Graphics Data Formats

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.

Table of Contents

About This Manual

This is the Runtime Library Release 2.4 version of the *Graphics Data Formats* manual.

It describes the eS Intermediate File Format, raw File Format, HiG Data Format, HiP Plugin Data Format and the GF-VU1 Standard Memory Formats.

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)

eS Intermediate File Format

eS is a very general-purpose file format for describing graphics, which can be used to link together various types of graphics authoring tools, middleware, and games.

eS is an easy-to-handle ASCII text file for which syntax rules are defined. Graphics functions that are specific to certain tools or middleware can be described with namespaces and by adding independently defined basic types and structures to a framework so that various kinds of data can be handled. Symbol references can be used in addition to immediate values for actual data descriptions.

Summary of eS Syntax

A summary of the eS syntax is presented below. A complete definition of the eS syntax is shown using BNF notation following the summary.

Comments

Everything from # or // until the end of a line is treated as a comment (single-line comment). Also, everything from /* until */ is treated as a comment that spans multiple lines (region comment).

Examples:

```
#  This is a comment
// This too
/*
This also
*/
```

The sample parser that is included in the eS package does not support region comments.

Line Feed Character

In addition to indicating the end of a comment, a line feed character acts as a character delimiter exactly like a space.

Version

The version identifier is specified in the following format as a comment "#" at the beginning of a file.

```
Format:
# eS vmajor.minor ascii
Example:
# eS v0.3 ascii
The current version is "# eS v0.6".
```

Reserved Words

The reserved words of eS are shown below.

Table 1

Reserved Word	Function
boolean	Used for a Boolean type declaration. The assigned data “TRUE”, “True”, or “true” indicates true, and “FALSE”, “False”, or “false” indicates false.
default	Indicates the default namespace.
enum	Used for an enumeration type declaration. The assigned data is a character string, and the initial value is bit shifted sequentially starting with 1.
integer	Used for an integer type declaration. The bit width of the sign part and exponent part is specified enclosed in [ ] following this specification. The bit width of the fractional part of an integer type is always zero.
floating	Used for a floating-point type declaration. The bit width of the sign part and exponent part is specified enclosed in [ ] following this specification. The bit width of the fractional part of an integer type is always greater than or equal to zero.
namespace	Used for a namespace declaration. An entity that is declared in that namespace can be accessed.
string	Used for a character string declaration. The assigned data is a character string enclosed in “ ”.
struct	Used for a structure declaration. Multiple basic types or structures within the scope are collected together in a single structure.
typedef	Used for a basic type declaration. This is used in together with the integer or floating keyword.

In addition, other C or C++ reserved words are also reserved words in eS. The C and C++ reserved words are shown below.

```
asm auto
break
case catch char class const continue
default delete do double
else enum extern
float for friend
goto
if inline int
long
new
operator
private protected public
register return
short signed sizeof static struct switch
template this throw try typedef
union unsigned
virtual void volatile
while
```

Declaration Statement

A declaration statement declares various kinds of basic types or structures. To package original data that is specific to various authoring tools or middleware, the C++ namespace concept is used. The following four declaration statements can be used.

- **namespace:** Declares an original namespace and name and packages the entire declaration.

```
namespace namespace.name {
     ...
}
```

- **typedef:** Uses bit fields to declare a basic integer or floating-point type.

```
typedef (integer|floating)[sign:exponent:fraction]
type.identifier
```

- **type:** Declares a data type and the name of that type.

```
type.identifier type.name
type.identifier type.name { initializer }
```

- **struct:** Declares a new structure by collecting together basic types or structures.

```
struct struct.identifier {
     type.identifier type.name
     type.identifier type.name { initial value }
     ...
}
```

Implementation Statements

An implementation statement uses data members that were defined in declaration statements to describe actual scene building data.

The basic format of an implementation statement is as follows.

```
namespace.name::struct.identifier struct.name {
        type.name { literal }
}
```

To define a default namespace name, specify the following.

```
default namespace namespace.name
struct.identifier struct.name {
             type.name { literal }
}
```

To define a structure reference with a symbol, specify the following (a basic type cannot be referenced).

```
struct.identifier struct.name { struct.name }
```

If “NULL” is specified within { }, it indicates that nothing is referenced.

Operators and Delimiters

{ } indicate the scope of the affected type or structure. When brackets having a different name appear within the scope, they indicate the next hierarchical level.

Include Files

In eS, declaration statements can be included from another file. An example is shown below.

Example:

```
$include “eSTD.es”
```

Specify the filename as a character string enclosed in double quotes. Implementation statements from ver eS v0.6 can now be included.

eS Syntax Using BNF

This section shows the syntax of eS using a simple BNF notation.

```
// ::=          is an assignment.
// ...|...      means either this construct or that construct.
// [...]        means the bracketed construct is optional.
// {...}        means the bracketed construct is repeated one
//              or more times.
// '...'        is a literal string definition.

   eS                  ::= version statement
                       ;

   version             ::= '# eS v0.6 ascii'
                       ;

   statement           ::= {declaration}
                       |   {declaration} default.namespace {implementation}
                       ;

   declaration         ::= include
                       |   type.declaration
                       |   namespace.declaration
                       ;

   include             ::= '$include' string.literal
                       ;

   type.declaration    ::= 'typedef' type.specifier '[' bitfield ']'
                       type.identifier
                       ;

   namespace.declaration      ::= namespace.identifier '{'{ struct.declaration } '}'
                              ;

   struct.declaration  ::= struct.identifier '{' { member.declaration } '}'
                       ;

   member.declaration  ::= type.declarator
                       ;

   type.declarator     ::= literal literal [ '{' initializer '}' ]
                       ;

   initializer         ::= integer.literal
                       |   floating.literal
                       |   string.literal
                       |   boolean.literal
                       |   literal
                       ;

   implementation      ::= definition
                       |   include
                       ;

   default.namespace   ::= 'default' 'namespace' namespace.identifier
```

```
                    ;

definition          ::= struct.definition '{' member.implementation '}'
                    ;

struct.definition   ::= literal '::' literal literal
                    |   literal literal
                    ;

member.implementation      ::= member.name '{' member.implementation '}'
                           |   member.name '{' literal '}'
                           ;

literal             ::= integer.literal
                    |   floating.literal
                    |   enum.literal
                    |   string.literal
                    |   boolean.literal
                    |   'NULL'
                    ;

integer.literal     ::= -?[0-9]+
                    |   0x[0-9A-Fa-f]+
                    ;

floating.literal    ::= -?[0-9]+\.?[0-9]+([eE][+-]?[0-9]+)?
                    ;

string.literal      ::= \"[A-Za-z0-9_\./]+\"
                    ;

boolean.literal     ::= 'TRUE'
                    |   'FALSE'
                    ;
enum.literal        ::= literal
                    |   enum.literal '|' literal
                    ;
```

eSTD

eSTD is a standard namespace that is provided by SCEI. Several basic types and structures are defined for describing standard scene graphs in an interactive world.

Basic Types

eSTD defines the following basic types using the typedef keyword.

Table 2

Basic Type	Contents	Definition
char	8-bit signed integer	integer[1:7:0]
short	16-bit signed integer	integer[1:15:0]
int	32-bit signed integer	integer[1:31:0]
long	64-bit signed integer	integer[1:63:0]
uchar	8-bit unsigned integer	integer[0:8:0]
ushort	16-bit unsigned integer	integer[0:16:0]
uint	32-bit unsigned integer	integer[0:32:0]
ulong	64-bit unsigned integer	integer[0:64:0]
fixeds	16-bit fixed point	integer[1:4:11]
fixedi	32-bit fixed point	integer[1:7:24]
float	32-bit single precision floating point (IEEE-754-compliant)	floating[1:8:23]
double	64-bit double precision floating point (IEEE-754-compliant)	floating[1:11:52]

Structures

eSTD defines the following structures.

Version

This structure describes eSTD version information. The current version is “eSTD v0.6”.

```
struct Version {
       string version      { "eSTD v0.6" }
       string author { "Copyright (C) 2001
              by Sony Computer Entertainment Inc."}
       string date   { "2001/07/01" }
}
```

Camera

This structure describes the viewpoint direction, display screen, and clipping.

```
struct Camera {
       float  fovy         //Field of vision angle
       float  aspect       //Aspect
       float  near         //Near clipping value
       float  far          //Far clipping value
       float  position     //Viewpoint position
       float  interest     //Line of sight direction position
       float  up           //Viewpoint up vector
}
```

Light

This structure describes the light source model. A directional light source, point light source, or spot light can be specified for the light source.

```
struct Light {
       enum   type   { DIRECTION POINT SPOT }   //Light source type
       float  coneangle     //Spot light cut on/off angle
       float  position      //Light source position
       float  direction     //Light source direction
       float  color         //Light source color
       float  attenuation   //Attenuation
}
```

Fog

This structure describes the fog model. LINEAR or EXPONENTIAL can be specified for the blend rate.

```
struct Fog {
       enum   type   { LINEAR EXPONENTIAL }     //Blend type
       float  density                           //Fog density
       float  start                             //Near view
       float  end                               //Distant view
       float  color                             //Fog color
}
```

Texel

This structure describes the texel information of a texture.

```
struct Texel {
       int    bpp          //Pixel bit width
       int    width        //Width
       int    height       //Height
       int    mipmap       //Mipmap level (0 - 6)
       uint   image        //Image pixels
}
```

Clut

This structure describes the color lookup table information of a texture.

```
struct Clut {
       int    bpp          //Pixel bit width
       int    width        //Width
       int    height       //Height
       uint   image        //Image pixels
}
```

Texture

This structure represents a texture map.

```
struct Texture {
       enum   type   { EXTERN INLINE CLUT TIM2} //Texture type
       string filename     //External reference file name
       int    region       //Region size when region repeating is used
       int    filter       //Texture filter (GS register-dependent)
       int    function     //Texture function (GS register-dependent)
       int    wrap         //S- or T-direction wrap mode (GS register-
                             dependent)
       Texel  texel        //Texel specification
       Clut   clut         //Color lookup table specification
}
```

For filename, specify the filename enclosed in double quotes ("").

For region, specify the region size { MINU MINV MAXU MAXV }.

For filter, specify the texture filter { MMAG MMIN } with the following values.

MMAG	0	NEAREST
	1	LINEAR
MMIN	0	NEAREST
	1	LINEAR
	2	NEAREST_MIPMAP_NEAREST
	3	NEAREST_MIPMAP_LINEAR
	4	LINEAR_MIPMAP_NEAREST
	5	LINEAR_MIPMAP_LINEAR

For function, specify the texture function with the following values.

0	MODULATE
1	DECAL
2	HIGHLIGHT
3	HIGHLIGHT2

For wrap, specify the S- or T-direction wrap mode with the following values.

WMS	0 REPEAT, 1 CLAMP, 2 REGION_CLAMP, 3 REGION_REPEAT
WMT	0 REPEAT, 1 CLAMP, 2 REGION_CLAMP, 3 REGION_REPEAT

Layer

This structure describes a multi-texture.

```
struct Layer {
       enum   blend  { NONE ADD SUB }    //Blend type
       float  alpha                      //Blend alpha value
       Texture      texture              //Texture specification
}
```

For blend, specify the blend type as follows.

NONE	No blending
ADD	Addition blending
SUB	Subtraction blending

For alpha, specify the blend alpha value. { 0.0 } is a completely transparent specification, { 128.0 } is an opaque specification, and { 255.0 } is a double brightness specification.

Material

This structure describes the material properties of a surface. This is usually used with Light in a set.

```
struct Material {
       enum    type  { CONSTANT LAMBERT PHONG BLINN VCOLOR }
                                 //Shading type
       float   shininess         //Specular coefficient. Specification of
                                   { 0.0 } or greater.
       float   transparency      //Transparency
       float   ambient           //Ambient color
       float   diffuse           //Diffuse color
       float   specular          //Specular color
       float   emission          //Emission color
       Texture texture           //Texture specification
       Layer   layer             //Multi-texture specification
       Bump    bump              //Bump mapping
       Environment  environment //Environment mapping
}
```

For type, specify the shading method. VCOLOR is a shading method that uses the vertex colors of Shape as the diffuse color.

For transparency, specify the transparency rate. { 0.0 } is a completely transparent specification, { 128.0 } is an opaque specification, and { 255.0 } is a double brightness specification. The initial value is { 128.0 }.

For ambient, diffuse, specular, and emission, specify the respective color values according to { R G B } values in the range from { 0.0 0.0 0.0 } to { 255.0 255.0 255.0 }.

Appearance

This structure describes appearance information. Material information or texture information that is to be used by the scene is described.

```
struct Appearance {
       Material     material     //Material specification
}
```

Vertex

This structure describes vertex information.

```
struct Vertex {
       int    num    //Number of vertices
       float  value  //Vertex coordinates
}
```

Normal

This structure describes normal line information.

```
struct Normal {
       int    num    //Number of normal lines
       float  value  //Normal line coordinates
}
```

Vcolor

This structure describes vertex colors.

```
struct Vcolor {
       int    num    //Number of vertex colors
       float  value  //Vertex color
}
```

Tcoord

This structure describes texture coordinates.

```
struct Tcoord {
       int    num    //Number of texture coordinates
       float  value  //Texture coordinates
}
```

Geometry

This structure describes a 3D object according to polygonal surfaces. Vertex coordinates, normal line coordinates, shared information, vertex colors, texture coordinates, and their index information are described.

```
struct Geometry {
     enum       type    { POINT LINE TRIANGLE QUAD POLYGON STRIP FAN }
                        //Drawing primitive type
     boolean    ccw     //Front surface direction
     boolean    npv     //Normal per vertex flag
     boolean    cpv     //Color per vertex flag
     boolean    tex     //Texture flag
     int        node    //Number of vertices per polygon
     int        length  //Number of primitives (number of polygons, strip
                          length, etc.)
     Vertex     vertex        //Vertex coordinates
     Normal     normal        //Normal coordinates
     Vcolor     vcolor        //Vertex color
     Tcoord     tcoord        //Texture coordinates (multiple specifications
                              for a multi-texture)
     int     vertexindex      //Vertex index
     int     normalindex      //Normal line index
     int     vcolorindex      //Vertex color index
     int     tcoordindex      //Texture coordinate index
     int     materialindex    //Material index (one specification per
                              primitive)
}
```

Shape

This structure describes shape information, which consists of Geometry and Appearance information.

```
struct   Shape {
     Geometry      geometry      //Specification of Geometry constituting the
                                   shape
     Appearance    appearance    //Specification of Appearance constituting
                                   the shape
}
```

Share

This structure describes shared information, which consists of Geometry and Appearance information.

```
struct Share {
     Geometry      geometry      //Specification of Geometry constituting the
                                   shape
     Appearance    appearance    //Specification of Appearance constituting
                                   the shape
}
```

Transform

This structure describes coordinate transformation information.

```
struct Transform {
     enum    order       { TRS TSR RTS RST STR SRT RXYZ RXZY RYXZ RYZX RZXY
RZYX }
             //Coordinate transformation order
     float   matrix
             { 1.0 0.0 0.0 0.0
               0.0 1.0 0.0 0.0
               0.0 0.0 1.0 0.0
               0.0 0.0 0.0 1.0 }              //Transformation matrix
     float   translate   { 0.0 0.0 0.0 }      //Translate
     float   rotate      { 0.0 0.0 0.0 }      //Rotate
     float   scale       { 1.0 1.0 1.0 }      //Scale
     float   pivot       { 0.0 0.0 0.0 }      //Center offset
}
```

Hierarchy

This structure describes the hierarchical structure that used the Shape.

```
struct Hierarchy {
     Transform   transform   //Transform specification
     Shape       shape       //Shape specification
     Hierarchy   parent      //Specification of Hierarchy assigned to parent
                               level
     Hierarchy   child       //Specification of Hierarchy assigned to child
                               level
     Hierarchy   sibling     //Specification of Hierarchy assigned to
                               sibling level
     Boundingbox boundingbox //Specification of bounding box
}
```

Keyframe

This structure describes keyframe information.

```
struct Keyframe {
     enum    interp     { CONSTANT LINEAR HERMITE BEZIER BSPLINE }
             //Interpolation type specification
     enum    fcurve     { TX TY TZ RX RY RZ SX SY SZ TXYZ
                          RXYZ RXZY RYXZ RYZX RZXY RZYX SXYZ }
                        //Function curve specification
     int     frame      //Specification of frame that is to be the key
     float   value      //Specification of control value that is to be the
                          key
}
```

Animation

This structure describes animation information.

```
struct Animation {
     enum     type { HIERARCHY }  //Animation type specification
     boolean    kpf        //Key per frame flag
     int        key        //Number of keyframes
     Hierarchy  hierarchy  //Specification of hierarchy to be animated
     Keyframe   keyframe   //Keyframe specification
}
```

SharedVN

This structure describes information for shared vertices and shared normal lines.

```
struct SharedVN {
     int       vertpos     //Position of shared vertex
     int       vertlen     //Length of shared vertex
     int       normpos     //Position of shared normal line
     int       normlen     //Length of shared normal line
     Hierarchy hierarchy   //Position specification of hierarchy
}
```

Bump

Structure describing bump-map information.

```
struct Bump {
       enum type { CLUT EMBOSS } // Bump type
       float scale // Bump alpha value: 0.0-255.0
       float shift // Bump-shift amount (used by Emboss)
       Texture texture // Bump texture
       Normal normal // Normal table (used with CLUT)
}
```

Fisheye

Structure describing fisheye-lens information (not defined).

```
struct  Fisheye {
       int    depth
       float  rmin
       float  rmax
       int    size
}
```

Reflection

Structure describing environment reflection map information.

```
struct Reflection {
       enum method { STATIC DYNAMIC SPHERE } // Method
       float alpha // Alpha value
       Texture texture // Environment reflection map texture (used with
STATIC)
       Fisheye fisheye // Fisheye lens (used with DYNAMIC)
}
```

Refraction

Structure describing environmental-refractance map information.

```
struct Refraction {
       enum method { STATIC DYNAMIC SPHERE } // Indicates method
       float alpha // Alpha value
       float index // Refraction index
       Texture texture // Indicates environment refraction map texture (used
with STATIC)
       Fisheye fisheye // Fisheye lens (used with DYNAMIC)
}
```

Environment

Structure describing environment map information

```
struct Environment {
       Reflection reflection // Environment reflection map
       Refraction refraction // Environment refraction map
}
```

BoundingBox

Structure describing bounding-box information.

```
struct BoundingBox {
       float min // Minimum size
       float max // Maximum size
}
```

ShadowMap

Structure describing shadow-map information.

```
struct ShadowMap {
       Hierarchy object // Shadow object
       Hierarchy reciever // Shadow receiver
       BoundingBox boundingbox // Bounding box of shadow object
}
```

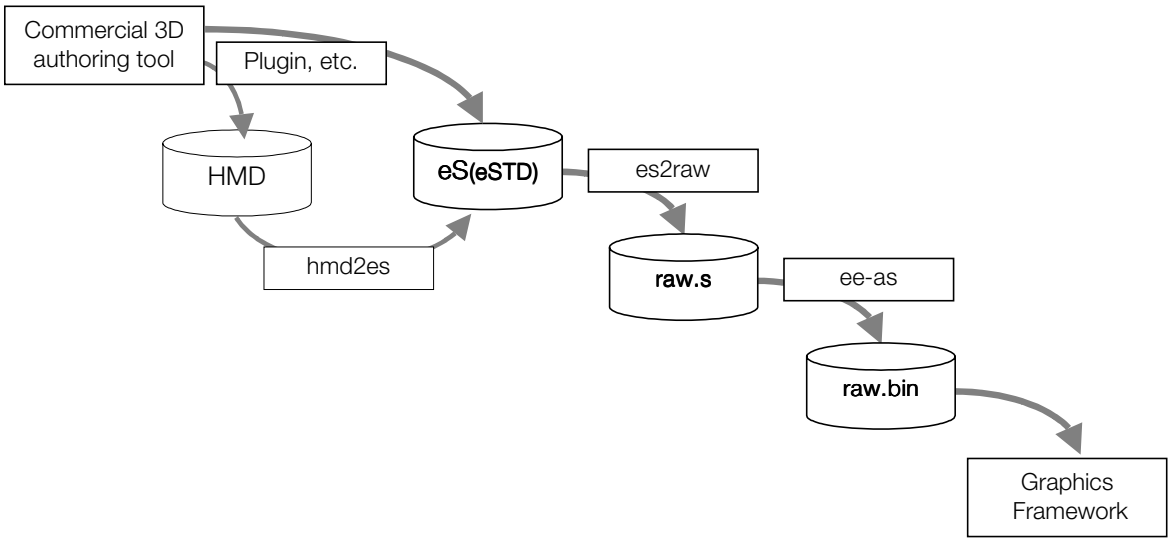
raw File Format

The raw file format is a data format that is used in the Graphics Framework to enable both drawing and development to be performed efficiently for the PlayStation 2 architecture. A raw file can be generated based on an eS (eSTD) file by using the es2raw converter.

Creating a raw File

The es2raw converter can be used to create a raw file from an eS file. The result of the conversion is an EE assembler source file. When necessary, a binary raw file can be obtained by making additional changes, then assembling the file.

Table 3



The header file eSTD.es is required to execute es2raw.

Correspondence with eSTD

Among the structures that are defined in the standard namespace eSTD, the following structures are supported by es2raw.

- Shape Structure that includes shape information and appearance information
- Geometry Geometry shape structure
- Material Material information structure
- Texture Texture information structure
- Share Shared shape structure
- Hierarchy Hierarchy structure
- Animation Keyframe animation structure

The internal structure of Shape differs in eSTD and raw. In eSTD, the relationship Material < Geometry holds, while in es2raw, Geometries are sorted for individual Materials, and the relationship Material > Geometry holds. This is done to reduce texture transfers because when texture transfers are performed frequently, drawing performance will drop significantly in the actual device.

raw File Structure

The block structure of raw data is roughly as follows.

Table 4

raw data header information
Shape data
Base matrix data
Texture data
Hierarchy data
Micro data
Animation data
Keyframe data
Key value data
Shared data
Shared vertex data
Shared normal data
Shared vertex index data
Shared normal index data
Shared vertex link data
Shared normal link data

The structure of each block is shown below. However, generally speaking, this structure consists of only the required data listed following each header. Some of the blocks may not exist, depending on the graphic contents.

raw data Header Information

The raw data header information is a collection of offsets to each of the other blocks. The offset data is arranged in the fixed order shown below, and if a relevant block does not exist, the offset will be NULL (=0).

```
.word   Address offset to shape data block
.word   Address offset to base matrix data block
.word   Address offset to texture data block
.word   Address offset to hierarchy data block
.word   Address offset to micro data block
.word   Address offset to animation data block
.word   Address offset to keyframe data block
.word   Address offset to key value data block
.word   Address offset to shared data block
.word   Address offset to shared vertex data block
.word   Address offset to shared normal data block
.word   Address offset to shared vertex index data block
.word   Address offset to shared normal index data block
.word   Address offset to shared vertex link data block
.word   Address offset to shared normal link data block
```

Shape Data Block

```
- header -
       .word  0, 0, 0, number of Shapes
- shape -
       .word  Shape ID, Shape word size, unused, number of Materials
       - material head -
             .word  Material ID, number of Geometries, Texture ID, number of
Textures
       - material info -
            .float    0.0, 0.0, 0.0, 0.0     (Unused)
            .float    0.0, 0.0, 0.0, 0.0     (Unused)
            - geometry head -
                  word   Geometry ID, Geometry word size, PRIM register,
                         number of PRIMs
                     - vertex -
                     .float     x, y, z, 1.0
                     (repeated for the number of PRIMs)
                     - normal -
                     .float     x, y, z, 1.0
                     (repeated for the number of PRIMs)
                     - st -
                     .float     s, t, 1.0, 0.0
                     (repeated for the number of PRIMs)
                     - color -
                     .float     r, g, b, a
                     (repeated for the number of PRIMs)
               - geometry head -
                      (Same as above)
               : Repeated for the number of Geometries
        - material head -
              (Same as above)
       : Repeated for the number of Materials
- shape -
      (Same as above)
: Repeated for the number of Shapes
```

Base Matrix Data Block

```
- header -
      .word  0, 0, 0, number of matrices
- matrix -
      .word  0, 0, 0, Shape ID
      sceVu0FMATRIX local_world  Local world matrix
      sceVu0FMATRIX  light_rot          Light rotation matrix
- matrix -
      (Same as above)
: Repeated for the number of matrices
```

Texture Data Block

```
- header -
      .word     0, 0, 0, number of textures
- texture -
      .dword    TEX0 register, 0
      .dword    CLUT word size | TEXEL word size
      .dword    CLUT width | CLUT height | TEXEL width | TEXEL height
      .word     TEXEL data
      (Repeated for the number of times equal to the TEXEL word size)
      .word     CLUT data
      (Repeated for the number of times equal to the CLUT word size)
- texture -
      (Same as above)
: Repeated for the number of textures
```

Hierarchy Data Block

```
- header -
      .word  0, 0, 0, number of hierarchies
- hierarchy -
      .float    trans X, trans Y, trans Z, 0.0
      .float    rot   X, rot   Y, rot   Z, 0.0
      .float    scale X, scale Y, scale Z, 0.0
      .word     Shape ID
      .word     Parent ID
      .word     -1 (Unused)
      .word     -1 (Unused)
- hierarchy -
      (Same as above)
: Repeated for the number of hierarchies
```

Micro Data Block

```
- header -
    sceVu0FMATRIX world_screen    Perspective transformation matrix
    sceVu0FMATRIX world_clip      World clipping matrix
    sceVu0FMATRIX clip_screen     Screen clipping matrix
    sceVu0FMATRIX init_data       FOG/ANIT setting data
    sceVu0FMATRIX light_vector0   Light source 0, 1, or 2 direction vector
                                  matrix + Light source spreading (for
                                  Spot)
    sceVu0FMATRIX light_point0    Light source 0, 1, or 2 position vector
                                  matrix + Light intensity (for point or
                                  spot)
    sceVu0FMATRIX light_color0    Light source 0, 1, or 2 color matrix +
                                  Ambient color
    sceVu0FMATRIX light_vector1   Light source 3, 4, or 5 direction vector
                                  matrix + Light source spreading (for
                                  Spot)
```

```
sceVu0FMATRIX light_point1    Light source 3, 4, or 5 position vector
                              matrix + Light intensity (for point or
                              spot)
sceVu0FMATRIX light_color1    Light source 3, 4, or 5 color matrix +
                              Ambient color
sceVu0FMATRIXlight_vector2    Light source 6, 7, or 8 direction vector
                              matrix + Light source spreading (for
                              Spot)
sceVu0FMATRIX light_point2    Light source 6, 7, or 8 position vector
                              matrix + Light intensity (for point or
                              spot)
sceVu0FMATRIX light_color2    Light source 6, 7, or 8 color matrix +
                              Ambient color
```

Animation Data Block

```
- header -
     .word     0, 0, 0, number of animation data
- animation -
     .word     Hierarchy data index
     .word     Number of keyframes
     .word     Keyframe data index
     .word     Key value data index
- animation -
     (Same as above)
: Repeated for the number of animation data
```

Keyframe Data Block

```
- header -
     .word     0, 0, 0, number of keyframe data
- keyframe -
     .word     0, index, word size number of keyframes
     .word     keyframe0, keyframe1, ... , keyframeN
- keyframe -
     (Same as above)
: Repeated for the number of keyframe data
```

Key Value Data Block

```
- header -
     .word     0, 0, 0, number of key value data
- keyvalue -
     .word     0, index, word size, number of keyframes
- keyframevalue-
     .float    trans X, trans Y, trans Z, 0.0
     .float    rot X, rot Y, rot Z, 0.0
- keyframevalue-
     (Same as above)
     : Repeated for the number of keyframes
- keyvalue -
     (Same as above)
: Repeated for the number of key value data
```

Shared Data Block

```
- header -
      .word     0, 0, 0, number of shared data
- share -
      .word     shared vertex starting position after calculation, shared
                vertex length after calculation
      .word     shared normal starting position after calculation, shared
                normal length after calculation
- share -
      (Same as above)
: Repeated for the number of shared data
```

Shared Vertex Data Block

```
- header -
      .word     0, 0, 0, number of shared vertex data
- share vertex -
      .float    x, y, z, 1.0       Shared vertex 0
- share vertex-
      (Same as above)
: Repeated for the number of shared vertex data
- calculated share vertex -
      .float    x, y, z, 1.0       Shared vertex 0 after calculation
- calculated share vertex -
      (Same as above)
: Repeated for the number of shared vertex data
```

Shared Normal Data Block

```
- header -
      .word     0, 0, 0, number of shared normal data
- share normal -
      .float    x, y, z, 1.0       Shared normal 0
- share normal -
      (Same as above)
: Repeated for the number of shared normal data
- calculated share normal -
      .float    x, y, z, 1.0       Shared normal 0 after calculation
- calculated share normal -
      (Same as above)
: Repeated for the number of shared normal data
```

Shared Vertex Index Data Block

```
- header -
      .word     0, 0, 0, number of shared vertex indices
- vertex index -
      .word     vertex 0,   vertex 1,   vertex 2,   vertex 3
      : Repeated until the number of shared vertex indices is reached
```

Shared Normal Index Data Block

```
- header -
      .word     0, 0, 0, number of shared normal indices
- normal index -
      .word     normal 0,   normal 1,   normal 2,   normal 3
      : Repeated until the number of shared normal indices is reached
```

Shared Vertex Link Data Block

```
- header -
      .word    0, 0, shared Shape ID, number of shared vertex link data
- share vertex -
      .word    shared vertex index data starting position, shared Geometry
ID, unused, number of shared vertices
- share vertex -
      (Same as above)
: Repeated for the number of shared vertex link data
```

Shared Normal Link Data Block

```
        - header -
            .word  0, 0, shared Shape ID, number of shared normal link data
        - share normal -
            .word  shared normal index data starting position, shared
Geometry ID, unused, number of shared normal lines
        - share normal -
            (Same as above)
        : Repeated for the number of shared normal link data
```

HiG Data Format

The HiG data format is a graphics data format used by the high-level graphics library (HiG and HiP). Refer to the high-level graphics library documents for more information.

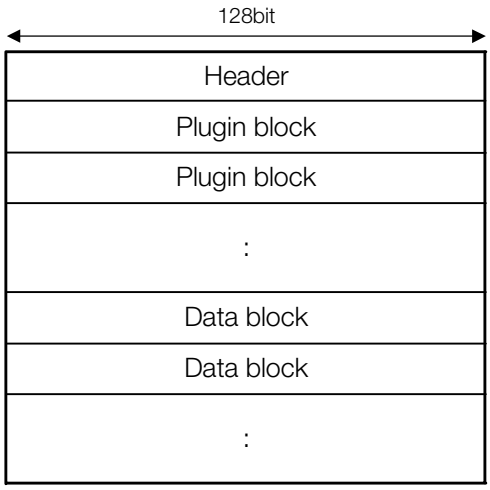
Overall Structure of HiG Data

The HiG data format consists of a header followed by an arbitrary number of plugin blocks and data blocks. However, since the blocks are linked by pointers, the memory images for the various blocks may be scattered.

The data width is 128 bits, and the beginning of a block must be aligned on a qword boundary. All sizes are in qword units (numeric value obtained by dividing the byte size by 16).

Part of the data (address information) is overwritten with real addresses. This is done by calling sceHiParseHeader() to perform parsing after the data is loaded into memory.

Figure 1



Header

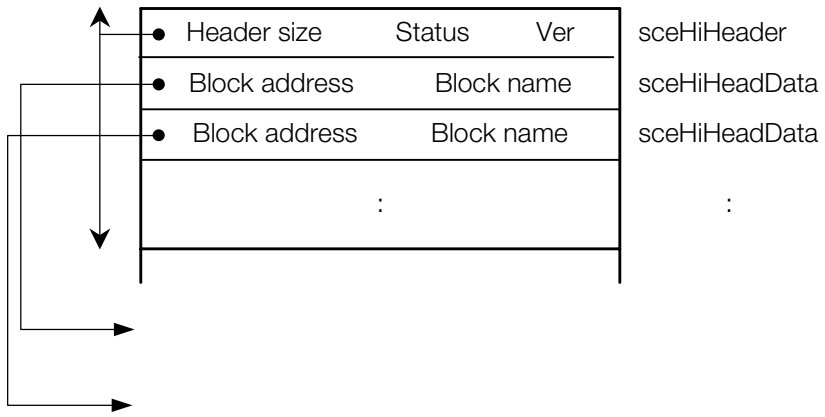
The header structure is as follows. The sceHiHeader structure, which corresponds to offset 0, and the sceHiHeadData structure, which corresponds to each qword for offset 1 and following, are defined in libhig.h.

Table 5

Offset	Bit Position	Contents
0	119-96	Header size (number of qwords)
0	31-24	Status (whether or not parsing is finished)
0	23-0	Version identifier
1	127-96	Plugin block address (overwritten with real address during parsing)
1	95-0	Plugin block name (up to 12 characters). If less than 12 characters, remaining character positions are filled with "\0".
2...		Similar to offset 1

Although the offset address from the beginning of the header is maintained at first for the plugin block address, it is overwritten with a real address during parsing.

Figure 2



Plugin Block

The plugin block structure is shown below. The entire plugin block contains a sceHiPlug structure and a sceHiList structure corresponding to each qword for offset 2 and following. These structures are defined in libhig.h.

Table 6

Offset	Bit Position	Contents
0	119-96	Block size (number of qwords)
0	95-64	Pointer to plugin function (overwritten with real address during parsing)
0	63-0	Type attribute
1	127-96	Argument area
1	95-64	Plugin stack
1	15-8	Number of entries in data block list (*n1*: maximum 255)
1	7-0	Number of entries in plugin block list (*n2*: maximum 255)
2	95-64	Pointer to data block (overwritten with real address during parsing)
2	63-0	Data block type attribute
		(repeated *n1* times)
2+*n1*	95-64	Pointer to plugin block (overwritten with real address during parsing)
2+*n1*	63-0	Plugin block type attribute Repeated *n2* times

If the pointer to the plugin function is NULL when the plugin block is loaded, it is overwritten with a real address during parsing.

The argument area is used for passing arguments or pointers to an argument area when the plugin block is called. For information about argument contents, refer to the corresponding document for each plugin.

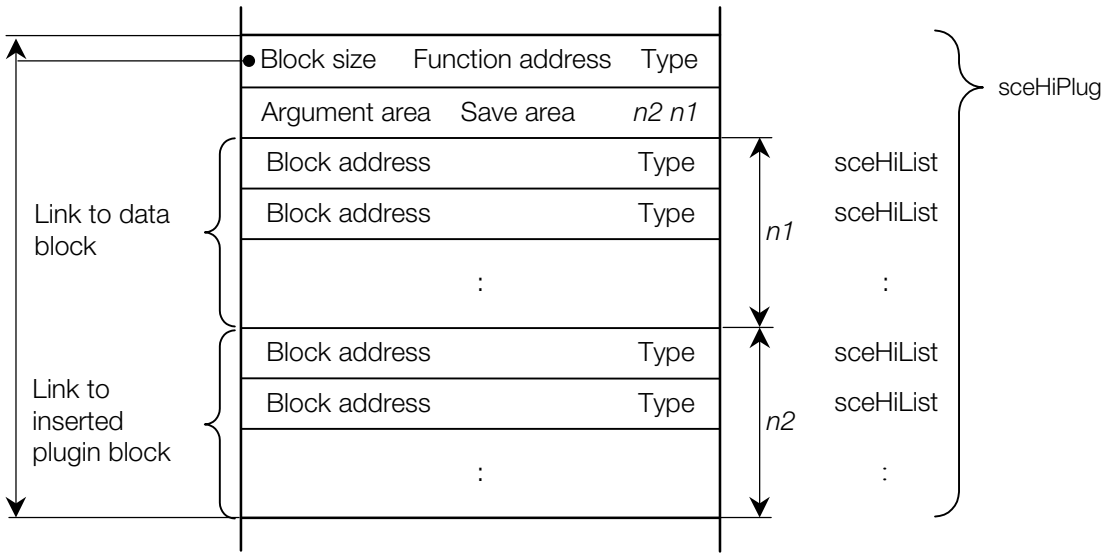
The plugin stack, which is an area used by the plugin side for saving data, is NULL when the plugin block is loaded. Although the application need not be aware of this area, make sure that the value of this area is not destroyed when the plugin is executed.

The pointers to the data blocks and plugin blocks that are the various entries in the data block list and plugin block list are originally offset addresses from the beginning of the header when the plugin block is loaded and are overwritten with real addresses during parsing.

The data block list is a list of the data blocks that are to be processed by this plugin block. Since the contents and order of the data blocks that must be specified here are determined for each plugin, refer to the corresponding plugin document.

The plugin block list is a list of the plugin blocks (inserted plugin blocks) that are called consecutively when this plugin block is called.

Figure 3



Data Block

The data block structure is shown below. The entire data block corresponds to a sceHiData structure, which is defined in libhig.h.

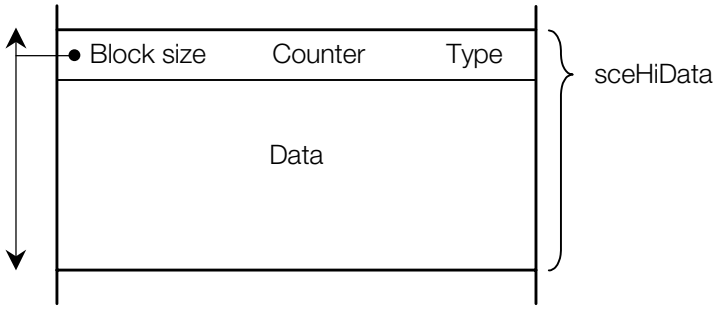
Table 7

Offset	Bit Position	Contents
0	119-96	Block size (number of qwords)
0	71-64	Reference counter (set during parsing)
0	63-0	Type attribute
1...		Data (contents differ according to plugin)

The reference counter indicates the number of the plugin block from which this data block is being referenced. Since it is set during parsing, the application need not be aware of it.

Since the data contents and structure differ according to the type attribute of the data block, refer to the documents corresponding to each plugin.

Figure 4



Type Attribute

Each plugin block and data block has a type attribute so that a plugin correspondence check can be performed.

The type attribute, which is 64 bits wide, consists of the following six fields.

Table 8

Field	Bit Position	Bit Width	Contents
Repository	7-0	8	Plugin provider code
Project	15-8	8	Project code
Category	23-16	8	Category code
Status	31-24	8	(Flag used internally by library)
Plugin ID / Data ID	55-32	24	Plugin identifier /data identifier
Revision	63-56	8	Revision number

Among these fields, the three levels corresponding to the Project, Category, and Plugin ID or Data ID are used for the correspondence check. The Repository has been excluded from the check on the assumption that the same data may be processed by a plugin provided by another company.

HiP Plugin Data Format

The HiP plugin data format shown below, which is part of the HiG data format, is a format for the plugin blocks and data blocks that are used by the various plugins included in the high-level graphics plugin library (HiP).

Type Attributes

The type attributes of the plugin blocks of HiP plugins are shown below. For all of these plugins, the Repository field value is SCE_HIP_COMMON and the Project field value is SCE_HIP_FRAMEWORK.

Table 9

Plugin	Category	PlugIn ID
Frame plugin	SCE_HIP_FRAME	SCE_HIP_FRAME_PLUG
Microcode plugin	SCE_HIP_MICRO	SCE_HIP_MICRO_PLUG
2D texture plugin	SCE_HIP_TEX2D	SCE_HIP_TEX2D_PLUG
Shape plugin	SCE_HIP_SHAPE	SCE_HIP_SHAPE_PLUG
Hierarchy plugin	SCE_HIP_HRCHY	SCE_HIP_HRCHY_PLUG
Animation plugin	SCE_HIP_ANIME	SCE_HIP_ANIME_PLUG
Share plugin	SCE_HIP_SHARE	SCE_HIP_SHARE_PLUG
TIM2 plugin	SCE_HIP_TIM2	SCE_HIP_TIM2_PLUG
ClutBump plugin	SCE_HIP_BUMP	SCE_HIP_CLUTBUMP_PLUG
FishEye plugin	SCE_HIP_REFLECT	SCE_HIP_FISHEYE_PLUG
Reflection plugin	SCE_HIP_REFLECT	SCE_HIP_REFLECT_PLUG
Refraction plugin	SCE_HIP_REFLECT	SCE_HIP_REFRACT_PLUG
ShadowMap plugin	SCE_HIP_SHADOW	SCE_HIP_SHADOWMAP_PLUG
ShadowBox plugin	SCE_HIP_SHADOW	SCE_HIP_SHADOWBOX_PLUG
LightMap plugin	SCE_HIP_LIGHT	SCE_HIP_LIGHTMAP_PLUG

The type attributes of the data blocks used by HiP plugins are as follows. For all of these data blocks, the Repository field value is SCE_HIP_COMMON, the Project field value is SCE_HIP_FRAMEWORK, and the Category field value is the same as that of the corresponding plugin block.

Table 10

Plugin	Data Block	Data ID
Frame plugin	(None)	(None)
Microcode plugin	Microcode data block	SCE_HIP_MICRO_DATA
2D texture plugin	2D texture data block	SCE_HIP_TEX2D_DATA
2D texture plugin	Texture environment data block	SCE_HIP_TEX2D_ENV
Shape plugin	Shape data block	SCE_HIP_SHAPE_DATA
Shape plugin	Base matrix data block	SCE_HIP_BASEMATRIX
Hierarchy plugin	Hierarchy data block	SCE_HIP_HRCHY_DATA
Hierarchy plugin	Pivot data block	SCE_HIP_PIVOT_DATA
Animation plugin	Keyframe data block	SCE_HIP_KEYFRAME
Animation plugin	Key value data block	SCE_HIP_KEYVALUE
Animation plugin	Animation data block	SCE_HIP_ANIME_DATA
Share plugin	Shared vertex data block	SCE_HIP_SRCDSTVERTEX

Plugin	Data Block	Data ID
Share plugin	Shared normal data block	SCE_HIP_SRCDSTNORMAL
Share plugin	Vertex index data block	SCE_HIP_VERTEXINDEX
Share plugin	Normal index data block	SCE_HIP_NORMALINDEX
Share plugin	Shared vertex link data block	SCE_HIP_SHAREVERTEX
Share plugin	Shared normal link data block	SCE_HIP_SHARENORMAL
Share plugin	Shared data block	SCE_HIP_SHARE_DATA
TIM2 plugin	TIM2 data block	SCE_HIP_TIM2_DATA
ClutBump plugin	ClutBump data block	SCE_HIP_CLUTBUMP_DATA
ClutBump plugin	ClutBump normal line data block	SCE_HIP_CLUTBUMP_NORMAL
ShadowBox plugin	ShadowBox data block	SCE_HIP_SHADOWBOX_DATA

Frame Plugin Data Structure

Frame Plugin Block

SCE_HIP_FRAME / SCE_HIP_FRAME_PLUG
Insertion plugin block information
Insertion plugin block information
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_FRAME,
          SCE_HIG_PLUGIN_STATUS
.word     SCE_HIP_FRAME_PLUG | (SCE_HIP_REVISION<<24), 0,
          frame plugin block size
.byte     number of insertion plugin blocks, 0, 0, 0
.word     0, 0, 0
- Insertion plugin block information -
          .byte      SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, insertion plugin
                      category, SCE_HIG_PLUGIN_STATUS
          .word      insertion plugin identifier | (SCE_HIP_REVISION<<24),
                     insertion plugin block address, 0
: Repeated for the number of insertion plugin blocks
```

Microcode Plugin Data Structure

Microcode Plugin Block

SCE_HIP_MICRO / SCE_HIP_MICRO_PLUG
Microcode data block information

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
          SCE_HIG_PLUGIN_STATUS
.word     SCE_HIP_MICRO_PLUG | (SCE_HIP_REVISION<<24), 0,
          microcode plugin block size
.byte     0, 1, 0, 0
.word     0, 0, 0
- Microcode data block information -
```

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_MICRO_DATA | (SCE_HIP_REVISION<<24),
          microcode data block address, 0
```

Microcode Data Block

SCE_HIP_MICRO / SCE_HIP_MICRO_DATA
Micro data

```
.byte SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
      SCE_HIP_DATA_STATUS
.word SCE_HIP_MICRO_DATA | (SCE_HIP_REVISION<<24), 0, micro data block size
- Micro data -
.float(4x4) world_view     Field of vision transformation matrix
                           Texture projection matrix
.float(4x4) material       Material color, transparency, specular
                           coefficient
                           Material information transferred from shape data
.float      x, y, z Camera position vector
.float      aa1     AA1 cut-off value 0.0 to 1.0+alpha
.float      fogA    Fog value 1/(start-end)
.float      fogB    Fog value end
.word       prmode  Primitive attribute (FGE|ABE|AA1)
.float      clipA   Clip parameter (far-near)/width
.float      clipB   Clip parameter (far-near)/height
.float      clipC   Clip parameter (far+near)/2
.float      clipD   Clip parameter (far-near)/2
.float      x, y, z, w     Special parameters (depends on microcode)


- Light source data -
.float      x, x, x, 0.0   Light source  0, 1, 2 direction vector X
.float      y, y, y, 0.0   Light source  0, 1, 2 direction vector Y
.float      z, z, z, 0.0   Light source  0, 1, 2 direction vector Z
.float      -, -, -, -     Unused
.float      x, y, z, a     Light source 0 position vector, angle
.float      x, y, z, a     Light source 1 position vector, angle
.float      x, y, z, a     Light source 2 position vector, angle
.float      -, -, -, -     Unused
.float      r, g, b, i     Light source 0 color, intensity
.float      r, g, b, i     Light source 1 color, intensity
.float      r, g, b, i     Light source 2 color, intensity
.float      -, -, -, -     Unused

     :  Repeated in order of parallel light source, point light source, and
        spot light.
```

2D Texture Plugin Data Structure

2D Texture Plugin Block

SCE_HIP_TEX2D / SCE_HIP_TEX2D_PLUG
Texture data block information
Texture environment data block information

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
          SCE_HIG_PLUGIN_STATUS
.word     SCE_HIP_TEX2D_PLUG | (SCE_HIP_REVISION<<24), 0,
          2D texture plugin block size
.byte     0, 2, 0, 0
.word     0, 0, 0
- Texture data block information -
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_TEX2D_DATA | (SCE_HIP_REVISION<<24),
          2D texture data block address, 0
- Texture environment data block information -
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_TEX2D_ENV | (SCE_HIP_REVISION<<24),
          texture environment data block address, 0
```

2D Texture Data Block

SCE_HIP_TEX2D / SCE_HIP_TEX2D_DATA
Texture
Texture
:

```
.byte    SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
         SCE_HIP_DATA_STATUS
.word    SCE_HIP_TEX2D_DATA | (SCE_HIP_REVISION<<24), 0,
         2D texture data block size
.word    0, 0, 0, number of textures
- Texture -
         .qword     TEX0 register
         .word      TEXEL word size, CLUT word size,
         .word      TEXEL width << 16 | TEXEL height, CLUT width << 16 |
CLUT height
         .word      TEXEL data
         (repeated for a number of times equivalent to the texel word size)
         .word      CLUT data
         (repeated for a number of times equivalent to the CLUT word size)
- Texture -
         (Same as above)
: Repeated for the number of textures
```

Texture Environment Data Block

SCE_HIP_TEX2D / SCE_HIP_TEX2D_ENV
Texture environment data
Texture environment data
:

```
.byte SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
      SCE_HIP_DATA_STATUS
.word SCE_HIP_TEX2D_ENV | (SCE_HIP_REVISION<<24), 0,
      texture environment data block size
.word        0, 0, 0, number of texture environments
- Texture environment -
     .qword      GIFtag
     .qword      TEX1 register
     .qword      TEX0 register
     .qword      CLAMP register
     .qword      MIPTBP1 register
     .qword      MIPTBP2 register
- Texture environment -
     (Same as above)
: Repeated for the number of texture environments
```

Shape Plugin Data Structure

Shape Plugin Block <when there is no texture>

SCE_HIP_SHAPE / SCE_HIP_SHAPE_PLUG
Shape data block information
Base matrix data block information Texture

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIG_PLUGIN_STATUS
.word     SCE_HIP_SHAPE_PLUG  | (SCE_HIP_REVISION<<24), 0,
          shape plugin block size
.byte     0, 2, 0, 0
.word     0, 0, 0
- Shape data block information -
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_SHAPE_DATA | (SCE_HIP_REVISION<<24),
          shape data block address, 0
- Base matrix data block information -
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_BASEMATRIX | (SCE_HIP_REVISION<<24),
          base matrix data block address,0
```

Shape Plugin Block <when there is a texture>

SCE_HIP_SHAPE / SCE_HIP_SHAPE_PLUG
Shape data block information
Base matrix data block information
Texture environment data block information

```
.byte      SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
           SCE_HIG_PLUGIN_STATUS
.word      SCE_HIP_SHAPE_PLUG  | (SCE_HIP_REVISION<<24), 0,
           shape plugin block size
.byte      0, 3, 0, 0
.word      0, 0, shape plugin packet size
- Shape data block information -
.byte      SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
           SCE_HIP_DATA_STAT
.word      SCE_HIP_SHAPE_DATA | (SCE_HIP_REVISION<<24),
           shape data block address, 0
- Base matrix data block information -
.byte      SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
           SCE_HIP_DATA_STATUS
.word      SCE_HIP_BASEMATRIX | (SCE_HIP_REVISION<<24),
           base matrix data block address, 0
- Texture environment data block information -
.byte      SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
           SCE_HIP_DATA_STATUS
.word      SCE_HIP_TEX2D_ENV | (SCE_HIP_REVISION<<24),
           texture environment data block address, 0
```

Shape Data Block

SCE_HIP_SHAPE / SCE_HIP_SHAPE_DATA
Shape data
Shape data
:

```
.byte    SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
         SCE_HIP_DATA_STATUS
.word    SCE_HIP_SHAPE_DATA | (SCE_HIP_REVISION<<24), 0, shape data block
         size
.word    0, 0, 0, number of Shapes
- Shape data -
         .word   Shape ID, Shape word size, unused, number of Materials
         - material head -
             .word  Material ID, number of Geometries, Texture ID, number of
                    Textures
             - material info -
             .word  0, 0, 0, material data QWORD size
             .qword GIFtag
             .qword ALPHA register
             .qword TEST register
             .float diffuse R, diffuse G, diffuse B, transparency
             .float specular R, specular G, specular B, shininess
             .float emission R, emission G, emission B, 0.0
             .float ambient R, ambient G, ambient B, 0.0
```

```
                - geometry head -
                      .word  Geometry ID, Geometry word size,
                             PRIM register, number of PRIMs
                      - vertex -
                             .float x, y, z, 1.0
                      : Repeated for the number of PRIMs
                      - normal -
                             .float x, y, z, 1.0
                      : Repeated for the number of PRIMs
                      - st -
                             .float s, t, 1.0, 0.0
                      : Repeated for the number of PRIMs
                      - color -
                             .float r, g, b, a
                      : Repeated for the number of PRIMs
                - geometry head -
                      (Same as above)
                : Repeated for the number of Geometries
            - material head -
                (Same as above)
            :  Repeated for the number of Materials
- Shape data -
            (Same as above)
: Repeated for the number of Shapes
```

Base Matrix Data Block

SCE_HIP_SHAPE / SCE_HIP_BASEMATRIX
Matrix
Matrix
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_BASEMATRIX | (SCE_HIP_REVISION<<24), 0,
          matrix data block size
.word     0, 0, 0, number of matrices
.word     0, 0, 0, Shape ID
- Matrix -
       sceVu0FMATRIX local_world     Local world matrix
       sceVu0FMATRIX light_rot       Light rotation matrix
- Matrix -
       (Same as above)
: Repeated for the number of matrices
```

Hierarchy Plugin Data Structure

Hierarchy Plugin Block

SCE_HIP_HRCHY / SCE_HIP_HRCHY_PLUG
Hierarchy data block information
Base matrix data block information
Pivot data block information

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_HRCHY,
          SCE_HIG_PLUGIN_STATUS
.word     SCE_HIP_HRCHY_PLUG | (SCE_HIP_REVISION<<24), 0,
          hierarchy plugin block size
.byte     0, 3, 0, 0
.word     0, 0, 0
- Hierarchy data block information -
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_HRCHY,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_HRCHY_DATA | (SCE_HIP_REVISION<<24),
          hierarchy data block address, 0
- Base matrix data block information -
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_BASEMATRIX | (SCE_HIP_REVISION<<24),
          base matrix data block address, 0
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_HRCHY,
          SCE_HIG_DATA_STATUS
- Pivot data block information -
.word     SCE_HIP_PIVOT_DATA | (SCE_HIP_REVISION<<24),
          pivot data block address, 0
```

Hierarchy Data Block

SCE_HIP_HRCHY / SCE_HIP_HRCHY_DATA
Hierarchy
Hierarchy
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_HRCHY,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_HRCHY_DATA | (SCE_HIP_REVISION<<24), 0,
          hierarchy data block size
.word     0, 0, rotation order, number of hierarchies
- hierarchy -
       .float     trans X, trans Y, trans Z, 0.0
       .float     rot   X, rot   Y, rot   Z, 0.0
       .float     scale X, scale Y, scale Z, 0.0
       .word      Shape ID
       .word      Parent ID
       .word      -1 (Unused)
       .word      -1 (Unused)
- hierarchy -
       (Same as above)
: Repeated for the number of hierarchies
```

The multiplication sequence beginning with the matrix at the left is represented with bit flags for the rotation order, as shown below.

Table 11

Matrix Multiplication Sequence	Rotation Order Value
RXYZ	(1<<6)
RXZY	(1<<7)
RYXZ	(1<<8)
RYZX	(1<<9)
RZXY	(1<<10)
RZYX	(1<<11)

Pivot Data Block

SCE_HIP_HRCHY / SCE_HIP_PIVOT_DATA
Pivot data
Pivot data
:

```
.byte SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_HRCHY,
      SCE_HIG_DATA_STATUS
.word SCE_HIP_PIVOT_DATA | (SCE_HIP_REVISION<<24), 0, pivot data block size
.word 0, 0, 0, number of pivots
- Pivot data -
        .float      X, Y, Z, 0.0
: Repeated for the number of pivots
```

Animation Plugin Data Structure

Animation Plugin Block

SCE_HIP_ANIME / SCE_HIP_ANIME_PLUG
Keyframe data block information
Key value data block information
Animation data block information

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
          SCE_HIG_PLUGIN_STATUS
.word     SCE_HIP_ANIME_PLUG | (SCE_HIP_REVISION<<24), 0,
          animation plugin block size
.byte     0, 4, 0, 0
.word     0, 0, 0
- Keyframe data block information -
          .byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
                    SCE_HIP_DATA_STATUS
          .word     SCE_HIP_KEYFRAME | (SCE_HIP_REVISION<<24),
                    keyframe data block address, 0
- Key value data block information -
          .byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
                    SCE_HIP_DATA_STATUS
```

```
          .word     SCE_HIP_KEYVALUE | (SCE_HIP_REVISION<<24),
                    key value data block address, 0
- Animation data block information -
          .byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
                    SCE_HIP_DATA_STATUS
          .word     SCE_HIP_ANIME_DATA | (SCE_HIP_REVISION<<24),
                    animation data block address, 0
- Hierarchy data block information –
          .byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_HRCHY,
                    SCE_HIP_DATA_STATUS
          .word     SCE_HIP_HRCHY_DATA | (SCE_HIP_REVISION<<24),
                    Hierarchy data block address, 0
```

Keyframe Data Block

SCE_HIP_ANIME / SCE_HIP_KEYFRAME
Keyframe data
Keyframe data
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_KEYFRAME | (SCE_HIP_REVISION<<24), 0,
          keyframe data block size
.word     0, 0, 0, number of keyframe data
- Keyframe data -
       .word  Interpolation type, index, word size, number of keyframes N
       .word  keyframe0, keyframe1, ... , keyframeN
- Keyframe data -
       (Same as above)
: Repeated for the number of keyframe data
```

The interpolation type is represented with bit flags as follows.

Table 12

Interpolation Method	Interpolation Type Value
Constant (CONSTANT)	(1<<0)
Linear interpolation (LINEAR)	(1<<1)
Hermite interpolation (HERMITE)	(1<<2)
Bezier interpolation (BEZIER)	(1<<3)
B-SPLINE interpolation (BSPLINE)*	(1<<4)

* Not supported

Key Value Data Block

SCE_HIP_ANIME / SCE_HIP_KEYVALUE
Key value data
Key value data
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_KEYVALUE | (SCE_HIP_REVISION<<24), 0,
          key value data block size
.word     0, 0, 0, number of key value data
- Key value data -
       .word  Curve specification (TXYZ|RXYZ|SXYZ), index, word size,
              number of keyframes
- Keyframe -
              (the contents, which are described separately, differ
                according to the interpolation method)
       : Repeated for the number of keyframes
- Key value data -
       (Same as above)
: Repeated for the number of key value data
```

- Keyframe - <for constant or linear interpolation>

```
           .float       trans X, trans Y, trans Z, 0.0
           .float       rot X, rot Y, rot Z, 0.0
           .float       scale X, scale Y, scale Z, 0.0
```

- Keyframe - <for Hermite interpolation>

```
           .float       trans X0, trans X1, trans X2, trans X3
           .float       trans Y0, trans Y1, trans Y2, trans Y3
           .float       trans Z0, trans Z1, trans Z2, trans Z3
           .float       rot X0, rot X1, rot X2, rot X3
           .float       rot Y0, rot Y1, rot Y2, rot Y3
           .float       rot Z0, rot Z1, rot Z2, rot Z3
           .float       scale X0, scale X1, scale X2, scale X3
           .float       scale Y0, scale Y1, scale Y2, scale Y3
           .float       scale Z0, scale Z1, scale Z2, scale Z3
```

The listed values represent the starting point, tangent vector from the starting point, tangent vector from the endpoint, and the endpoint, respectively.

- Keyframe - <for Bezier interpolation>

```
.float       trans X0, trans X1, trans X2, trans X3
.float       trans Y0, trans Y1, trans Y2, trans Y3
.float       trans Z0, trans Z1, trans Z2, trans Z3
.float       rot X0, rot X1, rot X2, rot X3
.float       rot Y0, rot Y1, rot Y2, rot Y3
.float       rot Z0, rot Z1, rot Z2, rot Z3
.float       scale X0, scale X1, scale X2, scale X3
.float       scale Y0, scale Y1, scale Y2, scale Y3
.float       scale Z0, scale Z1, scale Z2, scale Z3
```

The listed values represent the starting point, position of the tangent vector from the starting point, position of the tangent vector from the endpoint, and the endpoint, respectively.

Animation Data Block

SCE_HIP_ANIME / SCE_HIP_ANIME_DATA
Animation data
Animation data
:

```
.byte            SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_ANIME,
                 SCE_HIP_DATA_STATUS
.word            SCE_HIP_ANIME_DATA | (SCE_HIP_REVISION<<24), 0,
                 animation data block size
.word            0, 0, 0, number of animation data
- Animation data -
       .word     Hierarchy data index
       .word     Number of keyframes
       .word     Keyframe data index
       .word     Key value data index
- Animation data -
       (Same as above)
: Repeated for the number of animation data
```

Shared Plugin Data Structure

Shared Plugin Block

SCE_HIP_SHARE / SCE_HIP_SHARE_PLUG
Shared vertex data block information
Shared normal data block information
Shared vertex index data block information
Shared normal index data block information
Shared vertex link data block information
Shared normal link data block information
Shared data block information
Base matrix data block information
Shape data block information

```
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIG_PLUGIN_STATUS
  .word   SCE_HIP_SHARE_PLUG | (SCE_HIP_REVISION<<24), 0,
          shared plugin block size
  .byte   0, 9, 0, 0
  .word   0, 0, 0
- Shared vertex data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_SRCDSTVERTEX | (SCE_HIP_REVISION<<24),
          shared vertex data block address, 0
- Shared normal data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_SRCDSTNORMAL | (SCE_HIP_REVISION<<24),
          Shared normal data block address, 0
- Shared vertex index data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_VERTEXINDEX | (SCE_HIP_REVISION<<24),
          shared vertex index data block address, 0
- Shared normal index data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_NORMALINDEX | (SCE_HIP_REVISION<<24),
          shared normal index data block address, 0
- Shared index link data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_SHAREVERTEX | (SCE_HIP_REVISION<<24),
          shared index link data block address, 0
- Shared normal link data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_SHARENORMAL | (SCE_HIP_REVISION<<24),
          shared normal link data block address, 0
- Shared data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_SHARE_DATA | (SCE_HIP_REVISION<<24), shared data block
          address, 0
- Base matrix data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_BASEMATRIX | (SCE_HIP_REVISION<<24),
          base matrix data block address, 0
- Shape data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_SHAPE_DATA | (SCE_HIP_REVISION<<24),
          shape data block address, 0
```

Shared Vertex Data Block

SCE_HIP_SHARE / SCE_HIP_SRCDSTVERTEX
Shared vertex data
:
Shared vertex data after calculation
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_SRCDSTVERTEX | (SCE_HIP_REVISION<<24), 0,
          shared vertex data block size
.word     0, 0, 0, number of shared vertex data
.float    x, y, z, 1.0     Shared vertex
: Repeated for the number of shared vertex data
.float    x, y, z, 1.0     Shared vertex after calculation
: Repeated for the number of shared vertex data
```

Shared Normal Data Block

SCE_HIP_SHARE / SCE_HIP_SRCDSTNORMAL
Shared normal data
:
Shared normal data after calculation
:

```
.byte SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
      SCE_HIP_DATA_STATUS
.word SCE_HIP_SRCDSTNORMAL | (SCE_HIP_REVISION<<24), 0,
      shared normal data block size
.word     0, 0, 0, number of shared normal data
.float    x, y, z, 1.0        Shared normal line
: Repeated for the number of shared normal data
.float    x, y, z, 1.0        Shared normal line after calculation
: Repeated for the number of shared normal data
```

Shared Vertex Index Data Block

SCE_HIP_SHARE / SCE_HIP_VERTEXINDEX
Shared vertex index
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_VERTEXINDEX | (SCE_HIP_REVISION<<24), 0,
          shared vertex index data block size
.word     0, 0, 0, number of shared vertex indices
.word     Vertex 0,   vertex 1,   vertex 2,   vertex 3
: Repeated until the number of shared vertex indices is reached
```

Shared Normal Index Data Block

SCE_HIP_SHARE / SCE_HIP_NORMALINDEX
Shared normal index
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_NORMALINDEX | (SCE_HIP_REVISION<<24), 0,
          Shared normal index data block size
.word     0, 0, 0, number of shared normal indices
.word     Normal line 0,   normal line 1,   normal line 2,   normal line 3
: Repeated until the number of shared normal indices is reached
```

Shared Vertex Link Data Block

SCE_HIP_SHARE / SCE_HIP_SHAREVERTEX
Shared vertex link data
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_SHAREVERTEX | (SCE_HIP_REVISION<<24), 0,
          shared vertex link data block size
.word     0, 0, shared Shape ID, number of shared vertex link data
- Shared vertex link data -
.word     Starting position of shared vertex index data, shared Geometry ID,
          unused, number of shared vertices
: Repeated for the number of shared vertex link data
```

Shared Normal Link Data Block

SCE_HIP_SHARE / SCE_HIP_SHARENORMAL
Shared normal link data
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_SHARENORMAL | (SCE_HIP_REVISION<<24), 0,
          shared normal link data block size
.word     0, 0, shared Shape ID, number of shared normal link data
- Shared normal link data -
.word     Starting position of shared normal index data, shared Geometry ID,
          unused, number of shared normal lines
: Repeated for the number of shared normal link data
```

Shared Data Block

SCE_HIP_SHARE / SCE_HIP_SHARE_DATA
Shared data
:

```
.byte     SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHARE,
          SCE_HIP_DATA_STATUS
.word     SCE_HIP_SHARE_DATA | (SCE_HIP_REVISION<<24), 0, shared data block
          size
.word     0, 0, 0, number of shared data
- Shared data -
       .word     Starting position of shared vertex after calculation,
                  length of shared vertex after calculation
       .word     Starting position of shared normal after calculation,
                  length of shared normal after calculation
- Shared data -
       (Same as above)
: Repeated for the number of shared data
```

TIM2 Plugin Data Structure

TIM2 Plugin Block

SCE_HIP_TIM2 / SCE_HIP_TIM2_PLUG
TIM2 data block information
Texture environment data block information

```
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TIM2,
          SCE_HIG_PLUGIN_STATUS
  .word   SCE_HIP_TIM2_PLUG | (SCE_HIP_REVISION<<24), 0, TIM2 plugin block
          size
  .byte   0, 2, 0, 0
  .word   0, 0, 0
- TIM2 data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TIM2,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_TIM2_DATA | (SCE_HIP_REVISION<<24), TIM2 data block
          address, 0
- Texture environment data block information -
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_TEX2D_ENV | (SCE_HIP_REVISION<<24),
          texture environment data block address, 0
```

TIM2 Data Block

SCE_HIP_TIM2 / SCE_HIP_TIM2_DATA
TIM2 data
:

```
  .byte   SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TIM2,
          SCE_HIP_DATA_STATUS
  .word   SCE_HIP_TIM2_DATA | (SCE_HIP_REVISION<<24), TIM2 data block
          address, 0
  .word   0, 0, 0, TIM2 data count
- TIM2 data -
  .word    TIM2 ID, TIM2 file address, file size, filename character  count
  .ascii filename (QWORD fixed, at most 16 characters)
  : Repeated for the number of TIM2 data
```

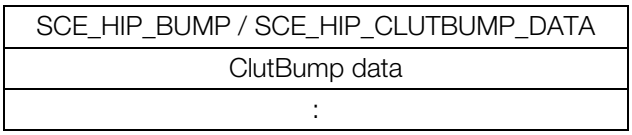
ClutBump Plugin Data Structure

ClutBump Plugin Block

SCE_HIP_BUMP / SCE_HIP_BUMP_PLUG
ClutBump data block
ClutBump normal line data block
Base matrix data block
Texture data block

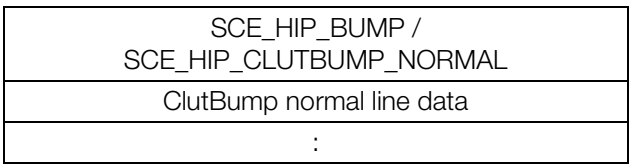
```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_BUMP,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_CLUTBUMP_PLUG | (SCE_HIP_REVISION<<24), 0, plugin block size
.byte  0, 4, 0, 0
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_BUMP, SCE_HIP_DATA_STATUS
.word  SCE_HIP_CLUTBUMP_DATA|(SCE_HIP_REVISION<<24), 0, ClutBump data block
address, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_BUMP,
SCE_HIP_DATA_STATUS
.word  SCE_HIP_CLUTBUMP_NORMAL|(SCE_HIP_REVISION<<24), 0, ClutBump normal
line data block address, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
SCE_HIP_DATA_STATUS
.word  SCE_HIP_BASEMATRIX|(SCE_HIP_REVISION<<24), 0, base matrix data block
address, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_TEX2D, SCE_HIP_DATA_STATUS
.word  SCE_HIP_TEX2D_DATA|(SCE_HIP_REVISION<<24), 0, texture block address,
0
```

ClutBump Data Block




```
.byte SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_BUMP,
SCE_HIG_DATA_STATUS
.word SCE_HIP_CLUTBUMP_DATA | (SCE_HIP_REVISION<<24), 0, ClutBump data block
size
.word 0, 0, 0, ClutBump data count
.word SHAPE ID, TEX2D ID, NORMAL LIST ID, 0
: Repeated ClutBump data count times
```

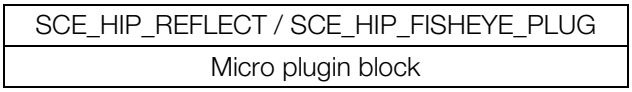
ClutBump Normal Line Data Block




```
.byte SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_BUMP,
SCE_HIG_DATA_STATUS
.word SCE_HIP_CLUTBUMP_NORMAL | (SCE_HIP_REVISION<<24), 0, data block size
.word 0, 0, 0, ClutBump normal line data count
- NORMAL LIST -
.float normal.x, normal.y, normal.z, 0.0      <= 256 times consecutively
: Repeated ClutBump normal line data count times
```

FishEye Plugin Data Structure

FishEye Plugin Block




```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_REFLECT,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_FISHEYE_PLUG|(SCE_HIP_REVISION<<24), 0, plugin block size
.byte  1, 0, 0, 0
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
SCE_HIP_PLUGIN_STATUS
.word  SCE_HIP_MICRO_PLUG|(SCE_HIP_REVISION<<24), 0, Micro plugin block
address, 0
```

Reflection Plugin Data Structure

Reflection Plugin Block

SCE_HIP_REFLECT / SCE_HIP_REFLECT_PLUG
Micro plugin block

```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_REFLECT,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_REFLECT_PLUG|(SCE_HIP_REVISION<<24), 0, plugin block size
.byte  1, 0, 0, 0
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
SCE_HIP_PLUGIN_STATUS
.word  SCE_HIP_MICRO_PLUG|(SCE_HIP_REVISION<<24), 0, Micro plugin block
address, 0
```

Refraction Plugin Data Structure

Refraction Plugin Block

SCE_HIP_REFLECT / SCE_HIP_REFRACT_PLUG
Micro plugin block

```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_REFLECT,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_REFRACT_PLUG|(SCE_HIP_REVISION<<24), 0, plugin block size
.byte  1, 0, 0, 0
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
SCE_HIP_PLUGIN_STATUS
.word  SCE_HIP_MICRO_PLUG|(SCE_HIP_REVISION<<24), 0, Micro plugin block
address, 0
```

ShadowMap Plugin Data Structure

ShadowMap Plugin Block

SCE_HIP_SHADOW/ SCE_HIP_SHADOWMAP_PLUG
Micro plugin block
ShadowBox data block
Microcode data block information

```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHADOW,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_SHADOWMAP_PLUG|(SCE_HIP_REVISION<<24), 0, plugin block size
.byte  1, 2, 0, 0
```

```
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
SCE_HIP_PLUGIN_STATUS
.word  SCE_HIP_MICRO_PLUG|(SCE_HIP_REVISION<<24), 0,
       Micro plugin block address, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHADOW,
SCE_HIP_DATA_STATUS
.word  SCE_HIP_SHADOWBOX_DATA|(SCE_HIP_REVISION<<24), 0,
       ShadowBox data block address, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO, SCE_HIP_DATA_STATUS
.word  SCE_HIP_MICRO_DATA | (SCE_HIP_REVISION<<24),
       Microcode data block address, 0
```

ShadowBox Plugin Data Structure

ShadowBox Plugin Block

SCE_HIP_SHADOW / SCE_HIP_SHADOWBOX_PLUG
ShadowBox data block
Base matrix data block

```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHADOW,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_SHADOWBOX_PLUG|(SCE_HIP_REVISION<<24), 0, plugin block size
.byte  0, 2, 0, 0
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHADOW,
SCE_HIP_DATA_STATUS
.word  SCE_HIP_SHADOWBOX_DATA|(SCE_HIP_REVISION<<24), 0, ShadowBox data
block address, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHAPE,
SCE_HIP_DATA_STATUS
.word  SCE_HIP_BASEMATRIX|(SCE_HIP_REVISION<<24), 0, base matrix data block
address, 0
```

ShadowBox Data Block

SCE_HIP_SHADOW / SCE_HIP_SHADOWBOX_DATA
ShadowBox data

```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_SHADOW,
SCE_HIG_DATA_STATUS
.word  SCE_HIP_SHADOWBOX_DATA|(SCE_HIP_REVISION<<24), 0, data block size
.float min x, min y, min z, 1.0
.float max x, max y, max z, 1.0
.float 0.0, 0.0, 0.0, 0.0  <= 8 vertices consecutively
```

LightMap Plugin Data Structure

LightMap Plugin Block

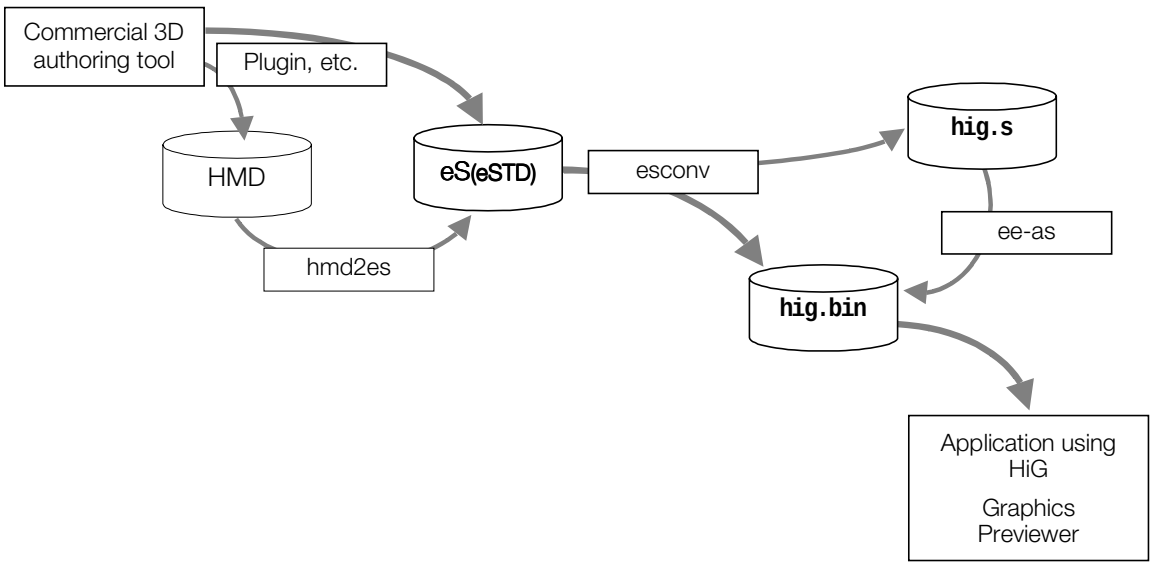
SCE_HIP_LIGHT/ SCE_HIP_LIGHTMAP_PLUG
Micro plugin block

```
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_LIGHT,
SCE_HIG_PLUGIN_STATUS
.word  SCE_HIP_LIGHTMAP_PLUG|(SCE_HIP_REVISION<<24), 0, plugin block size
.byte  1, 0, 0, 0
.word  0, 0, 0
.byte  SCE_HIP_COMMON, SCE_HIP_FRAMEWORK, SCE_HIP_MICRO,
SCE_HIP_PLUGIN_STATUS
.word  SCE_HIP_MICRO_PLUG|(SCE_HIP_REVISION<<24), 0, Micro plugin block
address, 0
```

esconv Converter

The esconv converter is provided for converting an eS (eSTD) file to HiG format. Although esconv can also directly generate binary data, outputting an EE assembler source file, making changes and assembling the file is also possible when necessary.

Figure 5



The header file eSTD.es is required to execute esconv. Refer to the esconv documentation for details.

The internal structure of Shape differs in eSTD and HiG. In eSTD, the relationship Material < Geometry holds, while in esconv, Geometries are sorted for individual Materials, and the relationship Material > Geometry holds. This is done to improve performance by reducing texture transfers, because the drawing performance in the actual device is significantly affected when texture transfers are performed frequently.

Overview of HiG Binary Files Output by esconv

Since the HiG binary files output by esconv are universally processed by the graphic artist tools, ClutBump/EmbossBump/ShadowMap have multiple frame plugins. Refer to the SAMPLE_DATA in the graphic artist tool for details.

ClutBump

Header section

"Bump0"
"Bump1"

"Bump0": Base object=frame having base texture

FRAME
...

"Bump1": ClutBump object= frame having ClutBump texture (Texel&Normal)

FRAME
...
CLUTBUMP

EmbossBump

Header section

"Bump0"
"Bump1"

"Bump0": Base object=frame having base texture

FRAME
...

"Bump1": EmbossBump object= frame having EmbossBump texture (Grayscale)

FRAME
...
CLUTBUMP

ShadowMap

- Shadow Object

Header section

"ShadowObj"
"ShadowTex"
"ShadowMap"

"ShadowObj": Shadow object=shadow object frame

FRAME
...
SHADOWBOX

"ShadowTex": Object for shadow texture rendering= shadow texture frame used with shadow object

SHADOWMAP
MICRO_PLUG (Micro for shadow texture rendering)
SHADOWBOX_DATA (SHADOWBOX_DATA "ShadowObj")
MICRO_DATA (MICRO_DATA "ShadowMap")

"ShadowMap": Object for shadow mapping rendering=shadow mapping frame (used with shadow receiver)

FRAME
MICRO_PLUG (Micro for shadow mapping rendering)

- Shadow receiver

Header section

"ShadowRec"
"ShadowRec1"

"ShadowRec": Shadow receiver object=shadow receiver frame

FRAME
...

"ShadowRec1": Object for shadow mapping rendering=shadow receiver frame having no texture (used with shadow object)

FRAME
SHAPE

GF-VU1 Standard Memory Format

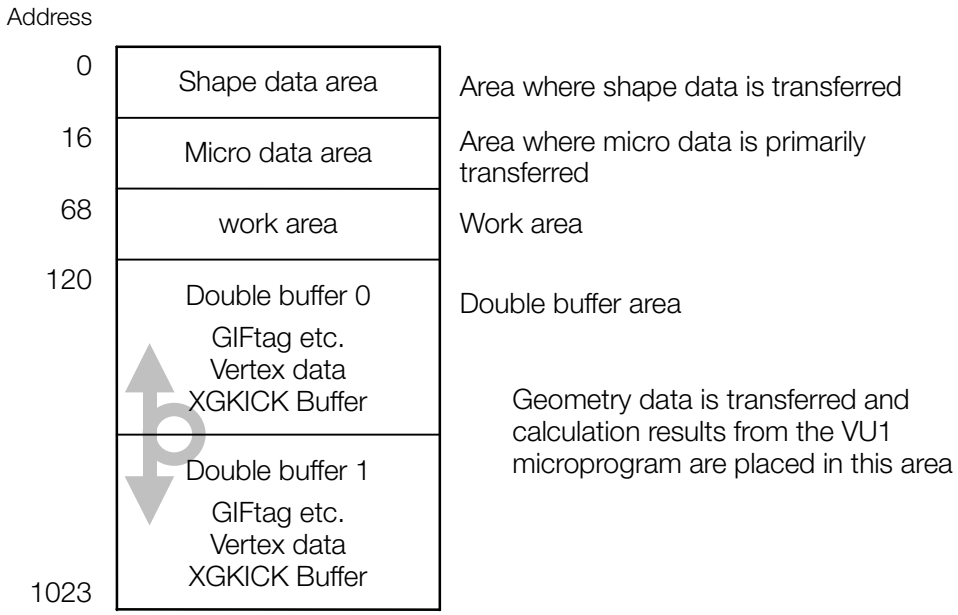
To make development and processing more efficient, the VUMem1 memory format, i.e. the data format processed by the VU1 and the organization of data in memory, have been standardized in HiG, HiP, and framework.

This format is called the GF-VU1 standard memory format and is described below. Please refer to the HiP plugin format together with this description.

GF-VU1 Standard Memory Format Overview

An overview of the GF-VU1 standard memory format is shown below. The addresses are given in qword units.

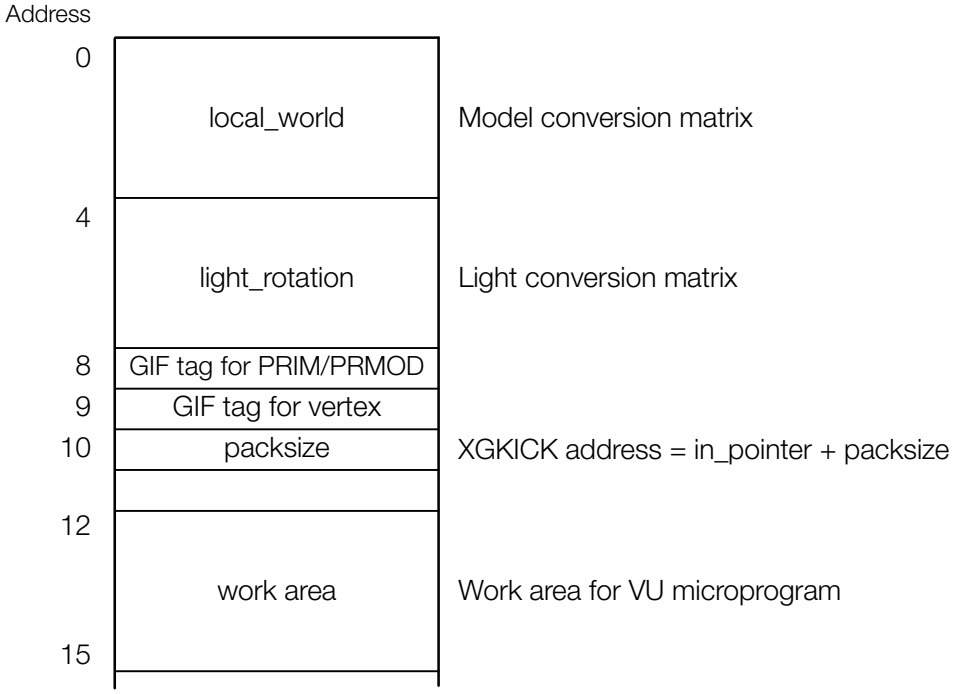
Figure 6: GF-VUI standard memory format overview



Shape Data Area

The data shown below is transferred to the shape data area for each shape data block.

Figure 7

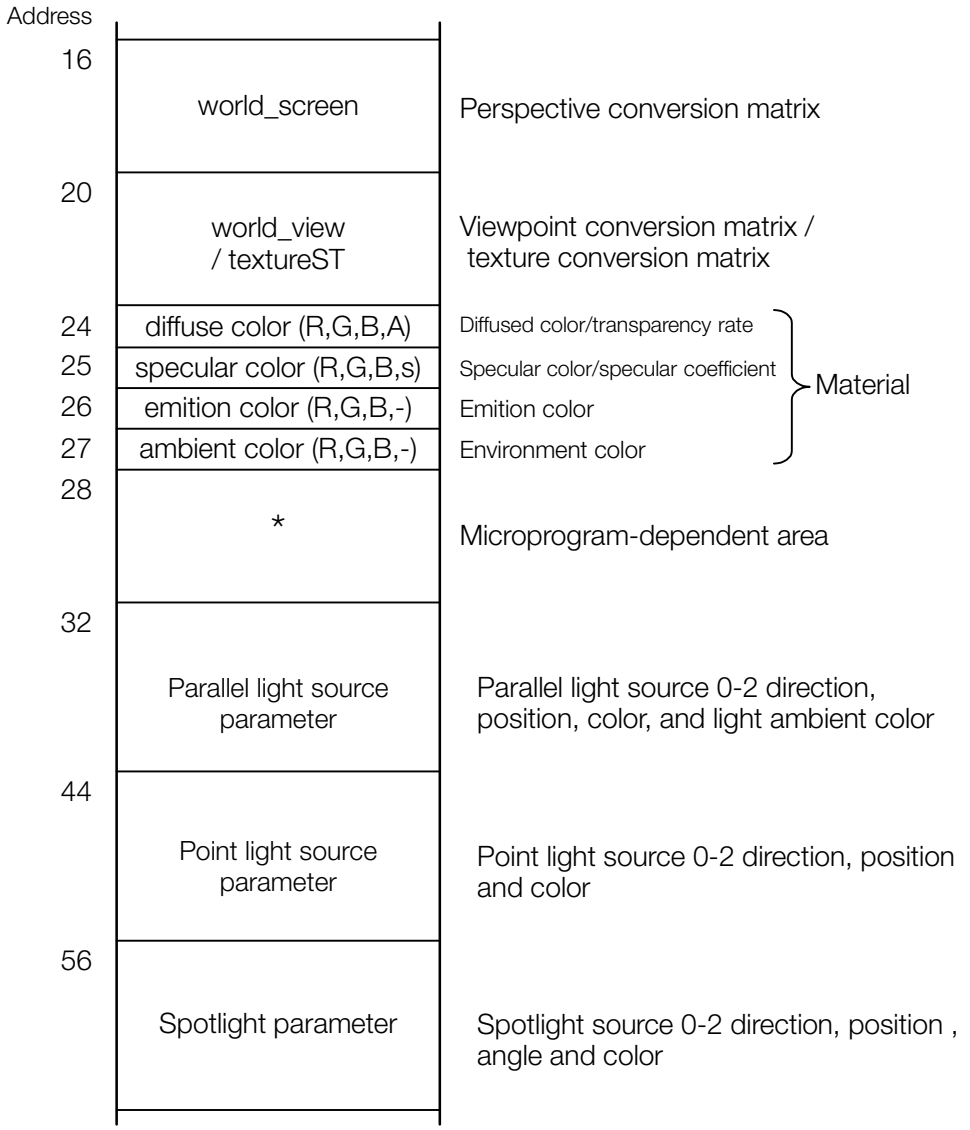


light_rotation is a matrix obtained by removing scan and trans from the local_world matrix.

Micro Data Area

The data shown below is transferred to the micro data area for each microcode data block. However, material data is transferred for each shape data block.

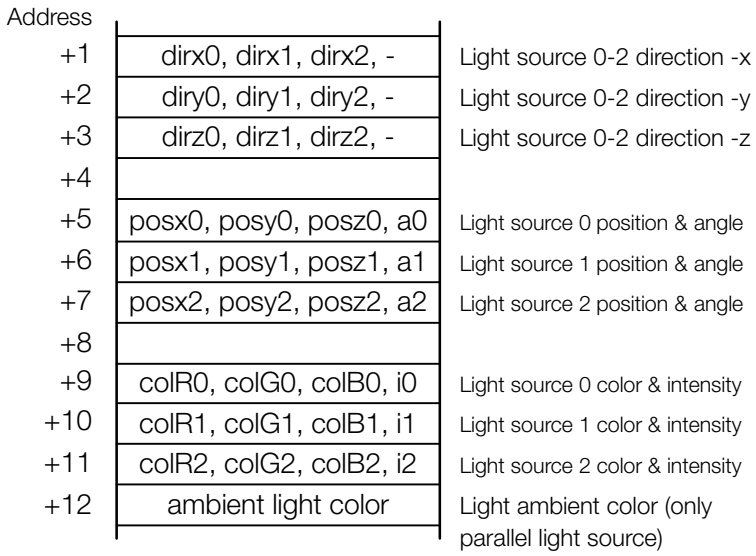
Figure 8



Addresses 20-23 are used as a field of vision transformation matrix or as a texture transformation matrix, depending on the individual microprogram. Addresses 28-31 are also used as a unique parameter area for each microprogram. However, some parameters such as PRMODE, clipping, and camera position are common to many microprograms, and for the most part, these will have a common layout. Details of usage methods with each microprogram are described later in this section.

The parallel light source parameters, point light source parameters, and spot light source parameters have a common format, which is shown below.

Figure 9

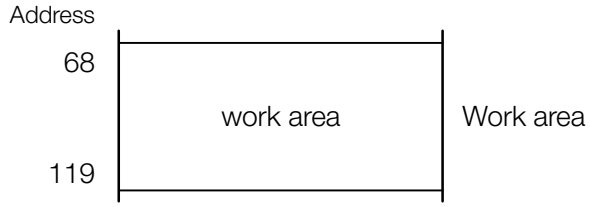


The parameters related to direction have a format that seems strange at first glance. However, that's because they form a transposition matrix. Also, although the position parameter for a parallel light source, for example, is meaningless, it may be used for other purposes, depending on the microprogram.

Work Area

The work area is used by some microprograms for saving parameters.

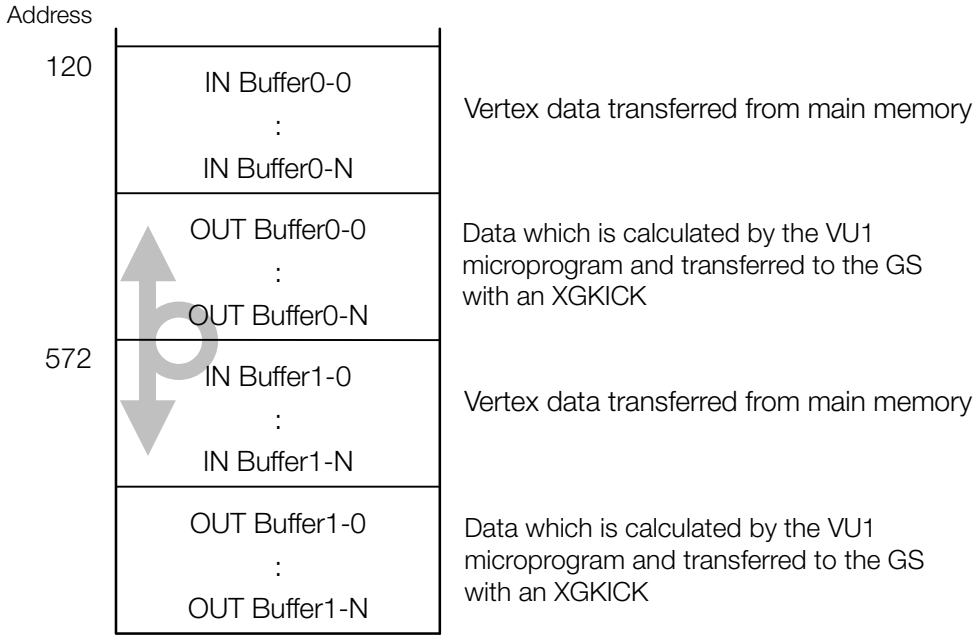
Figure 10



Double Buffer Area

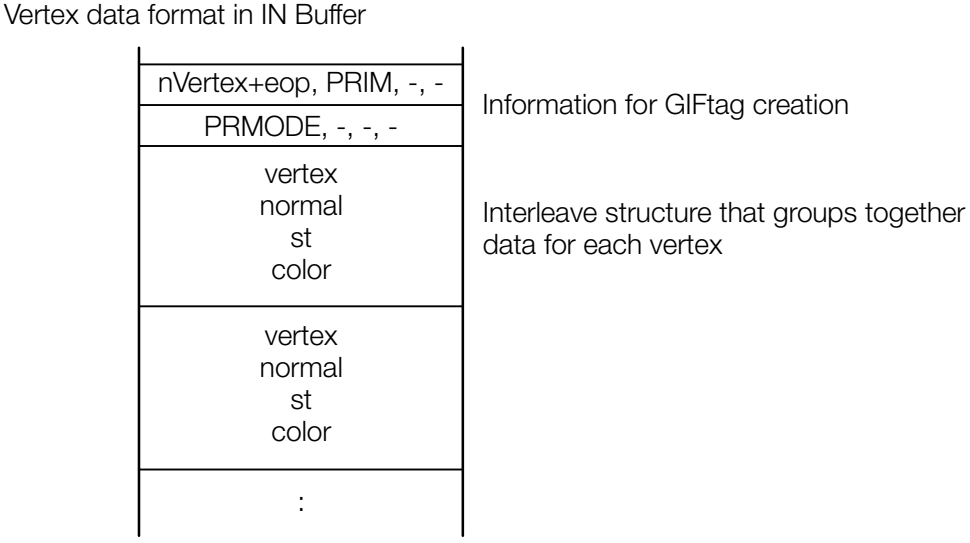
Vertex data, which is part of shape data, is transferred to the double buffer area for each Geometry. In addition, the calculation result is written by the VU1 and transferred to the GS with an XGKICK.

Figure 11



The vertex data format is shown below.

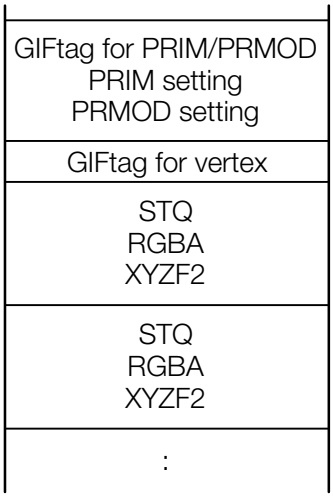
Figure 12



Since the w field of vertex, the w field of normal, and the z and w fields of st are empty, they can keep special information related to each vertex.

Figure 13

Vertex data format in OUT Buffer



Details of Microprogram Dependence Area

Although usage of the microdata area (addresses 16-31) differs for each microprogram, they are collectively displayed in the table below. Due to space limitations in the table, some variable names have been abbreviated (e.g. world_screen -> w_screen, world_view -> w_view, etc.)

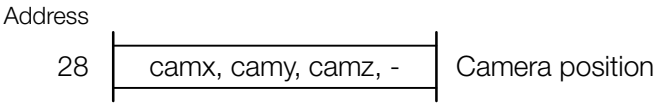
Table 13

Microprogram	16-19	20-23	24-27	28	29	30	31
vu1basicVo	w_screen	—	—	—	prmode	—	—
vu1cullVo / vu1pointVo / vu1spotVo / vu1basicClip / vu1colorSat	w_screen	—	—	—	prmode	clipping	—
vu1fogVo	w_screen	—	—	fog	prmode	clipping	—
vu1antiVo	w_screen	—	—	c_pos / AA1p	prmode	clipping	—
vu1basicLo	w_screen	—	material	—	prmode	—	—
vu1cullLo / vu1pointLo / vu1spotLo	w_screen	—	material	—	prmode	clipping	—
vu1fogLo	w_screen	—	material	fog	prmode	clipping	—
vu1antiLo	w_screen	—	material	c_pos / AA1p	prmode	clipping	—
vu1cullSo / vu1pointSo / vu1spotSo / vu1combiSo / vu1cullBlinno	w_screen	—	material	c_pos	prmode	clipping	—

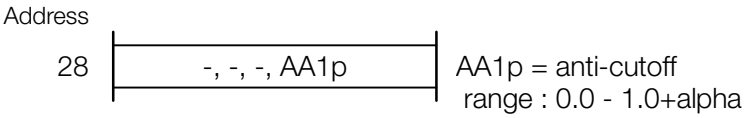
Microprogram	16-19	20-23	24-27	28	29	30	31
vu1embossDir / vu1embossPoint / vu1embossSpot	w_screen	—	—	—	prmode	clipping	emboss
vu1shadowTex	w_screen	—	—	—	prmode	—	—
vu1shadowSTQ / vu1shadowSTQCull / vu1lightmapSTQ / vu1lightmapSTQCull	w_screen	tex_proj	—	—	prmode	clipping	—
vu1fisheye	—	w_view	—	—	prmode	fisheye	—
vu1reflectS	w_screen	w_view	—	c_pos	prmode	clipping	zoom
vu1reflectR	w_screen	w_view	—	c_pos	prmode	clipping	zoom / zshift
vu1refractS	w_screen	w_view	—	c_pos	prmode	clipping	index / zoom
vu1refractR	w_screen	w_view	—	c_pos	prmode	clipping	index / zoom / zshift

Details of each variable are as follows:

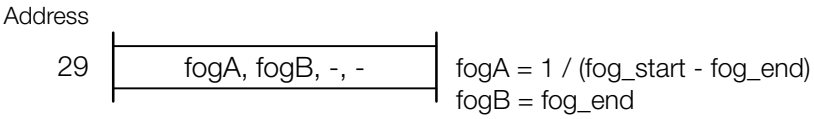
camera_pos（c_pos）



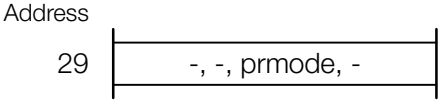
AA1p



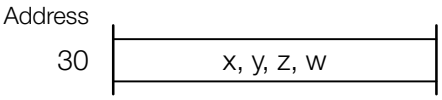
fog



prmode

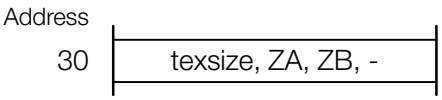


clipping



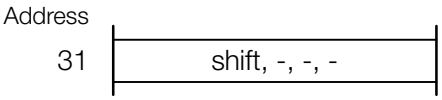
x = (far - near) / screen_width
y = (far - near) / screen_height
z = (far + near) / 2
w = (far - near) / 2

fisheye

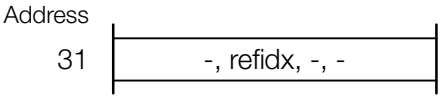


texsize = texture_size / 2
ZA= (rmax * near - rmin * far) / (near - far)
ZB= - (rmax * near * far - rmin * near * far) / (near - far)

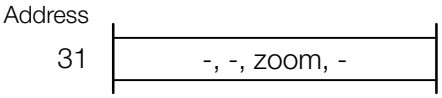
emboss



ref_index（index）



ref_zoom（zoom）



ref_zshift（zshift）