEL   model_nullmodel_cube1_sphere1[]
START
VList      vertex_nullmodel_cube1_sphere1,
PList      strip_nullmodel_cube1_sphere1,
Center     0.000000F, 0.000000F, 0.000000F,
Radius     5.000000F,
END

```

```

CNKOBJECT  object_nullmodel_cube1_sphere1[]
START
EvalFlags ( FEV_UR|FEV_US|FEV_BR ),
CNKModel   model_nullmodel_cube1_sphere1,
OPosition  ( 5.978683F, 0.000000F, 13.343729F ),
OAngle     ( 0.000000F, 0.000000F, 0.000000F ),
OScale     ( 1.000000F, 1.000000F, 1.000000F ),
Child      NULL,
Sibling     NULL,
OQuatRe    ( 0.000000 ),
END

```

CNKOBJECT_END オブジェクトアスキーチャンク終了。

CNKOBJECT_START

```

PLIST      strip_nullmodel_cube1_sphere2[]
START
    CnkM_DA( FBS_SA|FBD_ISA ), 4,
    MDiff( 255, 255, 255, 255 ),
    MAmbi( 0, 0, 0, 0 ),
    CnkS( FST_IL|FST_IS ), 149, _NB( UFO_0, 12 ),
    StripR(12), 7, 13, 6, 12, 5, 11, 4, 10, 3, 9,
        2, 0,
    ...
    CnkEnd()
END

```

```

VLIST      vertex_nullmodel_cube1_sphere2[]
START
    CnkV_VN(0x0, 349),
    OffnbIdx(0, 58),
    VERT( 0x00000000, 0xc0a00000, 0x00000000 ),
    NORM( 0xb23e0fcd, 0xbf800000, 0xb10e8bda ),
    ...
    CnkEnd()
END

```

```

CNKMODEL   model_nullmodel_cube1_sphere2[]
START
VList      vertex_nullmodel_cube1_sphere2,
PList      strip_nullmodel_cube1_sphere2,
Center      0.000000F, 0.000000F, 0.000000F,
Radius      5.000000F,
END

```

```

CNKOBJECT  object_nullmodel_cube1_sphere2[]
START
EvalFlags ( FEV_UR|FEV_US|FEV_BR ),
CNKModel   model_nullmodel_cube1_sphere2,
OPosition  ( 21.055365F, 0.000000F, 16.203100F ),
OAngle     ( 0.000000F, 0.000000F, 0.000000F ),
OScale     ( 1.000000F, 1.000000F, 1.000000F ),
Child      NULL,
Sibling     object_nullmodel_cube1_sphere1,
OQuatRe    ( 0.000000 ),

```

END

CNKOBJECT_END

CNKOBJECT_START

```
CNKOBJECT  object_nullmodel_cube1_null1[]
START
EvalFlags ( FEV_UT|FEV_UR|FEV_US|FEV_HD ),
CNKModel   NULL,
OPosition  ( 0.000000F, 0.000000F, 0.000000F ),
OAngle     ( 0.000000F, 0.000000F, 0.000000F ),
OScale     ( 1.000000F, 1.000000F, 1.000000F ),
Child      object_nullmodel_cube1_sphere2,
Sibling     NULL,
OQuatRe    ( 0.000000 ),
END
```

CNKOBJECT_END

CNKOBJECT_START

```
CNKOBJECT  object_nullmodel_cube1_inst1[] /* INSTANCE_OBJECT */ インスタンスオブジェクトであることを示す。
START
EvalFlags ( FEV_UR|FEV_US|FEV_HD ),
CNKModel   NULL,
OPosition  ( -12.943262F, 0.000000F, 11.039312F ),
OAngle     ( 0.000000F, 0.000000F, 0.000000F ),
OScale     ( 1.000000F, 1.000000F, 1.000000F ),
Child      object_nullmodel_cube1_sphere2, /* instance */ インスタンスオブジェクトポインタの参照。object_nullmo
del_cube1_sphere2のChild, Siblingが合わせてインスタンスの対象となる。
Sibling     NULL,
OQuatRe    ( 0.000000 ),
END
```

CNKOBJECT_END

CNKOBJECT_START

```
PLIST      strip_nullmodel_cube1_cube2[]
START
```



```

CnkM_DA( FBS_SA|FBD_ISA ), 4,
MDiff( 255, 255, 255, 255 ),
MAmbi( 0, 0, 0, 0 ),
CnkS( FST_IL|FST_IS ), 22, _NB( UFO_0, 3 ),
StripL(10), 7, 4, 6, 0, 2, 1, 3, 5, 7, 4,
...
CnkEnd()

```

END

```

VLIST      vertex_nullmodel_cube1_cube2[]
START
CnkV_VN(0x0, 49),
OffnbIdx(0, 8),
VERT( 0xc0a00000, 0xc0a00000, 0xc0a00000 ),
NORM( 0xbf13cd3a, 0xbf13cd3a, 0xbf13cd3a ),
...
CnkEnd()

```

END

```

CNKMODEL   model_nullmodel_cube1_cube2[]
START
VList      vertex_nullmodel_cube1_cube2,
PList      strip_nullmodel_cube1_cube2,
Center      0.000000F, 0.000000F, 0.000000F,
Radius      8.660254F,
END

```

```

CNKOBJECT  object_nullmodel_cube1_cube2[]
START
EvalFlags ( FEV_UR|FEV_US ),
CNKModel   model_nullmodel_cube1_cube2,
OPosition  ( 12.943262F, 0.000000F, 0.000000F ),
OAngle     ( 0.000000F, 0.000000F, 0.000000F ),
OScale     ( 1.000000F, 1.000000F, 1.000000F ),
Child      object_nullmodel_cube1_inst1,
Sibling     object_nullmodel_cube1_null1,
OQuatRe    ( 0.000000 ),
END

```

CNKOBJECT_END

CNKOBJECT_START

PLIST strip_nullmodel_cube1_cube1[]

START

 CnkM_DA(FBS_SA|FBD_ISA), 4,
 MDiff(255, 255, 255, 255),
 MAmbi(0, 0, 0, 0),
 CnkS(FST_IL|FST_IS), 22, _NB(UFO_0, 3),
 StripL(10), 7, 4, 6, 0, 2, 1, 3, 5, 7, 4,
 ...
 CnkEnd()

END

VLIST vertex_nullmodel_cube1_cube1[]

START

 CnkV_VN(0x0, 49),
 OffnbIdx(0, 8),
 VERT(0xc0a00000, 0xc0a00000, 0xc0a00000),
 NORM(0xbf13cd3a, 0xbf13cd3a, 0xbf13cd3a),
 ...
 CnkEnd()

END

CNKMODEL model_nullmodel_cube1_cube1[]

START

VList vertex_nullmodel_cube1_cube1,
PList strip_nullmodel_cube1_cube1,
Center 0.000000F, 0.000000F, 0.000000F,
Radius 8.660254F,

END

CNKOBJECT object_nullmodel_cube1_cube1[] ルートオブジェクト

START

EvalFlags (FEV_UT|FEV_UR|FEV_US),
CNKModel model_nullmodel_cube1_cube1,
OPosition (0.000000F, 0.000000F, 0.000000F),
OAngle (0.000000F, 0.000000F, 0.000000F),
OScale (1.000000F, 1.000000F, 1.000000F),
Child object_nullmodel_cube1_cube2,
Sibling NULL,
OQuatRe (0.000000),

END

CNKOBJECT_END

DEFAULT_START

...

DEFAULT_END

10.3.6 2 パラメータモディファイア nja ファイルの出力例

2パラメータモディファイアデータの出力はモデルにテクスチャを2枚貼りしコンバートオプションを指定することで可能となる。

```
/* NJA 2.10.00 Ninja2AsciiDataMix CnkModel (SI) */ ヘッダ情報。
```

```
/* ROOT OBJECT : object_2maiHari_cube7_cube7 n(1) d(1) v(8) */ objectデータ情報。
```

```
/* TEXTLIST      : texlist_2maiHari_cube7 n(2) */ texlistデータ情報。
```

本データにはOBJECT、TEXTLISTデータが格納されることを示す。モデル数1、PushPopの最大の深さ1、頂点バッファの最大インデックス8、texlistのエントリ数2。

TEXTURE_START texlistアスキーチャンク開始。

TEXTURENAME textures_2maiHari_cube7[]

START

TEXN("f40_number"), /* 0 64x64 565 TW_D MIP */ GlobalIndex 0, テクスチャサイズ64x64, RGB565, Twi
ddled dma形式、ミップマップあり。

TEXN("vip_emblem"), /* 1 64x64 565 TW_D MIP */

END

TEXTURELIST texlist_2maiHari_cube7

START

TextureList textures_2maiHari_cube7,

TextureNum 2,

END

TEXTURE_END texlistアスキーチャンク終了。

CNKOBJECT_START

PLIST strip_2maiHari_cube7_cube7[]

START

CnkM_DAS(FBS_SA|FBD_ISA), 6, マテリアル設定。
MDiff(255, 178, 178, 178), diffuse設定。
MAmbi(255, 127, 127, 127), ambient設定。
MSpec(11, 255, 255, 255), specular設定。
CnkS(0x0), 17, _NB(UFO_0, 2), テクスチャ無し、ユーザフラグ無し、ストリップ数 2。
StripR(10), 7, 4, 5, 0, 1, 2, 3, 6, 7, 4,
StripL(4), 4, 0, 6, 2,
CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_BF, 0), UVクリップ、D adjust 1.00、バイリニアフィルタ、texId
0。

CnkM_DAS2(FBS_SA|FBD_ISA), 6, 2パラメータモディファイアの第二マテリアル設定。
MDiff(113, 178, 178, 178), 第二diffuse設定。
MAmbi(255, 127, 127, 127), 第二ambient設定。
MSpec(11, 255, 255, 255), 第二specular設定。
CnkT_TID2(0), _TID(FFM_BF, 1), 第二テクスチャ設定。バイリニアフィルタ、texId 1。
CnkS_UVN2(FST_UA), 22, _NB(UFO_0, 1), 2パラメータモディファイアデータ。半透明、ユーザフラグ無し、
ストリップ数 1。

StripL(4), 左回り開始、長さ 4。
1, Uvn(0, 256), Uvn(0, 256), 各頂点に対し 8 ビット(256)精度で二組のUV値を設定する。
5, Uvn(256, 256), Uvn(256, 256),
3, Uvn(0, 0), Uvn(0, 0),
7, Uvn(256, 0), Uvn(256, 0),
CnkEnd()

END

VLIST vertex_2mai hari_cube7_cube7[]

START

CnkV_VN(0x0, 49),
OffnbIdx(0, 8),
VERT(0xc0a00000, 0xc0a00000, 0xc0a00000),
NORM(0xbf13cd3a, 0xbf13cd3a, 0xbf13cd3a),
...
CnkEnd()

END

CNKMODEL model_2mai hari_cube7_cube7[]

START

VList vertex_2mai hari_cube7_cube7,
PList strip_2mai hari_cube7_cube7,
Center 0.000000F, 0.000000F, 0.000000F,
Radius 8.660254F,

END

```
CNKOBJECT  object_2maihari_cube7_cube7[]
START
EvalFlags ( FEV_UT|FEV_UR|FEV_US|FEV_BR ),
CNKModel   model_2maihari_cube7_cube7,
OPosition  ( 0.000000F, 0.000000F, 0.000000F ),
OAngle     ( 0.000000F, 0.000000F, 0.000000F ),
OScale     ( 1.000000F, 1.000000F, 1.000000F ),
Child      NULL,
Sibling     NULL,
OQuatRe    ( 0.000000 ),
END
```

CNKOBJECT_END

DEFAULT_START

...

DEFAULT_END

10.3.7 FlatBump nja ファイルの出力例

平面に貼った時のサンプルである。性能も遅くハードウェア的に制限があるため使用に関しては注意のこと。

```
/* NJA 2.10.01 Ninja2AsciiDataMix CnkModel (SI) */ ヘッダ情報。
```

```
/* ROOT OBJECT : object_bts1_cube1_cube1 n(1) d(1) v(8) */ objectデータ情報。
```

```
/* TEXTLIST      : texlist_bts1_cube1 n(2) */ texlistデータ情報。
```

本データにはOBJECT、TEXTLISTデータが格納されることとその変数名を示す。モデル数1、PushPopの最大の深さ1、頂点バッファの最大インデックス8、texlistのエントリ数2。

TEXTURE_START texlistアスキーチャンク開始。

TEXTURENAME textures_bts1_cube1[]

START

```
        TEXN( "fuku05" ),          /* 0 128x128 565 TW_D MIP */ GlobalIndex 0, テクスチャサイズ128x128, RGB56
5, Twiddled dma形式、ミップマップあり。
```

```
        TEXN( "maru" ),           /* 1 256x256 BUMP TW_D MIP */ GlobalIndex 1, テクスチャサイズ256x256, バン
プ、Twiddled dma形式、ミップマップあり。
```

END

TEXTURELIST texlist_bts1_cube1

START

TextureList textures_bts1_cube1,

TextureNum 2,

END

TEXTURE_END texlistアスキーチャンク終了。

CNKOBJECT_START

PLIST strip_bts1_cube1_cube1[]

START

CnkM_DAS(FBS_SA|FBD_ISA), 6, マテリアル設定。

MDiff(255, 178, 178, 178), diffuse設定。

MAmbi(255, 127, 127, 127), ambient設定。

MSpec(11, 255, 255, 255), specular設定。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_BF, 0), UVクリップ、D adjust 1.00、バイリニアフィルタ、texId 0。

CnkS_UVN(0x0), 72, _NB(UFO_0, 5), ユーザフラグ無し、ストリップ数5。

StripL(4), 左回り開始、長さ4。

1, Uvn(0, 256), 8ビット(256)精度でUV値を出力。

2, Uvn(0, 0),

0, Uvn(0, 256),

6, Uvn(256, 0),

...

CnkM_D(FBS_SA|FBD_ISA), 2, マテリアル設定。diffuseのみ設定。

MDiff(127, 178, 178, 178), diffuse設定。第一引数127が半透明(127/255)を意味する。

CnkT_TID(FCL_U|FCL_V|FDA_100), _TID(FFM_BF, 1), UVクランプ、D adjust 1.00、バイリニアフィルタ、texId 1。texId 1のテクスチャはバンプタイプのテクスチャ。

CnkM_BU(0), 6, バンプパラメータの設定。

_BuDir(0.000000F, 0.000000F, -1.000000F), 貼った方向。

_BuUp(0.000000F, 1.000000F, 0.000000F), 貼った方向の上方向。

CnkS_UVN(FST_UA), 72, _NB(UFO_0, 5), 半透明、ユーザフラグ無し、ストリップ数5。

StripL(4), 左回り開始、長さ4。バンプポリゴン描画。

1, Uvn(0, 256),

2, Uvn(0, 0),

0, Uvn(0, 256),

6, Uvn(256, 0),

...

CnkEnd()

END

VLIST vertex_bts1_cube1_cube1[]

START

 CnkV_VN(0x0, 49),

 OffnbIdx(0, 8),

 VERT(0xc0a00000, 0xc0a00000, 0xc0a00000),

 NORM(0xbf13cd3b, 0xbf13cd3b, 0xbf13cd3b),

 ...

 CnkEnd()

END

CNKMODEL model_bts1_cube1_cube1[]

START

VList vertex_bts1_cube1_cube1,

PList strip_bts1_cube1_cube1,

Center 0.000000F, 0.000000F, 0.000000F,

Radius 8.660254F,

END

CNKOBJECT object_bts1_cube1_cube1[]

START

EvalFlags (FEV_UT|FEV_UR|FEV_US|FEV_BR),

CNKModel model_bts1_cube1_cube1,

OPosition (0.000000F, 0.000000F, 0.000000F),

OAngle (0.000000F, 0.000000F, 0.000000F),

OScale (1.000000F, 1.000000F, 1.000000F),

Child NULL,

Sibling NULL,

OQuatRe (0.000000),

END

CNKOBJECT_END

DEFAULT_START

...

DEFAULT_END

10.3.8 nsc ファイルの出力例

nscファイルはコンバートされたシーンファイルに格納されるnja, nam, mrsなどをくくるためのファイルである。

```
/* NSC 2.10.00 Ninja2Scene (LW) */
```

PATH: 現在ここには何も書き込まない。

polygon_shape256_poly_sakana_02_layer2.nja	モデルデータ 1。
polygon_shape256_poly_sakana_02_layer2.nam	モーションデータ。
polygon_shape256_poly_sakana_02_layer2.mrs	モデルデータ 1 のmrs。(コンバートオプションで省略可能)
polygon_shape256_poly_sakana.nja	モデルデータ 2。
polygon_shape256_poly_sakana.nam	モーションデータ 2。
polygon_shape256_poly_sakana.nas	CompactShapeデータ。
polygon_shape256_poly_sakana.mrs	モデルデータ 2 のmrs。(コンバートオプションで省略可能)
polygon_shape256_Camera.nac	カメラデータ。
polygon_shape256_Camera.nam	カメラモーション。
polygon_shape256_Light.nal	ライトデータ。
polygon_shape256_Light.nam	ライトモーションデータ。

11. バイナリフォーマット

11.1 バイナリフォーマット概要

バイナリフォーマットは単一のメモリ領域に必要な構造体データを格納しデータ先頭アドレスを 0 番地としたポインタアドレスをデータ構造に出現するポインタメンバーに埋め込む。この情報に加えデータの何バイト目にポインタメンバーがいるかを示すオフセット情報を持つ (ポインタオフセット (POF))。

データおよび POF は独立した IFF チャンクとして定義されファイルに連続して配置される。

ローダはデータチャンクのバイトサイズだけのバッファをメモリにアロケートしここにデータを読み込む。次に POF で与えられるポインタオフセットにアロケートされたメモリの先頭アドレスを足し込む。これによりデータ内でのポインタのアドレスは実アドレスになる。

POF はオフセット番号の集合データだが単純にこれを格納するとデータ量が大きくなるため簡単な圧縮をする。現在提供している手法をチャンク POF 0 と呼ぶ。今後より効率のより圧縮方法の適用、データサイズよりも展開速度の速い方法など必要に応じて POF 1、POF 2 などを必要であれば拡張する。POF の圧縮アルゴリズムが複数あってもローダでチャンクネームとして処理を分岐して展開するため互換を保ちつつかつ古いデータも問題なく展開することができる。

データチャンクのすぐ後ろに格納される POF チャンクを消した場合データのロードはできない。必ずデータと POF チャンクはセットで扱う。

バイナリの先頭アドレスにはデータの先頭データがくることを保証する。例えばモデルではモデルツリーのトップのオブジェクト構造体ポインタ、texlist では texlist ポインタ、モーションでは motion 構造体ポインタ。ユーザは各データのポインタをフリーすることでバイナリデータを破棄できる。

現在絶対アドレスのバイナリは定義はされるがコンバータでは未サポート。ユーザに指定されたアドレス値をあらかじめバイナリのポインタに足し込む。これは速度優先のストリーム再生などに利用する。

NJA2Conv というコンバータでアスキーからバイナリフォーマットへコンバートをサポートする。

Ninja2 用の Chunk Model のバイナリチャンク名は Ninja1 のデータとの識別用に 'NJCM' から 'N2CM' に変更した。

SimpleShape データで利用していた 'NSSM' は廃棄され CompactShape データは 'CPSM' を利用する。

11.2 バイナリチャンクの種類

'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。

11.3 バイナリ構造

バイナリ構造

Chunkname	bytesize
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	
Chunk POF0	bytesize
ポインタオフセットデータ	

データバイナリチャンク

< 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

P O F 0 チャンク

ポインタオフセットアルゴリズムタイプ 0	
----------------------	--

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。
現在デフォルト出力における texlist 付き Chunk Model の例を以下に示す。

(例) 拡張子.nj

'NJTL'	bytesize
texlist データ	
Chunk POF0	bytesize
texlist POF データ	
'N2CM'	bytesize
Chunk Model データ	
Chunk POF0	bytesize
Chunk Model POF データ	

texlist データ

Ninja2 Chunk Model データ

絶対アドレスバイナリ構造

‘NJAD’	4 (bytesize)	ユーザ指定アドレスを示す。
ユーザ指定アドレス		
Chunkname	bytesize	データバイナリチャンク
ユーザ指定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		

< 主なチャンクの種類 >

'N2CM' : ninja 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

NJAD チャンクとデータチャンクは離してはならない。ただしユーザ責任においてアドレスをあらかじめ知っている場合ははずすこと可能。先頭に NJAD を一つとその後ろに複数の同一種類のバイナリデータが格納することも可能。これはストリーム再生などで利用する。

11.4 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>
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