NINTENDO
NITRO-System

G3D Binary File Format

Version 1.0.1

> **The contents of this document are strictly confidential and the document should be handled accordingly.**

Table of Contents

Tables

Revision History

Version	Revision Date	Details of Revision
1.0.1	05/11/2005	Revised description of *XXX*_LAST_INTERP_MASK.
1.0.0	01/19/2005	Added support for the environmental mapping and projection mapping expansions.
0.5.0	12/06/2004	Introduced pseudo-structure notation, etc.
0.0.3	08/19/2004	Inserted table numbers.
0.0.2	08/09/2004	Created beta version.
0.0.1	07/23/2004	Initial version.

1 Overview

In order to display models and play animation using the G3D library, you must use `g3dcvtr` to convert NITRO intermediate files into binary files. This document explains the format of binary files converted with `g3dcvtr`. Note that the binary format specifications are subject to change without notice.

2 Characteristics of G3D Binary Format

The binary format used by G3D has the following characteristics:

- Multiple model or animation data sets can be stored in one binary file.

- The G3D library can directly process binary files loaded in memory. The G3D internal data structure and the binary file format match so it is not necessary to convert from binary file format to an object in memory during initialization. This minimizes the memory and processing overhead during initialization—there is no need to allocate additional memory.

- There are no internal pointers. All links in binary format are expressed as offsets based on the start of individual blocks in the files.

- There are dictionaries to allow searches by resource name. The name dictionary allows compact and fast search, and the resource search by name is performed efficiently.

- In order to reduce the calculation cost during drawing, various types of data that are calculated and aligned by `g3dcvtr` are kept.

3 Explanation of G3D Binary File Format

3.1   Data Structure Used by all Binary Files

The `pseudo_struct` pseudo-structure can change the data members of a structure and arrange them in variable length depending on the condition. The actual structure name defined in G3D appears in parentheses next to the name of the pseudo-structure.

3.1.1   File Headers

Data structures that show file types like the following example are stored at the beginning of the binary file.

```
pseudo_struct FileHeader {
    pseudo_struct HeaderInfo(NNSG3dResFileHeader) {
        u32 signature;
        u16 byteOrder = 0xfeff
        u16 version;
        u32 fileSize;
        u16 headerSize = 16;
        u16 dataBlocks;
    } info;
    u32 offset[dataBlocks];
};
```

Table 3-1: `FileHeader` Data Members

Name	Description
signature	Constant for determining binary file type (four characters).
byteOrder	Number for determining endianity `0xfeff` when little-endian
version	Binary file version (`0x0102` if version `1.2`)
fileSize	File size
headerSize	Size of `HeaderInfo` Fixed at 16 when G3D.
dataBlocks	Number of data blocks **1** or **2** for `.nsbmd` file **1** for other files.
offset	Stores offset from start of file to each block

3.1.2   Data Block Header

The data that shows the type and size of the data block like the following is stored at the beginning of each data block in the binary file.

```
pseudo_struct DataBlockHeader(NNSG3dResDataBlockHeader) {
    u32 kind;
    u32 size;
};
```

Table 3-2: `DataBlockHeader` Data Members

Name	Description
kind	Stores the symbol that represents the type of data block
size	Size of the entire data block

3.1.3   Dictionary

With G3D, it is possible to give names of up to 16 characters to various resources such as texture and material and access them by name. Name searches are performed using the same data structure. This section explains the data structure of the dictionary.

Expressed with a pseudo-structure, the dictionary will be as follows.

```
pseudo_struct Dictionary(NNSG3dResDict) {
    u8   revision = 0;
    u8   numEntry;
    u16  sizeDictBlk;
    PADDING(2 bytes);
    u16  ofsEntry;

    pseudo_struct PtreeNode(NNSG3dResDictTreeNode) {
        u8 refBit;
        u8 idxLeft;
        u8 idxRight;
        u8 idxEntry;
    } node[numEntry + 1];

    pseudo_struct DictEntry(NNSG3dResDictEntryHeader) {
        u16  sizeUnit;
        u16  ofsName;
        u8   data[numEntry][sizeUnit];
    } entry;

    pseudo_struct DictName(NNSG3dResName) {
        u8   name[16];
    } names[numEntry];
};
```

Table 3-3: `Dictionary` Data Members

Name	Description	
revision	Version of dictionary structure (zero only).	
numOfEntry	Number of entries registered in dictionary.	
sizeDictBlk	Size of dictionary (in bytes).	
ofsEntry	Offset from start of `Dictionary` to `DictEntry`.	
PtreeNode	refBit	Referenced from the start of the input string to the `rebBit` bit
	idxLeft	Index of node to reference next when referenced bit is `0`
	idxRight	Index of node to reference next when referenced bit is `1`
	idxEntry	Index of `DictEntry` and `DictName` corresponding to node
DictEntry	sizeUnit	Size per data entry. When equal to or greater than four bytes, in units of four bytes.
	ofsName	Offset from start of `DictEntry` to `DictName`
	data	Data storage section
DictName	name	Resource name string
		For names, zeros must be entered to fill any unused spaces within 16 characters. Cannot be treated as C string when all 16 characters are used up.

Normally, the offset to resource is stored in the data storage section (`DictEntry::data`), but data may be entered directly. When offset is stored, the function using the dictionary must request the pointer for the correct reference point.

Linear searches by name can be performed within `DictName`, but a tree (`node[]`) is provided for use with the `Patricia` algorithm. By using `Patricia`, it is possible to prevent the increase of search time when the number of entries increases. For the details of about this algorithm, refer to ***Algorithms in C++***[1].

`DictEntry` can also be scanned for references by index.

---

[1] Sedgwick, Robert. ***Algorithms in C++***
  ISBN: 0201510596
  Reading, MA; Addison Wesley Professional Publishing
  30 Apr 1992

Patricia tree is a type of radix search tree. The following information is stored in each node of the tree: "which bit of the search key to look up", "pointer to node (right) to proceed on to when searched bit is ON and to node (left) to proceed on to when searched bit is OFF", and "the key having the node". Patricia tree uses these data to perform the search.

To search the Patricia tree, first look up the "which bit to look up" information written in the node by applying it in the given key (in this case, the resource name) and determine whether to proceed right or left node. If the destination node is a normal child node, then the same process is repeated. If the destination node a node that is "upstream" (i.e., closer to the root in terms of links) of the node from which the search commenced, then the node movement is halted and a comparison of the key held by the node is performed.

As a result of the comparison, the process ends as found if it is the same key, and the process ends as failure if not.

Therefore, this algorithm allows greater search speeds because the comparison of entire keys is performed only at the end, and the comparison on each node uses only one bit.

3.1.4　Animation Header

Each animation resource has a header for classifying the animation type. The data structure is as follows.

```
pseudo_struct AnmHeader(NNSG3dResAnmHeader) {
    u8 category0;
    u8 revision;
    u16 category1;
};
```

Table 3-4: `AnmHeader` Data Members

Name	Description
category0	Specifies animation category: • *M* - material animation • *J* - joint animation • *V* - visibility animation
revision	Animation file format revision
category1	Specifies animation type: • *CA* - joint animation • *VA* - visibility animation • *MA* - material color animation • *TP* - texture pattern animation • *TA* - texture SRT animation

3.2   Model Data File (`.nsbmd`) Structure

An `.nsbmd` file can be divided roughly into the model section, and the texture and palette section. In the model block, the sets of models are stored, and joint structure, material, and shape are stored for each model. Sets of texture and palette are stored, and associated with each model loaded to VRAM during execution. Each model and texture and palette can be associated by a name of up to 16 characters.

The following shows the `.nsbmd` file using pseudo-structure.

```
pseudo_struct NSBMD {
    FileHeader fileHeader = {
        dataBlocks = 1 or 2,
        signature = '0DMB'
    };
    ModelSet modelSet;
    IF (there are textures) {
        TexPlttSet texPlttSet;
    }
};
```

Table 3-5: NSBMD Data Members

Name	Description
fileHeader	File header region
modelSet	Model block
texPlttSet	Texture and palette block

3.2.1 Model Block

The model block can contain multiple models. Each model can be accessed by its name of up to 16 characters.

3.2.1.1 Model Set

The sets of models are expressed with the pseudo-structure as shown below.

```
pseudo_struct ModelSet(NNSG3dResMdlSet) {
    DataBlockHeader  header = {
        kind = '0LDM',
        size = SIZEOF(ModelSet)
    };
    Dictionary       dict = {sizeUnit = 4 bytes};
    Model            models[# of models];
};
```

The data part of the dictionary is 32 bits, and stores the offset (in bytes) from the start of ModelSet.

Table 3-6: `ModelSet` Data Members

Name	Description
header	Model block header region
dict	`Dictionary` region for accessing each model
models	`Model` sets in model block

3.2.1.2 Model

Expressing a model as pseudo-structure will be as follows.

```
pseudo_struct Model(NNSG3dResMdl) {
    u32          size = SIZE_OF(Model);
    u32          ofsSbc;
    u32          ofsMat;
    u32          ofsShp;
    u32          ofsEvpMtx;
    ModelInfo    info;
    NodeSet      nodes;
    u8           sbc[ofsMat – ofsSbc];
    MaterialSet  materials;
    ShapeSet     shapes;
    EvpMatrices  evpMatrices;
};
```

In this instance, each offset is stored as number of bytes from beginning part of `Model`. One model is divided into the following: basic information regarding model; information on each node; information on each material; information on each shape; information for associating node, material and shape; and matrix storage region for envelope calculation.

Table 3-7: `Model` Data Members

Name	Description
size	Model size
ofsSbc	Offset from start of `Model` to SBC line
ofsMat	Offset from start of `Model` to material set
ofsShape	Offset from start of `Model` to shape set
ofsEvpMtx	Offset from start of `Model` to matrix storage region for envelope matrix calculation
info	Basic information about model
nodes	Location and position information of each node
sbc	Information for associating node, material, and shape
materials	Information on each material
shapes	Information on each shape
evoMatrices	Matrix storage region for envelope calculation

3.2.1.3   Basic Information about Models

Following is the basic information regarding models shown with the pseudo-structure.

```
pseudo_struct ModelInfo(NNSG3dResMdlInfo) {
    u8   sbcType;
    u8   scalingRule;
    u8   texMtxMode;
    u8   numNode;
    u8   numMat;
    u8   numShp;
    u8   firstUnusedMtxStackID;
    PADDING(1 byte);

    fx32 posScale;
    fx32 invPosScale;
    u16  numVertex;
    u16  numPolygon;
    u16  numTriangle;
    u16  numQuad;

    fx16 boxX, boxY, boxZ;
    fx16 boxW, boxH, boxD;
    fx32 boxPosScale;
    fx32 boxInvPosScale;
};
```

Table 3-8: `ModelInfo` Data Members

Name	Description
sbcType	SBC type. Zero is entered.
scalingRule	Scaling calculation method
texMtxMode	Texture matrix calculation method
numNode	Number of joints
numMat	Number of materials
numShp	Number of shapes
firstUnusedMtxStackID	Start of empty region in matrix stack (stack index)
posScale, invPosScale	Scale value applied to vertex position coordinate and its inverse number
vertexSize	Number of vertices
polygonSize	Number of polygons
triangleSize	Number of triangle polygons among polygons counted by polygonSize
quadSize	Number of quadrilateral polygons among polygons counted by polygonSize
boxX, boxY, boxZ	For box test (parameter that should be passed to G3_BoxTest)
boxW, boxH, boxD	Same as above
boxPosScale, boxInvPosScale	Scale value to apply before box test, and its inverse number

Table 3-9: Values Taken by `scalingRule`

Value	Description
0	Normal model
1	A model having a joint with Maya's segment scale compensate applied
2	A model with Softimage|3D and Softimage|XSI "classic scale off" specified

Table 3-10: Values Taken by `texMtxMode`

Value	Description
0	Use texture matrix calculation that supports Maya
1	Use texture matrix calculation that supports Softimage|3D
2	Use texture matrix calculation that supports 3dsmax
3	Use texture matrix calculation that supports Softimage|XSI

3.2.1.4   Node Information

Following is the collection of information regarding the location of each node and posture, shown with the pseudo-structure.

```
pseudo_struct NodeSet(NNSG3dResNodeInfo) {
    Dictionary dict = {sizeUnit = 4 bytes};
    pseudo_struct NodeData(NNSG3dResNodeData) {
    u16 flag;
    u16 _00;
    IF (!(flag & NNS_G3D_SRT_FLAG_TRANS_ZERO)) {
        fx32 Tx, Ty, Tz;
    }
    IF (!(flag & NNS_G3D_SRT_FLAG_ROT_ZERO) &&
    !(flag & NNS_G3D_SRT_FLAG_PIVOT_EXIST)) {
        fx16 _01, _02;
        fx16 _10, _11, _12;
        fx16 _20, _21, _22;
    }
    IF (!(flag & NNS_G3D_SRT_FLAG_ROT_ZERO) &&
    (flag & NNS_G3D_SRT_FLAG_PIVOT_EXIST)) {
        fx16 A, B;
    }
    IF (!(flag & NNS_G3D_SCALE_ONE)) {
        fx32 Sx, Sy, Sz;
        fx32 InvSx, InvSy, InvSz;
    }
} data[# of nodes];
};
```

The 32-bit data section of Dictionary stores the offset (in bytes) from the start part of NodeInfo. The size of NodeData differs depending on the flag value. Data is omitted if the identity matrix or zero vector. (See Table 3-11 on the next page.)

Table 3-11: `NodeData` flag field values

Definition Name	Value	Description
NNS_G3D_SRTFLAG_TRANS_ZERO	0x0001	If this bit is ON, translation component is zero
NNS_G3D_SRTFLAG_ROT_ZERO	0x0002	If this bit is ON, rotation matrix is identity matrix
NNS_G3D_SRTFLAG_SCALE_ONE	0x0004	If this bit is ON, scale is 1
NNS_G3D_SRTFLAG_PIVOT_EXIST	0x0008	If this bit is ON, rotation matrix is compressed form
NNS_G3D_SRTFLAG_PIVOT_MASK	0x00f0	When rotation matrix is compressed form, indicates location (0-8) of pivot element (element with absolute value of 1).
NNS_G3D_SRTFLAG_PIVOT_MINUS	0x0100	If this bit is ON, pivot element is negative (-1).
NNS_G3D_SRTFLAG_SIGN_REVC	0x0200	If this bit is ON, C has the opposite sign of B.
NNS_G3D_SRTFLAG_SIGN_REVD	0x0400	If this bit is ON, D has the opposite size of A.
NNS_G3D_SRTFLAG_IDXPIVOT_MASK	0x00f0	Specifies location of pivot element with the logical product of this value and `info`

Table 3-12: `NodeData` Data Members

Name	Description
flag	Flag Refer to the NodeData flag field value in Table 3-11.
Tx, Ty, Tz	Translation component set in node. Omitted when all zeros.
_00, _01, _02, _10, _11, _12 _20, _21, _22	Rotation matrix set in node. Omitted when identity matrix. Also, when any one of these elements is 1 or -1, data described below is used as rotation matrix.
A, B	Data format of rotation matrix used when the rotation matrix set in node is not an identity matrix and any element is 1 or -1. The details are explained outside the table.
Sx, Sy, Sz, InvSx, InvSy, InvSz	The scale value set in node and its inverse number Omitted when all ones.

The ABCD in the chart indicate the four elements in the matrix (elements in the small matrix relating to pivot element) that result when erasing the line and column that include the pivot element. C will be +B or –B, and D will be +A or -A. Specifically, writing the positional relationship of the pivot element and ABCD in the original matrix will be as follows.

$$\text{Pivot is 4} \rightarrow \begin{pmatrix} A & 0 & B \\ 0 & 1 & 0 \\ C & 0 & D \end{pmatrix}, \text{pivot is 0} \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & A & B \\ 0 & C & D \end{pmatrix}, \text{pivot is 8} \rightarrow \begin{pmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Here, the rotation matrix is an orthogonal matrix. Therefore, with the elements of rows and columns including pivot elements, elements other than pivot elements are 0, and C is +B or –B, D is +A or -A.

3.2.1.5   Material Information

The section that arranges the sets of material is shown below using the pseudo-structure.

```
pseudo_struct MaterialSet(NNSG3dResMat) {
    u16         ofsDictTexToMatList;
    u16         ofsDictPlttToMatList;
    Dictionary  dict = {sizeUnit = 4 bytes};
    Dictionary  dictTexToMatList = {sizeUnit = 4 bytes};
    Dictionary  dictPlttToMatList = {sizeUnit = 4 bytes};
    u8          matIdxData[];
    PADDING(4 bytes alignment);
    Material    materials[# of materials];
};
```

Table 3-13: `MaterialSet` Data Members

Name	Description
ofsDictTexToMatList	Offset from start of `MaterialSet` to `dictTexToMatList`
ofsDictPlttToMatList	Offset from start of `MaterialSet` to `dictPlttToMatList`
dict	Dictionary that references each material from material name or material ID
dictTexToMatList	Dictionary that references material ID list from texture name
dictPlttToMatList	Dictionary that references material ID list from palette name
matIdxData	Series of Material IDs. Accessed from `dictTexToMatList` and `dictPlttToMatList`.
materials	Series of each material

`MaterialSet` has three types of dictionaries. The dictionary `dict` is for referencing each material by material name. The dictionaries `dictTexToMatList` and `dictPlttToMatList` are for getting, from texture name and palette name, a list of materials IDs using that texture and palette. These two dictionaries can be used when associating textures and palettes with models.

The data storage section of the dictionary `dict` is 32 bits and stores the offset from the start of `MaterialSet`. Also, the data storage section of the dictionaries `dictTexToMatList` and

dictPlttToMatList is 32 bits, and stores the offset from the start of MaterialSet in the lower 16 bits, and references the start of the material ID arrangement stored in matIdxData. The number of material ID is stored in bits 16 through 23. 1 is entered in bits 24 to 31 when texture is bound, and 0 when not.

The data structure of each material shown in pseudo-structure is as follows.

```
pseudo_struct Material(NNSG3dResMatData) {
    u16   itemTag = 0;
    u16   size;
    u32   diffAmb, specEmi;
    u32   polyAttr, polyAttrMask;
    u32   texImageParam, texImageParamMask;
    u16   texPlttBase;
    u16   flag;
    u16   origWidth, origHeight;
    fx32  magW, magH;
    IF (!(flag & NNS_G3D_MATFLAG_TEXMTX_SCALEONE)) {
    fx32  scaleS, scaleT;
    }
    IF (!(flag & NNS_G3D_MATFLAG_TEXMTX_ROTZERO)) {
       fx32  rotSin, rotCos;
    }
    IF (!(flag & NNS_G3D_MATFLAG_TEXMTX_TRANSZERO)) {
       fx32  transS, transT;
    }
    IF (flag & NNS_G3D_MATFLAG_EFFECTMTX) {
       fx32 effectMtx[16];
    }
};
```

Table 3-14: `Material` Data Members

Name	Description
itemTag	Shows material data type (at present, zero only)
size	Appropriate material size
diffAmb	Diffuse and ambient specification Same bit pattern as the parameter of the geometry command `MaterialColor0`
specEmi	Specular and emission specification Same bit pattern as parameter of the geometry command `MaterialColor1`
polygonAttr	Polygon attribute value specification Same bit pattern as parameter of the geometry command `PolygonAttr`
polygonAttrMask	Mask that takes 1 for the bit valid as data in `polygonAttr` Invalid bits are set by combining with default settings.
texImageParam	Texture image parameter setting Same bit pattern as parameter of the geometry command `TexImageParam` Texture's VRAM start address, texture size, texture format, and palette's color 0 setting value enable flag are not set. Uses the texture settings to be bound during bind.
texImageParamMask	Mask that takes 1 for the bit valid as data in `TexImageParam` Invalid bits are set by combining with default settings.
texPlttBase	Texture palette base address setting Same bit pattern as lower 16 bits of the parameter for the geometry command `TexPlttBase`.
flag	Various flags regarding texture (described later)
origWidth, origHeight	Width and height of texture allocated to material during tool creation.
magW, magH	Region that stores the width and height of texture bound during execution divided by `origWidth` and `origHeight`.
scaleS, scaleT	Texture scale components
rotSin, rotCos	Sine and cosine of texture rotation angle
transS, transT	Texture translation components

Table 3-15: `Material` flag member values

Name	Value	Description
NNS_G3D_MATFLAG_TEXMTX_USE	0x0001	Determines use of texture matrix
NNS_G3D_MATFLAG_TEXMTX_SCALEONE	0x0002	ON if all texture scale components are 1.0
NNS_G3D_MATFLAG_TEXMTX_ROTZERO	0x0004	ON if texture does not rotate
NNS_G3D_MATFLAG_TEXMTX_TRANSZERO	0x0008	ON if texture does not translate
NNS_G3D_MATFLAG_ORIGWH_SAME	0x0010	Set when texture Width/Height is same as system (This bit is set during execution and initialization)
NNS_G3D_MATFLAG_WIREFRAME	0x0020	ON if wire frame display
NNS_G3D_MATFLAG_DIFFUSE	0x0040	ON if diffuse is specified for material
NNS_G3D_MATFLAG_AMBIENT	0x0080	ON if ambient is specified for material
NNS_G3D_MATFLAG_VTXCOLOR	0x0100	ON if `vtxcolor` flag is specified for material
NNS_G3D_MATFLAG_SPECULAR	0x0200	ON if specular is specified for material
NNS_G3D_MATFLAG_EMISSION	0x0400	ON if emission is specified for material
NNS_G3D_MATFLAG_SHININESS	0x0800	ON if shininess is specified for material
NNS_G3D_MATFLAG_TEXPLTTBASE	0x1000	ON if texture palette base address is specified
NNS_G3D_MATFLAG_ EFFECTMTX	0x2000	ON if effect matrix used in environment map and projection map exists

`Material` has parameters related to material color, polygon attribute, and texture. `TexImageParam`, `texPlttBase`, `magW`, and `magH` must reflect the texture settings to be bound during texture binding.

3.2.1.6　Shape Set and Shape

Shape set and shape shown in the pseudo-structure is as follows.

```
pseudo_struct ShapeSet(NNSG3dResShp) {
    Dictionary dict = {sizeUnit = 4 bytes};

    pseudo_struct Shape(NNSG3dResShpData) {
        u16 itemTag = 0;
        u16 size;
        u32 flag;
        u32 ofsDL;
        u32 sizeDL;
    } shape[# of shapes];

    u32 DL[SUM(Shape::sizeDL)];
};
```

Table 3-16: `ShapeSet` Data Members

Name	Description	
dict	Dictionary region for accessing each shape	
shape	itemTag	Shows types of shape data (at present, zero only)
	size	Appropriate shape size (size of Shape)
	flag	Flag showing characteristics of display list. Refer to Table 3-17.
	ofsDL	Offset from the start of Shape to display list
	sizeDL	Size of display list
DL	Array storing the display list of the shape belonging to ShapeSet	

Table 3-17: `ShapeSet::Shape::flag` member values

Name	Value	Description
NNS_G3D_SHPFLAG_USE_NORMAL	0x00000001	If ON, Normal command is on display list
NNS_G3D_SHPFLAG_USE_COLOR	0x00000002	If ON, Color command is on display list
NNS_G3D_SHPFLAG_USE_TEXCOORD	0x00000004	If ON, TexCoord command is on display list
NNS_G3D_SHPFLAG_USE_RESTOREMTX	0x00000008	If ON, RestoreMtx command is on display list

When converting `.imd` files with `g3dcvtr`, the first `RestoreMtx` command on the display list is encoded in the SBC mentioned later. Therefore, when not using envelope there is no `RestoreMtx` command on the display list.

3.2.1.7   Information for Associating Node, Material, and Shape

Information that associates each node, material, and shape is encoded as variable length byte code. This byte code is referred to as SBC (Structured Byte Code).

SBC stores parent/child relationship among nodes, association of nodes and matrix stack index, material and shape combination specification, and information that is a blend of association to node, billboard conversion, and modeling matrix. Each of these pieces of information is defined as a separate command. If processed in order, it is arranged so that the model is drawn.

Table 3-18: SBC Commands

Command Name (Symbol)	NOP(NNS_G3D_SBC_NOP)
Encoding	7 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
Operand	None
Process Details	No operation.

Command Name (Symbol)	RET(NNS_G3D_SBC_RET)
Encoding	7 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
Operand	None
Process Details	Exists at the end of the SBC line.

Command Name (Symbol)	NODE(NNS_G3D_SBC_NODE)
Encoding	7 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | NodeID | | --- | --- | --- | --- | --- | --- | --- | V |
Operand	*NodeID* - Specifies node corresponding to node ID. *V* - 1 when shape belonging to NodeID is visible. 0 when it is invisible.
Process Details	All MAT and SHP commands before the next NODE command appears are regarded as belonging to the node that has the NodeID specified by the NODE command.

Command Name (Symbol)	MTX(NNS_G3D_SBC_MTX)
Encoding	7 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | 0 | 0 | 0 | Idx |
Operand	*Idx* - matrix stack index
Process Details	Issues RestoreMtx command. Reads matrix from specified location of matrix stack of location coordinate matrix to current matrix.

Command Name (Symbol)	MAT(NNS_G3D_SBC_MAT)

Encoding	7 5 0
	
	  OPT 0 0 1 0 0
	
	           MatID

Operand	*MatID*: Material ID
Process Details	Sets the settings of the specified material to geometry engine. The *OPT* value is used as a hint for speeding up operation. When *OPT*=000: When the MatID specified by operand is the only one in this SBC. When *OPT*=001: The MatID specified by operand may be specified by subsequent MAT commands. When *OPT*=010: The MatID specified by operand may have been specified by previous MAT command, but won't be specified later.

Command Name (Symbol)	SHP(NNS_G3D_SBC_SHP)
Encoding	7 0 0 0 0 0 0 1 0 1 ShpID
Operand	*ShpID* - Shape ID
Process Details	Draws specified shape.

Command Name (Symbol)	NODEDESC(NNS_G3D_SBC_NODEDESC)
Encoding	7 5 4 3 2 1 0 OPT | 0 | 0 | 1 | 1 | 0 NodeID ParentNodeID 0 | 0 | 0 | 0 | 0 | 0 | P | S Exists when *OPT*=001,011 0 | 0 | 0 | DestIdx Exists when *OPT*=010,011 0 | 0 | 0 | SrcIdx
Operand	*NodeID* - Specifies node ID that requires modeling matrix. *ParentNodeID* - Specifies ID of parent node. *S* - Maya's Segment Scale Compensate is applied to this node. *P* - This node is the parent node of the node with Maya's Segment Scale Compensate applied. *DestIdx* - Matrix stack index is specified when storing calculation results in matrix stack. Specified when it is necessary to store calculation results in matrix stack. *SrcIdx* - Matrix stack index is specified when restoring matrix from matrix stack before calculation. Specified when extracting matrix corresponding to parent node from matrix stack.
Process Details	Calculates modeling matrix corresponding to node ID.

Command Name (Symbol)	BB(NNS_G3D_SBC_BB)

Encoding	7 0
	
	  0 0 0 0 0 1 1 1
	
	NodeID
	
	Exists when **OPT**=001,011
	
	  0 0 0 DestIdx
	
	Exists when **OPT**=010,011
	
	  0 0 0 SrcIdx

Operand	**NodeID** Node ID of matrix that applies billboard conversion. **DestIdx** - Matrix stack index is specified when storing calculation results in matrix stack. **SrcIdx** - Matrix stack index is specified when restoring matrix from matrix stack before calculation.
Process Details	Applies billboard conversion to matrix.

Command Name (Symbol)	BBY(NNS_G3D_SBC_BBY)

Encoding	7 0
	
	  0 0 0 0 1 0 0 0
	
	NodeID
	
	Exists when **OPT**=001,011
	
	  0 0 0 DestIdx
	
	Exists when **OPT**=010,011
	
	  0 0 0 SrcIdx

Operand	**NodeID** - Node ID of matrix that applies Y axis billboard conversion. **DestIdx** - Matrix stack index is specified when storing calculation results in matrix stack. **SrcIdx** - Matrix stack is specified when restoring matrix from matrix stack before calculation.
Process Details	Applies Y axis billboard conversion to matrix.

Command Name (Symbol)	NODEMIX(NNS_G3D_SBC_NODEMIX)
Encoding	7 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | | DestIdx | | NumMtx | Repeats the following values for each `NumMtx` | SrcIdx_N | | NodeID_N | | Ratio_N | Instruction length is `3+3*NumMtx`.
Operand	***DestIdx*** - Index of matrix stack where calculation results matrix is stored. ***NumMtx*** - Number of blended matrices. `SrcIdx_N`, `NodeID_N`, and `Ratio_N` are returned in order `NumMtx` times. ***SrcIdx_N*** - Index of matrix stack where blended matrix is stored. ***NodeID_N*** - Node ID of blended matrix. ***Ratio_N*** - Matrix blend ratio is a fixed decimal with unsigned decimal of 8 bits.
Process Details	Blends location coordinate matrix with specified ratio, and calculates matrix for weighted envelope. When calculating, uses inverse matrix (to convert from coordinate system of entire model to coordinate system of each joint) of modeling conversion matrix stored in `evpMatrices`.

Command Name (Symbol)	CALLDL(NNS_G3D_SBC_CALLDL)
Encoding	7 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 31 0 | RelAddr | 31 0 | Size |
Operand	***RelAddr*** - Relative address from start address of `CALLDL` instruction to display list ***Size*** - Size of display list (in bytes)
Process Details	Sends display list specified with operand to geometry engine.

Command Name (Symbol)	POSSCALE(NNS_G3D_SBC_POSSCALE)					
Encoding	7 5 4 3 2 1 0					
	OPT	0	1	0	1	1
Operand	None					
Process details	When *OPT*=000, applies scaling matrix (see *posScale* and *invPosScale* of `ModelInfo`) set for each model data in current matrix. When *OPT*=001, applies the inverse matrix.					

Command Name (Symbol)	ENVMAP(NNS_G3D_SBC_ENVMAP)					
Encoding	7 0					
	OPT=0	0	1	1	0	0
	MatID					
	Flag					
Operand	`MatID`: Material ID `Flag`: Flag for expansion (Currently always 0)					
Process Details	Calculates the texture matrix for environmental mapping. It is placed immediately after the `MAT` command, and the `OPT` value is always 0.					

Command Name (Symbol)	PRJMAP(NNS_G3D_SBC_PRJMAP)					
Encoding	7 0					
	OPT=0	0	1	1	0	1
	MatID					
	Flag					
Operand	`MatID`: Material ID `Flag`: Flag for expansion (Currently always 0)					
Process Details	Calculates the texture matrix for projection mapping. It is placed immediately after the `MAT` command, and the `OPT` value is always 0.					

3.2.1.8 Matrix Storage Region for Envelope Calculation

This region exists only for models with weighted envelope. The reverse matrix of the location coordinates matrix and direction vector matrix which convert the joint coordinate system of the still pose to the object coordinate system (in other words, the transformation matrix to the object coordinate system to the coordinate system of each joint) is stored. This is used for omitting inverse matrix calculation in NODEMIX command process of SBC.

```
pseudo_struct EvpMatrices {
    IF (Weighted envelope model) {
        pseudo_struct {
            MtxFx43 invM;
            MtxFx33 invN;
        } m[# of nodes];
    }
};
```

Table 3-19: `EvpMatrices` Data Members

Name	Description
invM	Inverse matrix of location coordinate matrix
invN	Inverse matrix of direction vector matrix

3.2.2 Texture and Palette Block

The texture and palette block can store multiple textures and palettes and can access each texture and palette by name of up to 16 characters.

3.2.2.1 Texture and Palette Sets

Texture and palette sets shown with pseudo-structure is as follows.

```
pseudo_struct TexPlttSet(NNSG3dResTex) {
    DataBlockHeader header = {
        kind = '0XET',
        size = SIZE_OF(TexPlttSet)
    };
    pseudo_struct TexInfo(NNSG3dResTexInfo) {
        u32 vramKey;
        u16 sizeTex;
        u16 ofsDict;
        u16 flag;
        PADDING(2 bytes);
        u32 ofsTex;
```

```
    } texInfo;


    pseudo_struct Tex4x4Info(NNSG3dResTex4x4Info) {
        u32 vramKey;
        u16 sizeTex;
        u16 ofsDict;
        u16 flag;
        PADDING(2 bytes);
        u32 ofsTex;
        u32 ofsTexPlttIdx;
    } tex4x4Info;


    pseudo_struct PlttInfo(NNSG3dResPlttInfo) {
        u32 vramKey;
        u16 sizePltt;
        u16 flag;
        u16 ofsDict;
        PADDING(2 bytes);
        u32 ofsPlttData;
    } plttInfo;


    Dictionary dictTex = {sizeUnit = 8 bytes};
    Dictionary dictPltt = {sizeUnit = 4 bytes};
    u8 texData[texInfo.sizeTex << 3];
    u8 tex4x4Data[tex4x4Info.sizeTex << 3];
    u8 tex4x4IdxData[tex4x4Info.sizeTex << 2];
    u8 plttData[plttInfo.sizePltt << 3];
};
```

Table 3-20: `TexPlttSet` **Data Members**

Name	Description		
header	Header region of texture and header blocks		
texInfo	vramKey	VRAM key storage location for texture data other than 4x4	
	sizeTex	Numerical value with size of `texData` shifted three bits to the right	
	ofsDict	Offset from start of `TexPlttSet` to `dictTex`	
	flag	Flag related to texture data other than 4x4	
	ofsTex	Offset from start of `TexPlttSet` to `texData`	
tex4x4Info	vramKey	VRAM key storage location for 4x4 texel compressed texture data	
	sizeTex	Numerical value with size of `tex4x4` shifted three bits to the right	
	ofsDict	Offset from start of `TexPlttSet` to `dictTex`	
	flag	Flag related to 4x4 texel compressed texture data	
	ofsTex	Offset from start of `TexPlttSet` to `tex4x4Data`	
	ofsTexPlttIdx	Offset from start of `TexPlttSet` to `tex4x4IdxData`	
plttInfo	vramKey	VRAM key storage location for palette	
	sizePltt	Numerical value with size of `plttData` shifted three bits to the right	
	flag	Flag related to palette data	
	ofsDict	Offset from start of `TexPlttSet` to `dictPltt`	
	ofsPlttData	Offset from start of `TexPlttSet` to `plttData`	
dictTex	Dictionary that accesses attribute value of each texture from texture name		
dictPltt	Dictionary that accesses attribute value of each palette from palette name		
texData	Array for texture data other than 4x4 texel compressed texture		
tex4x4Data	Array for texture data of 4x4 texel compressed texture		
tex4x4IdxData	Array for palette index data for 4x4 texel compressed texture		
plttData	Array for palette data		

Table 3-21: `TexPlttSet::TexInfo::flag` **member value**

Name	Value	Description
NNS_G3D_RESTEX_LOADED	0x0001	This is set when texture data other than 4x4 is loaded to VRAM.

Table 3-22: `TexPlttSet::Tex4x4Info::flag` **member value**

Name	Value	Description
NNS_G3D_RESTEX4x4_LOADED	0x0001	This is set when 4x4 texel compressed texture data is loaded to VRAM.

Table 3-23: `TexPlttSet::PlttInfo::flag` member values

Name	Value	Description
NNS_G3D_RESPLTT_LOADED	0x0001	This is set when Palette data loaded to VRAM.
NNS_G3D_RESPLTT_USEPLTT4	0x8000	Set when there is a four-color palette.

For the data in the dictionary `dictTex`, the data shown with the following pseudo-structure is stored, not the offset to the data.

```
pseudo_struct DictTexData(NNSG3dResDictTexData) {
      u32 texImageParam;
      u32 extraParam;
};
```

Table 3-24: Data stored in dictionary `dictTex`

Name	Description
texImageParam	31 30 29 28 26 25 23 22 20 19 16 15 0 | | P | Fmt | T | S | | OFS | Same layout as the parameters of the geometry command `TexImageParam`. However, the bits for flip, repeat and texture coordinates conversion mode are not set. *OFS* - Offset for `TexPlttSet::texData` or `TexPlttSet::tex4x4Data` shifted three bits to the right. *S* - Texture width *T* - Texture height *Fmt* - Texture format *P* - Palette color setting value enable flag
extraParam	31 30 22 21 11 10 0 | S | | OrigH | OrigW | *OrigW* - Texture width on tool *OrigH* - Texture height on tool *S* - 1 when `OrigW` and `OrigH` are the same as the width and height specified with `TexImageParam`.

Also, for data in the dictionary `dictPltt`, the data shown with the following pseudo-structure is stored, not the offset to the data.

```
pseudo_struct DictPlttData(NNSG3dResDictPlttData) {
    u16 offset;
    u16 flag;
};
```

Table 3-25: Data stored in dictionary `dictPltt`

Name	Description
offset	Offset for `TexPlttSet::plttData` shifted three bits to the right
flag	`1` when 4-color palette. `0` when something other than 4-color palette.

3.3   Structure of Joint Animation Data File (`.nsbca`)

The `.nsbca` file stores joint animation sets. Each joint animation is associated with the first 16 characters of the character string with the file extension removed (filled with NULL characters when there are less than 16 characters) from file name of .ica file. The `.nsbca` file format is explained below using pseudo-structure.

```
pseudo_struct NSBCA {
    FileHeader file_header = {
        dataBlocks = 1,
        signature = '0ACB'
    };
    JointAnmSet jntAnmSet;
};
pseudo_struct JointAnmSet(NNSG3dResJntAnmSet) {
    DataBlockHeader header = {
        kind = '0TNJ',
        size = SIZEOF(JointAnmSet)
    };
    Dictionary dict = {sizeUnit = 4 bytes};
    JointAnm jntAnm[dict.numEntry];
    pseudo_struct JointAnmRot3 {
        u16  info;
        fx16  A;
        fx16  B;
    } rot3[];
    PADDING(4 bytes alignment);
```

(continued on next page...)

```
    pseudo_struct JointAnmRot5 {
        fx16 data[5];
    } rot5[];
    PADDING(4 bytes alignment);


    u16 rotIdx[];
    PADDING(4 bytes alignment);


    pseudo_struct JointAnmScaleFx16 {
        fx16  scale, invScale;
    } scaleFx16[];
    PADDING(4 bytes alignment);


    fx16 transFx16[];
    PADDING(4 bytes alignment);


    pseudo_struct JointAnmScaleFx32 {
        fx32  scale, invScale;
    } scaleFx32[];


    fx32 transFx32[];
};
```

Table 3-26: **JointAnmSet Data Members**

Name	Description
header	Header region of joint animation block
dict	Dictionary region for accessing each joint animation
jntAnm	Main data of each joint animation
rot3	Rotation matrix data (6 bytes) array. Index referenced from rotIdx. 15 6 5 4 3 0 | — | SD | SC | M | IdxPivot | | A | | B | Meaning of data is same as that of NodeData. **idxPivot** - Shows location (0-8) of rotation matrix pivot element (element with absolute value of 1). **M** - When this bit is ON, pivot element is negative (-1) **SC** - If this bit is ON, C has opposite sign of B **SD** - If this bit is ON, D has opposite sign of A **A**, **B** - Rotation matrix elements

rot5	Rotation matrix data (10 bytes) array. Index referenced from `rotIdx`. The absolute value of each element value decreases by one.

```
15                                                            2           0
```

_00	_12(9-11)

_01	_12(6-8)

_02	_12(3-5)

_10	_12(0-2)

_11	_12(sign)

	`_00`, `_01`, `_02`, `_10`, `_11` - The rotation matrix element. Does 3 bit arithmetic shift to the right, and uses as `fx16`-type value.

`_12` (…) - The element in the 2nd row, third column of the rotation matrix is broken up and stored.

The third row can be obtained using the cross product. |
| rotIdx | Array storing index to rotation matrix data

The lower 15 bits are index, and `rot3` or `rot5` is selected with the highest 1 bit. References `rot3` when `1`. References `rot5` when `0`. References using the offset from `jntAnm`. |
| scaleFx16 | Data string storing scale component (`fx16`)

Referenced using the offset from `jntAnm`. |
| transFx16 | Array storing translation component (`fx16`)

Referenced using the offset from `jntAnm`. |
| scaleFx32 | Data string storing scale component (`fx32`)

Referenced using the offset from `jntAnm`. |
| transFx32 | Array storing translation component (`fx32`)

Referenced using the offset from `jntAnm`. |

The format of individual joint animation expressed as pseudo-structure is as follows. Internally holds the offsets for actual data string for scale, rotation and translation component.

```
pseudo_struct JointAnm(NNSG3dResJntAnm) {
    AnmHeader anmHeader = {
        category0 = 'J',
        category1 = 'CA'
    };
    u16 numFrame;
    u16 numNode;
    u32 annFlag;
    u32 ofsRot3;
    u32 ofsRot5;
    u16 ofsTag[numNode];
    PADDING(4 bytes alignment);
    pseudo_struct TagData(NNSG3dJntAnmSRTTag) {
        u32 flag;
        IF (!(flag & NNS_G3D_JNTANM_SRTINFO_IDENTITY)) {
            IF (!(flag & NNS_G3D_JNTANM_SRTINFO_IDENTITY_T) &&
            !(flag & NNS_G3D_JNTANM_SRTINFO_BASE_T)) {
                JointAnmTrans<flag & NNS_G3D_JNTANM_SRTINFO_CONST_TX> tx;
                JointAnmTrans<flag & NNS_G3D_JNTANM_SRTINFO_CONST_TY> ty;
                JointAnmTrans<flag & NNS_G3D_JNTANM_SRTINFO_CONST_TZ> tz;
            }
            IF (!(flag & NNS_G3D_JNTANM_SRTINFO_IDENTITY_R) &&
            !(flag & NNS_G3D_JNTANM_SRTINFO_BASE_R)) {
                JointAnmRot<flag & NNS_G3D_JNTANM_SRTINFO_CONST_R> r;
            }
            IF (!(flag & NNS_G3D_JNTANM_SRTINFO_IDENTITY_S) &&
            !(flag & NNS_G3D_JNTANM_SRTINFO_BASE_S)) {
                JointAnmScale<flag & NNS_G3D_JNTANM_SRTINFO_CONST_SX> sx;
                JointAnmScale<flag & NNS_G3D_JNTANM_SRTINFO_CONST_SY> sy;
                JointAnmScale<flag & NNS_G3D_JNTANM_SRTINFO_CONST_SZ> sz;
            }
        }
    } tagData[numNode];
};
```

(continued on next page...)

(... continued from previous page)

```
pseudo_struct JointAnmTrans<isConst> {
    IF (isConst) {
        fx32 const_trans;
    } ELSE {
        u32 info;
        u32 offset;
    }
};


pseudo_struct JointAnmRot<isConst> {
    IF (isConst) {
        u32 const_offset
    } ELSE {
        u32 info;
        u32 offset;
    }
};


pseudo_struct JointAnmScale<isConst> {
    IF (isConst) {
        fx32 const_scale;
        fx32 const_invScale;
    } ELSE {
        u32 info;
        u32 offset;
    }
};
```

Table 3-27: `JointAnm` **Data Members**

Name	Description		
anmHeader	Animation header		
numFrame	Number of animation frames		
numNode	Number of nodes in the model targeted for the joint animation		
anmFlag	Flag that specifies the joint animation option		
ofsRot3	Offset from start of `JointAnm` to rotation matrix data (6 bytes) array		
ofsRot5	Offset from the start of `JointAnm` to the rotation matrix data (10 bytes) array		
ofsTag	Offset from the start of `JointAnm` to `tagData` element corresponding to node		
tagData	flag	Flag group that determines data in each `tagData`
	tx	Data relating to x component of joint translation vector
	ty	Data relating to y component of joint translation vector
	tz	Data relating to z component of joint translation vector
	r	Data related to the rotation matrix of the joint
	sx	Data relating to joint x direction scale
	sy	Data relating to joint y direction scale
	sz	Data relating to joint z direction scale

Table 3-28: `JointAnmTrans` **Data Members**

Name	Description
const_trans	Value of constant translation component
info	Flag describing characteristics of `Translation` data string
offset	Offset from start of `JointAnm` to `Translation` data string

Table 3-29: `JointAnmRot` Data Members

Name	Description
const_offset	Index value to constant rotation matrix
info	Flag describing characteristics of `Rotation` data string
offset	Offset from start of `JointAnm` to `Rotation` data index string

Table 3-30: `JointAnmScale` Data Members

Name	Description
const_scale	Constant scale value
const_invScale	Inverse of constant scale value
info	Flag describing characteristics of `Scale` data string
offset	Offset from start of `JointAnm` to `Scale` data string

Table 3-31: Valid `JointAnm::TagData::flag` Values

Name	Value	Description
NNS_G3D_JNTANM_SRTINFO_IDENTITY	0x00000001	ON when there's no change in `SRT`
NNS_G3D_JNTANM_SRTINFO_IDENTITY_T	0x00000002	ON when no translation
NNS_G3D_JNTANM_SRTINFO_BASE_T	0x00000004	ON when using model value in `Trans`
NNS_G3D_JNTANM_SRTINFO_CONST_TX	0x00000008	ON when `Tx` is constant
NNS_G3D_JNTANM_SRTINFO_CONST_TY	0x00000010	ON when `Ty` is constant
NNS_G3D_JNTANM_SRTINFO_CONST_TZ	0x00000020	ON when `Tz` is constant
NNS_G3D_JNTANM_SRTINFO_IDENTITY_R	0x00000040	ON when there is no rotation
NNS_G3D_JNTANM_SRTINFO_BASE_R	0x00000080	On when using model value in `Rot`
NNS_G3D_JNTANM_SRTINFO_CONST_R	0x00000100	ON when `Rot` is constant
NNS_G3D_JNTANM_SRTINFO_IDENTITY_S	0x00000200	ON when `Scale` not applied
NNS_G3D_JNTANM_SRTINFO_BASE_S	0x00000400	ON when using model value in `Scale`
NNS_G3D_JNTANM_SRTINFO_CONST_SX	0x00000800	ON when `Sx` is constant
NNS_G3D_JNTANM_SRTINFO_CONST_SY	0x00001000	ON when `Sy` is constant
NNS_G3D_JNTANM_SRTINFO_CONST_SZ	0x00002000	ON when `Sz` is constant
NNS_G3D_JNTANM_SRTINFO_NODE_MASK	0xFF000000	Mask the location where the node ID targeting the animation is entered

Table 3-32: Valid `JointAnmTrans::info` Values

Name	Value	Description
NNS_G3D_JNTANM_TINFO_STEP_1	0x00000000	When there is data every frame
NNS_G3D_JNTANM_TINFO_STEP_2	0x40000000	ON when frame step is 2
NNS_G3D_JNTANM_TINFO_STEP_4	0x80000000	ON when frame step is 4
NNS_G3D_JNTANM_TINFO_FX16ARRAY	0x20000000	ON when animation data is `fx16` array
NNS_G3D_JNTANM_TINFO_LAST_INTERP_MASK	0x1FFF0000	When `frameStep`=2,4: ($numFrame - 1$) & ~($frameStep - 1$) When `frameStep`=1: `numFrame`

Table 3-33: Valid `JointAnmRot::info` Values

Name	Value	Description
NNS_G3D_JNTANM_RINFO_STEP_1	0x00000000	When there is data every frame
NNS_G3D_JNTANM_RINFO_STEP_2	0x40000000	ON when frame step is 2
NNS_G3D_JNTANM_RINFO_STEP_4	0x80000000	ON when frame step is 4
NNS_G3D_JNTANM_RINFO_LAST_INTERP_MASK	0x1FFF0000	When `frameStep`=2,4: (numFrame − 1) & ~(frameStep − 1) When `frameStep`=1: numFrame

Table 3-34: Valid `JointAnmScale::info` Values

Name	Value	Description
NNS_G3D_JNTANM_SINFO_STEP_1	0x00000000	When there is data every frame
NNS_G3D_JNTANM_SINFO_STEP_2	0x40000000	ON when frame step is 2
NNS_G3D_JNTANM_SINFO_STEP_4	0x80000000	ON when frame step is 4
NNS_G3D_JNTANM_SINFO_FX16ARRAY	0x20000000	ON when animation data is `fx16` array
NNS_G3D_JNTANM_SINFO_LAST_INTERP_MASK	0x1FFF0000	When `frameStep`=2,4: (numFrame − 1) & ~(frameStep − 1) When `frameStep`=1: numFrame

3.4   Structure of Texture Pattern Animation Data File (`.nsbtp`)

The `.nsbtp` file stores sets of texture pattern animation data. Each texture animation is associated with the first 16 characters (filled with NULL characters when there are fewer than 16 characters) of the character string with the file extension removed from the file name of the `.itp` file.

Below is an explanation of `.nsbtp` file format using pseudo-structure.

```
pseudo_struct NSBTP {
    FileHeader file_header = {
        dataBlocks = 1,
        signature = '0PTB'
    };
    TexPatAnmSet texPatAnmSet;
};
pseudo_struct TexPatAnmSet(NNSG3dResTexPatAnmSet) {
    DataBlockHeader header = {
        kind = '0TAP',
        size = SIZEOF(TexPatAnmSet)
    };
    Dictionary dict = {sizeUnit = 4 bytes};
    TexPatAnm texPatAnm[dict.numEntry];
};
```

Table 3-35: `TexPatAnmSet` Data Members

Name	Description
header	Header region of texture pattern animation block
dict	Dictionary region for access to each texture pattern animation
texPatAnm	Main data of each texture pattern animation

```
pseudo_struct TexPatAnm(NNSG3dResTexPatAnm) {
   AnmHeader anmHeader = {
      category0 = 'M',
      category1 = 'TP'
   };
   u16      numFrame;
   u8       numTex;
   u8       numPltt;
   u16      ofsTexName;
   u16      ofsPlttName;
   Dictionary  dict(sizeUnit = 8 bytes);

   pseudo_struct TexPatFV(NNSG3dResTexPatAnmFV) {
      u16  idxFrame;
      u8   idTex;
      u8   idPltt;
   } texPatFV[];

   pseudo_struct DictName(NNSG3dResName) {
      u8   name[16];
   } texName[numTex];

   pseudo_struct DictName(NNSG3dResName) {
      u8   name[16];
   } plttName[numPltt];
};
```

Table 3-36: `TexPatAnm` Data Members

Name	Description		
anmHeader	Animation header		
numFrame	Number of animation frames		
numTex	Total number of textures to animate		
numPltt	Total number of palettes to animate		
ofsTexName	Offset from start of `TexPatAnm` to `texName`		
ofsPlttName	Offset from start of `TexPatAnm` to `plttName`		
dict	Dictionary for accessing animation data from material name		
texPatFV	idxFrame	Frame number that switches to the following texture and palette	
	idTex	Index for `texName`	
	idPltt	Index for `plttName`	
texName	Array of texture names displayed by animation		
plttName	Array of palette names displayed by animation		

```
pseudo_struct DictTexPatAnmData(NNSG3dResDictTexPatAnmData) {
    u16  numFV;
    u16  flag;
    fx16 ratioDataFrame;
    u16  offset;
};
```

Table 3-37: Data stored in dictionary `TexPatAnm::dict`

Name	Description
numFV	Number of FV data
flag	15 1 0 `---` P *P*: If 1, palette animation does not exist.
ratioDataFrame	Number of FV data divided by number of frames Can be used as hint for looking up FV data from current frame.
offset	Offset to FV data arrangement (in `TexPatAnm::texPatFV`) using `TexPatAnm` as origin.

Texture pattern animation is a type of material animation, and switches the textures and palettes that belong to the specified material depending on the recorded FV (Frame-Value) data. Actual texture data does not exist in `.nsbtp` file, and only texture and palette name and information regarding the timing for switching are recorded.

3.5 Structure of Material Color Animation Data File (`.nsbma`)

The `.nsbma` file stores sets of material color animation. Each material color animation is associated with the first 16 characters (filled with NULL characters when there are fewer than 16 characters) of the character string with the file extension removed from the file name of the `.ima` file. Below is an explanation of `.nsbma` file format using pseudo-structure.

```
pseudo_struct NSBMA {
    FileHeader      file_header = {
        dataBlocks = 1,
        signature = '0AMB'
    };
    MatColAnmSet    matColAnmSet;
};


pseudo_struct MatColAnmSet(NNSG3dResMatCAnmSet) {
    DataBlockHeader  header = {
        kind = '0TAM',
        size = SIZEOF(MatColAnmSet)
    };
    Dictionary       dict = {sizeUnit = 4 bytes};
    MatColAnm        matColAnm[dict.numEntry];
};
```

Table 3-38: `MatColAnmSet` Data Members

Name	Description
header	Header region of material color animation block
dict	Dictionary region for access to each material color animation
matColAnm	Main data of each material color animation

```
pseudo_struct MatColAnm(NNSG3dResMatCAnm) {
   AnmHeader anmHeader = {
      category0 = 'M',
      category1 = 'MA'
   };
   u16 numFrame;
   u16 flag;
   Dictionary dict(sizeUnit = 20 bytes);
   u16 rgbData[];
   PADDING(4 bytes alignment);
   u8  alphaData[];
   PADDING(4 bytes alignment);
};
```

Table 3-39: `MatColAnm` Data Members

Name	Description
anmHeader	Animation header
numFrame	Number of animation frames
flag	Flag that specifies the material color animation option
dict	Dictionary for accessing animation data from material name
rgbData	RGB animation data
alphaData	Alpha animation data

Table 3-40: `MatColAnm::flag` Values

Name	Value	Description
NNS_G3D_MATCANM_OPTION_INTERPOLATION	0x0001	Interpolates color
NNS_G3D_MATCANM_OPTION_END_TO_START_INTERPOLATION	0x0002	Interpolates color from end frame to start frame (for loop)

The `DictMatColAnmData` pseudo-structure holds management information relating to color and alpha animation and is stored as a dictionary entry in `MatColAnm`. Definitions are shown below and explain the values which each data member takes using Table 3-41.

```
pseudo_struct DictMatColAnmData(NNSG3dResDictMatCAnmData) {
    u32 tagDiffuse;
    u32 tagAmbient;
    u32 tagSpecular;
    u32 tagEmission;
    u32 tagPolygonAlpha;
};
```

Table 3-41: `DictMatColAnmData` Data Members

Name	Description
tagDiffuse	Diffuse animation management information
tagAmbient	Ambient animation management information
tagSpecular	Specular animation management information
tagEmission	Emission animation management information
tagPolygonAlpha	Polygon Alpha animation management information

Table 3-42: `tagDiffuse/tagAmbient/tagSpecular/tagEmission/tagPolygonAlpha` Values

Name	Value	Description
NNS_G3D_MATCANM_ELEM_STEP_1	0x00000000	When there is data in every frame
NNS_G3D_MATCANM_ELEM_STEP_2	0x40000000	ON when frame step is 2
NNS_G3D_MATCANM_ELEM_STEP_4	0x80000000	ON when frame step is 4
NNS_G3D_MATCANM_ELEM_CONST	0x20000000	If ON, lower 16 bits are treated as constant data
NNS_G3D_MATCANM_ELEM_LAST_INTERP_MASK	0x1FFF0000	When $\texttt{frameStep}=2,4$: $(\texttt{numFrame}-1)$ & $\sim(\texttt{frameStep}-1)$ When $\texttt{frameStep}=1$: numFrame
NNS_G3D_MATCANM_ELEM_OFFSET_CONSTANT_MASK	0x0000FFFF	Offset to string using MatColAnm as origin, or constant value

3.6   Structure of Visibility Animation Data File (`.nsbva`)

The .nsbva file stores sets of visibility animation data. Each visibility animation is associated with the first 16 characters (filled with NULL characters when there are less than 16 characters) of the character string with the file extension removed from the file name of the .iva file. Following is an explanation of .nsbva file format using pseudo-structure.

```
pseudo_struct NSBVA {
   FileHeader      file_header = {
      dataBlocks = 1,
      signature = '0AVB'
   };
   VisAnmSet       visAnmSet;
};


pseudo_struct VisAnmSet(NNSG3dResVisAnmSet) {
   DataBlockHeader  header = {
      kind = '0SIV',
      size = SIZEOF(MatColAnmSet)
   };
   Dictionary      dict = {sizeUnit = 4 bytes};
   VisAnm          visAnm[dict.numEntry];
};
```

Table 3-43: `VisAnmSet` Data Members

Name	Description
header	Header region of visibility animation block
dict	Dictionary region for accessing each visibility animation
visAnm	Main data of each visibility animation

```
pseudo_struct VisAnm(NNSG3dResVisAnm) {
   AnmHeader anmHeader = {
      category0 = 'V',
      category1 = 'VA'
   };
   u16 numFrame;
   u16 numNode;
   u16 size = SIZEOF(VisAnm);
   PADDING(2 bytes);
   u32 visData[1 + (numFrame * numNode >> 5)];
};
```

Table 3-44: `VisAnm` Data Members

Name	Description
anmHeader	Animation header
numFrame	Number of animation frames
numNode	Number of nodes in the model to be animated
size	Size of `VisAnm` pseudo-structure
visData	Visibility animation data Each bit corresponds to the information about whether the node is visible or not. When little-endian, visibility information of `CurFrame` frame's `nodeID` node is stored in the bit at the position `CurFrame * numNode + nodeID`.

3.7　　　Structure of Texture SRT Animation Data File (`.nsbta`)

The `.nsbta` file stores sets of texture SRT animation data. Each texture SRT animation is associated with the first 16 characters (filled with NULL characters when there are less than 16 characters) of the character string with the file extension removed from the file name of the `.ita` file. Below is an explanation of `.nsbta` file format using pseudo-structure.

```
pseudo_struct NSBTA {
   FileHeader     file_header = {
      dataBlocks = 1,
      signature = '0ATB'
   };
   TexSRTAnmSet    texSRTAnmSet;
};


pseudo_struct TexSRTAnmSet(NNSG3dResTexSRTAnmSet) {
   DataBlockHeader  header = {
      kind = '0TRS',
      size = SIZEOF(TexSRTAnmSet)
   };
   Dictionary      dict = {sizeUnit = 4 bytes};
   TexSRTAnm       texSRTAnm[dict.numEntry];
};
```

Table 3-45: `TexSRTAnmSet` Data Members

Name	Description
header	Header region of visibility animation block
dict	Dictionary region for accessing each texture SRT animation
texSRTAnm	Main data of each texture SRT animation

```
pseudo_struct TexSRTAnm(NNSG3dResTexSRTAnm) {
   AnmHeader  anmHeader = {
      category0 = 'M',
      category1 = 'TA'
   };
   u16        numFrame;
   u8         flag;
   u8         texMtxMode;
   Dictionary  dict(sizeUnit = 40 bytes);
   u32        anmData[];
};
```

Table 3-46: `TexSRTAnm` Data Members

Name	Description
anmHeader	Animation header
numFrame	Number of animation frames
flag	Flag that specifies texture SRT animation option
texMtxNode	Texture matrix calculation method Refer to Table 3-10: Values Taken by `texMtxMode`
dict	Dictionary for accessing animation data from material name
anmData	Various numerical data strings referenced from DictTexSRTAnmData

Table 3-47: `TexSRTAnm::flag` Values

Name	Value	Description
NNS_G3D_TEXSRTANM_OPTION_INTERPOLATION	0x0001	Interpolation play specification
NNS_G3D_TEXSRTANM_OPTION_END_TO_START_INTERPOLATION	0x0002	Interpolates from end frame to start frame (for loop)

```
pseudo_struct DictTexSRTAnmData(NNSG3dResDictTexSRTAnmData) {
    u32 scaleS;
    u32 scaleSEx;
    u32 scaleT;
    u32 scaleTex;
    u32 rot;
    u32 rotEx;
    u32 transS;
    u32 transSEx;
    u32 transT;
    u32 transTEx;
};
```

Table 3-48: `DictTexSRTAnmData` Data Members

Name	Description
scaleS	Texture scale animation management information (vertical direction)
scaleSEx	Constant value (immediate) when NNS_G3D_TEXSRTANM_ELEM_CONST is set in scaleS Otherwise, the offset from the start of TexSRTAnm to data string.
scaleT	Texture scale animation management information (vertical direction)
scaleTEx	Constant value (immediate) when NNS_G3D_TEXSRTANM_ELEM_CONST is set in ScaleT Otherwise, the offset from the start of TexSRTAnm to data string.
rot	Texture rotation animation management information
rotEx	Constant value (immediate) with sin and cos packed when NNS_G3D_TEXSRTANM_ELEM_CONST is set in rot (upper 16 bits are fx16 cos value, lower 16 bits are fx16 sin value) Otherwise, the offset from the start of TexSRTAnm to the data string.
transS	Texture translation animation management information (horizontal direction)
transSEx	Constant value (immediate) when NNS_G3D_TEXSRTANM_ELEM_CONST is set in transS Otherwise, the offset from the start of TexSRTAnm to the data string.
transT	Texture translation animation management information (vertical direction)
transTEx	Constant value (immediate) when NNS_G3D_TEXSRTANM_ELEM_CONST is set in transT Otherwise, the offset from the start of TexSRTAnm to the data string.

Table 3-49: `scaleS/scaleT/rot/transS/transT` Values

Name	Value	Description
NNS_G3D_TEXSRTANM_ELEM_STEP_1	0x00000000	ON when there is data in every frame
NNS_G3D_TEXSRTANM_ELEM_STEP_2	0x40000000	ON when frame step is 2
NNS_G3D_TEXSRTANM_ELEM_STEP_4	0x80000000	ON when frame step is 4
NNS_G3D_TEXSRTANM_ELEM_CONST	0x20000000	If ON, lower 16 bits are treated as constant data
NNS_G3D_TEXSRTANM_ELEM_FX16	0x10000000	If ON, holds data in fx16 array
NNS_G3D_TEXSRTANM_ELEM_LAST_INTERP_MASK	0x0000FFFF	When `frameStep`=2,4: (`numFrame` − 1) & ~(`frameStep` − 1) When `frameStep`=1: `numFrame`

Maya is a registered trademark or trademark of the Alias Systems Corporation.

Avid, Softimage, SOFTIMAGE|3D, and SOFTIMAGE|XSI are a registered trademark or trademark of Avid Technology, Inc.

Discreet and 3dsmax are a registered trademark or trademark of Autodesk, Inc./Autodesk Canada, Inc.

Other company names, product names, etc. are the registered trademarks or trademarks of each company.