HMD Format

© 2001 Sony Computer Entertainment Inc.

Publication date: October 2001

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052, Japan

Sony Computer Entertainment America
919 E. Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.

The *HMD Format* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® license agreements.

The *HMD Format* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® license agreements.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® license agreements.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *HMD Format* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

and PlayStation are registered trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

Table of Contents

About This Manual

This is the Runtime Library Release 2.4 version of the *HMD Format* manual.

It describes HMD, which is a generic graphics format that allows model data, texture data, and animation data to be handled all within an integrated framework.

Changes Since Last Release

None

Related Documentation

Note: the Developer Support Web site posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Typographic Conventions

Certain Typographic Conventions are used throughout this manual to clarify the meaning of the text:

Convention	Meaning
`courier`	Indicates literal program code.
italic	Indicates names of arguments and structure members (in structure/function definitions only).
medium bold	Indicates data types and structure/function names (in structure/function definitions only).
blue	Indicates a hyperlink.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America:	*In North America:*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd. Foster City, CA 94404, U.S.A. Tel: (650) 655-8000	E-mail: PS2_Support@playstation.sony.com Web: http://www.devnet.scea.com/ Developer Support Hotline: (650) 655-5566 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In Europe:	*In Europe:*
Attn: Production Coordinator Sony Computer Entertainment Europe 30 Golden Square London W1F 9LD, U.K. Tel: +44 (0) 20 7859-5000	E-mail: ps2_support@scee.net Web: https://www.ps2-pro.com/ Developer Support Hotline: +44 (0) 20 7859-5777 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT)

Introduction

This document is being released as part of the PlayStation Programmer Tools Runtime Library. It is being released as a reference for PlayStation 2 graphics so its contents have not been updated. In addition, please note that the descriptions of library versions have not been revised in this document.

HMD Overview

HMD is a generic graphics format that allows model data, texture data, and animation data to be handled all within an integrated framework.

HMD can be easily extended to handle additional kinds of data with a unique identification code known as a type.

HMD data can be easily played back on the PlayStation using libhmd. A program that is used to playback HMD-formatted data is referred to as a primitive driver. Primitive drivers are linked to HMD data through their type.

Sony Computer Entertainment has created a set of standard primitive drivers for libhmd. These primitive drivers have standardized interfaces or APIs, so end users and middleware companies can also build their own primitive drivers.

The HMD format is supported by Library Version 4.0 and later.

Notes on HMD library version 4.3 and later

In previous versions, libhmd was provided as part of the libgs and libgte libraries, but it is now offered as a separate library. HMD-related functions, which were part of libgs and libgte in PlayStation library 4.2 and earlier, are now available separately. Consequently, HMD-related functions and declarations have been removed from the libgs and libgte libraries, and from the corresponding header files. The HMD library (libhmd) should now be used along with libhmd.h for HMD-related functions.

The environment map is provided only as a Beta version with this release. This is because future releases may introduce format changes that are not compatible with the current release. The Beta version primitives are currently not supported by SCE and should be used only at the licensee's discretion.

Abstract of the HMD (for All categories)

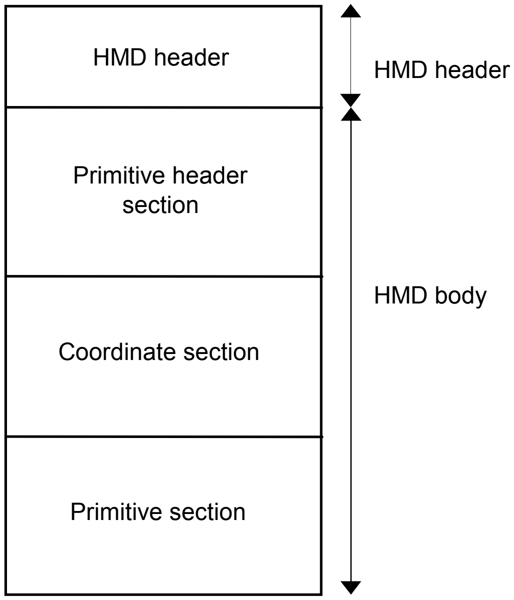
The HMD format supports several categories of data. Examples of categories include model data and image data. Each category can have its own individual data format. This chapter describes the HMD structures that are the same across all categories of data.

HMD data is divided into the following two main parts.

1. The HMD header
2. The HMD body

The HMD body is made up of areas known as sections. Two sections, one called the primitive section and the other known as the primitive header section, are always required. Other sections are included only if required by the specific type.

Figure 1:HMD Structure



In the example shown above, a coordinate section is included in addition to the required sections.

The following is a detailed description of the HMD format.

Notations

In this discussion, pointer values, which represent addresses, are converted at runtime into real addresses in memory. The process of converting pointer values to real addresses is known as mapping and is performed by the GsMapUnit() function.

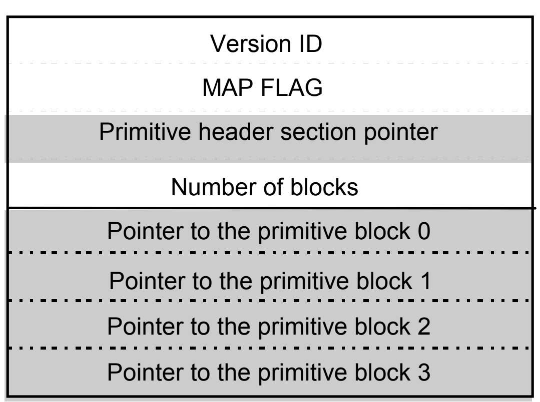
HMD data can be used only after addresses are mapped.

Pointers are shown highlighted in the figures. The initial value of a pointer is the number of words from the top of HMD data, where one word is equal to 32 bits.

HMD Header

The HMD header contains the version ID, MAP FLAG, the starting address of the primitive header section, and the number of primitive blocks. Primitive blocks and the primitive header section will be described in more detail later. The HMD header also contains a list of pointers to the primitive blocks.

Figure 2: HMD Header section



Version ID - 0x00000050

MAP FLAG - A flag that is used to indicate if the GsMapUnit() function has been called or not. The GsMapUnit() references this field and changes it. This value is 1 if mapped, otherwise 0.

Number of blocks - The number of primitive blocks pointers.

HMD Data

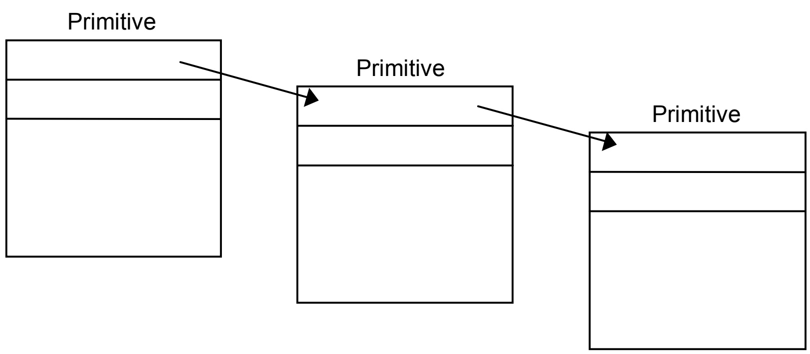
Primitive Section

A primitive section is defined to be a collection of primitive blocks.

Primitive Block

A primitive block is defined to be a chain of one or more primitives linked together by pointers. HMD data consists of one or more primitive blocks.

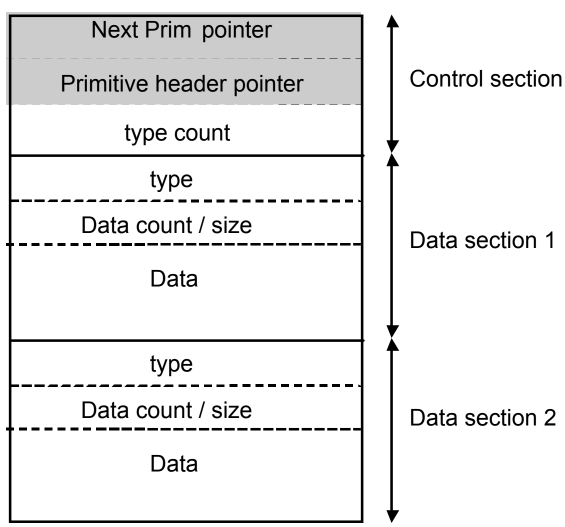
Figure 3: One primitive block which has been primitive chained



The Structure of a Primitive

Primitives consist of a control section and one or more data sections.

Figure 4: Primitive Structure



Control Section

- Next Prim pointer: Pointer to the header of the next primitive, thus forming the primitives chain. A value of 0xFFFFFFFF indicates that this is the last primitive in the chain.
- Primitive header pointer: Pointer to the primitive header. Primitive headers will be described later.
- Number of types: Number of data sections. The MSB serves as a flag indicating whether the NextPrim pointer and the primitive header pointer have been mapped. The MSB of the type count is 1 if UNMAPped, and 0 if MAPped.

Data Section

- type: Identifier of the data. Type is overwritten during a SCAN operation with the starting address of the driver used to process the data. Each type is unique within HMD. If the value of type is changed, the contents of the data and its driver can also be changed. SCAN and type will each be described in more detail later.
- Number of data / Size: The upper 16 bits of this field contain the data count for a single data section. A single type generally processes multiple sets of data. The data count indicates how many sets of data there are to process. In other words, how many times the data process will be repeated. The lower 16 bits contain the size of one data section in words.
- Data: The actual data is placed here. The data format depends on the value of the type field.

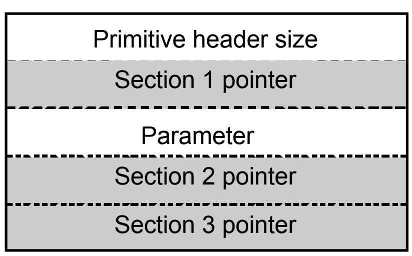
Primitive header

Primitive headers are grouped together and placed in the primitive header section. A pointer to the primitive header section is saved in the HMD header.

The first word of the primitive header structure is the size of the primitive header in words. Pointers to each of the sections follow. There is one primitive header for each primitive block. Within the primitive header are pointers to the sections referred to by the primitive block.

Setting the MSB of its data word to 1 identifies a pointer to a section. When the MSB is 0, the data is interpreted as a numeric value rather than as an address pointer. These unmapped values can be used as parameters for a primitive driver.

Figure 5: Primitive Header

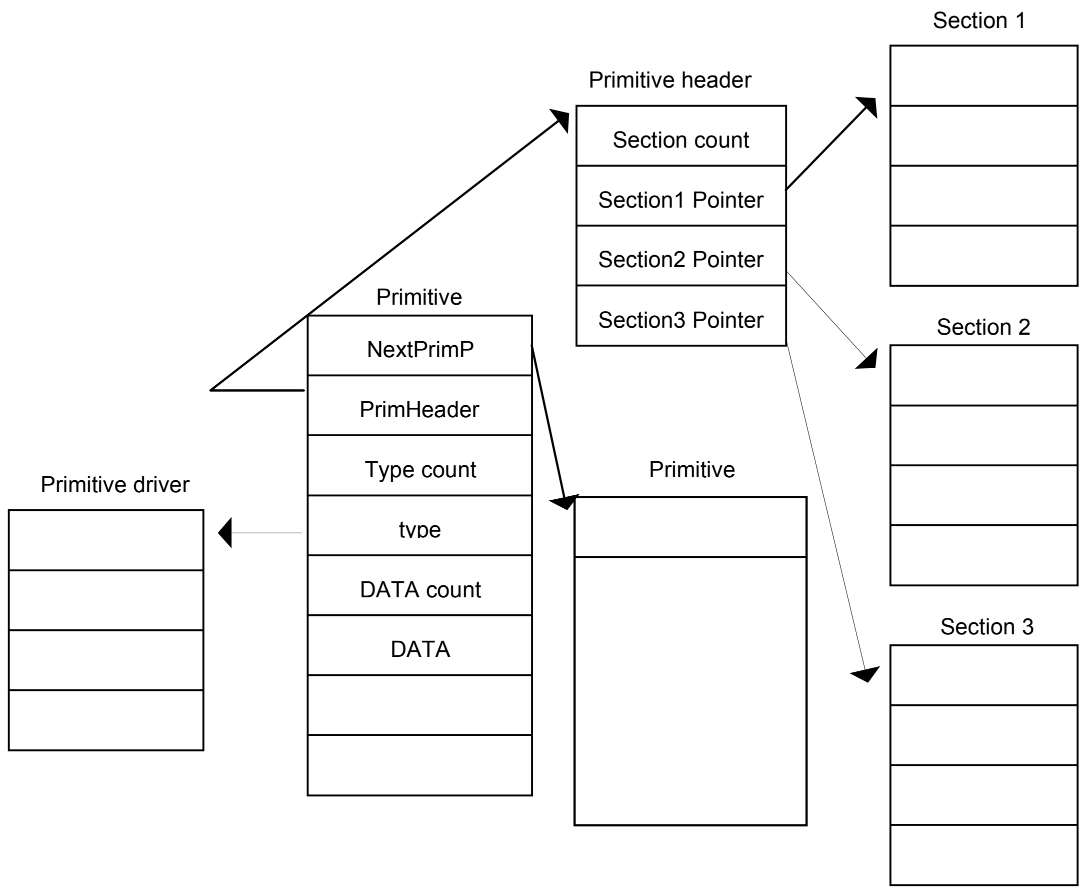


Basic structure of a primitive

A primitive is made up of three components: the primitive header, the primitive driver, and its data. The figure below shows the relationship between these components. The primitive header contains pointers to the beginning of the corresponding data sections. The data sections, which the pointers refer to depend on the type of primitive.

The primitive data and its corresponding sections are evaluated together by the primitive driver. The primitive driver is identified by the type field, which is overwritten, with the starting address of the driver. This process is known as a SCAN. SCAN uses the GsScanUnit() function to extract the address of the type field and its value for each primitive. Then it can be replaced with the starting address of the primitive driver into the type field for each primitive.

Figure 6: Primitive Structure



The following sections give examples of each of the data categories for use with HMD.

HMD Model Data (Category 0)

The model data must contain the following sections:

1. HMD header section
2. Primitive header section
3. Coordinate section
4. Primitive section
5. Polygon section
6. Vertex section

The following sections can also be included:

1. Normal section
2. Image section

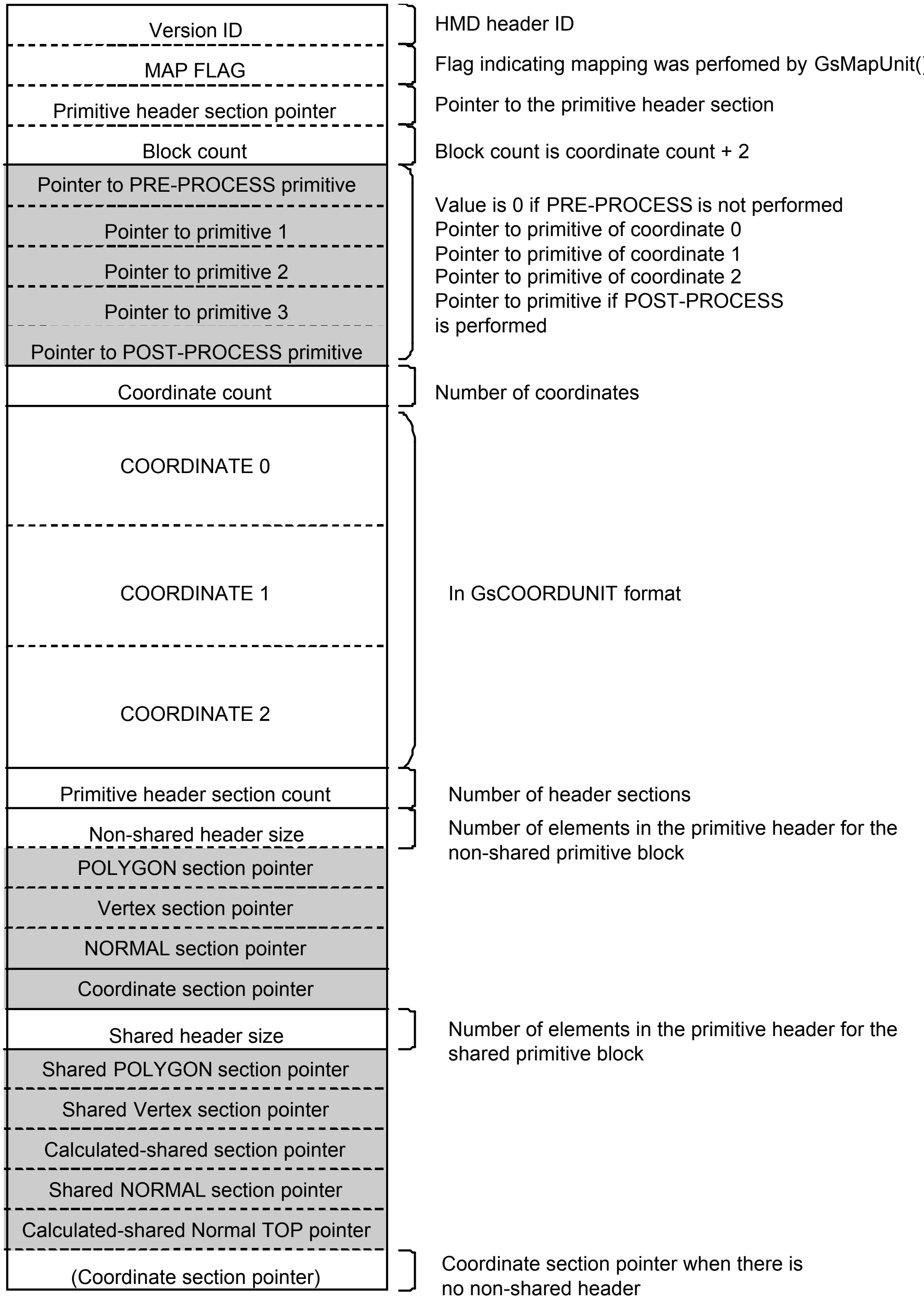
In HMD, model data consists of multiple coordinate systems and each coordinate system is assigned to a separate primitive block.

The HMD primitive header section contains a primitive header for each primitive block.

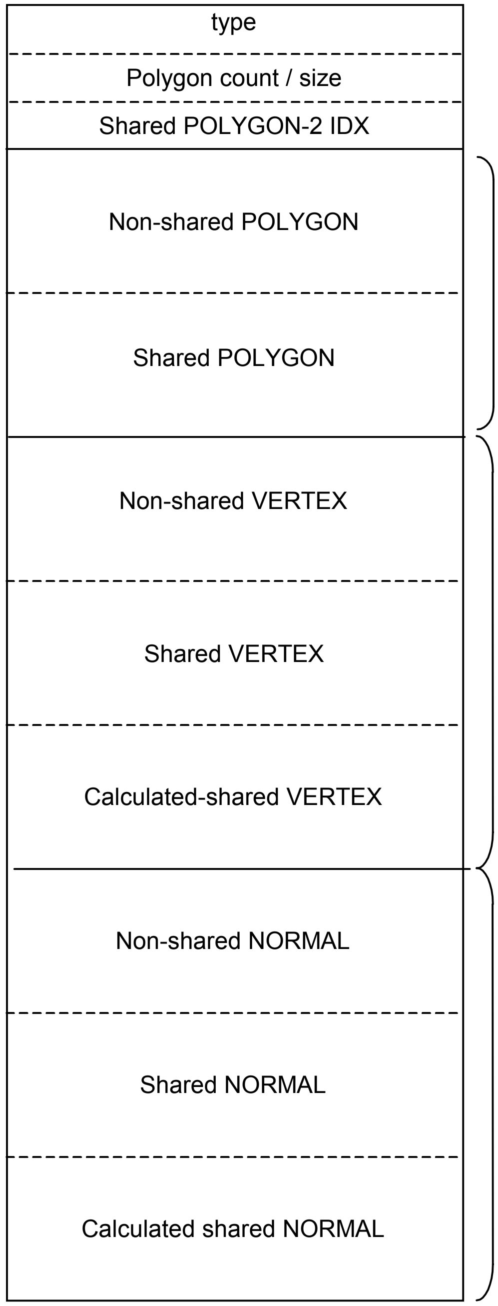
The following description is an example of the HMD format for model data.

Overall Structure

Figure 7: The structure of shared vertices HMD



NEXT Prim pointer	Pointer to the next primitive. The calculation process for the shared primitive's VERTEX and NORMAL is defined by the next chain of the non-shared primitive.
Non-shared header pointer	Pointer to primitive header of non-shared vertex
type count	Number of types
type	TYPE
Polygon count / size	The number of polygons and size for this type
POLYGON IDX	Index into the primitive type's polygon section
type	
Polygon count / size	
POLYGON IDX	
TERMINATE	TERMINATE indicates that this is the last primitive
Shared header pointer	
type count	MSB of the type count is a flag indicating map completed
type	Shared primitive type
Polygon count / size	
Shared VERTEX count	Offset specifies the number of words from the start of the shared VERTEX. Two buffers for input and output are independently defined allowing shared primitives to be reused.
Shared VERTEX offset (src)	
Shared VERTEX offset (dst)	
Shared NORMAL count	
Shared NORMAL offset (src)	
Shared NORMAL offset (dst)	
TERMINATE	This shared primitive is defined as the POST-PROCESS primitive. The header is the same as that for the shared primitive.
Shared header pointer	
type count	
type	
Polygon count / size	
Shared POLYGON IDX	



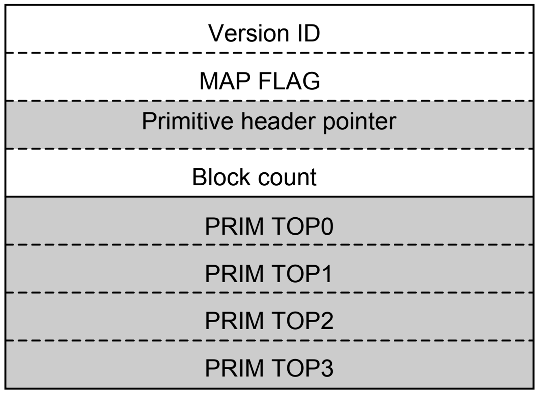
Polygon section
Connectivity data for POLYGONs (known as a PACKET) is placed here. The format (PACKET FORMAT) is described below.

Vertex section
The VERTEX and NORMAL sections are positioned such that non-shared, shared, and calculated-shared entities are arranged continuously.
This arrangement means a non-shared primitive can also scope a shared vertex.

NORMAL section

HMD Header Section

Figure 8: HMD Header Section



Version ID - Version number of the HMD format. Currently 0x00000050.

MAP FLAG - Flag indicating whether mapping was performed. This flag is accessed and updated by GsMapUnit(). This value is 0x00000000 if MAPped, and 0x00000001 if UNMAPped.

Primitive header top - Pointer to primitive header section (offset value from top, in words) MSB is 1 when data in the primitive header section has been mapped using GsMapUnit().

Block count - Number of blocks. There is 1 block per coordinate as well as a PRE-PROCESS block and a POST-PROCESS block. Therefore the block count is equal to the number of coordinates + 2.

Primitive pointer table - Contains a pointer to the primitive in each block. The first block is used for PRE-PROCESS and does not have a coordinate. The next blocks correspond to indexes from the coordinate tops. The last block is used for POST-PROCESS and does not have a coordinate.

Table 1: Primitive Pointer Table

Block	Coordinate	Primitive	Process
BLOCK 0		PRIM 0	PRE-PROCESS
BLOCK 1	COORDINATE0	PRIM 1	Block 1 process
BLOCK 2	COORDINATE1	PRIM 2	Block 2 process
BLOCK 3	COORDINATE2	PRIM 3	Block 3 process
BLOCK N	COORDINATE N-1	PRIM N	Block N process
BLOCK N+1		PRIM N+1	POST-PROCESS

COORDINATE Section

The coordinate section contains coordinate system data for each block.

The first word of the coordinate section indicates the number of coordinates.

Coordinates are represented in GsCOORDUNIT format as shown below.

```
GsCOORDUNIT {
unsigned long flg;
MATRIX matrix;
MATRIX workm;
SVECTOR rot;
struct GsCOORDUNIT *super;
}
```

The consistency between rot and matrix must be maintained during construction of HMD data.

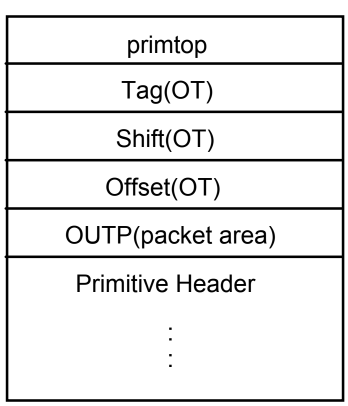
Primitive Header Section

The primitive header section contains a collection of primitive headers and global data for the primitive block. When a primitive driver is called, GsSortUnit() copies the data shown below to a variable transfer area. The size of the copied data is saved in the header size.

This process enables the primitive driver to access data in the primitive header. Since the primitive header contains pointers to normal and vertex section headers, the driver is able to access data in these sections.

The MSB of the data denotes whether or not the value will be mapped. If the value will not be mapped (MSB = 0), it is considered to be an ordinary number. If it will be mapped (MSB = 1), the value is treated as a pointer.

Figure 9: Variable transfer area transferred to the primitive driver



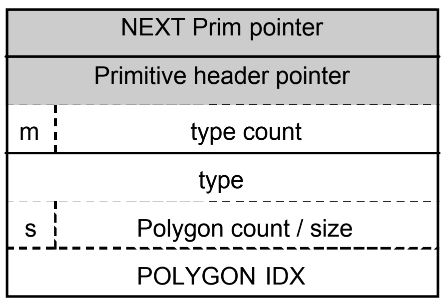
Primitive Section

The primitive section contains one or more primitive blocks. Each block corresponds to one coordinate. For model data, the first primitive is used for non-shared vertex data and the next block is a primitive that is used for shared vertex data. If non-shared vertex primitives and shared vertex primitives are not present in the model data, this section can be omitted.

Primitives

As shown below, primitives consist of several types of data.

Figure 10: One Primitive



The first three words in a primitive specify the control section. This section is made up of a NextPrim pointer, a primitive header pointer, and a type count. The MSB of the type count(m) serves as a flag indicating whether or not the NextPrim pointer and the primitive header pointer have been mapped. If m=1, the pointers have not been mapped. Conversely, if m=0, the pointers have been mapped.

The data section of a primitive is organized in three-word units. Each unit is made up of a type field, which serves as an identifying code, the number of polygons in the data for this type, and POLYGON IDX, which is a pointer to the actual polygon data. These three words are repeated according to the number of types in the control section.

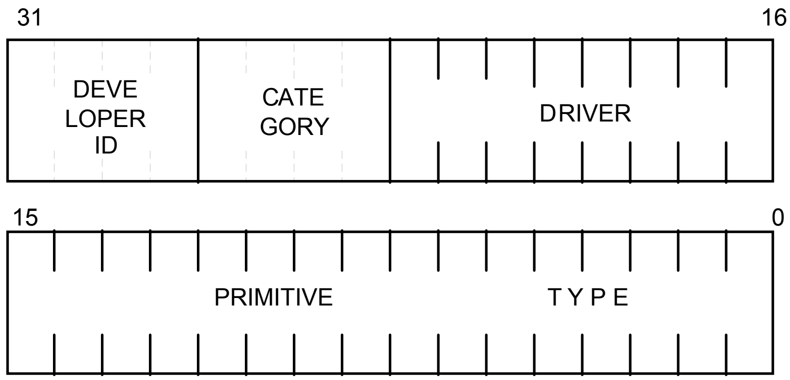
The MSB of the polygon count is a flag indicating whether a SCAN operation has been performed. The lower 16 bits indicate the size of the data contained in the type.

The upper 8 bits (n) of POLYGON IDX can be used as a parameter for the primitive driver. In the current implementation, DIV and ADV in the polygon data DRIVER bits (category 0) are used to control the number of polygon divisions. DIV stores the actual number of divisions (fixed divisions), while ADV stores the maximum number of divisions (automatic division). The allowed values of DIV and ADV are defined in libgs.h as GsUNIT_DIV1 - GsUNIT_DIV5. When using DIV or ADV, it is not advisable to set any other values to n. In particular, it is important to note that if the value is set to 0, the primitive driver will not function.

type

The type field consists of 32 bits arranged in four sections. The upper 8 bits contain data that is common to all categories.

Figure 11: Type Field



Common to all Categories

- DEVELOPER ID: Contains the ID of the developer who created this format. If ID is unique, the developer may use the other bits freely. A total of 16 IDs are available. The value zero is assigned to Sony Computer Entertainment.

  All 16 ID
  0x0: SCE
  0xf: User defined

- CATEGORY: This identifies the category of data such as polygon data or image data.
  16 categories are available.

  0: Polygon data
  1: Shared polygon data
  2: Image data
  3: Animation data
  4: MIMe data
  5: Ground data

Polygon Data (Category 0)

DRIVER

These bits can be used to change the behavior of the primitive driver for a given type of primitive data. For example, polygon subdivision can be enabled. 8 bits are available.

Figure 12: Polygon Primitive Driver

INI		STP	BOT	ADV	LGT	FOG	DIV

DIV
0: Disable subdivision
1: Perform subdivision

FOG
0: Turn FOG OFF
1: Turn FOG ON

LGT
0: Perform light-source calculation
1: Disable light-source calculation (forcibly mask off light-source calculation during execution)

ADV
0: Do not perform automatic division
1: Perform automatic division

BOT
0: Single-sided polygon
1: Double-sided polygon

STP
0: (Make semi-transparent if already semi-transparent. Make opaque if already opaque)
1: make all polygons semi-transparent

INI
0: Do not initialize
1: Initialize

When initialization is specified, an initialization function is called to set up the environment before SCAN is performed. In some cases, this bit is set when a type is first used.

PRIMITIVE TYPE

The value of these bits depends on the type of primitive.

Figure 13: Primitive Type of Polygon Primitive

						TILE	PST	MIP	LMD		CODE		IIP	COL	TME

TME
0: Disable texture mapping
1: Perform texture mapping

COL
0: Use one material color for identical polygons
1: Use a separate color for each vertex

IIP
0: Flat-shaded polygon
1: Gouraud-shaded polygon

CODE - Describes the shape of the polygon
0: Reserved by the system
1: Triangle
2: Quadrangle
3: Strip mesh
4-7: Reserved by the system

LMD
0: Has normal
1: Does not have normal

MIP - not implemented)
0: Disable MIP-mapping
1: Perform MIP-mapping

PST
0: No presets
1: Preset packet available

TILE
0: No information for tiled textures
1: Information available for tiled textures

MIMe
0: Normal polygon
1: MIMe polygon (not implemented)

Number of polygons / Size

Figure 14: Number and Size of Polygons

flg	Polygon count	Data size (in words)

flg - flag indicating whether or not SCAN was performed
0: SCAN was performed
1: SCAN was not performed

Number of polygons - Number of polygons in type.

Data size_@ - Size of data in type (in words)

Polygon Section

The polygon section contains polygon connection information. PACKETs are used to represent this information and are classified according to type.

A PACKET has NORMAL and VERTEX fields that are referenced by an index, and an RGB field that contains actual values.

The polygon type can be one of the following shapes:

1. Triangle
2. Quadrangle
3. MESH

For MESH, the first num field specifies the number of connections.

A list of PACKETs by type is shown below.

The type of polygon is shown at the upper left, and the value of the type field is shown at the upper right. The contents of the PACKET are drawn as a series of rows, with each row representing one word (32 bits). The meaning of the symbols shown is basically the same as that for TMD.

Polygon Types

With Light-source Calculation

```
Flat No-Texture Triangle
0x00000008; DRV(0)|PRIM_TYPE(TRI); GsUF3
B(r); B(g); B(b); B(0x20);
H(norm0); H(vert0);
H(vert1); H(vert2);

Gouraud No-Texture Triangle
```

```
0x0000000c; DRV(0)|PRIM_TYPE(TRI|IIP); GsUG3
B(r); B(g); B(b); B(0x30);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
```

Colored Flat No-Texture Triangle

```
0x0000000a; DRV(0)|PRIM_TYPE(TRI|COL); GsUCF3
B(r0); B(g0); B(b0); B(0x30);
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
H(norm0); H(vert0);
H(vert1); H(vert2);
```

Colored Gouraud No-Texture Triangle

```
0x0000000e; DRV(0)|PRIM_TYPE(TRI|IIP|COL); GsUCG3
B(r0); B(g0); B(b0); B(0x30);
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
```

Flat Texture Triangle

```
0x00000009; DRV(0)|PRIM_TYPE(TRI|TME); GsUFT3
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(vert1); H(vert2);
```

Gouraud Texture Triangle

```
0x0000000d; DRV(0)|PRIM_TYPE(TRI|IIP|TME); GsUGT3
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
```

Colored Flat Texture Triangle

```
0x0000000b; DRV(0)|PRIM_TYPE(TRI|COL|TME); GsUCFT3
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(vert1); H(vert2);
```

Colored Gouraud Texture Triangle

```
0x0000000f; DRV(0)|PRIM_TYPE(TRI|IIP|COL|TME); GsUCGT3
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
```

```
H(norm1); H(vert1);
H(norm2); H(vert2);
```

Flat No-Texture Quad
0x00000010; DRV(0)|PRIM_TYPE(QUAD); GsUF4

```
B(r); B(g); B(b); B(0x28);
H(norm0); H(vert0);
H(vert1); H(vert2);
H(vert3); H(0);
```

Gouraud No-Texture Quad
0x00000014; DRV(0)|PRIM_TYPE(QUAD|IIP); GsUG4

```
B(r); B(g); B(b); B(0x38);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
H(norm3); H(vert3);
```

Colored Flat No-Texture Quad
0x00000012; DRV(0)|PRIM_TYPE(QUAD|COL); GsUCF4

```
B(r0); B(g0); B(b0); B(0x38);
B(r1); B(g1); B(b1); B(0x38);
B(r2); B(g2); B(b2); B(0x38);
B(r3); B(g3); B(b3); B(0x38);
H(norm0); H(vert0);
H(vert1); H(vert2);
H(vert3); H(0);
```

Colored Gouraud No-Texture Quad
0x00000016; DRV(0)|PRIM_TYPE(QUAD|IIP|COL); GsUCG4

```
B(r0); B(g0); B(b0); B(0x38);
B(r1); B(g1); B(b1); B(0x38);
B(r2); B(g2); B(b2); B(0x38);
B(r3); B(g3); B(b3); B(0x38);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
H(norm3); H(vert3);
```

Flat Texture Quad
0x00000011; DRV(0)|PRIM_TYPE(QUAD|TME); GsUFT4

```
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
B(u3); B(v3); H(norm0);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Gouraud Texture Quad
0x00000015; DRV(0)|PRIM_TYPE(QUAD|IIP|TME); GsUGT4

```
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(norm0);
B(u3); B(v3); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
H(norm3); H(vert3);
```

Colored Flat Texture Quad
0x00000013; DRV(0)|PRIM_TYPE(QUAD|COL|TME); GsUCFT4

```
B(r0); B(g0); B(b0); B(0x3c);
B(r1); B(g1); B(b1); B(0x3c);
B(r2); B(g2); B(b2); B(0x3c);
B(r3); B(g3); B(b3); B(0x3c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
B(u3); B(v3); H(norm0);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Colored Gouraud Texture Quad
0x00000017; DRV(0)|PRIM_TYPE(QUAD|IIP|COL|TME); GsUCGT4

```
B(r0); B(g0); B(b0); B(0x3c);
B(r1); B(g1); B(b1); B(0x3c);
B(r2); B(g2); B(b2); B(0x3c);
B(r3); B(g3); B(b3); B(0x3c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(norm0);
B(u2); B(v2); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
H(norm3); H(vert3);
```

Flat No-Texture Mesh
0x00000018; DRV(0)|PRIM_TYPE(MESH); GsUMF3

```
H(num); H(0);
B(r); B(g); B(b); B(0x20);
H(norm2); H(vert0);
H(vert1); H(vert2);
/*------------------------*/
B(r); B(g); B(b); B(0x20);
H(norm3); H(vert3);
```

Gouraud No-Texture Mesh
0x0000001c; DRV(0)|PRIM_TYPE(MESH|IIP); GsUMG3

```
H(num); H(0);
B(r2); B(g2); B(b2); B(0x30);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
/*------------------------*/
B(r3); B(g3); B(b3); B(0x30);
H(norm1); H(norm2);
H(norm3); H(vert3);
```

Colored Flat No-Texture Mesh
0x0000001a; DRV(0)|PRIM_TYPE(MESH|COL)

```
H(num); H(0);
B(r0); B(g0); B(b0); B(0x30);
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
H(norm2); H(vert0);
H(vert1); H(vert2);
/*------------------------*/
B(r3); B(g3); B(b3); B(0x30);
H(norm3); H(vert3);
```

Colored Gouraud No-Texture Mesh

```
0x0000001e; DRV(0)|PRIM_TYPE(MESH|IIP|COL)
H(num); H(0);
B(r0); B(g0); B(b0); B(0x30);
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
/*-----------------------*/
B(r3); B(g3); B(b3); B(0x30);
H(norm1); H(norm2);
H(norm3); H(vert3);
```

Flat Texture Mesh

```
0x00000019; DRV(0)|PRIM_TYPE(MESH|TME); GsUMFT3
H(num); H(0);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm2); H(vert0);
H(vert1); H(vert2);
/*-----------------------*/
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(0);
H(norm3); H(vert3);
```

Gouraud Texture Mesh

```
0x0000001d; DRV(0)|PRIM_TYPE(MESH|IIP|TME); GsUMGT3
H(num); H(0);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
/*-----------------------*/
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(0);
H(norm1); H(norm2);
H(norm3); H(vert3);
```

Colored Flat Texture Mesh

```
0x0000001b; DRV(0)|PRIM_TYPE(MESH|COL|TME)
H(num); H(0);
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm2); H(vert0);
H(vert1); H(vert2);
/*-----------------------*/
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(r3); B(g3); B(b3); B(0x34);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
```

```
B(u3); B(v3); H(0);
H(norm3); H(vert3);
```

Colored Gouraud Texture Mesh
0x0000001f; DRV(0)|PRIM_TYPE(MESH|IIP|COL|TME)

```
H(num); H(0);
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
/*-----------------------*/
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(r3); B(g3); B(b3); B(0x34);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(0);
H(norm1); H(norm2);
H(norm3); H(vert3);
```

Without Light-source Calculation (Model Data without Normals)

Flat No-Texture Triangle
0x00040048; DRV(LGT)|PRIM_TYPE(LMD|TRI); GsUNF3

```
B(r); B(g); B(b); B(0x20);
H(vert0); H(vert1);
H(vert2); H(0);
```

Gouraud No-Texture Triangle
0x0004004c; DRV(LGT)|PRIM_TYPE(LMD|TRI|IIP); GsUNG3

```
B(r0); B(g0); B(b0); B(0x30);
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
H(vert0); H(vert1);
H(vert2); H(0);
```

Flat Texture Triangle
0x00040049; DRV(LGT)|PRIM_TYPE(LMD|TRI|TME); GsUNFT3

```
B(r); B(g); B(b); B(0x24);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
```

Gouraud Texture Triangle
0x0004004d; DRV(LGT)|PRIM_TYPE(LMD|TRI|IIP|TME); GsUNGT3

```
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
```

```
Flat No-Texture Quad
0x00040050; DRV(LGT)|PRIM_TYPE(LMD|QUAD); GsUNF4
B(r); B(g); B(b); B(0x28);
H(vert0); H(vert1);
H(vert2); H(vert3);

Gouraud No-Texture Quad
0x00040054; DRV(LGT)|PRIM_TYPE(LMD|QUAD|IIP); GsUNG4
B(r0); B(g0); B(b0); B(0x38);
B(r1); B(g1); B(b1); B(0x38);
B(r2); B(g2); B(b2); B(0x38);
B(r3); B(g3); B(b3); B(0x38);
H(vert0); H(vert1);
H(vert2); H(vert3);

Flat Texture Quad
0x00040051; DRV(LGT)|PRIM_TYPE(LMD|QUAD|TME); GsUNFT4
B(r); B(g); B(b); B(0x2c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
B(u3); B(v3); H(vert1);
H(vert2); H(vert3);

Gouraud Texture Quad
0x00040055; DRV(LGT)|PRIM_TYPE(LMD|QUAD|IIP|TME); GsUNGT4
B(r0); B(g0); B(b0); B(0x3c);
B(r1); B(g1); B(b1); B(0x3c);
B(r2); B(g2); B(b2); B(0x3c);
B(r3); B(g3); B(b3); B(0x3c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
B(u3); B(v3); H(vert1);
H(vert2); H(vert3);

Flat No-Texture Mesh
0x00040058; DRV(LGT)|PRIM_TYPE(LMD|MESH); GsUMNF3
H(num); H(0);
B(r2); B(g2); B(b2); B(0x20);
H(vert0); H(vert1);
H(vert2); H(0);
/*-----------------------*/
B(r3); B(g3); B(b3); B(0x20);
H(vert3); H(0);

Gouraud No-Texture Mesh
0x0004005c; DRV(LGT)|PRIM_TYPE(LMD|MESH|IIP); GsUMNG3
H(num); H(0);
B(r0); B(g0); B(b0); B(0x30);
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
H(vert0); H(vert1);
H(vert2); H(0);
/*-----------------------*/
B(r1); B(g1); B(b1); B(0x30);
B(r2); B(g2); B(b2); B(0x30);
B(r3); B(g3); B(b3); B(0x30);
H(vert3); H(0);
```

Flat Texture Mesh
0x00040059; DRV(LGT)|PRIM_TYPE(LMD|MESH|TME); GsUMNFT3

```
H(num); H(0);
B(r0); B(g0); B(b0); B(0x24);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
/*-----------------------*/
B(r3); B(g3); B(b3); B(0x24);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(vert3);
```

Gouraud Texture Mesh
0x0004005d; DRV(LGT)|PRIM_TYPE(LMD|MESH|IIP|TME); GsUMNGT3

```
H(num); H(0);
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
/*-----------------------*/
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g3); B(b3); B(0x34);
B(r3); B(g3); B(b3); B(0x34);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(vert3);
```

Tiled Textures

Tiled Textures with Light-source Calculation

- TUM: Tiling mask for the U coordinate of the texture pattern (5 bits)
- TVM: Tiling mask for the V coordinate of the texture pattern (5 bits)
- TUA: Upper address of U for tiling the texture pattern (5 bits)
- TVA: Upper address of V for tiling the texture pattern (5 bits)

A packet that is used for tiled textures contains a repetition parameter at the beginning of the packet, and a reset parameter at the end of the packet. This allows tiled and non-tiled textures to coexist.

tum, tvm, tua, tva serve as parameters for calculating UV' from given UV values (u,v) using the following equation.

```
UV' = ((~(tum << 3) & u)|((tum << 3) & (tua << 3)),
    (~(tvm << 3) & v)l((tvm << 3) & (tva << 3)));
```

In the following example, a texture window for tiling is set up in the texture page, with (x, y) representing the upper left corner, and (w, h) representing the width and height:

```
tum = (~(w - 1) & 0x0ff) >> 3;
tvm = (~(h - 1) & 0x0ff) >> 3;
tua = (x & 0x0ff) >> 3;
tva = (y & 0x0ff) >> 3;
```

At reset, all four parameters are set to zero.

Flat Texture Triangle
0x00000209; DRV(0)|PRIM_TYPE(TILE|TRI|TME); GsUTFT3

```
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
```

```
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(vert1); H(vert2);
```

Gouraud Texture Triangle
0x0000020d; DRV(0)|PRIM_TYPE(TILE|TRI|IIP|TME); GsUTGT3

```
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
Colored Flat Texture Triangle
0x0000020b; DRV(0)|PRIM_TYPE(TILE|TRI|COL|TME)
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(vert1); H(vert2);
```

Colored Gouraud Texture Triangle
0x0000020f; DRV(0)|PRIM_TYPE(TILE|TRI|IIP|COL|TME)

```
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
```

Flat Texture Quad
0x00000211; DRV(0)|PRIM_TYPE(TILE|QUAD|TME); GsUTFT4

```
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
B(u3); B(v3); H(norm0);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Gouraud Texture Quad
0x00000215; DRV(0)|PRIM_TYPE(TILE|QUAD|IIP|TME); GsUTGT4

```
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(norm0);
B(u3); B(v3); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
H(norm3); H(vert3);
```

Colored Flat Texture Quad

```
0x00000213; DRV(0)|PRIM_TYPE(TILE|QUAD|COL|TME)
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x3c);
B(r1); B(g1); B(b1); B(0x3c);
B(r2); B(g2); B(b2); B(0x3c);
B(r3); B(g3); B(b3); B(0x3c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
B(u3); B(v3); H(norm0);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Colored Gouraud Texture Quad

```
0x00000217; DRV(0)|PRIM_TYPE(TILE|QUAD|IIP|COL|TME)
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x3c);
B(r1); B(g1); B(b1); B(0x3c);
B(r2); B(g2); B(b2); B(0x3c);
B(r3); B(g3); B(b3); B(0x3c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(norm0);
B(u3); B(v3); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
H(norm3); H(vert3);
```

Flat Texture Mesh

```
0x00000219; DRV(0)|PRIM_TYPE(TILE|MESH|TME)
H(num); H(0);
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm2); H(vert0);
H(vert1); H(vert2);
/*-----------------------*/
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(0);
H(norm3); H(vert3);
```

Gouraud Texture Mesh

```
0x0000021d; DRV(0)|PRIM_TYPE(TILE|MESH|IIP|TME)
H(num); H(0);
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
/*-----------------------*/
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
```

```
B(u3); B(v3); H(0);
H(norm1); H(norm2);
H(norm3); H(vert3);
```

Colored Flat Texture Mesh
0x0000021b; DRV(0)|PRIM_TYPE(TILE|MESH|COL|TME)

```
H(num); H(0);
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm2); H(vert0);
H(vert1); H(vert2);
/*-----------------------*/
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(r3); B(g3); B(b3); B(0x34);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(0);
H(norm3); H(vert3);
```

Colored Gouraud Texture Mesh
0x0000021f; DRV(0)|PRIM_TYPE(TILE|MESH|IIP|COL|TME)

```
H(num); H(0);
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(0);
H(norm0); H(vert0);
H(norm1); H(vert1);
H(norm2); H(vert2);
/*-----------------------*/
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(r3); B(g3); B(b3); B(0x34);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(0);
H(norm1); H(norm2);
H(norm3); H(vert3);
```

Tiled Textures without Light-source Calculation

Flat Texture Triangle
0x00040249; DRV(LGT)|PRIM_TYPE(TILE|LMD|TRI|TME)

```
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r); B(g); B(b); B(0x24);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
```

```
Gouraud Texture Triangle
0x0004024d; DRV(LGT)|PRIM_TYPE(TILE|LMD|TRI|IIP|TME)
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x34);
B(r1); B(g1); B(b1); B(0x34);
B(r2); B(g2); B(b2); B(0x34);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);

Flat Texture Quad
0x00040251; DRV(LGT)|PRIM_TYPE(TILE|LMD|QUAD|TME)
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r); B(g); B(b); B(0x2c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
B(u3); B(v3); H(vert1);
H(vert2); H(vert3);
Gouraud Texture Quad
0x00040255; DRV(LGT)|PRIM_TYPE(TILE|LMD|QUAD|IIP|TME)
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x3c);
B(r1); B(g1); B(b1); B(0x3c);
B(r2); B(g2); B(b2); B(0x3c);
B(r3); B(g3); B(b3); B(0x3c);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
B(u3); B(v3); H(vert1);
H(vert2); H(vert3);

Flat Texture Mesh
0x00040259; DRV(LGT)|PRIM_TYPE(TILE|LMD|MESH|TME)
H(num); H(0);
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x24);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
/*------------------------*/
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r3); B(g3); B(b3); B(0x24);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(vert3);

Gouraud Texture Mesh
0x0004025d; DRV(LGT)|PRIM_TYPE(TILE|LMD|MESH|IIP|TME)
H(num); H(0);
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r0); B(g0); B(b0); B(0x35);
B(r1); B(g1); B(b1); B(0x35);
B(r2); B(g2); B(b2); B(0x35);
B(u0); B(v0); H(cba);
B(u1); B(v1); H(tsb);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
```

```
/*------------------------*/
TUM(tum)|TVM(tvm)|TUA(tua)|TVA(tva)|0xe2000000;
B(r1); B(g1); B(b1); B(0x35);
B(r2); B(g2); B(b2); B(0x35);
B(r3); B(g3); B(b3); B(0x35);
B(u1a); B(v1a); H(cba);
B(u2a); B(v2a); H(tsb);
B(u3); B(v3); H(vert3);
```

Preset model data

```
Flat No-Texture Triangle
0x00040148; DRV(LGT)|PRIM_TYPE(PST|LMD|TRI); GsUPNF3
DMAtag;
B(r); B(g); B(b); B(0x20);
H(x0); H(y0);
H(x1); H(y1);
H(x2); H(y2);
DMAtag;
B(r); B(g); B(b); B(0x20);
H(x0); H(y0);
H(x1); H(y1);
H(x2); H(y2);
H(vert0); H(vert1);
H(vert2); H(0);

Gouraud No-Texture Triangle
0x0004014c; DRV(LGT)|PRIM_TYPE(PST|LMD|TRI|IIP); GsUPNG3
DMAtag;
B(r0); B(g0); B(b0); B(0x30);
H(x0); H(y0);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
DMAtag;
B(r0); B(g0); B(b0); B(0x30);
H(x0); H(y0);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
H(vert0); H(vert1);
H(vert2); H(0);

Flat Texture Triangle
0x00040149; DRV(LGT)|PRIM_TYPE(PST|LMD|TRI|TME); GsUPNFT3
DMAtag;
B(r); B(g); B(b); B(0x24);
H(x0); H(y0);
B(u0); B(v0); H(cba);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
H(x2); H(y2);
B(u2); B(v2); H(0);
DMAtag;
B(r); B(g); B(b); B(0x24);
H(x0); H(y0);
B(u0); B(v0); H(cba);
H(x1); H(y1);
```

```
B(u1); B(v1); H(tsb);
H(x2); H(y2);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
```

Gouraud Texture Triangle
0x0004014d; DRV(LGT)|PRIM_TYPE(PST|LMD|TRI|IIP|TME); GsUNGT3

```
DMAtag;
B(r0); B(g0); B(b0); B(0x34);
H(x0); H(y0);
B(u0); B(v0); H(cba);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2); B(v2); H(0);
DMAtag;
B(r0); B(g0); B(b0); B(0x34);
H(x0); H(y0);
B(u0); B(v0); H(cba);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
```

Flat No-Texture Quad
0x00040150; DRV(LGT)|PRIM_TYPE(PST|LMD|QUAD); GsUPNF4

```
DMAtag;
B(r); B(g); B(b); B(0x28);
H(x0); H(y0);
H(x1); H(y1);
H(x2); H(y2);
H(x3); H(y3);
DMAtag;
B(r); B(g); B(b); B(0x28);
H(x0); H(y0);
H(x1); H(y1);
H(x2); H(y2);
H(x3); H(y3);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Gouraud No-Texture Quad
0x00040154; DRV(LGT)|PRIM_TYPE(PST|LMD|QUAD|IIP); GsUPNG4

```
DMAtag;
B(r0); B(g0); B(b0); B(0x38);
H(x0); H(y0);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
DMAtag;
B(r0); B(g0); B(b0); B(0x38);
H(x0); H(y0);
```

```
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Flat Texture Quad
0x00040151; DRV(LGT)|PRIM_TYPE(PST|LMD|QUAD|TME); GsUPNFT4

```
DMAtag;
B(r); B(g); B(b); B(0x2c);
H(x0); H(y0);
B(u0); B(v0); H(cba);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
H(x2); H(y2);
B(u2); B(v2); H(0);
H(x3); H(y3);
B(u3); B(v3); H(0);
DMAtag;
B(r); B(g); B(b); B(0x2c);
H(x0); H(y0);
B(u0); B(v0); H(cba);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
H(x2); H(y2);
B(u2); B(v2); H(0);
H(x3); H(y3);
B(u3); B(v3); H(0);
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Gouraud Texture Quad
0x00040155; DRV(LGT)|PRIM_TYPE(PST|LMD|QUAD|IIP|TME); GsUPNGT4

```
DMAtag;
B(r0); B(g0); B(b0); B(0x3c);
H(x0); H(y0);
B(u0); B(v0); H(cba);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2); B(v2); H(0);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
B(u3); B(v3); H(0)
DMAtag;
B(r0); B(g0); B(b0); B(0x3c);
H(x0); H(y0);
B(u0); B(v0); H(cba);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2); B(v2); H(0);
B(r3); B(g3); B(b3); B(0);
```

```
H(x3); H(y3);
B(u3); B(v3); H(0)
H(vert0); H(vert1);
H(vert2); H(vert3);
```

Flat No-Texture Mesh
0x00040158; DRV(LGT)|PRIM_TYPE(PST|LMD|MESH)

```
H(num); H(0);
DMAtag;
B(r2); B(g2); B(b2); B(0x20);
H(x0); H(y0);
H(x1); H(y1);
H(x2); H(y2);
DMAtag;
B(r2); B(g2); B(b2); B(0x20);
H(x0); H(y0);
H(x1); H(y1);
H(x2); H(y2);
H(vert0); H(vert1);
H(vert2); H(0);
/*------------------------*/
DMAtag;
B(r3); B(g3); B(b3); B(0x20);
H(x1); H(y1);
H(x2); H(y2);
H(x3); H(y3);
DMAtag;
B(r3); B(g3); B(b3); B(0x20);
H(x1); H(y1);
H(x2); H(y2);
H(x3); H(y3);
H(vert3); H(0);
```

Gouraud No-Texture Mesh
0x0004015c; DRV(LGT)|PRIM_TYPE(PST|LMD|MESH|IIP)

```
H(num); H(0);
DMAtag;
B(r0); B(g0); B(b0); B(0x30);
H(x0); H(y0);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
DMAtag;
B(r0); B(g0); B(b0); B(0x30);
H(x0); H(y0);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
H(vert0); H(vert1);
H(vert2); H(0);
/*------------------------*/
DMAtag;
B(r1); B(g1); B(b1); B(0x30);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
```

```
DMAtag;
B(r1); B(g1); B(b1); B(0x30);
H(x1); H(y1);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
H(vert3); H(0);
```

Flat Texture Mesh
0x00040159; DRV(LGT)|PRIM_TYPE(PST|LMD|MESH|TME)

```
H(num); H(0);
DMAtag;
B(r0); B(g0); B(b0); B(0x24);
H(x0); H(y0);
B(u0); B(v0); H(cba);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
H(x2); H(y2);
B(u2); B(v2); H(0);
DMAtag;
B(r0); B(g0); B(b0); B(0x24);
H(x0); H(y0);
B(u0); B(v0); H(cba);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
H(x2); H(y2);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
/*------------------------*/
DMAtag;
B(r1); B(g1); B(b1); B(0x24);
H(x1); H(y1);
B(u1a); B(v1a); H(cba);
H(x2); H(y2);
B(u2a); B(v2a); H(tsb);
H(x3); H(y3);
B(u3); B(v3); H(0);
DMAtag;
B(r1); B(g1); B(b1); B(0x24);
H(x1); H(y1);
B(u1a); B(v1a); H(cba);
H(x2); H(y2);
B(u2a); B(v2a); H(tsb);
H(x3); H(y3);
B(u3); B(v3); H(vert3);
```

Gouraud Texture Mesh
0x0004015d; DRV(LGT)|PRIM_TYPE(PST|LMD|MESH|IIP|TME)

```
H(num); H(0);
DMAtag;
B(r0); B(g0); B(b0); B(0x34);
H(x0); H(y0);
B(u0); B(v0); H(cba);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2); B(v2); H(0);
```

```
DMAtag;
B(r0); B(g0); B(b0); B(0x34);
H(x0); H(y0);
B(u0); B(v0); H(cba);
B(r1); B(g1); B(b1); B(0);
H(x1); H(y1);
B(u1); B(v1); H(tsb);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2); B(v2); H(vert0);
H(vert1); H(vert2);
/*------------------------*/
DMAtag;
B(r1); B(g1); B(b1); B(0x34);
H(x1); H(y1);
B(u1a); B(v1a); H(cba);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2a); B(v2a); H(tsb);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
B(u3); B(v3); H(0);
DMAtag;
B(r1); B(g1); B(b1); B(0x34);
H(x1); H(y1);
B(u1a); B(v1a); H(cba);
B(r2); B(g2); B(b2); B(0);
H(x2); H(y2);
B(u2a); B(v2a); H(tsb);
B(r3); B(g3); B(b3); B(0);
H(x3); H(y3);
B(u3); B(v3); H(vert3);
```

Shared Primitives (Category 1)

Two types of primitive drivers are available for shared primitives:

1. PRE-CALCULATION drivers
2. Shared drivers

For VERTEXes, a PRE-CALCULATION driver converts a three-dimensional shared vertex array into a perspective-transformed two-dimensional vertex array. For NORMALs, a PRE-CALCULATION driver performs vertex color calculations.

PRE-CALCULATION drivers are chained to each primitive block since they need to be called for each coordinate.

Shared drivers extract data from vertex arrays on which calculations have already been performed by a PRE-CALCULATION driver. The data is then used to create a GPU PACKET that is entered into the OT.

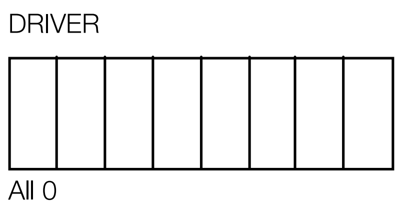
Shared drivers must be called last so they are chained to POST-PROCESS primitive blocks.

TYPE

PRE-CALCULATION driver 0x01000000

Shared Driver

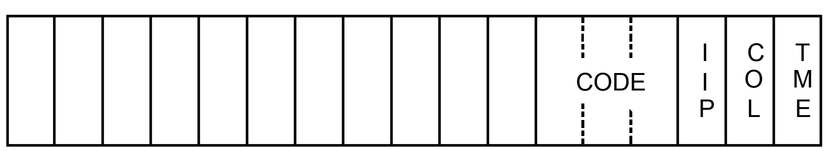
Figure 15: Shared Primitive Driver



PRIMITIVE TYPE

These bit assignments depend on the primitive type.

Figure 16: Primitive Type of Shared Primitive



TME
0: Disable texture mapping
1: Perform texture mapping

COL - (not implemented)
0: Use one material color for identical polygons
1: Each vertex has its own color

IIP
0: Flat-shaded polygon
1: Gouraud-shaded polygon

CODE - Describes shape of polygon
0: Reserved by the system
1: Triangle
2: Quadrangle
3: Strip mesh (not implemented)
4-7: Reserved by the system

The format of the connection data, which a shared driver refers to, is the same as the format for a non-shared polygon PACKET. The format of the calculated area, which a shared driver refers to, is shown below:

VERTEX

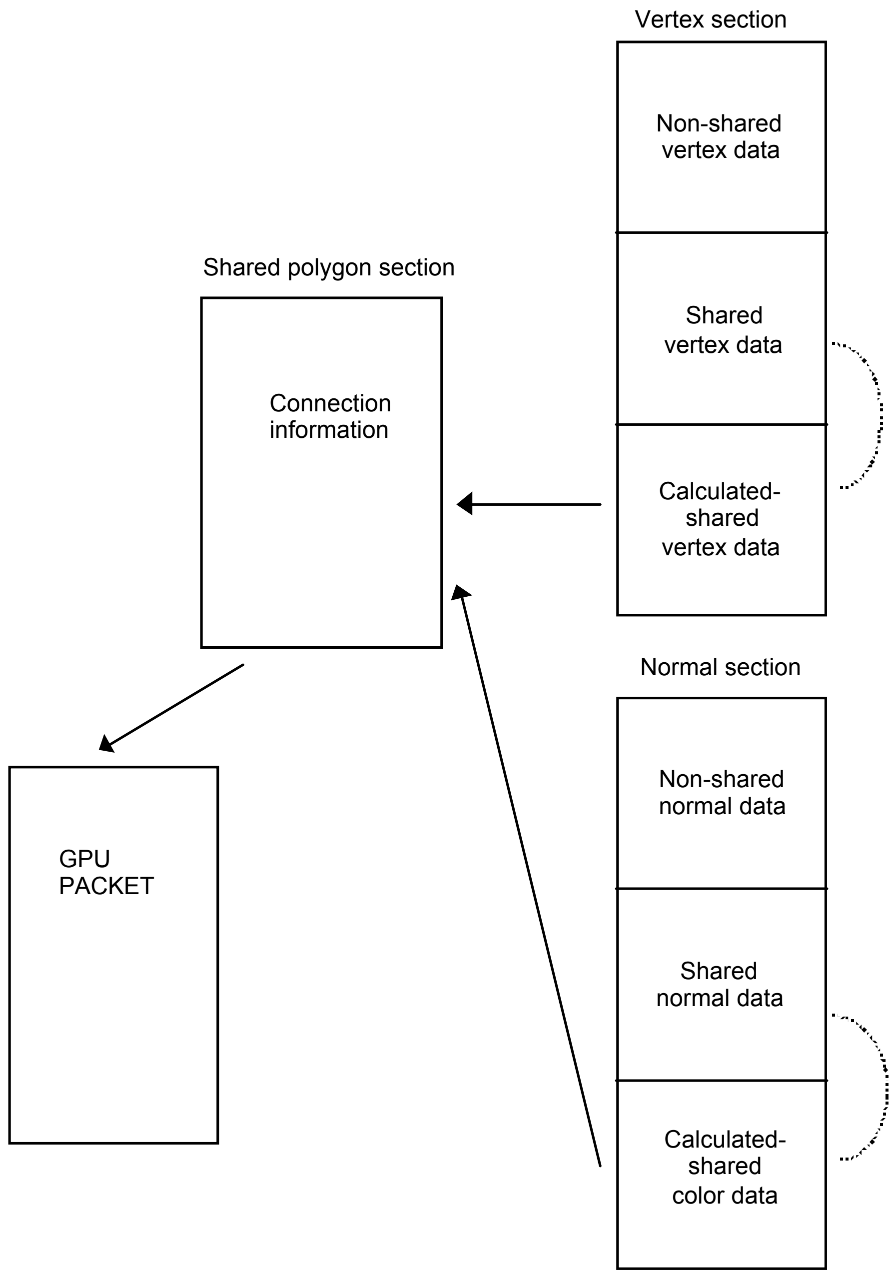
```
H(vx); H(vy);
H(otz); H(p);
```

NORMAL

```
H(r); H(g);
H(b); H(0);
```

Processing Flow for Shared Polygons

Figure 17: Shared Polygon Processing Flow



The arrows with the dotted line indicate the processing flow of the PRE-CALCULATION driver. Vertex and normal calculations are performed for each coordinate.

The arrows with the solid line indicate the processing flow of the shared driver. Pre-calculated vertex data and color data are used to create a GPU PACKET. The format of the connection data for the shared driver is the same as that for an independent PACKET and is identified by the type field.

Image Primitive Section (Category 2)

The HMD format is able to represent image data as a primitive. This allows HMD to provide integrated management of modeling data, image data, and animation data.

Of course, image data can be set up separately without including it in HMD data. For example, TIM can be used to represent image data. Conversely, HMD data can be created which contains only image data as well.

Figure 18

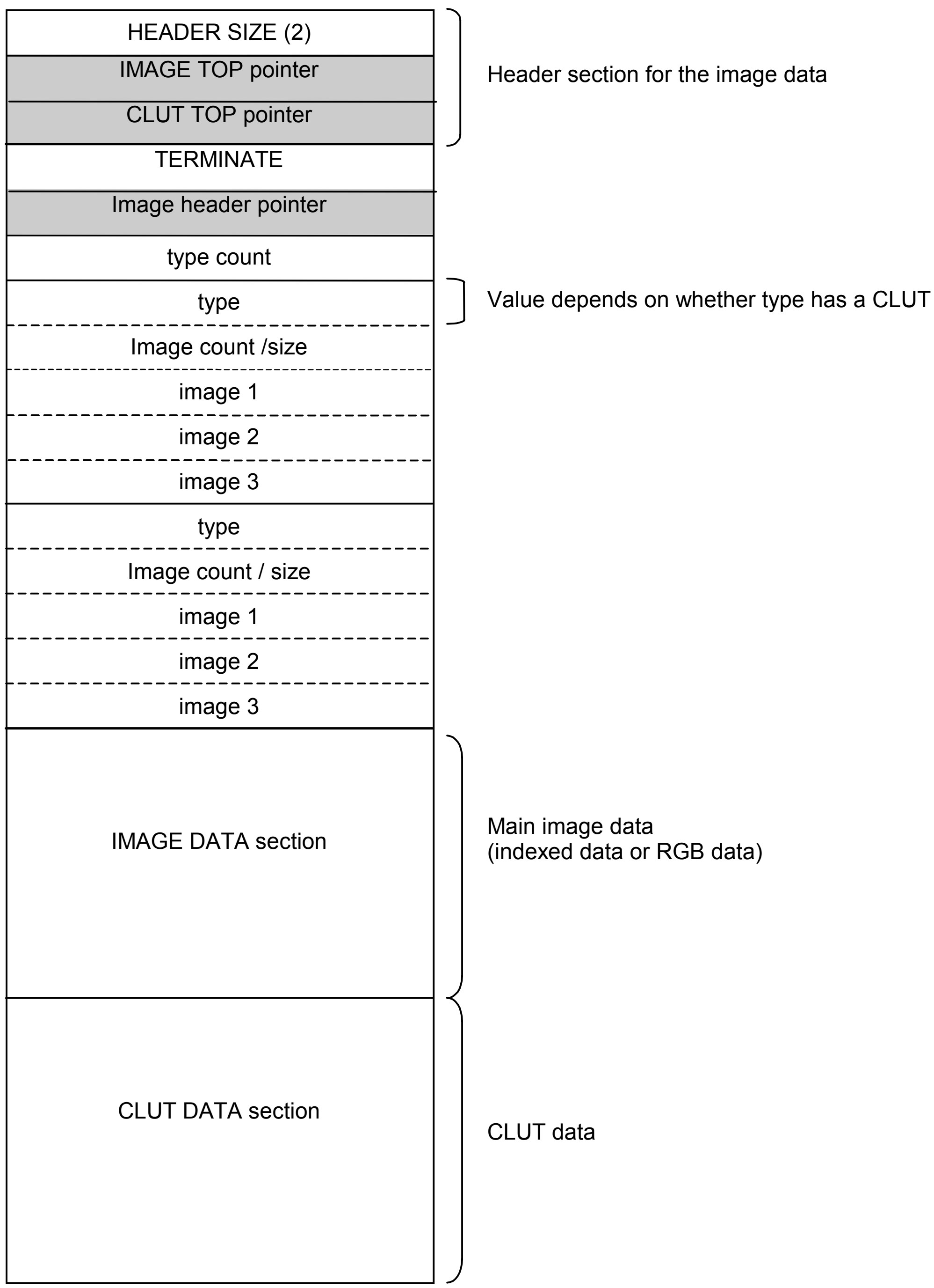
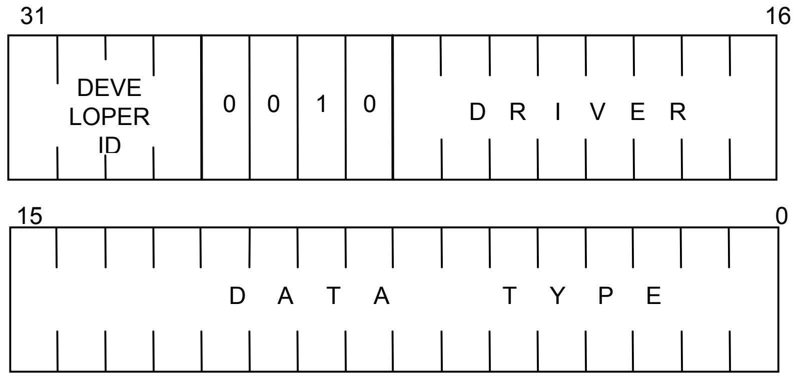


Image Type

Figure 19: Image Primitive Type Field



DRIVER - Currently all 0’s

DATA TYPE - Indicates type of data
0: No CLUT
1: CLUT

Non-CLUT Primitive

```
0x02000000; DEV_ID(SCE)|CTG(CTG_IMAGE)|DRV(0)|PRIM_TYPE(NOCLUT); GsUIMG0
H(dx); H(dy);
H(w); H(h);
image_idx;
```

Primitive with CLUT

```
0x02000001; DEV_ID(SCE)|CTG(CTG_IMAGE)|DRV(0)|PRIM_TYPE(WITHCLUT); GsUIMG1
H(dx); H(dy);
H(w); H(h);
image_idx;
H(dx); H(dy);
H(w); H(h);
clut_idx;
```

Run-time Environment for Image Primitive Driver

The image primitive driver is called with the following environment.

The following variables are copied to the parameter memory area.

Figure 20: Parameter Memory Area of Image Primitive Driver

primtop
tag(OT)
shift(OT)
offset(OT)
OUTP(packet area)
Image header pointer
CLUT top pointer

Parameter Settings

Behavior of the image primitive driver

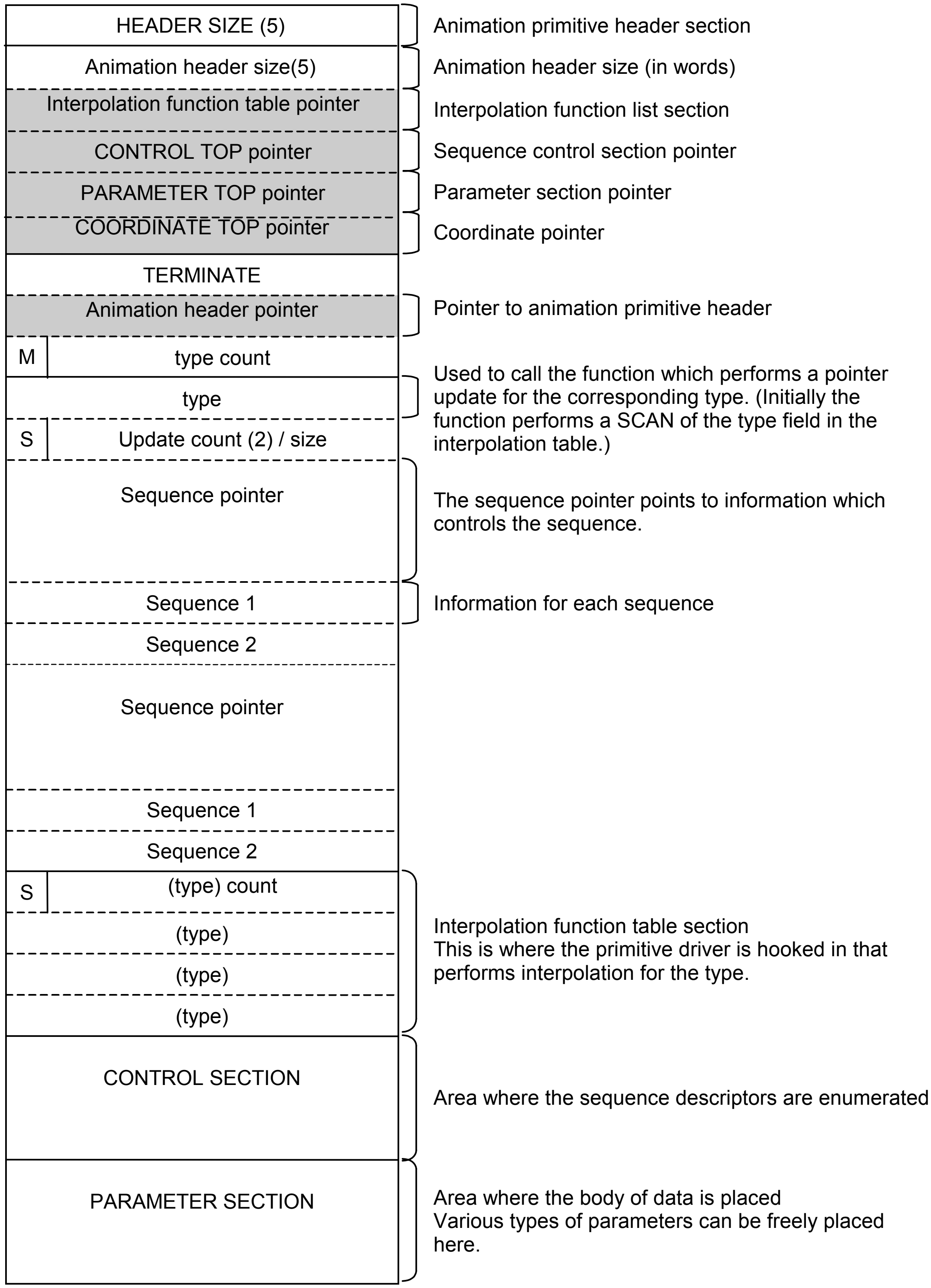
Image primitives are linked to the PRE-PROCESS at the beginning of HMD's coordinate section. A VRAM transfer function is called during the SCAN operation. A NULL driver (type=0x00000000, a primitive driver that does not do anything) can be set in the type field once the transfer is complete so that the transfer to VRAM will be performed only once.

Animation Primitive Section (Category 3)

An animation primitive section can be divided into the following five subsections:

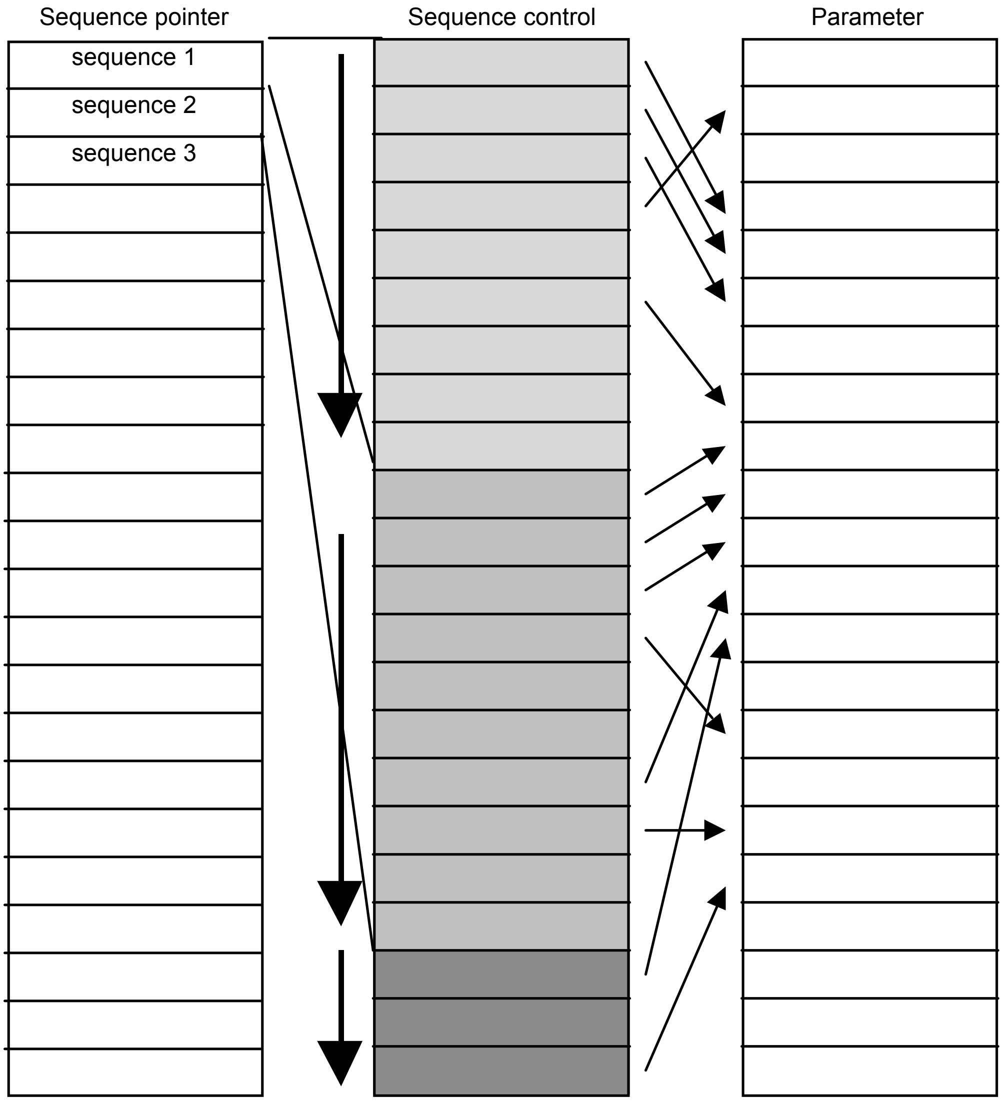
1. Animation primitive header section
2. Sequence pointer section
3. Interpolation function table section
4. Sequence control section
5. Parameter section

Figure 21: Animation Structure



Relationships Between Sections in Animation Data

Figure 22: Diagram Showing Correlation of All Animation Sections



Animation Primitive Header Section

The animation primitive header section must contain a pointer to the interpolation function table, a sequence control section, and a pointer to the start of the parameter section.

Pointers to the sections, which need to be updated, are placed in the corresponding low-order address. For example, when a COORDINATE is to be rewritten, COORDINATE TOP is saved. If a vertex is to be rewritten, VERTEX TOP is saved.

Sequence Pointer Section

The sequence pointer section contains the sequence pointer and sequence information for each sequence. The update index contains separate information for the upper 8 bits and the lower 24 bits.

Interpolation Function Table Section

The interpolation function table section contains the type fields for the interpolation functions referred to by the sequence descriptors. The type fields are stored in an array and the interpolation method to be used is determined from the index of this array. The GsScanAnim() function must first be used to extract the type field and perform a SCAN to obtain the starting address of the actual primitive driver.

Sequence Control Section

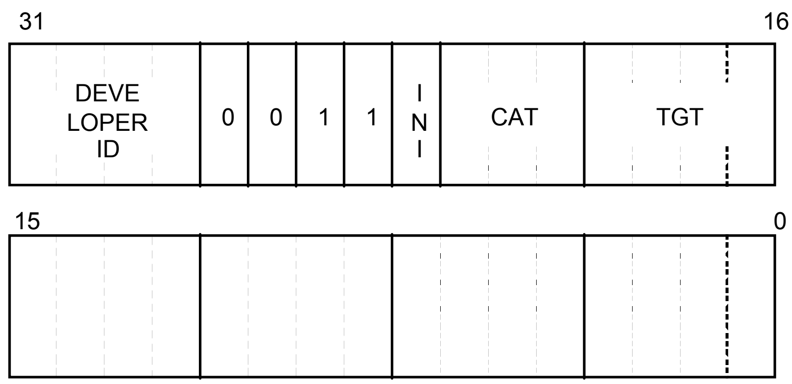
A sequence is represented as an array of sequence descriptors in the sequence control section. A sequence descriptor accesses the interpolation function table section and the parameter section using an index in order to specify the interpolation method that will be used between a key frame and the parameter of a key frame.

Parameter Section

A sequence descriptor accesses an interpolation function and an interpolation parameter using an index. The parameter section contains an array of interpolation parameters for various formats and interpolation functions.

Animation Type

Figure 23: Animation Primitive Type Field



INI - Determines whether a SCAN of the interpolation table section will be performed
0: Do not perform SCAN (SCAN already performed)
1: Perform SCAN for interpolation function table

CAT - Indicates category of the frame update driver
0: Standard frame update driver (performs frame updates and calls interpolation function)

TGT - Update target
0: Update the COORDINATE section
1: General update type

When TGT=0 (Update COORDINATE)

Figure 24: Type Field when TGT=0

31							16
DEVE LOPER ID	0	0	1	1	INI	CAT	0 0 0 0

15			0
ROT ORDER	SCAL INTR	ROT INTR	TRNS INTR

ROT ORDER - Specifies the rotation order. Valid only when ROT INTR is not 0.
The symbol indicates the applicable rotation matrix. When 0:XYZ, rotation is carried out in the following order: Z axis, Y axis, X axis.
0: XYZ
1: XZY
2: YXZ
3: YZX
4: ZXY
5: ZYX

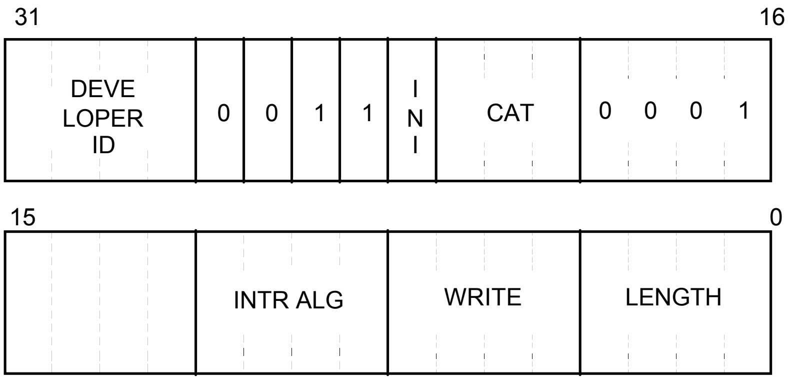
SCAL INTR - Specifies the interpolation method when scaling
0: Do not interpolate
1: LINEAR
2: BEZIER
3: B-SPRINE
4: beta-SPRINE
9: LINEAR (one parameter)
A: BEZIER (one parameter)
B: B-SPRINE (one parameter)

ROT INTR - Specifies the interpolation method when rotating
0: Do not interpolate
1: LINEAR
2: BEZIER
3: B-SPRINE

TRNS INTR - Specifies the interpolation method when translating
0: Do not interpolate
1: LINEAR
2: BEZIER
3: B-SPRINE
9: LINEAR(short)
A: BEZIER(short)
B: B-SPRINE(short)

When TGT=1 (General Purpose Update)

Figure 25: Type Field when TGT=1



LENGTH - 0: 32bit
1: 16bit
2: 8 bit

WRITE - Specify areas to update.
This field has 4bits, therefore, up to 4 units are allowed to update.

In the examples below, areas to update are colored with gray.

Figure 26: LENGTH=16 bit, WRITE=0x1

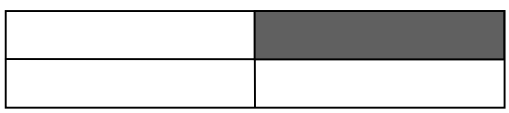


Figure 27: LENGTH=16bit, WRITE=0x7

Figure 28: LENGTH=8bit, WRITE=0x1

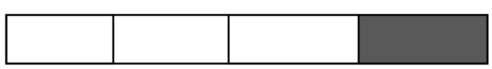


Figure 29: LENGTH=8bit, WRITE=0x7

INTR ALG - Interpolation algorithm
1. Linear
2. Bezier
3. B-Sprine

Sequence Header

The sequence header contains information that is used to manage the various sequences.

Figure 30: Sequence Header

Sequence pointer		
TRAVELING	STREAM ID	Start IDX
-	STREAM ID	Start IDX

Sequence Pointer

The sequence pointer holds sequence information during playback. When multiple sequences are set up to be played back simultaneously, a sequence pointer is assigned to each playback sequence. The programmer uses the sequence pointers to control the real time playback of sequences. The members of the sequence pointers are continuously referenced by the interpolation primitive driver, which provides instantaneous response.

The figure below shows the data format for a sequence pointer. The areas, which have been written with HMD data, are highlighted. The areas without highlighting are work areas used by the program for replacing values and controlling the sequence.

Figure 31: Sequence Pointer

Update index			
Sequence count / size			
A FRAME		INTR IDX	
SRC INTR IDX		SPEED	STREAM ID
TFRAME		RFRAME	
TCTR IDX		CTR IDX	
TRAVELING	START SID	START IDX	

Update Index

The update index contains the target address to be updated by the sequence. The upper 8 bits hold the section offset, and the lower 24 bits hold the offset within the section.

Figure 32: Setting Update Location

31	24 0
Section offset	Offset within the section

The primitive header contains a list of starting addresses, and the "section offset" is an index into that list specifying which section will be updated. For example, if the index is 0 the interpolation function table section will be used, and if the index is 1 the CONTROL section will be used.

The "offset within the section" is an index which points to the position within the section specified by the "section offset" that will be updated. The offset is specified in words. For example, if the second coordinate is to be rewritten, the offset would be sizeof(GsCOORDUNIT) /4+1. The +1 is included because the word at the beginning of the coordinate section is included in the coordinate count.

With some types of animation, vertices or normals may be updated instead of coordinates. In such cases, a pointer to the start of the section to be updated is added to the animation header, and the pointer is specified from the section offset of the update index. The position of the data to be updated can then be specified with the offset within the section. The type of data to be updated is identified with the type field.

Sequence Count/Size

The upper 16 bits hold the sequence count and the lower 16 bits hold the size. Sequence count is the number of sequences that are managed by the sequence pointer.

Size is the number of words remaining until the next sequence pointer.

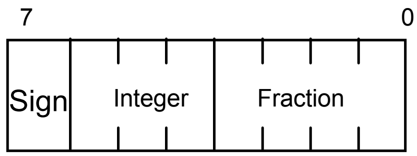
Figure 33: Sequence Count and Size

31	16 0
Sequence count	Size

- INTR IDX: The value in this field is an index into the key frame containing parameters to be used after interpolation of the current frame. The application can change this value if the sequence is to be dynamically switched.

  If this field contains the value 0xffff, updates will not be performed.
- A FRAME: The total frame count of the sequence. Setting AFRAME to 0 can stop the sequence. If ENDbit is detected in the sequence control descriptor, AFRAME will be automatically set to 0. If the value of AFRAME is set to 0xffff, the total frame count will be infinity and the value will not be decremented.
- SRC INTRIDX: Holds the work area to be assigned to INTR IDX.
- SPEED: Specifies the update speed for the sequence pointer.

Figure 34: Fixed Point Format Used in SPEED Specification



SPEED is a two's complement fixed-point number, with 1 bit for the sign, 3 bits for the integer part, and 4 bits for the fraction. If the value of SPEED is negative, the sequence pointer is decremented when it is updated, and animations will be played back in reverse.

- — If the sign bit is 1, the sequence pointer is decremented, resulting in the animation being played back in reverse.
- — The integer part has three bits, so animation playback can be sped up by a factor of 7.
- — The fractional part has four bits, so animation playback can be slowed down to 1/15.
- — If all 8 bits are 0, the update speed of the sequence pointer is set to the previous update rate. Note that operation may be unpredictable if 0 is specified as the initial value.

- TFRAME: The time between key frames for the data currently playing. This value is specified as a frame count and is updated automatically when the key frame is switched. TFRAME is represented as a fixed-point decimal integer, where the value 0x10 represents one frame.
- RFRAME: The time between the motion currently playing and the original key frame. This value is specified as a frame count and is re-read when the key frame is switched. RFRAME is represented as a fixed-point decimal integer, where the value 0x10 represents one frame.
- STREAM ID: Used for multiply-defined sequences. Sequence jumps take place only when STREAM IDs match. The STREAM ID can be changed dynamically during execution. This allows the efficient use of memory during interactive animation.

  The STREAM ID has 7 valid bits, ranging from 0 to 127.

  STREAM ID 0 has special meaning. This value matches to any SID. We do not recommend to use STREAM ID 127 as a condition of JUMP sequence. Then, STREAM ID 127 is possible to use with opposite meaning to STREAM ID 0.
- TCTR IDX: Holds the index of the target key frame (among the two key frames used for interpolation). The target key frame is the key frame that is in the direction of convergence. The index is automatically updated when the key frame is switched. To specify the start of a sequence, the index of the starting sequence descriptor should be placed in TCTR IDX and RFRAME should be set to 0.
- CTR IDX: Holds the index to the original key frame (among the two key frames used for interpolation). The original key frame is the key frame that has already passed. The index is automatically updated when the key frame is switched.
- START IDX: Holds the starting index for a sequence. When it is desired to start a sequence, START IDX should be placed in TCTR IDX, START SID should be placed in SID, and RFRAME should be set to 0.

  START IDX is also can be used as index to refer to control descriptor for sequence specific parameters. In this case, START IDX must not identical to starting index of sequence, the next sequence management data is allowed to use.
- START SID: Holds the stream ID of the sequence to be started.
- TRAVELING: Cleared to 0 when the key frame is switched. The programmer can use this variable freely. For example, to determine if the current interpolation is finished, a non-zero value can be entered in this field during key-frame interpolation. When the current interpolation completes, this field will be cleared to 0.

Sequence Management Data

A single sequence pointer can be used to define multiple sequences, and the sequences can be played back selectively. In these cases, the selected sequence data is added after the last sequence pointer.

This information is referred to as sequence management data. It consists of the final word of the sequence pointer with TRAVELING omitted.

Figure 35: Sequence Management Data

-	STREAM ID	START IDX

Sequence Index

This field holds the index of the sequence control descriptor at the starting point of the sequence. The application can start a sequence by copying the sequence index into the sequence pointer's TCTR IDX and setting RFRAME to 0.

STREAM ID

Holds the STREAM ID for the starting sequence. The application can start a sequence by copying this value into the STREAM ID of the sequence pointer.

Interpolation Functions Table Section

The interpolation method for key frames can be varied even within a single sequence. The interpolation method is specified with an index into a type array. All sequence descriptors except jumps have this index, which can be used to specify the interpolation method.

The interpolation function table section is an area that contains this type array.

The entry in the type array of the interpolation function table is converted beforehand to the starting address of the primitive driver for that type. This operation is performed by the SCAN function GsScanAnim(). When a SCAN is required, the INI bit of TYPE should be set to 1.

The SCAN function for the interpolation function table is called when the SCAN operation for the HMD data is performed. After the SCAN completes, the type is updated with the starting address of the frame update driver function and the INI bit is set to 0.

The first word of the interpolation function table section contains the number of types. The uppermost bit is used as a flag indicating whether a SCAN operation (GsScanAnim()) was performed. If the flag is set to 1, a SCAN has not been performed. 0 indicates that SCAN has been performed.

Sequence Control Section

The actual sequence is represented in the sequence control section as a list. One element of the list is defined as the sequence descriptor. Sequence descriptors can be classified as one of two types. One type is the descriptor for a sequential sequence. The other type is the descriptor for a branching sequence. The uppermost bit of the sequence descriptor determines the type.

- MSB: bit31 - Identifier that indicates whether or not the sequence control descriptor points to a normal key frame.

  0: PARAMETER IDX
  1: SEQUENCE IDX

Figure 36: Sequence Descriptor (Normal)

31	24	16	0
0	TYPE IDX	TFRAME	PARAMETER IDX

- TYPE IDX: This field is an index into the interpolation function table, which specifies the interpolation function to be used. Since seven bits are available. Up to 128 interpolation functions can be accessed.
- TFRAME: The frame number of the next sequence descriptor (in integer format). When this value is placed in the TFRAME member of the sequence pointer, it must be converted to fixed-point decimal format (with base 0x10).
- PARAMETER IDX: Index to parameter data for the key frame referred to by the sequence descriptor.

Figure 37: Sequence Descriptor (Jump)

31		23		16 … 0
1	0	SID DST	SID CND	SEQUENCE IDX

- STREAM ID @bit16-29: The STREAM ID can be used to define multiple sequence links in a single sequence. STREAM IDs are divided into a SID DST (upper 7 bits) and a SID CND (lower 7 bits). SID DST specifies the STREAM ID for the destination of the jump while SID CND determines whether a jump will be performed when the STREAM IDs matches.

  SID CND 0 matches to any current stream ID. In this case, SID DST will not be updated.

  SID 127 is reserved to use as an ID that never matches to any stream ID except 0.

  The Stream ID is updated according to the following rules.

  DST = 0 and CND = 0: Unconditional jump. The Stream ID is not updated.

  DST = 0 and CND != 0: Jump if the current SID matches CND.
  The Stream ID is set to 0.

  DST != 0 and CND = 0: Unconditional jump. The Stream ID is set to DST.

  DST != 0 and CND != 0: Jump if the current SID matches CND.
  The stream ID is set to DST.

  SID 127 is defined to not match any non-zero stream ID.
- SEQUENCE IDX: Contains the index of the control descriptor for the destination of the jump.

Figure 38: Sequence Descriptor (Control)

31		23		16 … 0
1	1	CODE	P1	P2

The parameters P1 and P2 can take on different values depending on CODE.

CODE: 0x01: END

If P1 matches the current STREAM ID, the sequence is halted.

CODE: 0X02: WORK

This indicates work area for each sequence pointer that is required by BSPLINE interpolation.

P1=127 Fixed

P2: Offset in parameter section indicates work area (in words).

Notes Regarding Switching of Interpolation Functions During a Sequence

A single interpolation function can be defined for each sequence control descriptor so that the interpolation function can be switched for each key frame. However, the parameters of the interpolation function must have the same format. Thus, if the interpolation function is switched, the program must ensure that the parameter format for the SRC FRAME and the DST FRAME match.

Example:

KEY 0 (parameter format A) TFRAME = 0

KEY 1 (parameter format B) TFRAME = 30

Interpolation cannot be performed here since the SRC FRAME and the DST FRAME has different parameter formats. In this case, a sequence control descriptor is added to unify the formats.

KEY 0 (parameter format A) TFRAME = 0

KEY 00 (parameter format B) TFRAME = 0

KEY 1 (parameter format B) TFRAME = 30

KEY00 performs parameter format conversion from A to B. The TFRAME of the descriptor must be 0 to perform this conversion. Note that the sequence will jump if there is a discontinuity between KEY0 and KEY00.

Behavior of Interpolation Driver When TFRAME is 0

Even if TFRAME is 0, interpolation driver is called. Thus, any interpolation driver should return without interpolation if TFRAME is 0. It is possible to use TFRAME=0 to change internal status of interpolation driver. For example, first 3 control points for spline function are written as key frames with TFRAME=0.

While TFRAME is 0 or return value of interpolation driver is 1, interpolation driver is called continuously, and RFRAME is not updated.

Parameter Section

The parameter section contains the actual parameters and that is referenced by an index in the sequence control section. The parameters in this section can take on various forms (for example, VECTORs and MATRIXes). The code, which accesses these parameters, is responsible for their management.

Run-time Environment of the Animation Primitive Driver

The animation frame update primitive driver and the interpolation primitive driver are called with the following environment.

Figure 39: Format of Parameters in the Argument Area

primtop
tag(OT)
shift(OT)
offset(OT)
OUTP(packet area)
Animation header size
Interpolation function table pointer
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
???
base
src
dst
intr

The colored areas must always be set. The other areas are copied from the primitive header, so these areas will be updated if the header format changes.

The animation header size specifies the number of elements after the interpolation function table pointer exclusive of the last four elements. In the example above, the header size would have a value of "???+4" with the "???" determined from the element count. The header size is used by the interpolation function to locate the start of the interpolation function's parameter section (described next).

The last four parameters are the arguments area for the interpolation function.

- base: starting address of the sequence pointer
- src: starting address of the source key frame to interpolate
- dst: starting address of the destination key frame to interpolate
- intr: address where parameters will be saved after interpolation (if this value is 0, the parameters will not be saved)

Behavior of the Primitive Driver

Primitive drivers can be divided into the following two types:

1. Frame update drivers
2. Interpolation drivers

Primitive drivers are called each time GsSortUnit() is called.

Animation primitives are linked in the PRE-PROCESS area at the beginning of HMD's coordinate section. The animation primitive driver is initialized in the following manner.

1. When HMD initialization is performed with GsScanUnit(), GsScanAnim() should be called to perform a SCAN operation.
2. The starting address of the frame update driver should be entered in the HMD type field. This ensures that the frame update driver will be called each time GsSortUnit() is called. The frame update driver will call the interpolation driver.

The frame update driver specifies the calling interface for the interpolation driver. Thus, the program must be aware of the relationship between the interpolation driver and the frame update driver. The three bits in the type field that identify the frame update driver must be the same for the corresponding interpolation function.

The calling interface used by the standard frame update driver to call an interpolation function is described below.

FUNC(sp)

sp is a pointer to the start of the parameter area.

As described above, the parameter area pointed to by sp contains the base, src, dst, and intr parameters.

- base: starting address of the sequence pointer corresponding to an update area which begins at the update index
- src: address of the interpolation source
- dst: address of the interpolation destination
- intr: address for holding interpolated data. Data is not saved if this value is 0. To make interpolation parameter, intr is allowed to use to indicate destination key frame created previously.

The frame update driver provided by Sony Computer Entertainment Inc., has type set to 0x03000000. The corresponding interpolation primitive driver needs to have the parameter format described above.

Interpolation Algorithms

The following 3 algorithms are available for interpolation driver.

1. LINEAR
2. BEZIER
3. BSPLINE

LINEAR

This interpolates linear between SRC KEY FRAME and DST KEY FRAME parameters.

T = (TFRAME-RFRAME)/TRFAME

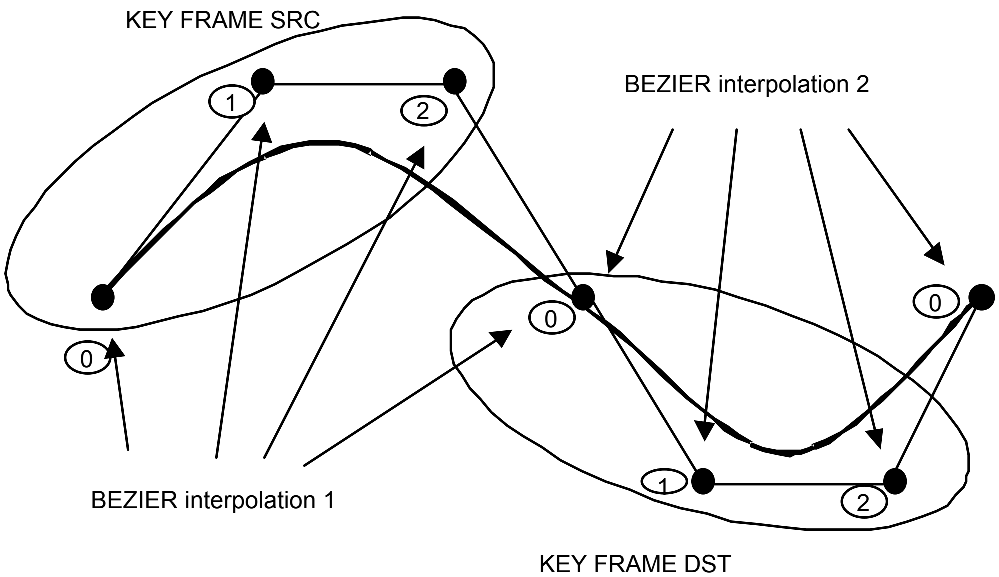
(1-T)*SRC_KEYFRAME + T*DST_KEY FRAME

BEZIER

BEZIER type KEY FRAME has 3 control points.

Interpolation is performed with control point 0, 1 and 2 of SRC KEY FRAME, and control point 0 of DST KEY FRAME.

Figure 40: Bezier Interpolation



BSPLINE

BSPLINE type KEY FRAME has a control point as same as LINEAR type.

BSPLINE interpolation is performed between SRC-2, SRC-1, SRC and DST KEY FRAME.

The beginning of sequence has no history of previous key frames, thus, 3 key frames are required to enumerated with TFRAME=0.

To make a history of key frames, 4 words in key frame area of parameter section are required. Sequence descriptor (control: work) that indexed by START IDX in sequence pointer, indicates this area.

Figure 41: BSPLINE Work Area

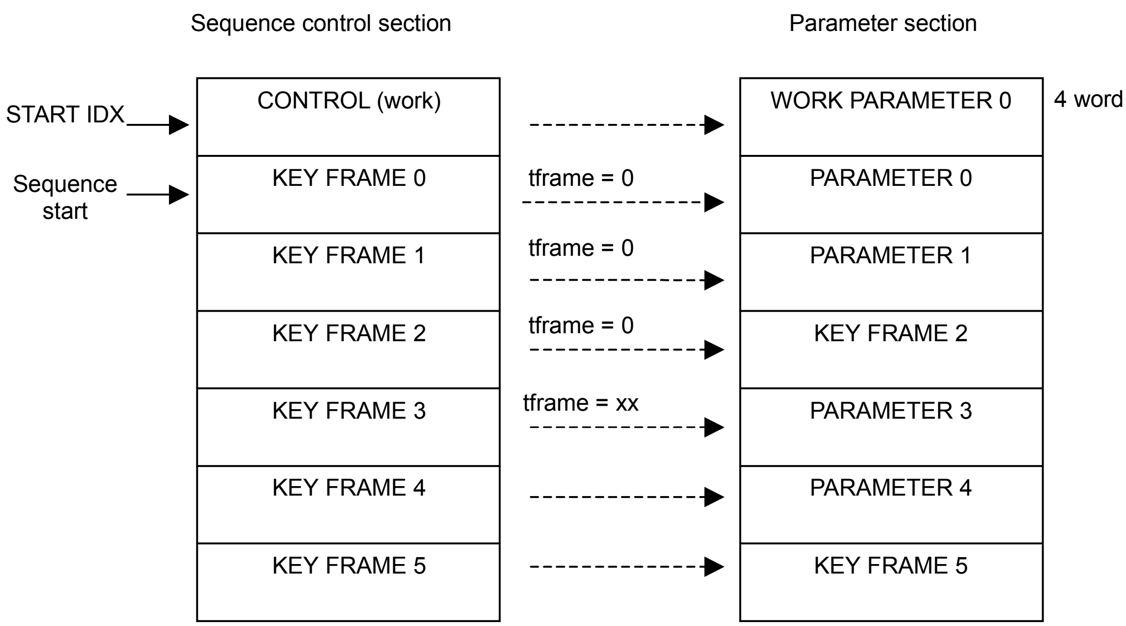
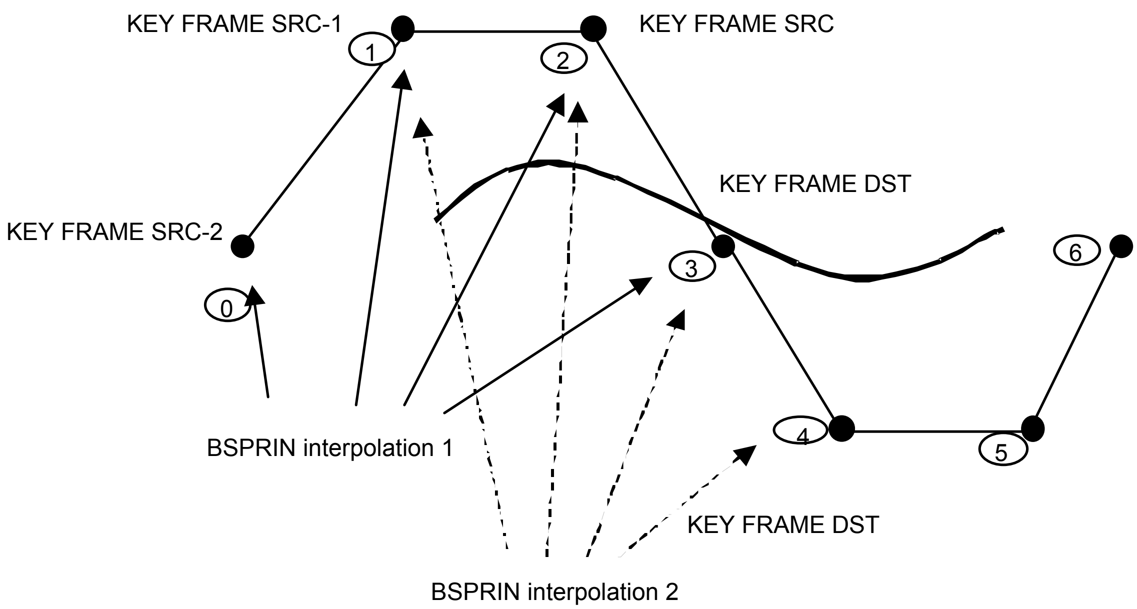


Figure 42: BSPLINE Interpolation



Animation Packets (COORDINATE)

```
DEV_ID(SCE)|CTG(CTG_ANIM)|DRV(CAT_STD|TGT_COORD)|PRIM_TYPE(x)
```

PARAMETER

```
0x03000010; SI_NONE|RI_LINEAR|TI_NONE
H(rx); H(ry); H(rz); H(0);

0x03000910; SI_LINEAR_1|RI_LINEAR|TI_NONE
H(rx); H(ry); H(rz);
H(scale);

0x03000030; SI_NONE|RI_BSPLINE|TI_NONE
H(rx); H(ry); H(rz); H(0);

0x03000001; SI_NONE|RI_NONE|TI_LINEAR
tx; ty; tz;

0x03000901; SI_LINEAR_1|RI_NONE|TI_LINEAR
tx; ty; tz;
H(scale); H(0);

0x03000011; SI_NONE|RI_LINEAR|TI_LINEAR
tx; ty; tz;
H(rx); H(ry); H(rz); H(0);

0x03000111; SI_LINEAR|RI_LINEAR|TI_LINEAR
tx; ty; tz;
H(rx); H(ry); H(rz);
H(sx); H(sy); H(sz);

0x03000911; SI_LINEAR_1|RI_LINEAR|TI_LINEAR
tx; ty; tz;
H(rx); H(ry); H(rz);
H(scale);

0x03000031; SI_NONE|RI_BSPLINE|TI_LINEAR
tx; ty; tz;
H(rx); H(ry); H(rz); H(0);

0x03000002; SI_NONE|RI_NONE|TI_BEZIER
tx0; ty0; tz0;
tx1; ty1; tz1;
tx2; ty2; tz2;

0x03000902; SI_LINEAR_1|RI_NONE|TI_BEZIER
tx0; ty0; tz0;
tx1; ty1; tz1;
tx2; ty2; tz2;
H(scale); H(0);

0x03000012; SI_NONE|RI_LINEAR|TI_BEZIER
tx0; ty0; tz0;
tx1; ty1; tz1;
tx2; ty2; tz2;
H(rx); H(ry); H(rz); H(0);

0x03000112; SI_LINEAR|RI_LINEAR|TI_BEZIER
tx0; ty0; tz0;
```

```
tx1; ty1; tz1;
tx2; ty2; tz2;
H(rx); H(ry); H(rz);
H(sx); H(sy); H(sz);
```

0x03000912; SI_LINEAR_1|RI_LINEAR|TI_BEZIER

```
tx0; ty0; tz0;
tx1; ty1; tz1;
tx2; ty2; tz2;
H(rx); H(ry); H(rz);
H(scale);
```

0x03000032; SI_NONE|RI_BSPLINE|TI_BEZIER

```
tx0; ty0; tz0;
tx1; ty1; tz1;
tx2; ty2; tz2;
H(rx); H(ry); H(rz); H(0);
```

0x03000003; SI_NONE|RI_NONE|TI_BSPLINE

```
tx; ty; tz;
```

0x03000013; SI_NONE|RI_LINEAR|TI_BSPLINE

```
tx; ty; tz;
H(rx); H(ry); H(rz); H(0);
```

0x03000033; SI_NONE|RI_BSPLINE|TI_BSPLINE

```
tx; ty; tz;
H(rx); H(ry); H(rz); H(0);
```

0x03000009; SI_NONE|RI_NONE|TI_LINEAR_S

```
H(tx); H(ty); H(tz); H(0);
```

0x03000909; SI_LINEAR_1|RI_NONE|TI_LINEAR_S

```
H(tx); H(ty); H(tz);
H(scale);
```

0x03000019; SI_NONE|RI_LINEAR|TI_LINEAR_S

```
H(tx); H(ty); H(tz);
H(rx); H(ry); H(rz);
```

0x03000119; SI_LINEAR|RI_LINEAR|TI_LINEAR_S

```
H(tx); H(ty); H(tz);
H(rx); H(ry); H(rz);
H(sx); H(sy); H(sz); H(0);
```

0x03000919; SI_LINEAR_1|RI_LINEAR|TI_LINEAR_S

```
H(tx); H(ty); H(tz);
H(rx); H(ry); H(rz);
H(scale); H(0);
```

0x03000039; SI_NONE|RI_BSPLINE|TI_LINEAR_S

```
H(tx); H(ty); H(tz);
H(rx); H(ry); H(rz);
```

0x0300000a; SI_NONE|RI_NONE|TI_BEZIER_S

```
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2); H(0);
```

0x0300090a; SI_LINEAR_1|RI_NONE|TI_BEZIER_S

```
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2);
H(scale);
```

0x0300001a; SI_NONE|RI_LINEAR|TI_BEZIER_S
```
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2);
H(rx); H(ry); H(rz);
```

0x0300011a; SI_LINEAR|RI_LINEAR|TI_BEZIER_S
```
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2);
H(rx); H(ry); H(rz);
H(sx); H(sy); H(sz); H(0);
```

0x0300091a; SI_LINEAR_1|RI_LINEAR|TI_BEZIER_S
```
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2);
H(rx); H(ry); H(rz);
H(scale); H(0);
```

0x0300003a; SI_NONE|RI_BSPLINE|TI_BEZIER_S
```
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2);
H(rx); H(ry); H(rz);
```

0x0300000b; SI_NONE|RI_NONE|TI_BSPLINE_S
```
H(tx); H(ty); H(tz); H(0);
```

0x0300001b; SI_NONE|RI_LINEAR|TI_BSPLINE_S
```
H(tx); H(ty); H(tz);
H(rx); H(ry); H(rz);
```

0x0300003b; SI_NONE|RI_BSPLINE|TI_BSPLINE_S
```
H(tx); H(ty); H(tz);
H(rx); H(ry); H(rz);
```

0x03000020; SI_NONE|RI_BEZIER|TI_NONE
```
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2); H(0);
```

0x03000021; SI_NONE|RI_BEZIER|TI_LINEAR
```
tx; ty; tz;
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2); H(0);
```

0x03000022; SI_NONE|RI_BEZIER|TI_BEZIER
```
tx0; ty0; tz0;
tx1; ty1; tz1;
tx2; ty2; tz2;
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2); H(0);
```

```
0x03000023; SI_NONE|RI_BEZIER|TI_BSPLINE
tx; ty; tz;
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2); H(0);

0x03000029; SI_NONE|RI_BEZIER|TI_LINEAR_S
H(tx); H(ty); H(tz);
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2);

0x0300002a; SI_NONE|RI_BEZIER|TI_BEZIER_S
H(tx0); H(ty0); H(tz0);
H(tx1); H(ty1); H(tz1);
H(tx2); H(ty2); H(tz2);
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2);

0x0300002b; SI_NONE|RI_BEZIER|TI_BSPLINE_S
H(tx); H(ty); H(tz);
H(rx0); H(ry0); H(rz0);
H(rx1); H(ry1); H(rz1);
H(rx2); H(ry2); H(rz2);
```

Animation Packets (General)

```
DEV_ID(SCE)|CTG(CTG_ANIM)|DRV(CAT_STD|TGT_GENERAL)|PRIM_TYPE(x)

LINEAR
General Single Linear(32bit)
0x03010110; GI_LINEAR|GI_WR(0x1)|GI_32
p0;

General Single Linear(32bit)
0x03010111; GI_LINEAR|GI_WR(0x1)|GI_16
0x03010121; GI_LINEAR|GI_WR(0x2)|GI_16
0x03010141; GI_LINEAR|GI_WR(0x4)|GI_16
H(p0); H(0);

General vector Linear(16bit)
0x03010171; GI_LINEAR|GI_WR(0x7)|GI_16
H(p0); H(p1); H(p2); H(0);

General Single Linear(8bit)
0x03010112; GI_LINEAR|GI_WR(0x1)|GI_8
0x03010122; GI_LINEAR|GI_WR(0x2)|GI_8
0x03010142; GI_LINEAR|GI_WR(0x4)|GI_8
B(p0); B(0); B(0); B(0);

General vector Linear(8bit)
0x03010172; GI_LINEAR|GI_WR(0x7)|GI_8
B(p0); B(p1); B(p2); B(0);
```

BEZIER

```
General single Bezier(32bit)
0x03010210; GI_BEZIER|GI_WR(0x1)|GI_32
p00; p10; p20;
```

General single Bezier(16bit)
```
0x03010211; GI_BEZIER|GI_WR(0x1)|GI_16
0x03010221; GI_BEZIER|GI_WR(0x2)|GI_16
0x03010241; GI_BEZIER|GI_WR(0x4)|GI_16
H(p00); H(p10); H(p20); H(0);
```

General vector Bezier(16bit)
```
0x03010271; GI_BEZIER|GI_WR(0x7)|GI_16
H(p00); H(p01); H(p02);
H(p10); H(p11); H(p12);
H(p20); H(p21); H(p22); H(0);
```

General single Bezier(8bit)
```
0x03010212; GI_BEZIER|GI_WR(0x1)|GI_8
0x03010222; GI_BEZIER|GI_WR(0x1)|GI_8
0x03010242; GI_BEZIER|GI_WR(0x1)|GI_8
B(p00); B(p10); B(p20); B(0);
```

General vector Bezier(8bit)
```
0x03010272; GI_BEZIER|GI_WR(0x7)|GI_8
B(p00); B(p01); B(p02); B(0);
B(p10); B(p11); B(p12); B(0);
B(p20); B(p21); B(p22); B(0);
```

BSPLINE

General Single Bspline(32bit)
```
0x03010310; GI_BSPLINE|GI_WR(0x1)|GI_32
p0;
```

General Single Bspline(16bit)
```
0x03010311; GI_BSPLINE|GI_WR(0x1)|GI_16
0x03010321; GI_BSPLINE|GI_WR(0x2)|GI_16
0x03010341; GI_BSPLINE|GI_WR(0x4)|GI_16
H(p0); H(0);
```

General vector Bspline(16bit)
```
0x03010371; GI_BSPLINE|GI_WR(0x7)|GI_16
H(p0); H(p1); H(p2); H(0);
```

General single Bspline(8bit)
```
0x03010312; GI_BSPLINE|GI_WR(0x1)|GI_8
0x03010322; GI_BSPLINE|GI_WR(0x2)|GI_8
0x03010342; GI_BSPLINE|GI_WR(0x4)|GI_8
B(p0); B(0); B(0); B(0);
```

General vector Bspline(8bit)
```
0x03010372; GI_BSPLINE|GI_WR(0x7)|GI_8
B(p0); B(p1); B(p2); B(0);
```

MIMe Primitive (Category 4)

Please refer to the following documents for more information on the MIMe primitive.

- libhmd reference, section on the GsARGUNIT_JntMIMe structure
- libhmd reference, section on the GsARGUNIT_RstJntMIMe structure
- libhmd reference, section on the GsARGUNIT_VNMIMe structure
- libhmd reference, section on the GsARGUNIT_RstVNMIMe structure
- libhmd reference, section on the GsInitRstVtxMIMe, GsInitRstNrmMIMe function
- libhmd reference, section on the GsU_04# function

The following symbols are used to indicate the MIMe type in a MIMe primitive.

- **JntMIMe** Joint MIMe (common to the following two types)
  JntAxesMIMe: Joint-axes MIMe (Joint MIMe using rotation-axes interpolation)
  JntRPYMIMe: Joint row-pitch-yaw MIMe (Joint MIMe using RPY interpolation)
- **RstJntMIMe** (common to the following two types)
  RstJntAxesMIMe: Reset MIMe based on rotation-axes interpolation
  RstJntRPYMIMe: Reset MIMe based on RPY interpolation
- **VNMIMe** Vertex / normal MIMe (common to the following two types)
  VtxMIMe: Vertex MIMe
  NrmMIMe: Normal MIMe
- **RstVNMIMe** Reset vertex / normal MIMe (common to the following two types)
  RstVtxMIMe: Reset vertex MIMe
  RstNrmMIMe: Reset normal MIMe

Areas needed specifically for MIMe primitives

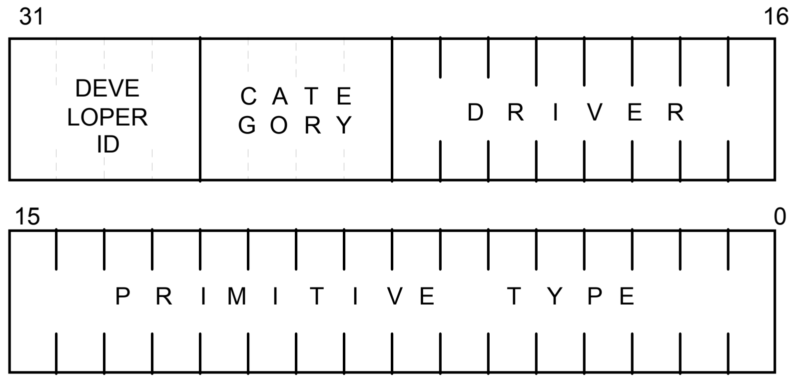
- MIMe DIFF section
- ORGSVN section (for VtxMIMe, NrmMIMe)
- MIMEPR area (when HMD contains MIMEPR)

Notes on Formats

- Up to 32 MIMe differences can be used for a single primitive.
- The JntMIMe function uses the same primitive block as the corresponding reset function (RstJntMIMe). However, VNMIMe and RstVNMIMe do not share this block and use their own primitive.
- When two or more JntMIMe primitives are used for a single joint, the corresponding reset functions (RstJntMIMe) must be called in reverse order otherwise, the state will not be correct).

type

Figure 43: Primitive Type Field



DEVELOPER ID - 0: SCE

CATEGORY - 4: MIMe data

DRIVER

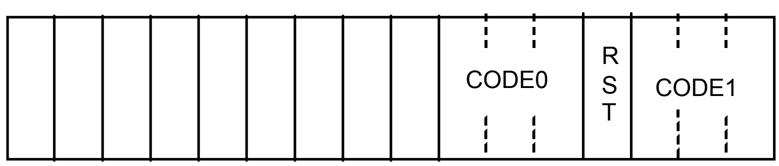
In MIMe category, DRIVER bits are defined as below.

Figure 44: MIMe Primitive DRIVER

Always 0x01

PRIMITIVE TYPE

Figure 45: Primitive Type of MIMe Primitive



RST
0: MIMe primitive to do MIMe
1: Reset MIMe primitive

CODE0 - Major categorization of interpolation method
0: JntMIMe
1: VNMIMe

CODE1 - Minor categorization of interpolation method (depends on value of CODE0)
CODE0=0 (JntMIMe)
  0: JntAxesMIMe
  1: JntRPYMIMe
CODE0=1(VNMIMe)
  0: VtxMIMe
  1: NrmMIMe

Format

Header for MIMe Primitive Block

- HEADLEN: Length of primitive header.
  This value will be changed by GsMap...MIMe(), GsMapRst....MIMe() functions.
- COORD TOP: Starting address of COORDINATE section (the number of long words from start of HMD)
- MIMEPR PTR: If HMD contains MIMEPR, the number of long words from start of HMD.
  If MIMEPR is outside of HMD, the value is 0.
- MIMENUM: The number of the MIMe keys.
  reserved(16bit): reserved (0)
- MIMEID(16bit): ID of the primitive (this area can be used freely by user and modeler)
- MIMe DIFF TOP: starting address of MIME DIFF section (number of long words from start of HMD)
- ORGSVN TOP: starting address of ORGSVN section (number of long words from start of HMD)
- VERTEX TOP: starting address of VERTEX section (number of long words from start of HMD)
- NORMAL TOP: starting address of NORMAL section (number of long words from start of HMD)

```
MIMeHeader(JntMIMe)
5;    /* header size */
M(CoordSect / 4);
M(MIMePr_ptr / 4);
MIMe_num;
H(MIMeID); H(0 /* reserved */);
M(MIMeDiffSect / 4);

MIMeHeader(RstJntMIMe)
3;     /* header size */
M(CoordSect / 4);
H(MIMeID); H(0 /* reserved */);
M(MIMeDiffSect / 4);

MIMeHeader(VNMIMe)
7;     /* header size */
M(MIMePr_ptr / 4);
MIMe_num;
H(MIMeID); H(0 /* reserved */);
M(MIMeDiffSect / 4);
M(MIMeOrgsVNSect / 4);
M(VertSect / 4);
M(NormSect / 4);

MIMeHeader(RstVNMIMe)
5;     /* header size */
H(MIMeID); H(0 /* reserved */);
M(MIMeDiffSect / 4);
M(MIMeOrgsVNSect / 4);
M(VertSect / 4);
M(NormSect / 4);
```

MIMe Primitive

- TYPE: type of the primitive.
- m(1bit): Initial value is 1 (changes to 0 during execution when TYPE is scanned and the function pointer is embedded)
- Num of DIFFs: MIMe DIFF IDX count

- MIME DIFF IDX: starting address of MIME DIFF (number of long words from MIMe DIFF TOP)

```
MIMe primitive
DEV_ID(SCE)|CTG(CTG_MIMe)|DRV(MIMe_PRIM)|PRIM_TYPE(x)
H(size); M(H(num_diffs));  /* size = num_diffs + 1 */
(MIMeDiff0 - MIMeDiffSect) / 4;
     :
(MIMeDiffN - MIMeDiffSect) / 4; /* N = num_diffs - 1 */
```

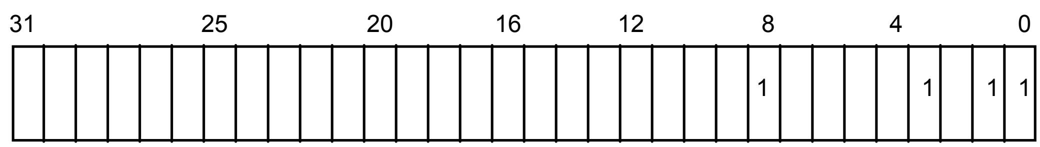
MIMe DIFF

Data in the MIMe DIFF section related to differences.

- DIFFS NUM: number of DIFFS (DIFFS for Rst are not counted)
- COORDID: COORDINATE ID (the joint to apply MIMe)
- ONUM: Number of RstVNMIMe blocks that correspond to VNMIMe.
- dflags: bits with differences (DIFFS) are set to 1, 0 otherwise.

Example: When MIMe-key #0, #1, #3, #8 have differences

Figure 46: dflags Example



->In this case, dflags=0x0000010B

- VNMIMe Changed:

  The changed address within RstVNMIMeDiffData of the corresponding RstVNMIMe

- MIMeDiffData:

  For Rst, original data for resets.

  Otherwise, actual difference values for each key. The DIFFS must be ordered in the same sequence as the dflags bits.

  Details of the formats are shown below.

```
JntMIMeDiff/RstJntMIMeDiff
JntMIMe and RstJntMIMe are paired and use the same MIMeDIFF.
H(coord_ID); H(diffs_num);
dflags;
JntMIMeDiffData0:
     :     /* Jnt???MIMeDiffData format */
JntMIMeDiffDataN:
     :     /* N = diffs_num - 1 */
RstJntMIMeDiffData:
     :     /* RstJnt???MIMeDiffData format */

VNMIMeDiff
VNMIMeDiff:
H(onum); H(diffs_num);
dflags;
(VNMIMeDiffData0 - VNMIMeDiff) / 4;
     :
(VNMIMeDiffDataN - VNMIMeDiff) / 4;     /* N = diffs_num - 1 */
(VNMIMeChanged0 - MIMeDiffSect) / 4;
     :
```

```
(VNMIMeChangedM - MIMeDiffSect) / 4;    /* M = onum - 1 */
VNMIMeDiffData0:
     :     /* VNMIMeDiffData format */
VNMIMeDiffDataN:
     :     /* N = diffs_num - 1 */
```

RstVNMIMeDiff

```
H(0); H(diffs_num);
RstVNMIMeDiffData0:
     :     /* RstVNMIMeDiffData format */
RstVNMIMeDiffDataN:
     :     /* N = diffs_num - 1 */
```

MimeDiffData

Actual difference values for each key.

The format and contents vary according to the interpolation method.

Difference Value Data

dtp:

Bit 0 is 0 when the rotation values (dvx-dvz and m) are all 0. Otherwise, Bit 0 is 1.

Bit 1 is 0 when the translation values (dtx-dtz) are all 0. Otherwise, Bit 1 is 1.

JntRPYMIMeDiffData

```
H(dvx); H(dvy); H(dvz); H(dtp);     /* rot difference value */
dtx; dty; dtz;                      /* t[0-2] difference value */
```

JntAxesMIMeDiffData

```
H(dvx); H(dvy); H(dvz); H(dtp);     /* rot difference value rotation vector
*/
dtx; dty; dtz;                      /* t[0-2] difference value */
```

VNMIMeDiffData

```
vstart;                             /* number of first different vertex */
H(0 /* reserved */); H(vnum);       /* number of difference vertices */
H(dvx0); H(dvy0); H(dvz0); H(0);
     :
H(dvxN); H(dvyN); H(dvzN); H(0);    /* N = vnum - 1 */
```

Original reset data

RstVNMIMeDiffData

```
vstart;         /* Number of the first vertex/normal which is */
                /* different */
ostart;         /* Number of ORGSVN area start which is used */
VNMIMeChanged:  /* Referred from VNMIMeDiff */
H(changed);     /* Initial value 0 */
                /* At runtime, this value will be changed to 1 */
                /* when the vertices or normal vectors in this */
                /* region are changed to 0 when RstMIMe is reset*/
H(vnum);        /* Number of different vertices/normals */
```

RstJntRPYMIMeDiffData

```
H(dvx); H(dvy); H(dvz);    /* Initial value is undefined */
                /* The original coordinate's rot value will */
                /* be saved here during execution */
H(changed);     /* Initial value is 0; flag indicating data was */
                /* saved */
dtx; dty; dtz;  /* Initial value is undefined. The */
```

```
                /* original coordinate's t[0-2] value */
                /* will be saved here during execution */

RstJntAxesMIMeDiffData
H(m00); H(m01); H(m02);     /* Initial value is undefined */
                /* The original coordinate's m[0-2] */
                /* [0-2] value will be saved here during */
                /* execution*/
H(m10); H(m11); H(m12);
H(m20); H(m21); H(m22);
H(changed);     /* Initial value is 0; flag indicating data was */
                /* saved */
dtx; dty; dtz;  /* Initial value is undefined. The */
                /* original coordinate's t[0-2] value */
                /* will be saved here during execution */
```

MIMeOrgsVN Section

Initial values are not defined. These values are used in the following manner during execution. dx-z is the original vertex/normal data that had been saved.

```
MIMeOrgsVN
H(dvx0); H(dvy0); H(dvz0); H(0);
     :
H(dvxN); H(dvyN); H(dvzN); H(0);
```

Ground Primitives (Category 5)

Ground primitive is allowed to use as one of HMD primitive. This primitive generates packets at run time based on width and height of a grid, and count of grids. Thus, data amount can be reduced in HMD data.

Primitive Header Section

Primitive header section

Primitive header format depends on texture is used or not.

(1) Non-textured

```
4;      /* header size */
M(GndPolySect / 4);     /* Polygon section */
M(GndGridSect / 4);     /* Grid section */
M(GndVertSect / 4);     /* Vertex section */
M(GndNormSect / 4);     /* Normal section */
```

(2) Textured

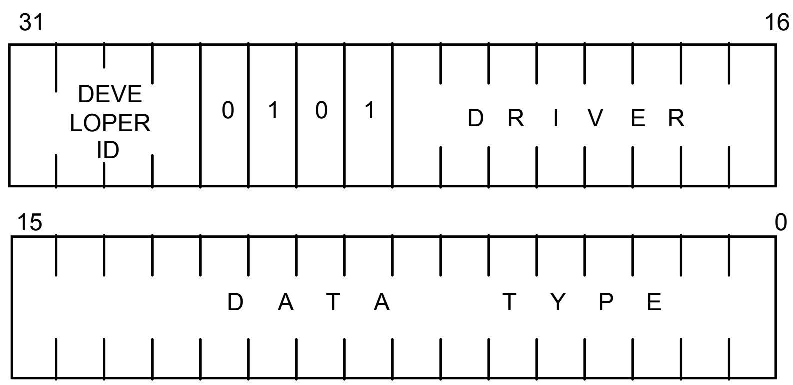
```
5;      /* header size */
M(GndPolySect / 4);     /* Polygon section */
M(GndGridSect / 4);     /* Grid section */
M(GndVertSect / 4);     /* Vertex section */
M(GndNormSect / 4);     /* Normal section */
M(GndUVSect / 4);       /* UV section */
```

Type

Type of ground primitive is defined as below.

Ground TYPE

Figure 47: Ground Primitive Type Field



DRIVER: All 0 in this version
DATA TYPE: Defines type of data
0: Flat
1: Flat texture

Primitive Section

Primitive section is common for non-textured and textured type.

```
Ground primitive
DEV_ID(SCE)|CTG(CTG_GND)|DRV(x)|PRIM_TYPE(y)
H(size); H(0);
(GndPoly - GndPolySect) / 4;
(GndGrid - GndGridSect) / 4;
(GndVert - GndVertSect) / 4;
```

Polygon Section

Required information to generate actual polygons is saved in polygon section.

Polygon section is common for non-textured and textured type.

```
H(x0); H(y0);        /* Start point X coordinate; start point Y */
                     /* coordinate */
H(w); H(h);          /* 1 grid width; 1 grid height */
H(m); H(n);          /* Vertices count (horizontal); vertices */
                     /* count (vertical) */
H(size); H(base);    /* Size; base vertex */
H(v0); H(c0);        /* Start vertex number 0; grids count 0 */
     :
H(vN); H(cN);        /* Start vertex number N; Grids count N; N; N = */
                     /* size - 1 */
```

Grid Section

Grid section has information for each grid, for example, indexes to normal vectors, RGB value and UV.

Grid section format depends on non-textured or textured type.

(1) Non-textured

```
B(r); B(g); B(b); B(0);
H(norm_idx); H(0);
     :
B(r); B(g); B(b); B(0);
H(norm_idx); H(0);
```

(2) Textured

```
H(norm_idx); H(UV_idx);
     :
H(norm_idx); H(UV_idx);
```

Vertex Section

Vertex section has information for each vertex, for example, Z value.

```
H(z0); H(z1);
     :
H(zN-1); H(zN);
```

UV section

UV section has actual texture UV values that are referred from grid section.

```
H(uv0); H(cba);
H(uv1); H(tsb);
```

```
H(uv2); H(uv3);
     :
H(uv0); H(cba);
H(uv1); H(tsb);
H(uv2); H(uv3);
```

Device Primitives Section (Category 7)

Device primitives are primitives that perform settings such as camera (viewpoint) and light (light source). By using these primitives, it is possible to maintain camera and light settings that used to be made within the application. With the exception of certain cases, linking should be performed as a standard preprocess.

Currently, the following primitives are supported as device primitives.

- Camera primitive
- Light primitive

Camera Primitives

With camera primitives, settings such as projection and camera position and direction can be made. The following types of camera primitives are available.

Projection

Adjusts the field of view. Projection refers to the distance from the viewpoint to the projection plane. The size of the projection plane is determined by the resolution for the GsInitGraph() function.

WORLD Camera

Sets the camera position on the WORLD coordinate system and calculates WSMATRIX.

FIX Camera

Sets the camera position on a coordinate system other than world and calculates WSMATRIX.

AIM Camera

A position on one coordinate system is referenced from a camera position on another coordinate system, and WSMATRIX is calculated.

Light Primitives

With light primitives, settings such as ambient color and lighting direction can be made. The following types of light primitives are available:

Ambient Color

Sets the ambient color.

WORLD Light

Sets light (flat light source) on the WORLD coordinate system.

FIXCg

Sets light (flat light source) on a coordinate system other than WORLD.

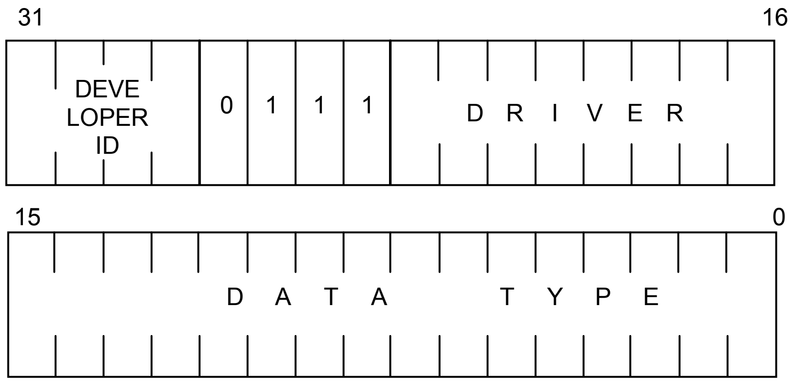
AIM Camera

A position on one coordinate system is referenced from a camera position on another coordinate system, and light (flat light source) is set.

Types

The following types of device primitives are available

Figure 48: Type fields for device primitives



DATA TYPE

Specifies the type of data

0x0100: Camera primitive

0x0200: Light primitive

DRIVER

Specifies the type of primitive operation. Varies according to DATA TYPE.

Camera Primitive

0x00: Projection

0x01: WORLD camera

0x02: FIX camera

0x03: AIM camera

Light Primitive

0x00: Ambient color

0x01: WORLD light

0x02: FIX light

0x03: AIM light

Primitive Header Section

Camera primitives and light primitives have different primitive headers.

Camera Primitive Header

```
3;                       /* header size :
                         Projection, WORLD camera  1
                         FIX camera                2
                         AIM camera                3 */
M(CameraParamSect / 4);  /* Camera parameter section */
M(CameraCoord / 4);      /* Coordinate system in which camera is positioned :
                         Nothing for projection, WORLD camera*/
M(ReferenceCoord / 4);   /* Coordinate system referenced by camera :
                         Nothing for projection, WORLD camera,
                         FIX camera*/
```

Light Primitive Header

```
3;                       /* header size :
                         Ambient color, WORLD light  1
                         FIX light                   2
                         AIM light                   3 */
M(LightParamSect / 4);   /* Light parameters section */
M(LightCoord / 4);       /* Coordinate system in which light is positioned :
                         Nothing for ambient color, WORLD light */
M(ReferenceCoord / 4);   /* Coordinate system referenced by light :
                         Nothing for ambient color, WORLD light,
                         FIX light */
```

Primitive Section

Camera primitives and light primitives have different primitive sections.

Camera Primitives

```
DEV_ID(SCE)|CTG(CTG_EQUIP)|DRV(x)|PRIM_TYPE(CAMERA)
H(1); H(0);
```

Light Primitives

```
DEV_ID(SCE)|CTG(CTG_EQUIP)|DRV(x)|PRIM_TYPE(LIGHT)
H(2); H(1);     /* size, data */
H(n); H(idx);   /* n: light number(0,1,2)
                   idx: light parameter index (number of words) */
```

Parameter Section

Camera primitives and light primitives have different parameter sections.

Camera Primitives

```
proj;               /* Projection */
rot;                /* Camera rotation; 4096 is equivalent to 1 degree*/
vx, vy, vz;         /* Camera position
                    WORLD camera:  in WORLD coordinate system
                    FIX camera, AIM camera: in local coordinate system */
rx, ry, rz;         /* position of target point
                    WORLD camera:  in WORLD coordinate system
                    FIX camera:    in local coordinate system to which
                                   camera belongs
                    AIM camera:    in local coordinate system to which
                                   target point belongs */
```

Light Primitive

```
B(r);B(g);B(b);B(0);/* color of light */
vx, vy, vz;         /* position of light
                    ambient color: none
                    WORLD light:   in WORLD coordinate system
                    FIX light, AIM light: in local coordinate system */
rx, ry, rz;         /* position of target point
                    ambient color: none
                    WORLD light:   in WORLD coordinate system
                    FIX light:     in local coordinate system to which light
                                   belongs
                    AIM light:     in local coordinate system to which
                                   target point belongs */
```

Appendix A: HMD Library Primitive Types

The list of installed primitives which previously appeared here has been moved into the excel spreadsheet called “Installation status of HMD primitive drivers. Following is an explanation of the primitive type list description rules.

The “libhmd” sheet in this spreadsheet presents a list of primitive types implemented in the HMD library. The list is shown in HMD assembler (labp) format. The following notation is used:

```
DEV_ID(SCE)|CTG(CTG_POLY)|DRV(BOT)|PRIM_TYPE(TRI); /* 00100008; 4.2 */
```

In this example, the developer ID is "SCE" (0; standard primitive driver), the category is "CTG_POLY" (polygon primitive), the driver bit is "BOT" (double-sided flag ON), and the primitive type is "TRI" (triangle). The actual bit pattern is "00100008" in hexadecimal. A primitive driver function name can be obtained by adding "GsU_" to the actual bit pattern value. If there is no designation, the primitive type was implemented in version 4.1 or earlier.

Library 4.3 provides a beta release of a pseudo-environment map driver. These are expressed using the following notation.

```
DEV_ID(SCE)|CTG(6)|DRV(0x00 /* ??? */)|PRIM_TYPE(0x0100 /* ??? */);
/* 06000100; 4.2 */
```

Since the pseudo-environment map driver is a beta release, symbol definitions are not included in the "hmd.def" HMD assembler definition file. Also, symbolic output is not supported in the "xhmd" HMD disassembler. This document, "hmd.doc", does not describe pseudo-environment mapping. A brief description is provided in the sample data directory.

Appendix B: HMD Animation

The HMD library also supports animation. Since HMD holds coordinate information, the motion of a hierarchical model can be described.

A special characteristic of HMD animation is the interactive control of animation sequences via the Realtime Motion Switch. This technique enables movement at arbitrary times between multiple pre-defined motion sequence patterns. This technique allows interactivity to be implemented — a feature which is indispensable in games. It also makes it possible to tune the authoring level (i.e. create apparent motion).

The amount of memory used for HMD animation data has been minimized by enabling sequences to be used. Entities are not represented in the data, as everything is referenced according to indexes and pointers.

Since the key frame interpolation method for HMD animation is managed by HMD Type, various interpolation methods can be used. A new interpolation method can also be defined by adding a Type.

A library for performing LINEAR, BEZIER and B-SPLINE coordinate rotation, translation and scaling with LINEAR, BEZIER and B-SPLINE interpolation is provided as a Primitive Driver. Also, the common interpolation functions for animating the optional data within HMD are provided by the LINEAR, BEZIER and B-SPLINE algorithms. In this way, animation of vertices, colors, etc. is possible.

Animation Definition

Sequence control descriptor

One animation sequence is defined by a list of 16-bit sequence control descriptors (SC). There are three kinds of SCs. One is a key frame descriptor (SCK), another is a jump descriptor (SCJ) and the third is for control (SCC).

A key frame descriptor (SCK) holds an index to the area in memory area that represents the key frame entity. A jump descriptor (SCJ) holds the index of an SCK jump destination. The SCK also holds the amount of time until the next key frame (TFRAME). It also maintains an index of interpolation functions (Type Idx). SCC displays control information such as sequence stoppage.

All sequences contain a 7-bit ID called a stream ID (SID). An SCJ holds both a source stream ID (SSID) and a destination stream ID (DSID). A jump due to an SCJ is performed only when the SSID matches the SID of the relevant sequence. If it does not match, the pointer moves directly to the next SC. The DSID determines the SID of the jump destination when a jump occurs. However, when the SID is equal to 0, it unconditionally matches all SIDs. As an operational rule, it is advisable that SID127 not be matched.

Sequence header

The sequence header, which unifies the management of individual pieces of sequence information, consists of the following two parts:

1. Sequence pointer
2. Sequence information

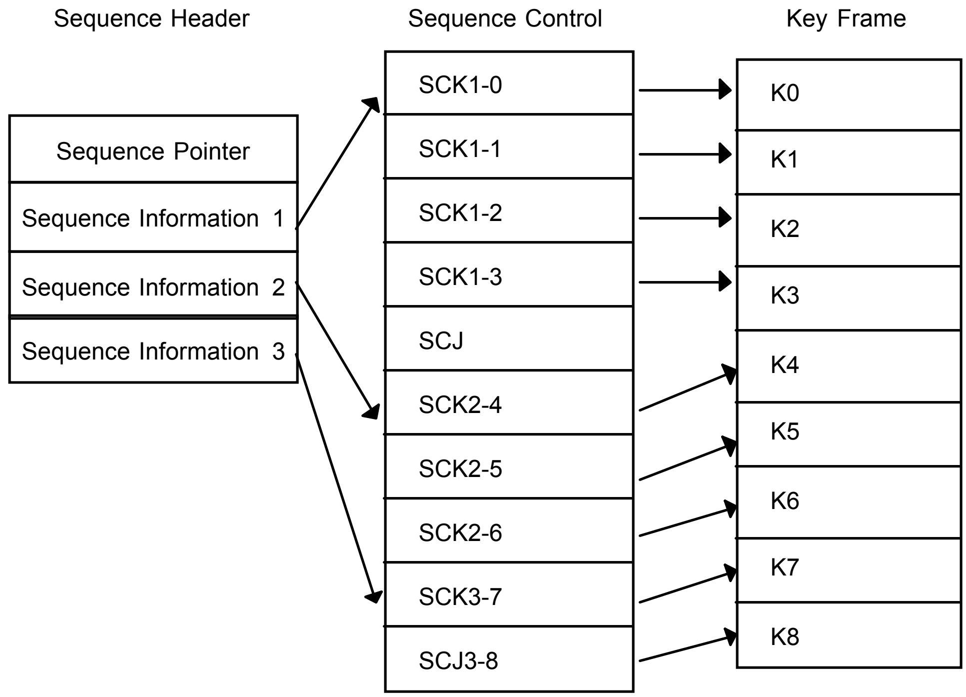
The sequence pointer directs animation playback control, which is described later.

The sequence information holds information on individual sequences. (Entries are listed for the number of sequences.)

The information for one sequence consists of the sequence starting index and the stream ID. The sequence starting index contains the index where that sequence begins of the area in which the SCs are listed. The stream ID indicates the SID at the time that that sequence is to be played back.

This information is referenced by all user programs. A user program controls a sequence by notifying the library via the sequence pointer.

Figure 49: Sequence Management Construction



Animation Playback

Frame update driver

A frame update driver interprets a sequence according to a time series and calls the appropriate interpolation function for performing the interpolation.

Frame update drivers are included according to the same framework as HMD primitive drivers. The frame update driver GsU_03000000(), which is provided with Version 4.0 of libgs, provides such features as the Realtime Motion Switch, forward and reverse playback, slow-motion playback up to 1/16 speed, and high-speed playback up to 8-times normal speed.

Interpolation driver

The interpolation driver is a function for performing key frame interpolation. Although the interpolation driver is identified according to Type in a similar manner as the frame update driver, it is not implemented by the HMD standard primitive driver framework. Instead, the special-purpose SCAN function GsScanAnim() is used, rather than the standard SCAN function GsScanUnit().

When the SCAN function ends, the pointers to interpolation drivers are listed in a special-purpose area (interpolation function table section).

The SCK specifies the interpolation driver that should be called for each key frame according to Type Idx. This enables the key frame interpolation method to be switched within a single sequence.

Sequence pointer

The sequence pointer holds the playback point information of an animation. The playback of an animation can be controlled via this pointer.

The following elements are maintained in the sequence pointer:

- Rewrite IDX: Specifies the areas that are to be updated by the animation.
- NUM: Holds the number of sequences which can be substituted for that sequence pointer.
- INTR IDX: SCK index indicating the area for holding the current parameters.
- AFRAME: Manages the absolute frame numbers of the sequence.
- SRC INTRIDX: Contains the area where parameters to be specified for INTR IDX are held.
- SPEED: Playback speed.
- TFRAME: Time interval between key frames.
- RFRAME: Time interval from a key frame (decremented).
- Stream: IDSequence ID number.
- TCTR IDX: Index to the SC that holds the target key frame.
- CTR IDX: Index to the SC that holds the source key frame.
- START IDX: Holds the starting index of the sequence.
- START SID: Holds the SID when the sequence starts.
- TRAVELING: A variable that is reset to 0 at a key frame transition point. This can be freely used.

Some of these parameters can be set only by the programmer, and others can be updated by a frame update driver. For details, see the GsSEQ structure reference.

Realtime Motion Switch

This function makes interactive animation possible. It is implemented by the HMD frame update drivers and interpolation drivers.

The Realtime Motion Switch is divided into two functions. One function switches sequences in terms of key frame units according to the SID, and the other switches sequences immediately during interpolation.

Sequence switching using the SID

Normal sequence

Figure 50: Sequence With No Jumps

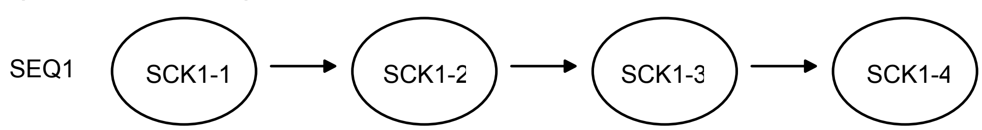
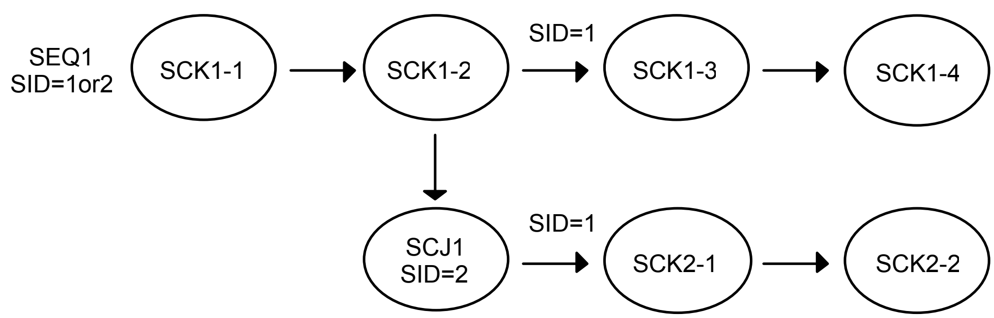


Figure 51: Sequence With Jumps

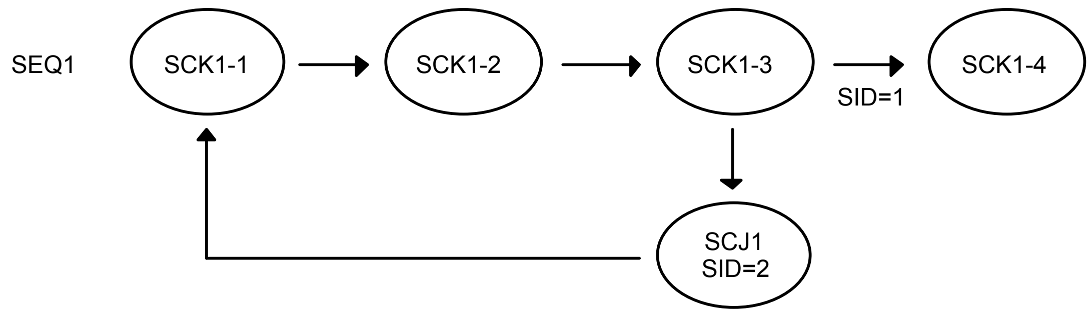


If the SCJ1 descriptor is written in advance and the Sid is 1, this kind of sequence branches to SCK2-1 after SCK1-2. If the information that the Sid is to be set to 0 after the jump is written for the SCJ1 descriptor, the Sid is set to 0 after the jump. Since SCJ descriptors can be arranged in multiple series, individual jump destinations can be specified for various Sids

Sequence branching can be controlled at execution time by changing (rewriting) the Sid from 0 to 1 before the sequence pointer passes the SCJ1 descriptor.

Loop sequence

Figure 52: Loop Sequence

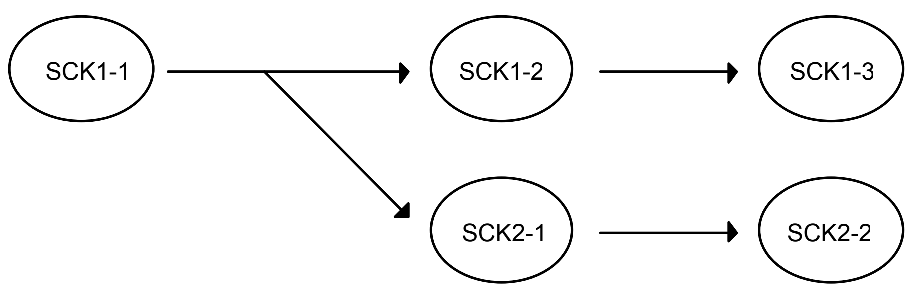


A loop sequence is realized by jumping forward according to a jump descriptor. Looping continues while the SID is 2, and control escapes the loop when the SID is set to 1. The loop can be controlled interactively by rewriting the SID at execution time.

Immediate sequence switching

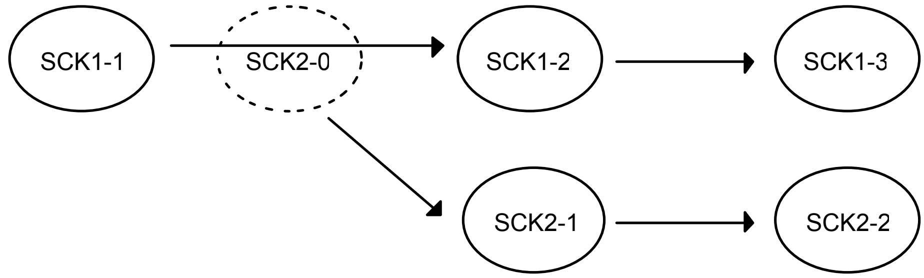
With sequence switching via the SID, a sequence is switched only when the key frame changes. This has the advantage that the switching is completely controllable because the sequence is switched only at points where an SID change can be issued and only at intended locations. However, since no response appears unless the key frame is reached, this method presents a problem from the standpoint of responsiveness. The figure below illustrates immediate sequence switching.

Figure 53: Immediate Sequence Switching 1



The sequence can be changed to key frame SCK2-1 at any time during interpolation between key frame SCK1-1 and key frame SCK1-2. In this case, a new virtual key frame SCK2-0 is defined at the branch point.

Figure 54: Immediate Sequence Switching 2



To implement this function, the sequence pointer is set as described below. An area for saving the current parameters is created in advance by entering a DUMMY key frame. This is defined as SCK2-0. Then, SCK2-0 is entered in INTR IDX at the frame at which the sequence was switched. 0xffff is entered in INTR IDX at the next frame. 0xffff prohibits parameter updating. This process enters the current location's parameters in the key frame entity pointed to by SCK2-0.

At the stage where SCK2-0 is captured, the SCK2-1 index is entered in TCTR IDX and the SCK2-0 index is entered in CTR IDX. Also, the time interval from SCK2-0 to SCK2-1 is entered in TFRAME and RFRAME. The next SID is entered in SID.

This implements immediate sequence switching.