



# **Ninja2 pvm 仕様書**

**(07/10/2000)**

## 1. 概要

pvm (PVr Manipulation) は pvr ファイルの拡張です。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します。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です。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ます。以下に特徴をまとめます。

IFF チャンク形式を採用しています。

複数のテクスチャを一括して扱えます。

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ます。

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です。

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ます。

モデル出力ファイル名を持ちモデルデータの texId の更新が可能です。

pvr ファイルへの分解が可能です。編集後再び pvm へのパックができます。

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です。

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ます。ただしデータ量は大きくなるので注意が必要です。

利用するコンバータを指定できます。そのオプションも保存できます。

Ninja ライブラリは pvm から texlist への変換をサポートします。

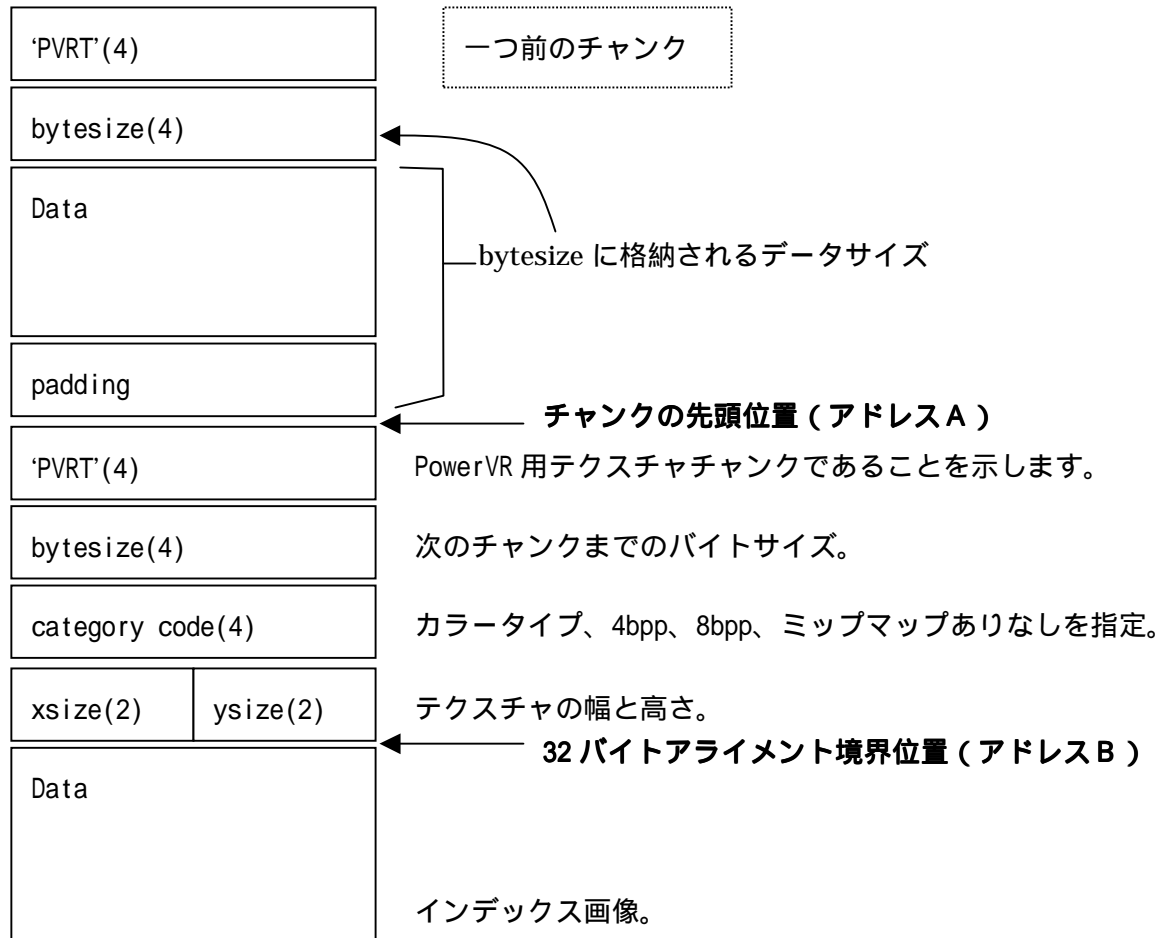
パレットバンク ID を格納できます。globalIndex フィールドの上位 6 ビットを利用します。

パレットデータ PVPL チャンクを格納できます。ロード時にパレットの設定ができます。

### (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ります。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します。アライメント調整は性能に重要な意味を持つため pvm の仕様の一部とします。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です。

PVRT における 3 2 バイトアライメント調整は次の通りです。



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してください。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  $A = B - 16$  になります。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ります。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ません。

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ませす。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 に含まれます。

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ます (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ります)。構造を複雑にしないため pvm も同様の構造をとることとします。次節で説明する PVMH チャンクは pvm 全体の情報を管理しますがここで globalIndex データの上位 6 ビットにパレットのバンク ID を格納します。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します。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ます。

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 が格納されます。上位 6 ビット以外は無効です。

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ります。

パレットは総エントリ数が 1024 エントリです。これを 4bpp の 16 色単位で 64 (0 ~ 63) バンクに分割します。このバンク ID が globalIndex の上位 6 ビットに格納されます。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 として有効です。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ます。詳細はパレットテクスチャ仕様書を参照してください。

## 2. チャンク

### 2.1 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です。括弧内はバイト数を示します。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です。

<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ます。

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

PVPN は PVRL のファイル名を格納します。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 を配置します。

pvm ファイルの先頭には必ず PVMH チャンクがきます。ただしコメントチャンク COMM は PVMH の前にあってもいいです。(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ません。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)です。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ります。

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ます。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します。

## 2.2 チャンク仕様

### ChunkName : 'PVMH'

(PVM Header Information)

概要 :

pvm ファイルのヘッダ情報です。pvm のファイルの先頭には必ず PVMH チャンクを置きます。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、タイプ情報を持ちます。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 を格納します。

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 分だけ繰り返されます。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ります。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ます。

pvmstatus :

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

```
#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します。  
**PVMH\_PS\_CATEGORYCODE** CategoryCode フィールドがあることを示します。  
**PVMH\_PS\_ENTRYINFO** EntryStatus フィールドがあることを示します。  
**PVMH\_PS\_GLOBALINDEX** GlobalIndex フィールドがあることを示します。  
(注意) Ninja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を扱います。

**PVMH\_PS\_CHUNK\_MDLN** チャンク MDLN があることを示します。  
**PVMH\_PS\_CHUNK\_CONV** チャンク CONV があることを示します。  
**PVMH\_PS\_CHUNK\_PVMI** チャンク PVMI があることを示します。  
**PVMH\_PS\_CHUNK\_IMGC** チャンク IMGC があることを示します。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ます。  
**PVMH\_PS\_CHUNK\_PVRT** チャンク PVRT があることを示します。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ます。

以下はパレットが使われる場合のみに利用されます。

**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します。

**PVMH\_PS\_CHUNK\_PVPL** PVPL チャンクが pvm 内部にあることを示します。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ます。

**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

つことを示します。

entrycount :

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

entryId :

データ生成時のエントリのシリアルナンバーです。通常は 0 から始まります。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 に一致します。

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します。

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かがわかります。

pvrname :

Texlist のテクスチャの名前に相当します。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ます。

categoryCode :

PVR のフォーマットの種類を示すカテゴリコードです。本来 PVRT の中を見ればこのコードは格納されているのです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 にも持たせています。

entryStatus :

pvm に格納されている pvr データの情報です。CategoryCode とともにユーザに現在のテクスチャの状態を示します。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します。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与えます。

#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します。

**PVMH\_ES\_LOCK\_PVR** すでに存在する pvr ファイルの上書きを禁止します。

**PVMH\_ES\_DITHER** 変換時にディザを使ったことを示します。

**PVMH\_ES\_ADITHER** 変換時に ディザを使ったことを示します。

**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です。

## ChunkName : 'COMM'

(pvm COMMENT)

### 概要 :

情報を文字列で格納 (コメント) できます。pvm ファイルの任意の位置に配置できます。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ます。

### 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です。Bytesize が string の長さになります。

## 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です。pvm 側からモデルを見つけ texId の変更ができます。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ます。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ります。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ります。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サーチもできます。

### 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ます。TexId 更新のためのモデル群を与えます。

### modelcount：

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

### model name string：

モデルの名前です。通常絶対パス付きで記述します。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です。この場合 pvm のあるカレントディレクトリが利用されます。

## ChunkName : 'CONV'

(pvm CONVerter)

### 概要 :

デフォルトの pvm 生成コンバータは PVMConv です。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します。絶対パスでもいいです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です。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もよく、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ます。

(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ます。

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) で保存されます。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保存/再利用できます。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します。

### 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 : 'PVM'

(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します。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ます。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ます。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ます。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です。エントリの数は PVMH チャンクの entrycount と同じでなければなりません。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です。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ります。

### 形式 :

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

### 説明 :

{ } は PVMH チャンクの entrycount 分だけ繰り返されます。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ります。

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します。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です。

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します。

## ChunkName : 'IMGC'

(Original IMaGe Container)

概要 :

コンバータで変換する前の画像を格納します。他のマシンにコピーしても再変換が可能です。ただしファイルサイズが大きくなるので注意が必要です。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います。

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 から確認されます。

type :

**IMGC\_FMT\_ARGB** ARGB8888 でテクスチャを格納します。

image data : [ARGB8888]...

**IMGC\_FMT\_PAL8** palette 8 でテクスチャを格納します。

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

**IMGC\_FMT\_BMP** bmp を格納します。

**IMGC\_FMT\_TGA** tga を格納します。

**IMGC\_FMT\_PIC** pic を格納します。

**IMGC\_FMT\_PIX** pix を格納します。

## ChunkName : 'PVRT'

(PVR Texture)

概要 :

テクスチャデータです。

形式 :

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

説明 : PVRConv が出力する pvr テクスチャデータです。



## ChunkName : 'PVPN'

(PVP Name)

概要 :

パレットの PVPL のファイル名を格納します。

形式 :

['COMM'(4)][byteSize(4)]

[pallette name string]

説明 :

pallette filename string :

パレットの PVPL のファイル名を格納します。ByteSize が string の長さになります。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ます。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します。

## ChunkName : 'PVPL'

(PVr PaLette)

概要 :

パレットデータです。

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です。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エントリばnテクスチャを描画します。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です。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みます。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します。

### 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ちます。これをインフォレベルと呼びます。インフォレベルが高ければより多くのことができます。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します。この状態では再加工できなくなります。インフォレベルの変更はPVMConv もしくは他のツールされます。

#### 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 pvmstatus 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します。

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ません。

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くなります（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ます）。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ません。PVMH\_PS\_GLOBALINDEX がないと Global Index を設定できません。Ninja でのテクスチャ使用ができなくなります。

以上の理由により上記の四つのデータはNinja用では必須です。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 の更新ができません。

PVMH\_PS\_CHUNK\_CONV がいない場合は PVMConv が利用されます。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\_PVMH がないとテクスチャの再変換ができません。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ります。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くなります。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します。

PVMH\_PS\_CHUNK\_PVRT がないと基本的にはゲームには使えません。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します。この状態であればゲームで使うこともできます。この場合 pvm は texlist 生成のための情報ファイルとして動作します。

PVMH\_PS\_CHUNK\_PVPL は pvm にパレット情報が含まれることを示します。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ます。

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します。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ます。

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 )だけの構成となります。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です。

```
[ '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ます。  
texlist の生成ができます。  
グローバルインデックスの付け替えができます。  
テクスチャ変換時の履歴は見えません。  
テクスチャの再変換はできません。

パレットが必要な場合は PVPL チャンクが付加されます。pvmstatus には globalIndex 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 フラグが設定されます。

```

    ['PVMH'][bytesize]
[pvmstatus][entrycount]
{ [entryId][pvrname][bankId|globalIndex] }
...
{ ['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です。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ります。

## 4. オペレーション

### 4.1 規約

ここで pvm 用ツールを作る場合に守るべきルールをまとめます。

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します。Pvm ツール製作者は次のチェックをパスできるように pvm ファイルを生成できなければいけません。

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

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必要があります。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します。

PVMH の entrycount は pvm ファイルに含まれるテクスチャの数を与えます。

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りません。一致しない場合エラーとします。

### 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です。

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

ここで pvmstatus は以下の通りです。

```
pvmstatus=PVMH_PS_PVRNAME  
    | PVMH_PS_CATEGORYCODE  
    | PVMH_PS_ENTRYINFO  
    | PVMH_PS_GLOBALINDEX  
    | PVMH_PS_CHUNK_MDLN  
    | PVMH_PS_CHUNK_PVMH
```

| PVMH\_PS\_CHUNK\_PVRT;

パレット使用時は PVMH\_PS\_BANKID、PVMH\_PS\_CHUNK\_PVPN、PVMH\_PS\_CHUNK\_PVPL フラグが追加されます。

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ます。  
texlist の生成ができます。  
グローバルインデックスの付け替えができます。  
カテゴリコードによるテクスチャのグループ分けができます。スモールVQの最適化に重要です。  
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ます。  
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ます。  
各テクスチャごとにオプション指定が可能です。  
pvm のテクスチャの順番を入れ替えたり複数の pvm をマージできます。またこの時参照される複数のモデルの texId を更新できます。  
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ます。  
パレットの取り込み、取り出しが可能です。

#### 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ます。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 でできます。また次の操作は動的に行うことができます。

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。

#### 4.4 グローバルインデックスの更新手順

ディレクトリに pvm ファイルを集め pvm 対応版 gigen を実行します。  
内部では次の手順で処理されます。

ディレクトリ内の pvm のリストを作ります。

各 pvm を開き PVMH の情報を参照します。

pvmstatus から pvrname と globalIndex のフィールドを持つことを確認します。持たない pvm ファイルは処理の対象から外します。

pvrname を読み出しリストを作ります。pvrname が重複しないリストを作ります。

ベースナンバーから通し番号を振ります。

pvrname を見ながら PVMH の globalIndex フィールドを更新します。

また pvr の pvm からの展開を使うと次のような操作が可能です。

カレントディレクトリ上ですべての pvm ファイルの pvr を取り出す。同一ファイル名は上書きされるので重複は無くなります。

pvr ファイルに対し gigen します。各 pvr ファイルに globalIndex がつきます。

すべての pvr ファイルの pvm への取り込みをします。この時 pvr ファイルの globalIndex が PVMH チャンクの globalIndex フィールドに格納されます。取り込み完了後展開されていた pvr ファイルを消去します。

## 4.5 model データの texId の更新手順

スモールVQ（コードブック共有タイプ）のエントリを連続させたり、複数の pvm をマージしたりする場合にモデルの texId をそのエントリの変更にあわせて更新する必要があります。

### 4.5.1 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ます。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です。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できます。

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ます。

この時のモデルの texId の更新は次の通りです。

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

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



ーブルを作成します。

`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します。

#### 4.6.2 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` を加算します。また二つの `entrycount` の合計を新しい `pvm` の `entrycount` に入れます。この状態では MLDN チャンクには二つのモデルネームが格納されます。

### 5. P V M C o n v

PVMConv は `pvm` ファイルを操作する基本ツールです。`pvm` のすべての機能を包みます。次に説明する `pvmUtil` ライブラリを元に `pvm` を操作/編集します。

#### 5.1 pvmUtil

次の機能を持ちます。

`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します。

`pvr` 変換時に `pvm` の各チャンクデータを生成できます。

すべてのチャンクデータを `pvm` に書き込みます。

`pvm` ファイルからチャンクデータを読み出せます。

二つの `pvm` をマージできます。

`pvm` から `pvr` を展開できます。

`pvm` から元画像を展開できます。

モデルの `texId` の更新ができます。

`pvm` の COMM チャンクをテキストで表示できます。

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ます。

`pvm` の中身の情報を表示できます。

`pvm` のモデルのパスを書き換えられます。

`pvm` の元画像のパスを書き換えられます。

以 上