

SEGATM

Kamui



KAMUI

Table of Contents

1. OVERVIEW	KAM-1
2. BASIC PROCESSING FLOW	KAM-3
3. KAMUI FUNCTIONS	KAM-7
3.1 NAMING RULES	KAM-7
3.2 INITIALIZATION FUNCTION	KAM-8
3.2.1 Initializing the Rendering Chip	KAM-8
3.3 SURFACE HANDLING FUNCTION	KAM-9
3.3.1 Setting the Display Mode	KAM-9
3.3.2 Creating the Primary Surface and Off-Screen Surface	KAM-11
3.3.3 Creating the Texture Surface	KAM-12
3.3.4 Creating the Texture Surface for Mipmap / VQ Texture	KAM-14
3.3.5 Creating the Texture Surface in Contiguous Address Areas	KAM-15
3.3.6 Using Frame Buffer as Texture Surface	KAM-17
3.3.7 Setting the a Threshold Value	KAM-19
3.3.8 Determining the Display Screen	KAM-20
3.3.9 Flipping the Display Screen	KAM-21
3.4 SETTING PARAMETERS FOR EACH SCENE SEPARATELY	KAM-22
3.4.1 Setting the Culling Parameter	KAM-22
3.4.2 Setting the Color Clamp Value	KAM-23
3.4.3 Setting the Fog Color	KAM-24
3.4.4 Specifying a Fog Density	KAM-25
3.4.5 Setting the Fog Table	KAM-26
3.4.6 Setting the On-Chip Palette Mode	KAM-27
3.4.7 Setting the On-Chip Palette	KAM-28
3.4.8 Setting the Border Color	KAM-29
3.4.9 Registering the Rendering Parameter of the Background Plane	KAM-29
3.4.10 Setting the Background Plane	KAM-30
3.4.11 Setting Autosort Mode	KAM-31
3.4.12 Specifying Pixel-Unit Clipping	KAM-31
3.4.13 Specifying the Stride Size	KAM-32

3.4.14 Setting the CheapShadowMode	KAM-33
3.4.15 Specifying the Number of VsyncWait States	KAM-34
3.4.16 Forced Reset of Renderer	KAM-35
3.5 SETTING PARAMETERS OF EACH VERTEX	KAM-36
3.5.1 VERTEXCONTEXT	KAM-36
3.5.2 Setting Rendering Parameters for Each Vertex	KAM-46
3.5.3 Registering Rendering Parameters for Each Vertex	KAM-47
3.5.4 Registering Rendering Parameters of a Modifier Volume	KAM-48
3.5.5 Setting Global Clipping	KAM-49
3.5.6 Setting a Strip Length	KAM-50
3.5.7 Setting a User Clipping Area	KAM-51
3.5.8 Setting a User Clipping Area (Direct Specification)	KAM-53
3.5.9 Direct Rewrite Mode for Rendering Status	KAM-54
3.6 RECORDING VERTEX DATA	KAM-56
3.6.1 Recording Vertex Data	KAM-56
3.6.2 Vertex Data Structure	KAM-58
3.6.3 Vertex Parameter	KAM-59
3.6.4 Setting a Modifier Volume	KAM-64
3.6.5 Allocating a Vertex Data Buffer	KAM-65
3.6.6 Releasing Vertex Data Registration Buffers	KAM-66
3.6.7 Allocating the Tile Accelerator Output Buffer	KAM-66
3.6.8 Writing Global Parameters in a Buffer	KAM-67
3.6.9 Directly Writing Global Parameters	KAM-68
3.6.10 Writing Vertex Data in a Buffer	KAM-69
3.6.11 Directly Writing Vertex Data	KAM-71
3.6.12 Reporting the End of Vertex Registration	KAM-73
3.6.13 Notifying the End of Vertex Data Writing (When the Data Is Written into Buffers)	KAM-73
3.6.14 Notifying the End of Vertex Data Writing (When Data Is Directly Written)	KAM-74
3.6.15 Rendering into the Texture Memory (When Data Is Written into Buffers)	KAM-74
3.6.16 Rendering into the Texture Memory (When Data Is Directly Written)	KAM-75
3.6.17 Specifying a Modifier Volume List	KAM-76
3.6.18 Obtaining Current Writing Position of VertexBuffer	KAM-77
3.6.19 Changing Current Writing Position of VertexBuffer	KAM-78
3.6.20 Direct Rewriting of Vertex Control Word	KAM-79
3.6.21 Flushing Opaque VertexBuffer	KAM-80
3.7 CALLBACK FUNCTIONS AND CALLBACK AUXILIARY FUNCTIONS	KAM-81
3.7.1 Specifying a Rendering End Callback Function	KAM-81
3.7.2 Specifying a V-Sync Callback Function	KAM-82
3.7.3 Specifying an H-Sync Interrupt Callback Function	KAM-83
3.7.4 Setting the H-Sync Interrupt Line	KAM-84
3.7.5 Reading the Current H-Sync Line	KAM-84
3.7.6 Specifying a Texture Memory Overflow Callback Function	KAM-85
3.7.7 Specifying a Strip Buffer Overrun Callback Function	KAM-86
3.7.8 Specifying a Vertex Data Transfer End Callback Function	KAM-87

3.8 OTHER FUNCTIONS	KAM-88
3.8.1 Stopping the Frame Buffer Display	KAM-88
3.8.2 Obtaining the Version Information	KAM-88
3.9 TEXTURE HANDLING FUNCTIONS OF KAMUI	KAM-89
3.9.1 Loading Texture Data	KAM-90
3.9.2 Re-reading the Code Book Portion of VQ Texture	KAM-91
3.9.3 Reloading a Particular Mipmap Texture	KAM-92
3.9.4 Reading the YUV-Format Texture Data	KAM-95
3.9.5 Deleting Texture Data	KAM-96
3.9.6 Obtaining the Available Texture Memory Space	KAM-97
3.9.7 Reading the Texture in Texture Memory	KAM-98
3.9.8 Garbage Collection of Texture Memory	KAM-99
4. KAMUI UTILITY LIBRARY KAM-101	
4.1 TEXTURE-RELATED FUNCTIONS	KAM-102
4.1.1 Conversion from KAMUI Bit Map Format to Twiddled Format	KAM-102
4.1.2 Conversion from Frame Buffer Format (Rectangle) to Windows BMP Format	KAM-103
5. STRUCTURES KAM-105	
5.1 FRAME BUFFER/TEXTURE SURFACE STRUCTURE	KAM-105
5.2 VERSION INFORMATION STRUCTURE	KAM-106
5.3 VERTEX CONTEXT	KAM-107
5.4 PACKED 32-BIT COLORS	KAM-108
5.5 PALETTE DEFINITION STRUCTURE	KAM-108
5.6 VERTEX DATA BUFFER STRUCTURE	KAM-108
6. TEXTURE FORMAT KAM-109	
6.1 TEXTURE FORMATS SUPPORTED BY ARC1/CLX1	KAM-109
6.2 ARC1/CLX1 TEXTURE FORMAT	KAM-110
6.2.1 Texture Format of KAMUI	KAM-110
6.2.2 Twiddled Format and Twiddled Mipmap Format	KAM-113
6.2.3 VQ Format and VQ Mipmap Format	KAM-116
6.2.4 Palettized 4-bpp/8-bpp Format	KAM-120
6.2.5 Rectangle Format	KAM-123
6.2.6 Stride Format	KAM-124
6.2.7 BUMP-Mapping Format	KAM-125
6.2.8 Kamui Bit Map Format	KAM-127



1. OVERVIEW

KAMUI™ is a driver maintenance layer used by drivers and other software in an environment wherein the ARC1, which is the next-generation PowerVR™ chip, is added with a Tile accelerator. These drivers and software are expected to run at a high speed in this environment.

KAMUI supports the following environments:

- a) PC IRIS + ARC1 evaluation board
- b) SH4 COSMOS + ARC1 evaluation board
- c) SH4 HOLLY (CLX1)

The basic input/output parameters for KAMUI remain the same in any of the above environments, except that some functions are unavailable in items a) and b) because the HOLLY has functions such as BumpMapping and Trilinear Filter, but the ARC1 does not.

KAMUI supports the following functions:

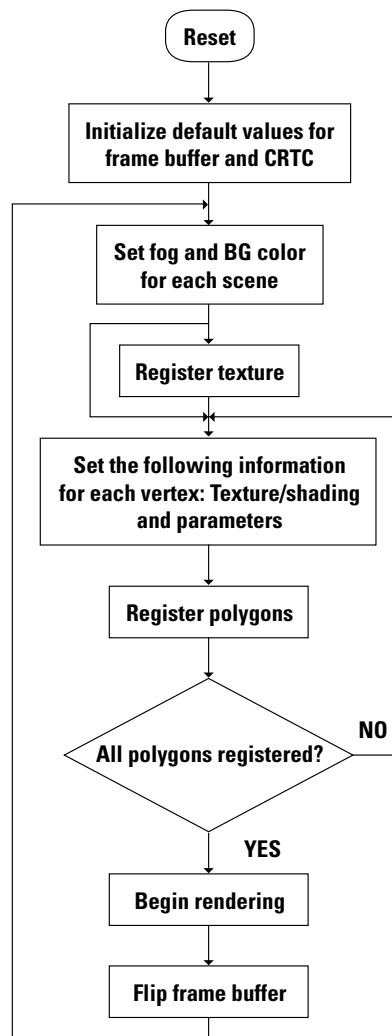
- Setting the rendering context
- Registering triangular polygon strips and tetragonal polygons with scenes.
- Handling the frame buffer
- Handling the texture
- Setting various special effects (such as fog and modifier volume)
- Other service routines

KAMUI performs only minimum error checks. If invalid parameters (those not stipulated, for example) are given to KAMUI, its normal operation is not guaranteed.



2. BASIC PROCESSING FLOW

The following flowchart shows the basic processing flow of KAMUI.



After a reset occurs, an application must first set the frame buffer and display controller. The frame buffer is allocated two areas: Primary surface for ordinary display, and off-screen surface that is a rendering target of a nondisplay area. The frame buffer is configured in a double buffer system, in which the Flip command causes the two areas to be exchanged with each other.

After initializing the frame buffer, the application sets parameters (such as fog table data and background color) related to an entire scene.

Now, the application registers texture. This is done by transferring texture data from system memory to texture memory. (Texture memory will not receive texture data directly from file I/O units.) The application simply transfers data from an area allocated in system memory to texture memory. In the previous example, texture is registered after parameters related to the entire scene are registered. Texture registration can be done at any point in the above flowchart. (Do not rewrite texture data during the course of rendering, however.)

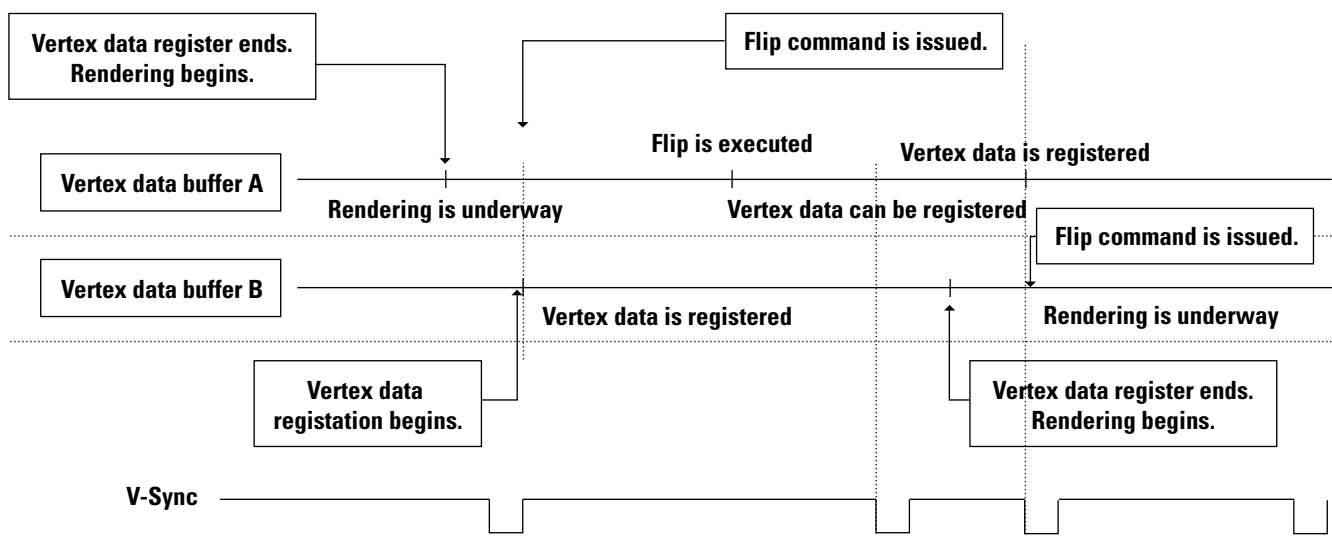
Next, the application sets rendering conditions (such as coordinate space, texture, and texture filter) for each polygon to be registered. Registering all rendering conditions every time would be extremely inefficient in some cases. So, the application only has to set changes from the previous setting. After setting is completed, polygon vertex data is registered. **All vertex data is registered in the Strip format.** If you want to specify a single polygon, register it as a polygon with Strip = 1.

Upon completion of polygon registration, a rendering start command is issued. Processing to be performed at the end of rendering can be added using a polling-based wait function or a callback function to be executed at the end of rendering.

The frame buffer can be flipped after the rendering start command is issued. When the frame buffer Flip command is issued, it is stacked in a queue. **If rendering is completed before the first V-Sync after the Flip command is executed, the primary surface and off-screen surface are exchanged automatically.** This way, continuous display becomes possible.

The Flip command is only enqueued. It does not wait for V-Sync to be detected. After executing the Flip command, a program utilizing KAMUI can start registering vertex data for the next scene immediately, as long as vertex data registration is possible (if the registration area is not currently being used for rendering). In KAMUI, vertex data for rendering is submitted to double buffering, similarly to the frame buffer. This is because vertex data being currently rendered and parameters being registered are necessary. Programs need not be aware of that, however.

The following chart shows the timing when vertex data is registered, rendering is started, and the Flip command is issued.



The processing shown above is performed as pipelining. If the following conditions occur, wait states are inserted:

- An attempt is made to issue a rendering start command for the next scene when hardware rendering is under way (this can occur if processing ends within a short time because only a small amount of vertex data is registered after the Flip command is issued). Alternatively, an attempt is made to issue a rendering start command for the next scene when hardware rendering is under way (this can occur if hardware rendering takes much time).

Environment mapping requires that rendering be applied to rectangular texture. So, special frame buffer control is necessary. See the description of the frame buffer handling function for details.



3. KAMUI FUNCTIONS

3.1 NAMING RULES

The following naming rules apply to the KAMUI functions and structures.

Function	kmXXXX
ENUM value, #define	KM_XXXX
Structure and variable	KMXXXX
Structure pointer	PKMXXXX

An attempt to call a function that has not been implemented will be responded with `KMSTATUS_NOT_IMPLEMENTED`.

3.2 INITIALIZATION FUNCTION

This function initializes the hardware. Video signals are output to blank out the screen.

3.2.1 Initializing the Rendering Chip

```
KMSTATUS kmInitDevice (KMVIDEOMODE nVideoMode)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function initializes the hardware to output the default background color (black) to the screen. First call this API when using KAMUI.

The ARC1 always displays in VGA mode regardless of parameter settings.

Argument:

nVideoMode (input)

This argument specifies a video mode by selecting one from:

```
KM_NTSC ..... NTSC bloc (North America and Japan)
KM_PAL ..... PAL bloc (European countries)
KM_VGA ..... VGA (ARC1)
```

For the ARC1, KM_VGA is assumed no matter what mode is specified.

Return values:

```
KMSTATUS_SUCCESS Function successful
KMSTATUS_INVALID_VIDEOMODE Invalid video mode specified
KMSTATUS_HARDWARE_NOT_PRESENTED
```

Example: `kmInitDevice(KM_NTSC);`

3.3 SURFACE HANDLING FUNCTION

This function generates, erases, or flips the frame buffer and texture buffer. It also manages the states of the buffers.

3.3.1 Setting the Display Mode

```
KMSTATUS kmSetDisplayMode(KMDISPLAYMODE nDisplayMode
                           KMBPPMODE nBpp,
                           KMBOOLEAN bDither,
                           KMBOOLEAN bAntiAlias)
KMSTATUS kmChangeDisplayFilterMode(KMBOOLEAN bDither,
                                   KMBOOLEAN bAntiAlias)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Implemented

Note: The ARC1 can be used only in VGA mode.

Explanation: This function sets the display mode of the frame buffer. `KmChangeFilterMode` is used to turn on/off the dithering and antialiasing filters later.

Arguments:

nDisplayMode (input)

This argument specifies a display mode.

```
KM_DSPMODE_VGA.....VGA (640 x 480) 60 Hz
KM_DSPMODE_NTSCNI320x240.....320 x 240 Noninterlace 60 Hz
KM_DSPMODE_NTSCI320x240.....320 x 240 Interlace1 30 Hz
KM_DSPMODE_NTSCNI640x240.....640 x 240 Noninterlace 60 Hz
KM_DSPMODE_NTSCI640x240.....640 x 240 Interlace 30 Hz
KM_DSPMODE_NTSCI640x480.....640 x 480 Interlace2 30 Hz
KM_DSPMODE_PALNI320x240.....320 x 240 Noninterlace 50 Hz
KM_DSPMODE_PALI320x240.....320 x 240 Interlace 25 Hz
KM_DSPMODE_PALNI640x240.....640 x 240 Noninterlace 50 Hz
KM_DSPMODE_PALI640x240.....640 x 240 Interlace 25 Hz
KM_DSPMODE_PALI640x480.....640 x 480 Interlace 25 Hz
```

1. In interlace mode, the same picture is drawn in both odd and even frames.

2. This is a high-resolution mode, in which different pictures are drawn in odd and even fields.

nBpp (input)

This argument specifies a frame buffer color mode, using a predefined constant listed below.

KM_DSPBPP_RGB565 RGB565 format
KM_DSPBPP_RGB555 RGB555 format
KM_DSPBPP_ARGB1555 ARGB1555 format
KM_DSPBPP_RGB888 RGB888 format
KM_DSPBPP_ARGB8888 ARGB8888 format

bDither (input)

TRUEDithering is used when a rendering result is written to the 16-bit frame buffer.
(The argument is ineffective when the RGB888 or ARGB8888 frame buffer is used.)

FALSEDithering is not used.

bAntiAlias (input)

TRUEThe antialiasing filter is used, in which case the operation speed may get lowered.

FALSENo antialiasing filter is used.

Return values:

KMSTATUS_SUCCESS	Set successfully
KMSTATUS_INVALID_DISPLAY_MODE	Invalid display mode

3.3.2 Creating the Primary Surface and Off-Screen Surface

```

KMSTATUS kmCreateFrameBufferSurface (
    PKMSURFACEDESC pSurfaceDesc,
    KMINT32 nWidth,
    KMINT32 nHeight,
    KMBOOLEAN fStripBuffer,
    KMBOOLEAN fBufferClear)

```

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Implemented

Explanation: This function allocates a display surface in frame buffer memory and initializes it. Before this function is called, `kmSetDisplayMode` must be executed.

Arguments

pSurfaceDesc (output)

This argument is a pointer to a surface information structure. Surface information is returned using the pointer. It becomes undefined if `KMSTATUS` is responded with `KMSTATUS_NOT_ENOUGH_MEMORY`.

nWidth and nHeight (input)

These arguments specify the horizontal and vertical surface sizes. If both sizes are specified as 0, the screen size becomes the one specified in `kmSetDisplayMode`.

The HOLLY can use a strip buffer. To use the strip buffer, it is necessary to specify a multiple of 32.

The ARC1 has no strip buffer function. So, both arguments must always be 0 for the ARC1.

fStripBuffer (input)

If this argument is `TRUE`, a frame buffer in `StripBuffer` format is created. No strip buffer function is available to the ARC1. So, this argument must always be `FALSE` for the ARC1.

fBufferClear (input)

`TRUE` When a surface is created, it is cleared to 0.

`FALSE` When a surface is created, it is not cleared to 0.

Return values:

<code>KMSTATUS_SUCCESS</code>	Function successful
<code>KMSTATUS_NOT_ENOUGH_MEMORY</code>	Failed because of insufficient memory

Examples:

```

KMSTATUS status;
SURFACEDESC SurfDesc;
status = kmCreateFrameBufferSurface(&SurfDesc, 0, 0, FALSE,
TRUE);

```

3.3.3 Creating the Texture Surface

```
KMSTATUS kmCreateTextureSurface(  
    PKMSURFACEDESC pSurfaceDesc,  
    KMINT32 nWidth,  
    KMINT32 nHeight,  
    KMTEXTURETYPE nTextureType)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function secures a texture surface in texture memory. This API can allocate texture surfaces in all formats.

ARC1 interleaves a VQ/Twiddled mipmap format texture. Therefore, a texture surface may not be created even if the total amount of free space in the frame buffer exceeds the size of the VQ/Twiddled mipmap to be secured.

ARC1 searches for the smallest texture pair that is larger than the specified size if the texture surface of a VQ/Twiddled mipmap is specified, and creates a texture surface in a single bank of that pair if it is vacant.

Efficient use of texture memory

(Common to ARC1 and CLX1)

- Secure/release as many texture surfaces as possible at the same time.
- Call the creation of a frame buffer area or TA output data area (kmCreateFrameBufferSurface, kmCreateVertexBuffer API) before the creation of a texture surface, and avoid releasing and re-creating these until AP ends.

(ARC1-specific)

- Execute kmCreateVertexBuffer before kmCreateFrameBufferSurface.
- Allocate Mipmap, VQ texture surface first.
- Avoid alternately allocating texture surfaces of different sizes. Whenever possible, simultaneously allocate surfaces of the same size and format.

Arguments:

pSurfaceDesc (output)

This argument is a pointer to a surface information structure. Surface information is returned using the pointer. It becomes undefined if KMSTATUS is responded with KMSTATUS_NOT_ENOUGH_MEMORY.

nWidth and nHeight (input)

These arguments specify the horizontal and vertical texture sizes. If MIPMAP is used, the top-level texture size must be specified. If square texture such as Twiddled, VQ, or Palettized is used, only a value specified in nWidth is used. If rectangular texture is used, both nWidth and nHeight are valid.

A value specified in nWidth or nHeight must be 8, 16, 32, 64, 128, 256, 512, or 1024.

nTextureType (input)

This argument specifies a texture format. The texture format is specified by ORing a category code and pixel format code selected from those listed below.

Category codes

KM_TEXTURE_TWIDDLED	// Twiddled format
KM_TEXTURE_TWIDDLED_RECTANGLE	// Rectangular Twiddled format (Cannot be used with ARCl.)
KM_TEXTURE_TWIDDLED_MM	// Twiddled format with Mipmap
KM_TEXTURE_VQ	// VQ compression format
KM_TEXTURE_VQ_MM	// VQ compression format with Mipmap
KM_TEXTURE_PALETTIZE4	// 4-bpp palette format (Cannot be used with ARCl.)
KM_TEXTURE_PALETTIZE4_MM	// 4-bpp palette format with Mipmap (Cannot be used with ARCl.)
KM_TEXTURE_PALETTIZE8	// 8-bpp palette format (Cannot be used with ARCl.)
KM_TEXTURE_PALETTIZE8_MM	// 8-bpp palette format with Mipmap (Cannot be used with ARCl.)
KM_TEXTURE_RECTANGLE	// Rectangle
KM_TEXTURE_STRIDE	// Rectangle (stride specification)

Pixel format codes

KM_TEXTURE_ARGB1555	
KM_TEXTURE_RGB565	
KM_TEXTURE_ARGB4444	
KM_TEXTURE_YUV422	(Cannot be used with ARCl.)
KM_TEXTURE_BUMP	(Cannot be used with ARCl.)
KM_TEXTURE_8888	

Note: With the palette format texture (KM_TEXTURE_PALETTIZED4, KM_TEXTURE_PALETTIZED4_MM, KM_TEXTURE_PALETTIZED8, KM_TEXTURE_PALETTIZED8_MM), the pixel format cannot be specified. Specify the pixel format of the palette format texture by setting a palette (kmSetPaletteMode).

Return values:

KMSTATUS_SUCCESS	Secured successfully
KMSTATUS_INVALID_TEXTURE_TYPE	Invalid texture type specified
KMSTATUS_INVALID_MIPMAPPED_TEXTURE	
KMSTATUS_NOT_ENOUGH_MEMORY	Insufficient memory capacity

3.3.4 Creating the Texture Surface for Mipmap/VQ Texture

```

KMSTATUS kmCreateCombinedTextureSurface(
    PKMSURFACEDESC pSurfaceDesc1,
    PKMSURFACEDESC pSurfaceDesc2,
    KMINT32 nWidth,
    KMINT32 nHeight,
    KMTEXTURETYPE nTextureType)

```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function secures a texture surface in texture memory. It secures two texture surfaces of the same size and format.

This API explicitly secures two surfaces interleaved by the ARC1 (Twiddled-mipmap or VQ (mipmap) in pairs. Like `kmCreateTextureSurface`, however, this API can allocate texture surfaces in all formats.

Arguments:

pSurfaceDesc1 (OutPut)

This argument is a pointer (No. 1) to surface structure information. Surface information is returned using the pointer. It becomes undefined if, for `KMSTATUS`, `KMSTATUS_NOT_ENOUGH_MEMORY` is returned.

pSurfaceDesc2 (OutPut)

This argument is a pointer (No. 2) to surface structure information. Surface information is returned using the pointer. It becomes undefined if, for `KMSTATUS`, `KMSTATUS_NOT_ENOUGH_MEMORY` is returned.

nWidth, nHeight (Input)

These arguments specify the horizontal and vertical texture sizes. If `MIPMAP` is used, the top-level texture size must be specified. If a square texture such as `Twiddled`, `VQ`, or `Palettized` is used, only the value specified for `nWidth` is used. If a rectangular texture is used, both `nWidth` and `nHeight` are valid.

The value specified for `nWidth` or `nHeight` must be 8, 16, 32, 64, 128, 256, 512, or 1024.

nTextureType (Input)

This argument specifies a texture format. See `kmCreateTextureSurfaceCategory` code

<code>KMSTATUS_SUCCESS</code>	Secured successfully
<code>KMSTATUS_INVALID_TEXTURE_TYPE</code>	Invalid texture type specified
<code>KMSTATUS_NOT_ENOUGH_MEMORY</code>	Insufficient memory

3.3.5 Creating the Texture Surface in Contiguous Address Areas

```

KMSTATUS kmCreateContiguousTextureSurface
    PPKMSURFACEDESC          ppSurfaceDesc,
    KMINT32                   nTexture,
    KMINT32                   nWidth,
    KMINT32                   nHeight,
    KMTEXTURETYPE             nTextureType)

```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function simultaneously secures two or more texture surfaces at contiguous addresses in the frame buffer. It is used to read two or more textures of YUV422 type in succession, by using the YUV converter of Tile Accelerator of the CLX1 (see `kmLoadYUVTexture`).

Like `kmCreateTextureSurface`, however, this API can also allocate texture surfaces in all formats.

However, the ARC1 cannot secure surfaces interleaved by this API (VQ, VQ-mipmap, Twiddled-mipmap).

Argument:

ppSurfaceDesc (output)

This is a pointer to a `KMSURFACEDESC` structure. Secured texture surface information is returned. It becomes undefined if, for `KMSTATUS`, `KMSTATUS_NOT_ENOUGH_MEMORY` is returned.

nTexture (input)

This argument specifies the number of Texture Surfaces to be secured in succession.

nWidth, nHeight (input)

These arguments specify the horizontal size and vertical size of the texture. The value specified for `nWidth` or `nHeight` must be 8, 16, 32, 64, 128, 256, 512, or 1024.

nTextureType (input)

This argument specifies a texture format. The texture format is specified by ORing a category code and pixel format code selected from those listed below.

Category codes

```
KM_TEXTURE_TWIDDLED           // Twiddled format
KM_TEXTURE_TWIDDLED_MM        // Twiddled format with Mipmap
                                (Cannot be specified with ARC1)
KM_TEXTURE_TWIDDLED_RECTANGLE // Twiddled-Rectangle
                                (Cannot be used with ARC1)
KM_TEXTURE_VQ                 // VQ compression format
                                (Cannot be specified with ARC1)
KM_TEXTURE_VQ_MM              // VQ compression format with Mipmap
                                (Cannot be specified with ARC1)
KM_TEXTURE_PALETTIZE4         // 4-bpp palette format
                                (Cannot be used with ARC1)
KM_TEXTURE_PALETTIZE4_MM      // 4-bpp palette format with Mipmap
                                (Cannot be used with ARC1)
KM_TEXTURE_PALETTIZE8         // 8-bpp palette format
                                (Cannot be used with ARC1)
KM_TEXTURE_PALETTIZE8_MM      // 8-bpp palette format with Mipmap
                                (Cannot be used with ARC1)
KM_TEXTURE_RECTANGLE          // Rectangle
KM_TEXTURE_STRIDE              // Rectangle
                                (with stride specification)
```

Pixel format codes

```
KM_TEXTURE_ARGB1555
KM_TEXTURE_RGB565
KM_TEXTURE_ARGB4444
KM_TEXTURE_YUV422             // (Cannot be used with ARC1)
KM_TEXTURE_BUMP                // (Cannot be used with ARC1)
```

Return values:

```
KMSTATUS_SUCCESS              Texture memory secured successfully
KMSTATUS_INVALID_TEXTURE_TYPE Invalid texture type specified
KMSTATUS_NOT_ENOUGH_MEMORY    Insufficient memory
```

3.3.6 Using Frame Buffer as Texture Surface

```
KMSTATUS kmSetFramebufferTexture(
    PKMSURFACEDESC pSurfaceDesc)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Does not work

Explanation: This function reads frame buffer information into `SurfaceDesc` for texture. The contents of the frame buffer immediately before are set in `SurfaceDesc` as Stride-Rectangular-Texture. The contents of the frame buffer immediately before can be used as a texture by mapping textures by using this `SurfaceDesc`.

As the frame buffer to be used as a texture must be `KM_DSPBPP_ARGB1555`, `KM_DSPBPP_ARGB4444`, `KM_DSPBPP_RGB565`, or `KM_DSPBPP_RGB555` must be specified in `kmSetDisplayMode` as `BitDepth`.

Each parameter of `SurfaceDesc` is set as follows:

- `SurfaceType` `KM_SURFACETYPE_TEXTURE`
- `BitDepth` `KM_BITDEPTH_16`

PixelFormat

`PixelFormat` is as follows depending on the setting of the display mode of the frame buffer:

Display Mode	PixelFormat
<code>KM_DSPBPP_ARGB1555</code> <code>KM_DSPBPP_RGB555</code>	<code>KM_PIXELFORMAT_ARGB1555</code>
<code>KM_DSPBPP_RGB565</code>	<code>KM_PIXELFORMAT_RGB565</code>
<code>KM_DSPBPP_ARGB4444</code>	<code>KM_PIXELFORMAT_ARGB4444</code>

- `Usize`, `Vsize` Always 1024 x 1024.
- `dwTextureSize` The capacity of the actual frame buffer is set. The vertical size (pixels) x horizontal size (pixel) x 2 (bytes/pixel) of the screen size is set.
- `fSurfaceFlags` `KM_SURFACEFLAGS_STRIDE`
- `pSurface` First address of the displayed frame buffer

When a 640 x 480 frame buffer is used, for example, the entire screen can be pasted as a texture if the stride value is specified to be 640 by `kmSetStrideWidth` and U and V coordinates are set to (0.0f, 0.0f) - (0.625f, 0.46875f).



Caution: The values (1024 x 1024) set for `USize` and `VSize` of output `SurfaceDesc` do not coincide with the size set to `dwTextureSize` (the actual capacity of the frame buffer memory). Care must be exercised when specifying U and V coordinates and reading a texture surface (When specifying U and V coordinates, determine a value as a texture of 1024 x 1024).

When pasting the contents of the frame buffer to a background plane by using `kmSetFramebufferTexture`, the screen mode of the frame buffer (`kmSetDisplayMode`) must be other than `KM_DSPBPP_RGB565` or `bDither = FALSE`.

If the mode is `RGB565`, the PowerVR decrements the value of G to make the luminance of R, G, and B even when data is written into the frame buffer. Consequently, if the frame buffer is pasted to the back plane, the value of G is decremented every frame, resulting in an illegal screen color. If `bDither = TRUE`, dither processing is performed repeatedly on the image on the same background plane, causing a moire pattern.

Argument:

pSurfaceDesc (output)

This argument is a pointer to a `KMSURFACEDESC` structure area. Frame buffer information is returned.

Return values:

<code>KMSTATUS_SUCCESS</code>	Reading frame buffer information successful
<code>KMSTATUS_INVALID_TEXTURE_TYPE</code>	Current setting of frame buffer cannot be used as texture.

Examples:

```
kmSetStrideWidth(nScreenWidth);           // Set screen width
....(Setup vertex)....
kmRender();                               // Generate previous Frame-Buffer
kmSetFramebufferTexture(pSurfaceDesc);    // Get Framebuffer information
VertexContext.pTextureSurfaceDesc = &pSurfaceDesc;
                                           // Set texture(Framebuffer)
kmProcessVertexRenderState(&VertexContext); // Use Framebuffer as texture
kmSetVertexRenderState(&VertexContext);    // Set new vertex-context
....(Setup vertex)....
kmRender();                               // Render!!
```


3.3.7 Setting the α Threshold Value

KMSTATUS kmSetAlphaThreshold(KMINT32 nThreshold)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: The threshold value specified by this function is used to determine whether to set the α bit to 1 if the 1555 surface is a rendering surface. If the rendering result is greater than or equal to the specified threshold value, the α bit is set to 1. This bit is used for chroma-key composition by RAMDAC.

Argument:

nThreshold (input)

This argument specifies a threshold value from 0 to 255. If a value less than 0 or greater than 255 is specified, 0 or 255 is assumed, respectively.

Return value:

KMSTATUS_SUCCESS Set successfully

3.3.8 Determining the Display Screen

```
KMSTATUS kmActivateFrameBuffer(  
    PKMSURFACEDESC pPrimarySurfaceDesc,  
    PKMSURFACEDESC pBackBufferSurfaceDesc,  
    KMBOOLEAN fStripBuffer,  
    KMBOOLEAN bWaitVSync)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Does not work

Explanation: This function starts displaying the content of the frame buffer. After this API has been issued, `kmSetBackGroundRenderState` and `kmSetBackGroundPlane` cannot be set.

Arguments:

pPrimarySurfaceDesc (input)

This argument is the surface structure of a surface to be used for displaying. The surface structure must be one obtained by securing a surface using `kmCreateFrameBufferSurface`.

pBackBufferSurfaceDesc (input)

This argument is the surface structure of a surface to be submitted to rendering.

fStripBuffer (input)

This argument must be `TRUE` if the strip buffer is used. This function cannot be used with ARC1.

bWaitVSync (input)

This argument specifies whether to defer displaying a surface till the timing of Vsync. If the argument is `TRUE`, this function defers displaying till the timing of Vsync.

Return value:

`KMSTATUS_SUCCESS` Display switched successfully

3.3.9 Flipping the Display Screen

KMSTATUS kmFlipFrameBuffer(KMVOID)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: If the frame buffer is configured for double buffering³, this function exchanges the display primary surface with the off-screen surface (rendering target). The Flip command is enqueued. It is executed if rendering for the off-screen surface has been completed at the first Vsync timing after the execution of the previous Flip command.

If the strip buffer is used, software-based Flip is not needed.

Argument:

None

Return values:

KMSTATUS_SUCCESS	Flip command issued successfully
KMSTATUS_CANT_FLIP_SURFACE	Failure in issuing Flip command

3. Unnecessary if the strip buffer is used.

3.4 SETTING PARAMETERS FOR EACH SCENE SEPARATELY

3.4.1 Setting the Culling Parameter

```
KMSTATUS kmSetCullingRegister(KMFLOAT fCullVal)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies a threshold value for culling small polygons.

Argument:

fCullVal (input)

This argument sets a determinant value for a plane parameter.

Return value:

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.4.2 Setting the Color Clamp Value

```
KMSTATUS kmSetColorClampValue(  
                                KMPACKEDARGB MaxVal,  
                                KMPACKEDARGB MinVal)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function specifies the color clamp value. Color clamping is applied ahead of fogging. If you want to change the clamp color when rendering, do so within a callback function for rendering termination. If an attempt is made to change the clamp color at any other timing, a screen image may become invalid.

Arguments:

MaxVal (input)

This argument specifies a maximum value for color clamping. It is a packed 32-bit color. If you want to specify the RGB color with a brightness of 128, enter 0x00808080.

MinVal (input)

This argument specifies a minimum value for color clamping. It is a packed 32-bit color. If you want to specify the RGB color with a brightness of 20, enter 0x00141414.

Return value:

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.4.3 Setting the Fog Color

```
KMSTATUS kmSetFogTableColor(KMPACKEDARGB FogTableColor)
KMSTATUS kmSetFogVertexColor(KMPACKEDARGB FogVertexColor)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Implemented

Explanation: This function specifies a fog color. For the HOLLY, different fog colors can be specified in FogVertex and FogTable. For the ARC1, however, the most recently specified value is valid even if different fog colors are specified using FogVertex and FogTable, because they are identical for the ARC1. If you want to change the fog color when rendering, do so within a callback function for rendering termination. If an attempt is made to change the fog color at any other timing, a screen image may become invalid.

Arguments:

FogTableColor and FogVertexColor (input)

These arguments specify the packed 32-bit color to be used in FogTable and FogVertex.

Return value:

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.4.4 Specifying a Fog Density

KMSTATUS kmSetFogDensity(KMDWORD fogDensity)

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Implemented

Explanation: This function assigns a coefficient (scale factor) to TSP Fog.

FogDensity consists of two bytes. The higher byte indicates the mantissa and the lower byte indicates the exponent (the nth power of 2).

Example	0x0100 = 0.00000001(b)	= 0.0078125
	0x8000 = 0.1(b)	= 0.5
	0xFF00 = 0.11111111(b)	= 0.9921875
	0xFF01 = 1.11111111(b)	= 1.984375
	0xFF07 = 1111111.1(b)	= 128.5
	0xFF08 = 1111111.0(b)	= 255
	0xFF09 = 11111110.0(b)	= 510
	0xFF0A = 111111100.0(b)	= 1020
	0xFF0B = 2040	
	0xFF0C = 4080	
	0xFF0D = 8160	

If a low value is specified for FogDensity, the effect of Fog appears from the polygon with the higher $1/w$ (Fog density increases).

If a high value is specified for FogDensity, the effect of Fog can be seen only on the polygon with a low $1/w$ (Fog density decreases).

For details, see the description of kmSetFogTable.

Argument:

fogDensity (input)

This argument is a coefficient of TSP Fog (scale factor).

Specify this argument as "kmSetFogDensity (0xFF09)".

Return Value

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.4.5 Setting the Fog Table

```
KMSTATUS kmSetFogTable(PKMFLOAT pfFogTable)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function registers the FogTable. A pointer to an array holding 128 different float values is passed via the argument. The fog table takes effect on the polygon for which FogTable is specified by VERTEXCONTEXT of $1/w = 0.0$ (infinite point) to 1.0 (depth = 1.0).

An element of FogTable that is 0.0 has an attenuation rate of, and an element that is 1.0 has the maximum attenuation rate.

FogTable consists of 128 elements with indexes 0 to 127.

The element of Index of FogTable specifies Fog density of the following position with a depth of ($1/w$ value):

Depth = $(\text{pow}(2.0, \text{Index} \gg 4) * (\text{float}) ((\text{Index} \& 0x0F) + 16) / 16.0f) / \text{FogDensity}$

Therefore, specify the density starting from the most distant point, in sequence.

Specify FogDensity with kmSetFogDensity.

Argument:

pfFogTable (input)

This argument specifies a pointer to an array of fog table values. The array consists of 256 different parameters.

Return value:

KMSTATUS_SUCCESS Set successfully

3.4.6 Setting the On-Chip Palette Mode

```
KMSTATUS kmSetPaletteMode(KMPALETTEMODE Palettemode)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function specifies an on-chip palette mode.

Argument:

PaletteMode (input)

KM_PALETTE_16BPP_ARGB1555	16-BPP mode, ARGB1555 format
KM_PALETTE_16BPP_RGB565	16-BPP mode, RGB565 format
KM_PALETTE_16BPP_ARGB4444	16-BPP mode, ARGB4444 format
KM_PALETTE_32BPP_ARGB8888	32-BPP mode, ARGB8888 format

Return value:

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.4.7 Setting the On-Chip Palette

```
KMSTATUS kmSetPaletteData(PKMPALETTE_DATA pPaletteTable)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function sets the on-chip palette used by the Palettize texture. A palette has a total of 1024 entries. The number of entries is the same regardless of whether the screen mode is 16 bpp or 32 bpp. Because a palette can be read at 1 clock/pixel in 16-bpp screen mode, the speed is higher than in 32-bpp mode (2 clocks/pixel).

With Palettized-4bpp, the 1024 entries are divided into 64 banks (1024 entries/16 colors = 64 banks). With Palettized-8bpp, the 1024 entries are divided into four banks (1024 entries/256 colors = 4 banks). The banks are not separated physically, each bank being created by calculating pointers to the 1024 entries.

The 4-bpp texture and 8-bpp texture can exist together in one scene, but the overlapping portion of the 1024 entries is shared. Changing the contents of a palette, therefore, affects both the 4-bpp and 8-bpp textures.

The bank of a palette can be specified in units of VERTEX (polygon). Specify a bank number by using the `PaletteBank` member of `KMVERTEXCONTEXT`. The entry that is actually used is selected as follows, depending on the palette bank number (`PaletteBank`) and index value of each pixel of the texture (`index_data`).

```
if (PixelFormat == 8BPP)
{
    palette_entry = (PaletteBank << 4) & 0x300 + index_data;
}

if (PixelFormat == 4BPP)
{
    palette_entry = (PaletteBank << 4) + index_data;
}
```

A value of 0 to 63 can be specified for `PaletteBank` in 4-bpp mode.

Also, 0 to 63 can be specified in 8-bpp mode, but only four types of values, 0 (0 to 15), 16 (16 to 31), 32 (32 to 47), and 48 (48 to 63), can be used in this mode because only the higher two bits of the six are valid.

Argument:

pPaletteTable (input)

This argument specifies a pointer to a palette setting array. The array is defined as follows:

```
KMPACKEDARGB PackdColor[1024];
```

Return value:

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.4.8 Setting the Border Color

```
KMSTATUS kmSetBorderColor(KMPACKEDARGB BorderColor)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function sets the color for borders (for portions outside the display screen).

Argument:

BorderColor (input)

This argument specifies a packed ARGB color.

Return value:

KMSTATUS_SUCCESS Set successfully

3.4.9 Registering the Rendering Parameter of the Background Plane

```
KMSTATUS kmSetBackGroundRenderState (
    PKMVERTEXCONTEXT pVertexContext)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function registers the GLOBALPARAMBUFFER, ISPPARAMBUFFER, TSPPARAMBUFFER, and TexturePARAMBUFFER members in KMVERTEXCONTEXT set by kmProcessVertexRenderState to the system as the rendering parameters of the background plane. When subsequently setting the background plane (kmSetBackGroundPlane), KMVERTEXCONTEXT specified here becomes valid. In VERTEXCONTEXT of Background, DSTBlendingMode must be zero.

To change the background plane, it is necessary to execute kmProcessVertexRenderState, kmSetBackGroundRenderState, then kmSetBackGroundPlane.

Argument:

pVertexContext (input)

This argument sets a pointer to context.

Return value:

KMSTATUS_SUCCESS Registering rendering parameters successful

3.4.10 Setting the Background Plane

```
KMSTATUS kmSetBackGroundPlane(PVOID pVertex[3],  
                               KMVERTEXTYPE VertexType,  
                               KMINT32 StructSize)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function registers a background plane. Before using this function, it is necessary to call `kmSetBackGroundRenderState`. If the Z value of BackgroundPlane is greater than the object to be displayed, the object may not be displayed (keep the value of $1/w$ less than the object to be displayed. The default $1/w$ value of a background plane is 0.001).

To change the background plane, it is necessary to execute `kmProcessVertexRenderState`, `kmSetBackGroundRenderState`, and `kmSetBackGroundPlane` in this order.

Arguments:

pVertex[3] (input)

This is a pointer to a vertex data structure that indicates coordinates on a background plane. For details, see the description of `kmSetVertex`.

VertexType (input)

This argument indicates the data type of vertex data. For details, see the description of `kmSetVertex`.

StructSize (input)

This argument indicates the data type size of vertex data. Specify it like `sizeof (KMVERTEX_01)` in accordance with the type used for the vertex data. For details, see the description of `kmSetVertex`.

Return value:

KMSTATUS_SUCCESS	Registering background plane successful
------------------	---

3.4.11 Setting Autosort Mode

```
KMSTATUS kmSetAutoSortMode(KMBOOLEAN fEnable)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function turns on/off translucent autosort mode.

Argument:

fEnable (input)

If it is TRUE, this argument specifies autosort mode for translucent planes. If it is FALSE, it emulates software-based sorting, which is the conventional sorting type.

Return value:

```
KMSTATUS_SUCCESS          Set successfully
```

3.4.12 Specifying Pixel-Unit Clipping

```
KMSTATUS kmSetPixelClipping (KMINT32   Xmin,
                              KMINT32   Ymin,
                              KMINT32   Xmax,
                              KMINT32   Ymax)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function sets pixel-unit clipping for rendering output to the frame buffer.

Arguments:

Xmin, Ymin, Xmax, Ymax (input)

These arguments specify the coordinates of the upper-left and lower-right corners of a clipping area in pixel units. "(Xmin, Ymin) - (Xmax, Ymax)" cannot be larger than the screen size. If screen mode is 24 bpp, coordinates specified for a clipping area must be even numbers; in other words, the clipping area can be specified only in two-pixel units. If they are not even, values that are 1 greater than specified are assumed for (Xmin, Ymin), and values that are 1 less than specified are assumed for (Xmax, Ymax).

Return values:

```
KMSTATUS_SUCCESS          Set successfully
KMSTATUS_INVALID_PARAMETER Invalid parameter
```

3.4.13 Specifying the Stride Size

```
KMSTATUS kmSetPixelClipping      (KMINT32      Xmin,  
                                KMINT32      Ymin,  
                                KMINT32      Xmax,  
                                KMINT32      Ymax)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function sets the stride size when the stride texture is used. The stride size must be a multiple of 32. The value that can be set is a multiple of 32 in the range of 32 to 992.

It is also necessary to call this API before executing kmProcessVertexRenderState to VERTEXCONTEXT of the polygon to which Cheap Shadow is to be applied.

Argument:

fIntensity (input)

This argument sets the luminance of the polygon entering the modifier volume. A value of 0.25, 0.5, or 0.75 can be input. If any other positive value is input, it is rounded to any of the above values.

If a negative value is input, the setting of CheapShadowMode is completed, and the normal 2-parameter polygon becomes valid.

Return value:

KMSTATUS_SUCCESS	Set successfully
KMSTATUS_INVALID_PARAMETER	Invalid parameter

3.4.14 Setting the CheapShadowMode

```
KMSTATUS kmSetCheapShadowMode(KMFLOAT fIntensity)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function selects the cheap (simple) shadow mode. After a value has been set by this function, all the ModifierVolume values are set in cheap shadow mode. Coexistence with two-parameter polygons is not allowed. To terminate CheapShadowMode, enter a negative number as the argument, then call this function.

It is also necessary to call this API before executing `kmProcessVertexRenderState` to `VERTEXCONTEXT` of the polygon to which Cheap Shadow is to be applied.

Argument:

fIntensity (input)

This argument sets the luminance of the polygon entering the modifier volume. A value of 0.25, 0.5, or 0.75 can be input. If any other positive value is input, it is rounded to any of the above values.

If a negative value is input, the setting of CheapShadowMode is completed, and the normal 2-parameter polygon becomes valid.

Return values:

KMSTATUS_SUCCESS	Set successfully
KMSTATUS_INVALID_PARAMETER	Invalid parameter

3.4.15 Specifying the Number of VsyncWait States

KMSTATUS kmSetWaitVsyncCount (KMINT32 nCount)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies how many times Vsync has been passed before the display is Flipped. This function is used to obtain a constant frame rate.

For example, Flip is executed at the second Vsync after rendering has been completed if `VwaitNum` is set to 2. In this case, even if rendering has been completed within 16.67 ms in NTSC mode, Flip is not executed at the next Vsync, but at the Vsync after that. If `VwaitNum` is set to 2 and if the time required for rendering is longer than 2Vsync, Flip is executed at the next Vsync after rendering has been completed.

The default value that is assumed if this function is not called is 1. Therefore, Flip is executed at the next Vsync after rendering has been completed.

Argument:

VwaitNum (input)

This argument specifies the number of Wait states of Vsync. Set a value of 1 or greater. A negative value is ignored even if specified.

Return value:

KMSTATUS_SUCCESS Set successfully

3.4.16 Forced Reset of Renderer

KMSTATUS kmResetRenderer(void)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function resets the rendering pipeline through software. It is used for forced reset if data in Strip cannot be fully drawn when Strip Buffer is used.

Argument:

none

Return value:

KMSTATUS_SUCCESS

Reset successfully

3.5 SETTING PARAMETERS OF EACH VERTEX

3.5.1 VERTEXCONTEXT

KAMUI has all parameters that can be set for each vertex (strip) in a KMVERTEXCONTEXT structure. An application can have and selectively use two or more KMVERTEXCONTEXT structures.

To do this, the application first allocates the KMVERCONTEXT structure and define member values. Next, KMVERTEXCONTEXT is completed by kmProcessVertexRenderState. Then, the structure is registered in the system by kmSetVertexRenderState. The finished structures can be switched just by executing kmSetVertexRenderState. To modify some members of a finished KMVERTEXCONTEXT structure, kmProcessVertexRenderState and kmSetVertexRenderState must be executed again.

A polygon influenced by a modifier volume requires two structures inside and outside the modifier volume. The application allocates two KMVERTEXCONTEXT structures and specifies the parameters inside the modifier volume in one structure and the parameters outside the modifier volume in the other structure. The parameters outside the modifier volume are registered in the system by kmProcessVertexRenderState and kmSetVertexRenderState. The parameters inside the modifier volume are registered in the system by kmProcessVertexRenderState and kmSetVertexRenderState.

When using VERTEXCONTEXT for the first time, all the parameters must be specified. Set all the flags for RenderState and define all the parameters. The operation is not guaranteed if any bit is undefined.

<KMVERTEXCONTEXT>

Structure holding parameters that can be specified for each vertex (strip). This structure has the following members.

```
typedef struct tagKMVERTEXCONTEXT
{
    KMDWORD                RenderState;           // Render Context

    // for Global Parameter
    KMPARAMTYPE             ParamType             // Parameter Type
    KMLISTTYPE              ListType              // List Type
    KMCOLORTYPE             ColorType             // Color Type
    KMUVFORMAT              UVFormat              // UV format

    // for ISP/TSP Instruction Word
    KMDEPTHMODE             DepthMode;            // Specified Depth Mode
    KMCULLINGMODE           CullingMode;          // Culling Mode
    KMSCREENCOORDINATION    ScreenCoordination;   // Screen Coordination
    KMSHADINGMODE           ShadingMode;          // Shading Mode
    KMMODIFIER              SelectModifier;       // Modifier Volume Valiant
    KMBOOLEAN               bZWriteDisable;       // Z Write Disable
    KMBOOLEAN               bDCalcExact;          // D-param calc
```

```
// for TSP Control Word
KMBLENDINGMODE          SRCBlendingMode;          // Source Blending Mode
KMBLENDINGMODE          DSTBlendingMode;          // Destination Blending Mode
KMBOOLEAN               bSRCSel;                  // Source Select
KMBOOLEAN               bDSTSel;                  // Destination Select
KMFOGMODE               FogMode;                  // Fogging
KMBOOLEAN               bUseSpecular;             // Specular Highlight
KMBOOLEAN               bUseAlpha;                // Alpha
KMBOOLEAN               bIgnoreTextureAlpha;      // Ignore Texture Alpha
KMFLIPMODE              FlipUV;                   // Flip U,V
KMCLAMPMODE             ClampUV;                   // Clamp U,V
KMFILTERMODE            FilterMode;               // Texture Filter
KMBOOLEAN               bSuperSample;             // Anisotropic Filter
KMDWORD                 MipMapAdjust;             // Mipmap D Adjust
KMTEXTURESHADINGMODE    TextureShadingMode;       // Texture Shading Mode
KMBOOLEAN               bColorClamp;              // Color Clamp
KMDWORD                 PaletteBank;              // Palette Bank

// for Texture Control Bits/Address
PKMSURFACEDESC          pTextureSurfaceDesc;      // Texture DESC Pointer

// Intensity FaceColor Setting
KMFLOAT                 fFaceColorAlpha;          // Face Color Alpha
KMFLOAT                 fFaceColorR;              // Face Color Red
KMFLOAT                 fFaceColorG;              // Face Color Green
KMFLOAT                 fFaceColorB;              // Face Color Blue

// Intensity Specular Highlight Setting
KMFLOAT                 fOffsetColorAlpha;         // Specular Alpha
KMFLOAT                 fOffsetColorR;             // Specular Red
KMFLOAT                 fOffsetColorG;             // Specular Green
KMFLOAT                 fOffsetColorB;             // Specular Blue

/* Variables Internally Used */
KMDWORD                 GLOBALPARAMBUFFER;        // Global Parameter Buffer
KMDWORD                 ISPPARAMBUFFER;           // ISP Parameter Buffer
KMDWORD                 TSPPARAMBUFFER;           // TSP Parameter Buffer
KMDWORD                 TexturePARAMBUFFER;       // TextureParameter Buffer
} KMVERTEXCONTEXT, *PKMVERTEXCONTEXT;

/* for ModifierInstruction */
KMDWORD                 ModifierInstruction;       /* ModifierInstruction*/
KMFLOAT                 fBoundingBoxXmin;          /* BoundingBoxXmin(ShadowVolume)*/
KMFLOAT                 fBoundingBoxYmin;          /* BoundingBoxYmin(ShadowVolume)*/
KMFLOAT                 fBoundingBoxXmax;          /* BoundingBoxXmax(ShadowVolume)*/
KMFLOAT                 fBoundingBoxYmax;          /* BoundingBoxYmax(ShadowVolume)*/
} KMVERTEXCONTEXT, *PKMVERTEXCONTEXT;
```

To modify some or all members, specify the type of the member to be modified in a RenderState structure member using the following status flag. At the same time, specify a value for the corresponding structure member. Then, execute `kmProcessVertexRenderState`.

The following status flags can be specified in RenderState.

KM_PARAMTYPE	0x00100000,	/* Parameter Type */
KM_LISTTYPE	0x00200000,	/* List Type */
KM_COLORTYPE	0x00400000,	/* Specified Color Format */
KM_UVFORMAT	0x00800000,	/* Specified UV Format */
KM_DEPTHMODE	0x00000001,	/* Z-value Comparison Mode Setting */
KM_CULLINGMODE	0x00000002,	/* Culling Mode Setting */
KM_SCREENCOORDINATION	0x00000004,	/* Coordinate System Setting */
KM_SHADINGMODE	0x00000008,	/* Texture Gouraud, Texture Flat, Non-Texture Gouraud */
KM_MODIFIER	0x00000010,	/* Modifier Volume Valiant No or A */
KM_ZWRITEDISABLE	0x00000020,	/* Z Write Disable or not */
KM_SRCBLENDINGMODE	0x00000040,	/* Blending Mode */
KM_DSTBLENDINGMODE	0x00000080,	/* Blending Mode */
KM_SRCSELECT	0x01000000,	/* SRC Blending Select */
KM_DSTSELECT	0x02000000,	/* DST Blending Select */
KM_FOGMODE	0x00000100,	/* Fog Non or Vertex or Table */
KM_USESPECULAR	0x00000200,	/* Specular Highlighted or not */
KM_USEALPHA	0x00000400,	/* Alpha Blended or not */
KM_IGNORETEXTUREALPHA	0x00000800,	/* Ignore Texture Alpha */
KM_FLIPUV	0x00002000,	/* Texture Flipping */
KM_CLAMPUV	0x00001000,	/* Texture Clamping */
KM_FILTERMODE	0x00004000,	/* Point-sample or Bilinear or Trilinear */
KM_SUPERSAMPLE	0x00008000,	/* Anisotropic Filter */
KM_MIPMAPDADJUST	0x00010000,	/* MipMap D Adjust */
KM_TEXTURESHADINGMODE	0x00020000,	/* Modulate, Decal Alpha, Modulate Alpha */
KM_COLORCLAMP	0x00040000	/* Color Clamping */
KM_PALETTEBANK	0x00080000	/* Palette Bank */
KM_DCALCEXACT	0x04000000	/* DCALC Exact */

The following values can be specified in the members:

ParamType

Specify a vertex parameter type.

One of the following values can be specified:

<u>KMPARAMTYPE</u>		
KM_POLYGON	= 0	// Normal triangular polygon
KM_MODIFIERVOLUME	= 1	// Modifier volume (shadow/light)
KM_SPRITE	= 2	// Sprite (tetragonal polygon)

ListType

Specify a list type.

One of the following values can be specified:

KMLISTTYPE

KM_OPAQUE_POLYGON	= 0	// Opaque polygon
KM_OPAQUE_MODIFIER	= 1	// Opaque modifier volume
KM_TRANS_POLYGON	= 2	// Translucent/transparent polygon
KM_TRANS_MODIFIER	= 3	// Translucent/transparent modifier volume

ColorType

Specify a vertex color format.

One of the following values can be specified:

KMCOLORTYPE

KM_PACKEDCOLOR	= 0	// 32bit ARGB packed color format
KM_FLOATINGCOLOR	= 1	// 32bit x 4 floating color format
KM_INTENSITY	= 2	// Intensity format
KM_INTENSITY_PREV_FACE_COL	= 3	//Intensity format (FaceColor registered immediately before is used as is: Cannot be used with ARCl.)

UVFormat

Specify a U/V coordinate format. The 32-bit UV expresses the U, V coordinates in a 32-bit Float format conforming to IEEE754. The 16-bit UV expresses a value with an accuracy of the 32-bit UV with the lower 16 bits of the Mantissa deleted.

One of the following values can be specified:

KMUVFORMAT

KM_32BITUV	= 0	// 32bit KMFLOAT format
KM_16BITUV	= 1	// 16bit KMFLOAT format

DepthMode

Specify one of the following for Z value comparison.

KMDEPTHMODE

KM_IGNORE	= 0
KM_LESS	= 1
KM_EQUAL	= 2
KM_LESSEQUAL	= 3
KM_GREATER	= 4
KM_NOTEQUAL	= 5
KM_GREATEREQUAL	= 6
KM_ALWAYS	= 7

CullingMode

No culling, clockwise culling, counterclockwise culling, or small polygon culling can be specified. One of the following values can be specified:

<u>KMCULLINGMODE</u>		
KM_NOCULLING	= 0	// No culling
KM_CULLSMALL	= 1	// Small polygon culling
KM_CULLCCW	= 2	// Counterclockwise culling
KM_CULLCW	= 3	// Clockwise culling

When setting KM_CULLSMALL, it is necessary to execute kmSetCullingRegister for global setting. If KM_CULLCCW or KM_CULLCW is specified, small polygon Culling is performed at the same time.

ScreenCoordination

Select the screen coordinate system (1/w) or projection coordinate system (w).

<u>KMSCREENCOORDINATION</u>		
KM_SCREEN	= 0	// Screen coordinate system (1/w)
KM_PROJECTIVE	= 1	// Projection coordinate system (w)= 1

bDCalcExact

KMDCALCEXACT

If TRUE is set, the D parameter added by CLX1 is calculated accurately. With ARC1, a symptom where the selection of MipMap differs between the upper-right and lower-left of the screen is observed. This symptom can be eliminated by accurately calculating CLX1. However, the operation speed may drop because the calculation is time-consuming. This flag of ARC1 is meaningless. If FALSE is set, the D parameter is calculated in the same manner as ARC1.

ShadingMode

Specify one of four shading modes, which are combinations of presence or absence of texture and flat or Gouraud shading.

<u>KMSHADINGMODE</u>		
KM_NOTEXTUREFLAT ⁴	= 0	// No texture, flat shading
KM_NOTEXTUREGOURAUD	= 1	// No texture, Gouraud shading
KM_TEXTUREFLAT	= 2	// Texture, flat shading
KM_TEXTUREGOURAUD	= 3	// Texture, Gouraud shading

When the KM_TEXTUREFLAT is specified, the first and second color are neglected. (for the first polygon, only 3rd color is valid.)

4. ARC1 does not have this mode. Never try to use this mode in the ARC1 environment.

SelectModifier

Specify whether the a polygon (two-parameter polygon or a polygon to which Cheap Shadow is to be applied in Cheap Shadow mode) is influenced by a modifier volume. If it is specified that the polygon is to be influenced by a modifier volume, use the Vertex structure for a two-parameter polygon.

KMMODIFIER

```
KM_NOMODIFIER          = 0          // Not used
KM_MODIFIER_A          = 1          // Influenced by modifier volume A
```

ARC1 and HOLLY support modifier volume A only.

bZWriteDisable

When TRUE is specified, modification of the Z value is disabled.

ARC1 does not have this function.

SRCBlendingMode, DSTBlendingMode

These members correspond to the D3D blending mode. One of the following values can be specified.

KMBLENDINGMODE**KM_BOTHINVSRCALPHA,**

The source blending parameter is set as $(1 - \alpha_s, 1 - \alpha_s, 1 - \alpha_s, 1 - \alpha_s)$. The destination blending parameter is set as $(\alpha_d, \alpha_d, \alpha_d, \alpha_d)$. When this value is set for SRCBlendingMode, DSTBlendingMode is overridden. When this value is set for DSTBlendingMode, SRCBlendingMode is overridden.

KM_BOTHSRCALPHA,

The source blending parameter is set as $(\alpha_s, \alpha_s, \alpha_s, \alpha_s)$. The destination blending parameter is set as $(1 - \alpha_d, 1 - \alpha_d, 1 - \alpha_d, 1 - \alpha_d)$. When this value is set for SRCBlendingMode, DSTBlendingMode is overridden. When this value is set for DSTBlendingMode, SRCBlendingMode is overridden.

KM_DESTALPHA,

The blending parameter is set as $(\alpha_d, \alpha_d, \alpha_d, \alpha_d)$.

KM_DESTCOLOR,

The blending parameter is set as $(\alpha_d, R_d, G_d, B_d)$.

KM_INVDESTALPHA,

The blending parameter is set as $(1 - \alpha_d, 1 - \alpha_d, 1 - \alpha_d, 1 - \alpha_d)$.

KM_INVDESTCOLOR,

The blending parameter is set as $(1 - \alpha_d, 1 - R_d, 1 - G_d, 1 - B_d)$.

KM_INVSRCALPHA,

The blending parameter is set as $(1 - \alpha_s, 1 - \alpha_s, 1 - \alpha_s, 1 - \alpha_s)$.

KM_INVSRCOLOR,

The blending parameter is set as $(1 - \alpha_s, 1 - R_s, 1 - G_s, 1 - B_s)$.

KM_SRCALPHA,

The blending parameter is set as (α_s , α_s , α_s , α_s).

KM_SRCCOLOR,

The blending parameter is set as (α_s , R_s , G_s , B_s).

KM_ONE,

The blending parameter is set as (1, 1, 1, 1).

KM_ZERO

The blending parameter is set as (0, 0, 0, 0).

Note: (α_s , R_s , G_s , B_s) represents a source color, and (α_d , R_d , G_d , B_d) represents a destination color.

Set `DSTBlendingMode` to zero for `VERTEXCONTEXT` for the background plane (see the description of `kmSetBackGroundRenderState`).

bSRCSel

The contents of the second accumulation buffer are used as the source color.

bDSTSel

The contents of the second accumulation buffer are used as the destination color.

FogMode

Specify a fog mode. In the ARC1 and HOLLY environments, two fog modes can be used: In `FogTable` mode, fogging is performed with reference to a table according to the depth value; In `FogVertex` mode, fog parameters are specified at transformation to the specular α channel. One of the following values can be specified:

KMFOGMODE

KM_FOGTABLE

KM_FOGVERTEX

KM_NOFOG

bUseSpecular

Specify whether specular highlight (offset color) is used. When `TRUE` is specified, specular is used. When using specular highlight, the Vertex structure for specifying the offset color must be used.

bUseAlpha

Specify whether the α bit in the shading color is enabled. When `TRUE` is specified, the bit is enabled.

bIgnoreTextureAlpha

When `TRUE` is specified, the α bit in the texture data is ignored. The transparency information included in the texture is ignored.

FlipUV

Specify whether a repeated texture is flipped.

One of the following values can be specified. If FlipUV and ClampUV are specified at the same time, the specification of FlipUV is ignored.

KMFLIPMODE

```
KM_NOFLIP          // Not flipped
KM_FLIP_V          // Flipped in the V-coordinate direction
KM_FLIP_U          // Flipped in the U-coordinate direction
KM_FLIP_UV         // Flipped in the U-coordinate and V-coordinate directions
```

ClampUV

Specify a texture clamp.

One of the following values can be specified. If FlipUV and ClampUV are specified at the same time, the specification of FlipUV is ignored.

KMCLAMPMODE

```
KM_NOCLAMP         // Not clamped
KM_CLAMP_V         // Clamped in the V-coordinate direction
KM_CLAMP_U         // Clamped in the U-coordinate direction
KM_CLAMP_UV        // Clamped in the U-coordinate and V-coordinate directions
```

FilterMode

Specify a texture filter mode. The point-sample, bilinear, or trilinear filter mode can be selected. However, Bilinear is assumed if Trilinear is specified with the ARC1. One of the following values can be specified:

KMFILTERMODE

```
KM_POINT_SAMPLE    (= 0)
KM_BILINEAR         (= 1)
KM_TRILINEAR        (= 2)
```

bSuperSample

When TRUE is specified, the quad-speed super sampling filter (anisotropic filter) is selected.

This selection causes the quality of the texture filter to be improved.

MipMapAdjust

The D parameter calculation for MIPMAP level selection is multiplied by a coefficient. By this multiplication, aliasing can be adjusted. The four low-order bits of the value are effective. The four bits form floating point data consisting of a two-bit integer portion and a two-bit fractional portion.

Example	4 low-order bits	Coefficient to be used
	0000	Invalid (Do not use this value)
	0001	0.25
	0100	1.0
	1111	3.75

Usually, specify 0100 (1.0).

TextureShadingMode

Specify a D3D texture blending mode. This cannot be used when the BumpMap texture is selected. One of the following values can be specified:

KMTEXTURESHADINGMODE

KM_MODULATE	= 1
KM_DECAL_ALPHA	= 2
KM_MODULATE_ALPHA	= 3

KM_MODULATE

The texture color is multiplied by the color resulting from shading. Texture α is replaced by shading α .

Pixel Color = Shading_{RGB} × Texture_{RGB} + Offset_{RGB}

Pixel α = Texture α

KM_DECAL_ALPHA

The texture color is blended with the shading color, according to texture α .

Pixel Color = (Texture_{RGB} × Texture α) + {Shading_{RGB} × (1- Texture α)} + Offset_{RGB}

Pixel α = Shading α

KM_MODULATE_ALPHA

The texture color is multiplied by the shading color. Texture α is multiplied by shading α .

Pixel Color = (Texture_{RGB} × Shading_{RGB}) + Offset_{RGB}

Pixel α = Shading α × Texture α

bColorClamp

Specify whether a color is clamped. When TRUE is specified, a pixel color is clamped to the clamp value specified by kmSetColorClampValue. ARC1 does not support this function.

PaletteBank

Specify a palette bank number. This value is valid only when palettized texture is specified. This value is valid only when palettized texture is specified. The value that can be specified is 0 to 63 in the Palettized -4bpp mode. It is also 0 to 63 in the Palettized -8bpp mode, but only four types of values, 0 (0 to 15), 16 (16 to 31), 32 (32 to 47), and 48 (48 to 63), can be used in this mode because only the higher two bits of the six are valid (for details, see the description of `kmSetPaletteData`).

ARC1 does not support this function.

pTextureSurfaceDesc

Specify a pointer to the surface structure of the texture surface.

To change only the texture without changing the other parameters of `VERTEXCONTEXT`, only change the texture address of `pTextureSurfaceDesc` and call `kmProcessVertexRenderState` and `kmSetVertexRenderState`. If `ShadingMode` is specified as `KM_TEXTUREFLAT` or `KM_TEXTUREGOURAUD`, `pTextureSurfaceDesc` is loaded into the system each time `kmProcessVertexRenderState` is called.

If `NULL` is specified as `pTextureSurfaceDesc`, the texture address is not loaded even if `ShadingMode` is `KM_TEXTUREFLAT` or `KM_TEXTUREGOURAUD`. If the texture is not determined and if other parameters are to be set in advance, specify `NULL` as `pTextureSurfaceDesc`.

ModifierInstruction

Specify the type of the polygon data to be registered when a modifier volume is registered. Specify any of the following:

`KM_MODIFIER_INCLUDE_FIRST_POLY` Indicates the first polygon of the Inclusion modifier volume.

`KM_MODIFIER_EXCLUDE_FIRST_POLY` Indicates the first polygon of the Exclusion modifier volume.

`KM_MODIFIER_INCLUDE_LAST_POLY` Indicates the first polygon of the Inclusion modifier volume.

`KM_MODIFIER_EXCLUDE_LAST_POLY` Indicates the first polygon of the Exclusion modifier volume.

`KM_MODIFIER_NORMAL_POLY` Indicates a polygon other than those above.

This member does not have to be specified for anything other than a modifier volume.

fboundingBoxXmin, fBoundingBoxYmin**fboundingBoxXmax, fBoundingBoxYmax**

Specify an area on the screen in which the effect of a modifier volume is valid. The effect of a modifier volume becomes valid only in the range specified here. By setting this value as necessary, the rendering speed can be improved.

This member does not have to be specified for anything other than modifier volume.

This member is valid only with the ARC1.

3.5.2 Setting Rendering Parameters for Each Vertex

```
KMSTATUS kmProcessVertexRenderState(  
    PKMVERTEXCONTEXT pVertexContext)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Implemented

Note: In the ARC1 environment, some functions are restricted.

Explanation: This function specifies rendering parameters that are used for each vertex (strip). The function performs preprocessing for registering to the system the values set by pVertexContext. When this function is called, the global parameter, combined ISP/TSP instruction word, TSP control word, texture control bits are generated from the KMVERTEXCONTEXT members and set in the GLOBALPARAMBUFFER, ISPPARAMBUFFER, TSPPARAMBUFFER, and TexturePARAMBUFFER members of the same KMVERTEXCONTEXT. When kmSetVertexRenderState is called after this function, the values are registered in the system.

When using VERTEXCONTEXT for the first time, the values of all the members of VERTEXCONTEXT must be specified (initialized). Set all the flags for RenderState and define all the parameters. The operation is not guaranteed if any bit is undefined.

When using kmSetUserClipping or kmSetStripLength, call these APIs before executing kmProcessVertexRenderState of the polygon.

Argument:

pVertexContext (input)

Specify a pointer to the context.

Return values:

KMSTATUS_SUCCESS	Set successfully
KMSTATUS_INVALIDSETTING	Illegal mode setting

3.5.3 Registering Rendering Parameters for Each Vertex

```
KMSTATUS kmSetVertexRenderState (
    PKMVERTEXCONTEXT pVertexContext)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function registers to the system the GLOBALPARAMBUFFER, ISPPARAMBUFFER, TSPPARAMBUFFER, and TexturePARAMBUFFER members of KMVERTEXCONTEXT specified by kmProcessVertexRenderState. In subsequent vertex strip registrations, KMVERTEXCONTEXT specified here will be used.

Argument:

pVertexContext (input)

Specify a pointer to the context.

Return value:

```
KMSTATUS_SUCCESS Successful registration of rendering parameters
```

3.5.4 Registering Rendering Parameters of a Modifier Volume

```

KMSTATUS          kmSetModifierRenderState(
                    PKMVERTEXCONTEXT pVertexContext)

```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function registers to the system the TSPPARAMBUFFER and TexturePARAMBUFFER members of KMVERTEXCONTEXT specified by kmProcessVertexRenderState as the second rendering parameters of the polygon using a modifier volume. In subsequent vertex strip registrations of a polygon using a modifier volume, KMVERTEXCONTEXT specified here will be used as the second rendering parameters.

When using kmSetUserClipping or kmSetStripLength, call these APIs before executing kmProcessVertexRenderState of the polygon.

The following members of the KMVERTEXCONTEXT structure are invalid.

(The settings by kmSetVertexRenderState are valid.)

```

// for Global Parameter
KMPARAMTYPE          ParamType          // Parameter Type
KMLISTTYPE            ListType           // List Type
KMCOLORTYPE          ColorType          // Color Type
KMUVFORMAT            UVFormat           // UV Format

// for ISP/TSP Instruction Word
KMDEPTHMODE          DepthMode;          // Specified Depth Mode
KMCULLINGMODE         CullingMode;        // Culling Mode
KMSCREENCOORDINATION  ScreenCoordination; // Screen Coordination
KMSHADINGMODE         ShadingMode;        // Shading Mode
KMMODIFIER            SelectModifier;     // Modifier Volume Valiant
KMBOOLEAN            bZWriteDisable;     // Z Write Disable
/* for ModifierInstruction */
KMDWORD              ModifierInstruction; // ModifierInstruction*/
KMFLOAT              fBoundingBoxXmin;   /* BoundingBoxXmin(ShadowVolume)*/
KMFLOAT              fBoundingBoxYmin;   /* BoundingBoxYmin(ShadowVolume)*/
KMFLOAT              fBoundingBoxXmax;   /* BoundingBoxXmax(ShadowVolume)*/
KMFLOAT              fBoundingBoxYmax;   /* BoundingBoxYmax(ShadowVolume)*/

```

Argument:

pVertexContext (input)

Specify a pointer to the rendering context.

Return value:

KMSTATUS_SUCCESS Successful registration of rendering parameters

3.5.5 Setting Global Clipping

KMSTATUS kmSetGlobalClipping(KMINT32 nWidth, KMINT32 nHeight)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies a global clipping area. Rendering is performed only in the area determined by the 0,0 origin, Width, and Height.

Arguments:

nWidth, nHeight (input)

Specify a multiple of the tile size. To specify a 128 x 64 area in the HOLLY environment, in which 32 x 32 tiles are used, for example, set Width to 4 and Height to 2. For the ARC1 environment, in which 32 x 8 tiles are used, KAMUI internally converts the parameters specified in the HOLLY environment (by multiplying Height by 4).

Return values:

KMSTATUS_SUCCESS Set successfully
KMSTATUS_ERROR Setting ended in failure

3.5.6 Setting a Strip Length

```
KMSTATUS kmSetStripLength(KMLISTTYPE ListType,
                          KMSTRIPLENGTH StripLength)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function specifies the length of strip, which is the number of polygons. The hardware internally divides the input vertex strip data into strips of the specified length. This set value is embedded in a global parameter and is reported to the hardware. Therefore, this API must be issued before `kmProcessVertexRenderState` is called for `VERTEXCONTEXT` of the subjected polygon. In subsequent vertex strip registrations, the strip length specified here will be used.

In the ARC1 environment, the hardware always divides the data into strips of strip length 1. (Internally, when the first vertex data is registered by the specified list type after the function is issued, the global parameter `Strip_Len` is specified.)

Arguments:

ListType (input)

Specify a vertex data list.

One of the following values can be specified:

KMLISTTYPE

```
KM_OPAQUE_POLYGON          = 0          // Opaque polygon
KM_OPAQUE_MODIFIER         = 1          // Opaque modifier volume
KM_TRANS_POLYGON           = 2          // Translucent/transparent polygon
KM_TRANS_MODIFIER          = 3          // Translucent/transparent modifier volume
```

StripLength (input)

One of the following values can be specified:

KMSTRIPLENGTH

```
KM_STRIP_01                = 0          // Divided into strips of strip length 1
KM_STRIP_02                = 1          // Divided into strips of strip length 2
KM_STRIP_04                = 2          // Divided into strips of strip length 4
KM_STRIP_06                = 3          // Divided into strips of strip length 6
```

Return value:

```
KMSTATUS_SUCCESS           Set successfully
```


3.5.7 Setting a User Clipping Area

```

KMSTATUS kmSetUserClipping(
    KMLISTTYPE ListType,
    KMUSERCLIPMODE ClipMode,
    KMINT32      Xmin, KMINT32      Ymin,
    KMINT32      Xmax, KMINT32      Ymax)

```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function specifies a user clipping area. This set value is embedded in a global parameter and is reported to the hardware. Therefore, this API must be issued before `kmProcessVertexRenderState` is called for `VERTEXCONTEXT` of the subjected polygon. In subsequent vertex strip registrations, the user clipping area specified here will be used.

(Internally, the control parameter of user tile clip is put in the vertex data buffer of the specified list type.)

Arguments:

ListType (input)

Specify a vertex data list.

One of the following values can be specified:

```

KMLISTTYPE
KM_OPAQUE_POLYGON      = 0          // Opaque polygon
KM_OPAQUE_MODIFIER      = 1          // Opaque modifier volume
KM_TRANS_POLYGON        = 2          // Translucent/transparent polygon
KM_TRANS_MODIFIER        = 3          // Translucent/transparent modifier volume

```

ClipMode (input)

One of the following values can be specified:

```

KMUSERCLIPMODE
KM_USERCLIP_DISABLE     = 0          // The user clip is disabled.
KM_USERCLIP_RESERVE      = 1          // This should not be specified.
KM_USERCLIP_INSIDE       = 2          // The user clip is enabled only inside the
                                   specified area.
KM_USERCLIP_OUTSIDE      = 3          // The user clip is enabled only outside the
                                   specified area.

```

Xmin, Ymin, Xmax, Ymax (input)

Specify the coordinates of the top left and bottom right corners of the user clip area. Specify the values in tiles (1 = 32 pixels).

Just the six low-order bits of Xmin and Xmax are valid. As for Ymin and Ymax, just the four low-order bits are valid.

Return value:

KMSTATUS_SUCCESS

Set successfully



Caution: kmSetUserClipping must not be issued between kmStartVertexStrip and kmSetVertex.

3.5.8 Setting a User Clipping Area (Direct Specification)

```
KMSTATUS kmSetUserClippingDirect(KMUSERCLIPMODE ClipMode,
                                  KMINT32 Xmin, KMINT32 Ymin,
                                  KMINT32 Xmax, KMINT32 Ymax)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function specifies a user clipping area. In subsequent vertex strip registrations, the user clipping area specified here will be used. For the difference between this function and `kmSetUserClipping`, see Section 3.6.

(Internally, the control parameter of the user tile clip is directly written to the tile accelerator (hardware).)

Arguments:

ClipMode (input)

One of the following values can be specified:

```
KMUSERCLIPMODE
KM_USERCLIP_DISABLE    = 0    // User clipping is disabled.
KM_USERCLIP_RESERVE    = 1    // This should not be specified.
KM_USERCLIP_INSIDE     = 2    // User clipping is enabled only inside the specified area.
KM_USERCLIP_OUTSIDE    = 3    // User clipping is enabled only outside the specified area.
```

Xmin, Ymin, Xmax, Ymax (input)

Specify the coordinates of the upper-left and lower-right corners of the user clip area. Specify the values in tiles (1 = 32 pixels).

Just the six low-order bits of `Xmin` and `Xmax` are valid. As for `Ymin` and `Ymax`, just the four low-order bits are valid.

Return value:

```
KMSTATUS_SUCCESS      Successful setting
```



Caution: `kmSetUserClippingDirect` must not be issued between `kmStartVertexStripDirect` and `kmSetVertexDirect`.

3.5.9 Direct Rewrite Mode for Rendering Status

Explanation: After rendering conditions are set in a rendering context and `kmProcessVertexRenderState` is executed to perform preprocessing, `kmSetVertexRenderState` is usually to register the global current rendering state as the rendering status. The global value is utilized as the rendering status when a vertex is registered. In some cases, the global value of the previous rendering state may have to be modified a little. It may be very inefficient to reset all values and to preprocess the rendering context. KAMUI provides a service function for rewriting a particular rendering state.

IRIS+ARC1	COSMOS+ARC1	Holly
Partially Implemented	Partially Implemented	Implemented

Note: In the ARC1 environment, some functions are restricted.

Arguments:

The arguments to be used for this function are the same as those specified for the rendering context. The function restriction in the ARC1 environment is also the same as those for the rendering context.

```
KMSTATUS kmChangeContextColorType(KMCOLORTYPE Color)
```

Explanation: This function modifies the vertex color format.

```
KMSTATUS kmChangeContextDepthMode(KMDEPTHMODE DepthMode)
```

Explanation: This function selects a Z-value comparison mode.

```
KMSTATUS kmChangeContextCullingMode(KMCULLINGMODE CullingMode)
```

Explanation: This function selects a culling mode.

```
KMSTATUS kmChangeContextZWriteDisable (  
                                KMBOOLEAN fZwriteDisable)
```

Explanation: This function enables or disables the modification of the Z value.

```
KMSTATUS kmChangeContextSRCBlendMode(KMBLENDINGMODE SRCBlend)
```

Explanation: This function selects a source blending mode.

```
KMSTATUS kmChangeContextDSTBlendMode(KMBLENDINGMODE DSTBlend)
```

Explanation: This function selects a destination blending mode.

```
KMSTATUS kmChangeContextFogMode(KMFOGMODE FogMode)
```

Explanation: This function selects a fog mode.

```
KMSTATUS kmChangeContextFlipUV(KMFLIPMODE FlipMode)
```

Explanation: This function sets a texture flip.

KMSTATUS kmChangeContextClampUV(KMCLAMPMODE ClampMode)

Explanation: This function sets a texture clamp.

KMSTATUS kmChangeContextFilterMode(KMFILTERMODE FilterMode)

Explanation: This function sets a texture filter.

KMSTATUS kmChangeContextSuperSample(KMBOOLEAN bEnable)

Explanation: When the argument is TRUE, an anisotropic filter is used.

KMSTATUS kmChangeContextTextureShadingMode(
KMTEXTURESHADINGMODE TexShadingmode)

Explanation: This function specifies a texture blending mode.

KMSTATUS kmChangeContextColorClamp(KMBOOLEAN fColorClamp)

Explanation: When the argument is TRUE, the color is clamped according to the value of the global register.

KMSTATUS kmChangeContextPaletteBank(KMDWORD Bank)

Explanation: This function changes the current palette bank.

3.6 RECORDING VERTEX DATA

3.6.1 Recording Vertex Data

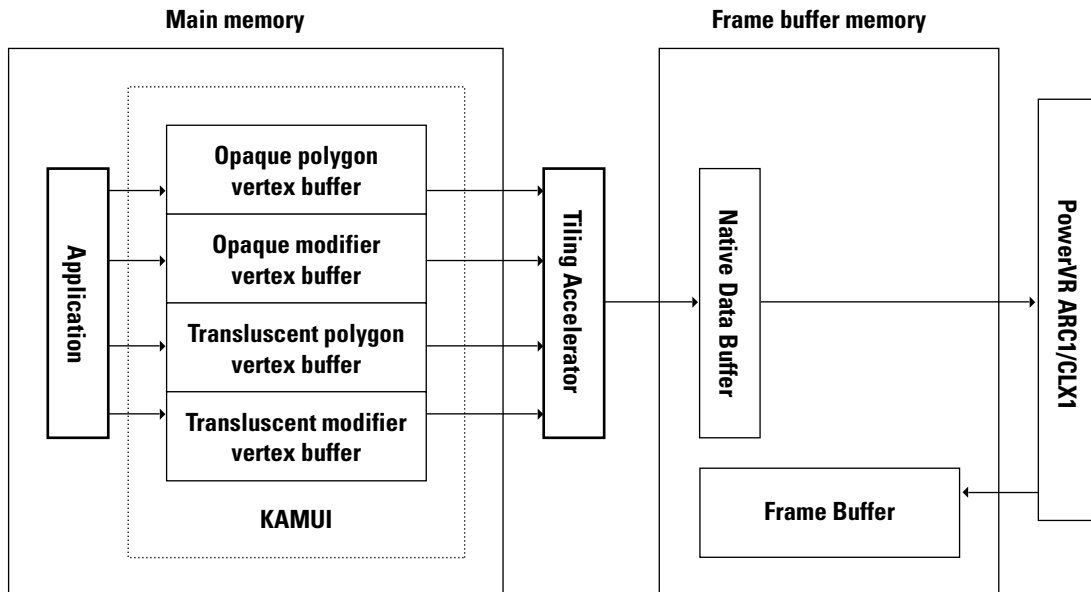
KAMUI allocates buffers for recording vertex data. Drawing is performed by storing necessary vertex data in the buffers and issuing a rendering command. All vertex data is defined as strip-type data. When specifying a single polygon, specify strip length 1.

First, `kmCreateVertexBuffer` is executed to allocate four vertex data buffers (main memory) and a tile accelerator output buffer (native data buffer: frame buffer). These four vertex data buffers are:

- Buffer for opaque polygon (opaque polygon)
- Buffer for opaque modifier volume (opaque modifier)
- Buffer for translucent/translucent polygon (translucent polygon)
- Buffer for translucent/translucent modifier volume (translucent modifier)

All vertex data is stored in any of the four buffers.

Then, the parameters of common to scenes and the parameters common to vertices to be recorded are specified. (See Sections 3.4 and 3.5.)



To record a vertex, first execute `KmStartVertexStrip` to write global parameters into the data buffer. Then, execute `kmSetVertex` any number of times to record the vertex data. All vertex data defined by `kmSetVertex` is assumed to be a single strip. To record a new strip, issue `KmStartVertexStrip` again. At the end of a scene, issue `kmRender`.

The vertex data of a single strip is defined as indicated below:

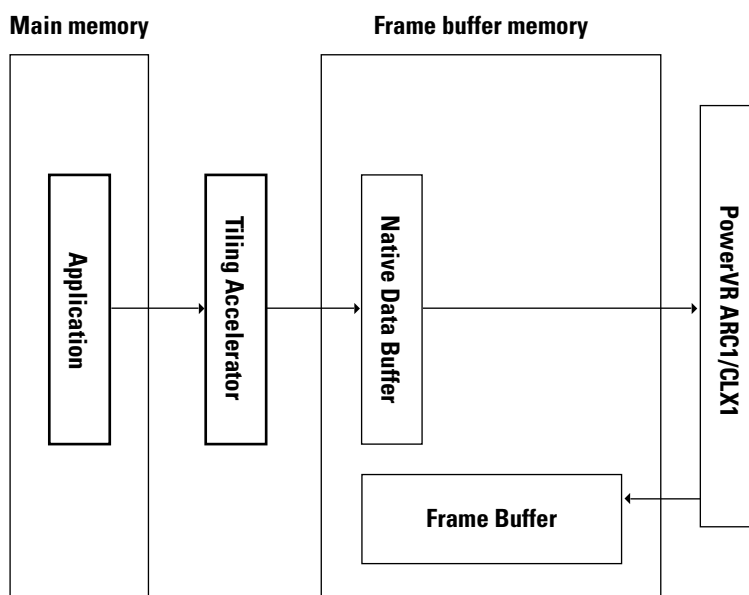
```
kmStartVertexStrip();    // Start of strip
do{
    .....
    kmSetVertex(pVertex, VertexType, sizeof(VertexType));
    // Registration of one vertex
} while(...)
```



Caution: kmSetUserClipping must not be issued between kmStartVertexStrip and kmSetVertex. If no kmSetVertex is issued after kmStartVertexStrip, the operation may be adversely affected.

The vertex data can also be directly written into the hardware (tile accelerator) without allocating four vertex data buffers in main memory.

To do this, first execute kmCreateTABuffer to allocate a tile accelerator output buffer (native data buffer) in a frame buffer. Then, specify the general parameters of the scene and the parameters common to vertices to be recorded. (See Sections 3.4 and 3.5.)



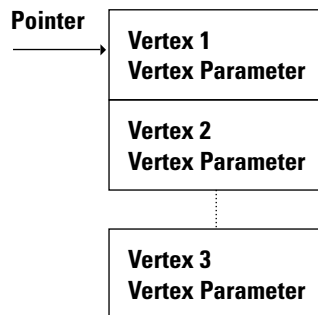
To record a vertex, first execute KmStartVertexStripDirect to write global parameters into the hardware. Then, execute kmSetVertexDirect any number of times to write the vertex data directly into the hardware. To record a new strip, issue KmStartVertexStripDirect again. When the registration of data of a specific vertex type (Opaque Polygon, Opaque Modifier, Translucent Polygon, Translucent Modifier) has been completed, kmSetEndOfListDirect is issued. At the end of a scene, issue kmRenderDirect

The global parameters and vertex data of a single scene to be input to the hardware (tile accelerator) must be classified by vertex type (opaque polygon, opaque modifier, translucent polygon, translucent modifier). If the vertex data of a single type is input twice (the data of opaque polygon and translucent polygon is input, then the data of opaque polygon is input again, for example), the vertex data input earlier becomes invalid.

To specify user clipping, `kmSetUserClippingDirect` must be used instead of `kmSetUserClipping`. `kmSetUserClippingDirect` must not be issued between `kmStartVertexStripDirect` and `kmSetVertexDirect`. If no `kmSetVertexDirect` is issued after `kmStartVertexStripDirect`, the operation may be adversely affected.

3.6.2 Vertex Data Structure

Vertex data consists of the Global Parameter followed by as many Vertex Parameter blocks as the number of vertices (strip). Several strips are combined to create a scene of one screen. To register a strip, create the Global Parameter with `kmStartVertexStrip`, as described earlier, then issue `kmSetVertex` for each vertex. At least three Vertex Parameters are necessary. The end of the strip is specified by Parameter Control Word at the beginning of the Vertex Parameter.



Select either of the following Parameter Control Words:

```
KM_VERTEXPARAM_NORMAL      = 0xE0000000    // Normal vertex data
KM_VERTEXPARAM_ENDOFSTRIP   = 0xF0000000    // Vertex data at end of strip
```

If Parameter Control Word of the vertex data at the end of the strip is not `KM_VERTEXPARAM_ENDOFSTRIP`, the operation is not guaranteed.

If `KM_TEXTUREFLAT` is specified as `ShadingMode` of `VERTEXCONTEXT`, the color data of the first and second vertices of the vertex strip is invalid.

3.6.3 Vertex Parameter

Type0 (Non-Textured, Packed Color)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	Base Color
0x14	Reserved
0x18	Reserved
0x1C	Reserved

Type2 (Non-Textured, Intensity)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	Base Intensity
0x14	Reserved
0x18	Reserved
0x1C	Reserved

Type3 (Packed Color)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U
0x14	V
0x18	Base Color
0x1C	Offset Color

Type1 (Non-Textured, Floating Color)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	Base Color Alpha
0x14	Base Color R
0x18	Base Color G
0x1C	Base Color B

Type4 (Packed Color, 16bit UV)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U V
0x14	Reserved
0x18	Base Color
0x1C	Offset Color

Type5(Floating Color)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U
0x14	V
0x18	Reserved
0x1C	Reserved
0x20	Base Color Alpha
0x24	Base Color R
0x28	Base Color G
0x2C	Base Color B
0x30	Offset Color Alpha
0x34	Offset Color R
0x38	Offset Color G
0x3C	Offset Color B

Type6 (Floating Color, 16bit UV)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U V
0x14	Reserved
0x18	Reserved
0x1C	Reserved
0x20	Base Color Alpha
0x24	Base Color R
0x28	Base Color G
0x2C	Base Color B
0x30	Offset Color Alpha
0x34	Offset Color R
0x38	Offset Color G
0x3C	Offset Color B

Type7 (Intensity)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U
0x14	V
0x18	Base Intensity
0x1C	Offset Intensity

Type8 (Intensity, Compact UV)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U V
0x14	Reserved
0x18	Base Intensity
0x1C	Offset Intensity

Type9 (Non-Textured, Packed Color, with Two Volumes)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	BaseColor 0
0x14	BaseColor 1
0x18	Reserved
0x1C	Reserved

Type10 (Non-Textured, Intensity, with Two Volumes)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	Base Intensity 0
0x14	Base Intensity 1
0x18	Reserved
0x1C	Reserved

Type11 (Textured, Packed Color, with Two Volumes)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U0
0x14	V0
0x18	Base Color 0
0x1C	Offset Color 0
0x20	U1
0x24	V1
0x28	Base Color 1
0x2C	Offset Color 1
0x30	Reserved
0x34	Reserved
0x38	Reserved
0x3C	Reserved

Type12 (Textured, Packed Color, 16bit UV, with Two Volumes)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U0 V0
0x14	Reserved
0x18	Base Color 0
0x1C	Offset Color 0
0x20	U1 V1
0x24	Reserved
0x28	Base Color 1
0x2C	Offset Color 1
0x30	Reserved
0x34	Reserved
0x38	Reserved
0x3C	Reserved

Type13 (Textured, Intensity, with Two Volumes)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U0
0x14	V0
0x18	Base Intensity 0
0x1C	Offset Intensity 0
0x20	U1
0x24	V1
0x28	Base Intensity 1
0x2C	Offset Intensity 1
0x30	Reserved
0x34	Reserved
0x38	Reserved
0x3C	Reserved

Type14 (Textured, Intensity, 16-bit UV, with Two Volumes)	
0x00	Parameter Control Word
0x04	X
0x08	Y
0x0C	Z
0x10	U0 V0
0x14	Reserved
0x18	Base Intensity 0
0x1C	Offset Intensity 0
0x20	U1 V1
0x24	Reserved
0x28	Base Intensity 1
0x2C	Offset Intensity 1
0x30	Reserved
0x34	Reserved
0x38	Reserved
0x3C	Reserved

Sprite type 0 (Type15)	
0x00	Parameter Control Word
0x04	AX
0x08	AY
0x0C	AZ
0x10	BX
0x14	BY
0x18	BZ
0x1C	CX
0x20	CY
0x24	CZ
0x28	DX
0x2C	DY
0x30	Reserved
0x34	Reserved
0x38	Reserved
0x3C	Reserved

Sprite type 1 (Type16)	
0x00	Parameter Control Word
0x04	AX
0x08	AY
0x0C	AZ
0x10	BX
0x14	BY
0x18	BZ
0x1C	CX
0x20	CY
0x24	CZ
0x28	DX
0x2C	DY
0x30	Reserved
0x34	AU AV
0x38	BU BV
0x3C	CU CV

Shadow Volume (Type 17)	
0x00	Parameter Control Word
0x04	AX
0x08	AY
0x0C	AZ
0x10	BX
0x14	BY
0x18	BZ
0x1C	CX
0x20	CY
0x24	CZ
0x28	Reserved
0x2C	Reserved
0x30	Reserved
0x34	Reserved
0x38	Reserved
0x3C	Reserved

The corresponding vertex data structures (18 types) are defined as follows:

```
typedef struct tagKMVERTEX_n (n=00...17)
{
    KMDWORD ParamControlWord;
    KMFLOAT fX;
    KMFLOAT fY;
    union{
        KMFLOAT fZ;
        KMFLOAT fInvW;
    } u;
    .....
} KMVERTEX_n, *PKMVERTEX_n;
```

In the IRIS plus ARC1 environment, the words reserved in the parameter structure are not actually present in the structure. The SH4-based system inserts a padding word in the header file.

The in-structure parameter types and names are listed below:

KMFLOAT fX, fY, fZ(fInvW)	Vertex coordinates
KMFLOAT fU, fV	Texture UV
KMDWORD dwUV	Packed texture UV
KMFLOAT fBaseIntensity	Base intensity
KMFLOAT fBaseAlpha	Base color *
KMFLOAT fBaseRed	Base color red
KMFLOAT fBaseGreen	Base color green
KMFLOAT fBaseBlue	Base color blue
KMPACKEDARGB uRGB	Packed color ARGB
KMFLOAT fModifierU, fModifierV	Modifier texture UV
KMDWORD dwModifierUV	Modifier packed texture UV
KMFLOAT fModifierBaseIntensity	Modifier base intensity
KMFLOAT fModifierBaseAlpha	Modifier base color *
KMFLOAT fModifierBaseRed	Modifier base color red
KMFLOAT fModifierBaseGreen	Modifier base color green
KMFLOAT fModifierBaseBlue	Modifier base color blue
KMPACKEDARGB uModifierRGB	Modifier packed color ARGB
KMFLOAT fAX, fAY, fAZ...fDY	Sprite/modifier coordinates
KMDWORD dwUVA...dwUVC	Sprite packed texture UV

3.6.4 Setting a Modifier Volume

This section explains how to set a modifier volume.

Setting a two-parameter volume

(Polygon influenced by modifier volume)

- Setting of VERTEXCONTEXT

Set `KM_MODIFIER_A` as the `SelectModifier` member. Set the two parameters (CONTEXT) with "`kmSetVertexRenderState`" and "`kmSetModifierRenderState`".

- Format of vertex data

Define the vertex data with one of `KMVERTEX_09` to `KMVERTEX_14`.

- When using Cheap Shadow mode

When issuing `kmProcessVertexRenderState` to the VERTEXCONTEXT of the polygon to be influenced by Cheap Shadow, it is necessary to set Cheap Shadow mode (`kmSetCheapShadowMode`).

Setting of modifier volume

- A modifier volume is registered to dedicated VertexBuffer.

Secure Buffer for Opaque-Modifier and Trans-Modifier by `kmCreateVertexBuffer`.

Setting of VERTEXCONTEXT

The polygons constituting a modifier volume must be classified into three types by using VERTEXCONTEXT: the first, the last, and others.

Specify `KM_MODIFIER_INCLUDE_FIRST_POLY` or `KM_MODIFIER_EXCLUDE_FIRST_POLY` as the `ModifierInstruction` member for the first polygon. For the last polygon, specify `KM_MODIFIER_INCLUDE_LAST_POLY` or `KM_MODIFIER_EXCLUDE_LAST_POLY` as the `ModifierInstruction` member.

Specify `KM_MODIFIER_NORMAL_POLY` as the `ModifierInstruction` member for the other polygons. In any case, specify `KM_MODIFIERVOLUME` as the `ParamType` member and `KM_OPAQUE_MODIFIER` or `KM_TRANS_MODIFIER` as `ListType`. ARC1 sets the screen coordinates at which the modifier volume is validated for each member of `fBoundingBoxXmin`, `fBoundingBoxYmin`, `fBoundingBoxXmax`, and `fBoundingBoxYmax` (this setting is not necessary with CLX1).

- Format of vertex data

Define vertex data by `KMVERTEX_17`.

3.6.5 Allocating a Vertex Data Buffer

```

KMSTATUS kmCreateVertexBuffer(PKMVERTEXBUFFDESC pBufferDesc,
    KMINT32 OpaquePolygonBuffer,
    KMINT32 OpaqueModifierBuffer,
    KMINT32 TransPolygonBuffer,
    KMINT32 TransModifierBuffer)

```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function allocates four buffers (double buffers) for recording vertex data. The function also allocates a tile accelerator output buffer (native data buffer), which commensurate in size with the four buffers, in the frame buffer memory. The data of one scene is placed in the four buffers by `kmStartVertexStrip`, `kmSetVertex`, and `kmRender`, then all of the corresponding vertex data is sent to the hardware in a batch. One application cannot use `kmCreateVertexBuffer` and `kmCreateTABuffer` simultaneously.

Arguments:

pBufferDesc (I/O)

PKMVERTEXBUFFDESC-type pointer to the vertex buffer descriptor. The first addresses of the four buffers allocated here are returned to the members of the vertex buffer descriptor.

OpaquePolygonBuffer (input)

Size of the vertex data buffer in which the data of opaque polygons is stored.

Given as the number of DWORDs (number of bytes/4).

OpaqueModifierBuffer (input)

Size of the vertex data buffer in which the data of opaque modifier volumes is stored.

Given as the number of DWORDs (number of bytes/4).

TransPolygonBuffer (input)

Size of the vertex data buffer in which the data of translucent/transparent polygons is stored.

Given as the number of DWORDs (number of bytes/4).

TransModifierBuffer (input)

Size of the vertex data buffer in which the data of translucent/transparent modifier volumes is stored.

Given as the number of DWORDs (number of bytes/4).

Return values:

KMSTATUS_SUCCESS	Vertex data buffers successfully allocated
KMSTATUS_NOT_ENOUGH_MEMORY	Insufficient memory space

3.6.6 Releasing Vertex Data Registration Buffers

```
KMSTATUS kmDiscardVertexBuffer(PKMVERTEXBUFFDESC pBufferDesc)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function releases the four buffers (double buffer) for vertex data registration that have been allocated by `kmCreateVertexBuffer`. It also releases the tile accelerator output buffer (Native Data Buffer) that was allocated at the same time.

Argument:

pBufferDesc (I/O)

This argument inputs a pointer to a vertex buffer descriptor of `PKMVERTEXBUFFDESC` type.

Return value:

`KMSTATUS_SUCCESS` Vertex data buffer released successfully

3.6.7 Allocating the Tile Accelerator Output Buffer

```
KMSTATUS kmCreateTABuffer(
    PKMVERTEXBUFFERDESC pBufferDesc,
    KMINT32 TABuffer)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function allocates a tile accelerator output buffer (native data buffer) in the frame buffer memory. Vertex data is directly written into the tile accelerator by `kmStartVertexStripDirect`, `kmSetVertexDirect`, and `kmRenderDirect`. One application cannot use `kmCreateVertexBuffer` and `kmCreateTABuffer` simultaneously.

Argument:

pBufferDesc (input)

Pointer to a vertex buffer descriptor of `PKMVERTEXBUFFDESC` type.

TABuffer (input)

Size of the buffer that stores the tile accelerator output (PowerVR native data) (bytes)

Return values:

`KMSTATUS_SUCCESS` Vertex data buffer successfully allocated
`KMSTATUS_NOT_ENOUGH_MEMORY` Insufficient memory space

3.6.8 Writing Global Parameters in a Buffer

KMSTATUS kmStartVertexStrip (PKMVERTEXBUFFDESC pBufferDesc)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function generates global parameters from KMVERTEXCONTEXT specified by kmSetVertexRenderState and kmSetModifierRenderState and writes the parameters in any of the four vertex data buffers. The vertex data buffer to be written is determined by the ListType member of KMVERTEXCONTEXT.

Argument:

pBufferDesc (input)

PKMVERTEXBUFFDESC-type pointer to the vertex buffer descriptor

Return values:

KMSTATUS_SUCCESS Global parameters successfully written

KMSTATUS_NOT_ENOUGH_MEMORY Insufficient vertex data buffer space



Caution: The format of the global parameters written here depends on KMVERTEXCONTEXT specified by kmSetVertexRenderState. If the format of the global parameters written here does not match the format of vertex parameters written by following kmSetVertex, the operation may be adversely affected.

<This function is defined as a macro of inline expansion because high-speed execution is demanded.>

3.6.9 Directly Writing Global Parameters

```
KMSTATUS kmStartVertexStripDirect(void)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function generates global parameters from KMVERTEXCONTEXT specified by kmSetVertexRenderState and kmSetModifierRenderState and directly writes the parameters into the hardware (tile accelerator).

The global parameters and vertex data of a single scene to be input to the hardware (tile accelerator) must be classified by vertex type (opaque polygon, opaque modifier, translucent polygon, translucent modifier).

Argument:

None

Return value:

KMSTATUS_SUCCESS Global parameters successfully written



Caution: The format of the global parameters written here depends on KMVERTEXCONTEXT specified by kmSetVertexRenderState. If the format of the global parameters written here does not match the format of vertex parameters written by following kmSetVertex, the operation may be adversely affected.

<This function is defined as a macro of inline expansion because high-speed execution is demanded.>

3.6.10 Writing Vertex Data in a Buffer

```
KMSTATUS kmSetVertex(PKMVERTEXBUFFDESC pBufferDesc,
                    PVOID pVertex,
                    KMVERTEXTYPE VertexType,
                    KMINT32 StructSize)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function writes one vertex data specified by `pVertex` to any of four types of vertex data buffers. The vertex data buffer to which the data is to be written is determined by the `ListType` member of `KMVERTEXCONTEXT`. If Parameter Control Word of the vertex data at the end of the strip is not `KM_VERTEXPARAM_ENDOFSTRIP`, the operation is not guaranteed.

Arguments:

pBufferDesc (input)

`PKMVERTEXBUFFDESC`-type pointer to the vertex buffer descriptor

pVertex (input)

Pointer to the vertex data structure

VertexType (input)

Vertex data type. Specify one of the following:

```
KM_VERTEXTYPE_00    // Vertex data Type 0
KM_VERTEXTYPE_01    // Vertex data Type 1
KM_VERTEXTYPE_02    // Vertex data Type 2
KM_VERTEXTYPE_03    // Vertex data Type 3
KM_VERTEXTYPE_04    // Vertex data Type 4
KM_VERTEXTYPE_05    // Vertex data Type 5
KM_VERTEXTYPE_06    // Vertex data Type 6
KM_VERTEXTYPE_07    // Vertex data Type 7
KM_VERTEXTYPE_08    // Vertex data Type 8
KM_VERTEXTYPE_09    // Vertex data Type 9
KM_VERTEXTYPE_10    // Vertex data Type 10
KM_VERTEXTYPE_11    // Vertex data Type 11
KM_VERTEXTYPE_12    // Vertex data Type 12
KM_VERTEXTYPE_13    // Vertex data Type 13
KM_VERTEXTYPE_14    // Vertex data Type 14
KM_VERTEXTYPE_15    // Vertex data Type 15
KM_VERTEXTYPE_16    // Vertex data Type 16
KM_VERTEXTYPE_17    // Vertex data Type 17
```

StructSize (input)

Vertex data type size. Specify a size according to the selected vertex data type in the size of (KMVERTEX01) format.



Caution: If the format of the vertex parameters written here does not match the format of global parameters specified by preceding `kmSetVertexRenderState`, the operation may be adversely affected.

Return values:

KMSTATUS_SUCCESS	Vertex data successfully written
KMSTATUS_NOT_ENOUGH_MEMORY	Insufficient vertex data buffer space

<This function is defined as a macro of inline expansion because high-speed execution is demanded.>

3.6.11 Directly Writing Vertex Data

```
KMSTATUS kmSetVertexDirect(
    PVOID pVertex,
    KMVERTEXTYPE VertexType,
    KMINT32 StructSize)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: The function directly writes the single-vertex data specified by `pVertex` into the hardware (tile accelerator).

The global parameters and vertex data of a single scene to be input to the hardware (tile accelerator) must be classified by vertex type (Opaque Polygon, Opaque Modifier, Translucent Polygon, Translucent Modifier).

If Parameter Control Word of the vertex data at the end of the strip is not `KM_VERTEXPARAM_ENDOFSTRIP`, the operation is not guaranteed.

Arguments:

pVertex (input)

Pointer to the vertex data structure

VertexType (input)

Vertex data type. Specify one of the following:

```
KM_VERTEXTYPE_00    // Vertex data Type 0
KM_VERTEXTYPE_01    // Vertex data Type 1
KM_VERTEXTYPE_02    // Vertex data Type 2
KM_VERTEXTYPE_03    // Vertex data Type 3
KM_VERTEXTYPE_04    // Vertex data Type 4
KM_VERTEXTYPE_05    // Vertex data Type 5
KM_VERTEXTYPE_06    // Vertex data Type 6
KM_VERTEXTYPE_07    // Vertex data Type 7
KM_VERTEXTYPE_08    // Vertex data Type 8
KM_VERTEXTYPE_09    // Vertex data Type 9
KM_VERTEXTYPE_10    // Vertex data Type 10
KM_VERTEXTYPE_11    // Vertex data Type 11
KM_VERTEXTYPE_12    // Vertex data Type 12
KM_VERTEXTYPE_13    // Vertex data Type 13
KM_VERTEXTYPE_14    // Vertex data Type 14
KM_VERTEXTYPE_15    // Vertex data Type 15
KM_VERTEXTYPE_16    // Vertex data Type 16
KM_VERTEXTYPE_17    // Vertex data Type 17
```

StructSize (input)

Vertex data type size. Specify a size according to the selected vertex data type in the size of(KMVERTEX01) format.



Caution: If the format of the vertex parameters written here does not match the format of global parameters specified by preceding kmSetVertexRenderState, the operation may be adversely affected.

Return value:

KMSTATUS_SUCCESS

Vertex data successfully written

<This function is defined as a macro of inline expansion because high-speed execution is demanded.>

3.6.12 Reporting the End of Vertex Registration

KMSTATUS kmSetEndOfListDirect(VOID)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function directly reports the end of a vertex list of a specific type to the hardware (tile accelerator). Global parameters and vertex data must be classified by the type of vertex (Opaque Polygon, Opaque Modifier, Translucent Polygon, Translucent Modifier) and input to the hardware (tile accelerator) in the same scene. The end of the type of the currently registered vertex is reported by issuing `kmSetEndOfListDirect` at the end of each of these four types of vertices.

This function is not necessary when vertices are registered by using `kmSetVertex`.

Argument:

None

Return Value:

KMSTATUS_SUCCESS

Reported successfully

3.6.13 Notifying the End of Vertex Data Writing (When the Data Is Written into Buffers)

KMSTATUS kmRender(VOID)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function notifies KAMUI that all vertex data of a single scene has been written into the four vertex data buffers.

KAMUI starts rendering for the back buffer.

(Internally, DMA sends the contents of the four vertex data buffers and the End Of List control parameter into the hardware (tile accelerator) in succession.)

Argument:

None

Return value:

KMSTATUS_SUCCESS

Successful notification

3.6.14 Notifying the End of Vertex Data Writing (When Data Is Directly Written)

```
KMSTATUS kmRenderDirect(VOID)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function notifies KAMUI that all vertex data of a single scene has been written into the hardware (tile accelerator).

KAMUI starts rendering for the back buffer.

Argument:

None

Return value:

KMSTATUS_SUCCESS Successful notification

3.6.15 Rendering into the Texture Memory (When Data Is Written into Buffers)

```
KMSTATUS kmRenderTexture(PKMSURFACEDESC pTextureSurface)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: The function notifies KAMUI that all vertex data of a single scene has been stored in the four vertex data buffers.

KAMUI starts rendering for a specified texture.

(Internally, DMA sends the contents of the four vertex data buffers and the End Of List control parameter into the hardware (tile accelerator) in succession.)

Usually, when 640 × 480 rendering is performed, a 1024 × 512 texture surface is specified as the rendering destination. As a result, the result of rendering is written to the UV coordinates (0.0f, 0.0f)-(0.625f, 0.9375f) of this texture (the top line on the screen is written to the bottom line of the texture). If a texture surface less than the screen resolution is specified as the rendering destination surface, the upper-left part of the screen is rendered to the texture. For example, if rendering is performed to a 256 × 256 texture surface where the screen resolution is 640 × 480, the upper-left part (0,0)-(255, 255) of the screen is written to the texture.

When using this API, make sure that BPP of the frame buffer is the same as BPP of the texture at the rendering destination by using `kmSetDisplayMode`. Otherwise, the performance will drop. Note that the texture surface specified here must be of `RECTANGLE/STRIDE` type.

Argument:

pTextureSurface (input)

Texture of which rendering result is stored

Return values:

KMSTATUS_SUCCESS Successful notification

KMSTATUS_INVALID_TEXTURE Invalid texture specified

3.6.16 Rendering into the Texture Memory (When Data Is Directly Written)

```
KMSTATUS kmRenderTextureDirect(
    PKMSURFACEDESC pTextureSurface)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function notifies KAMUI that all vertex data of a single scene has been written into the hardware (tile accelerator).

KAMUI starts rendering for a specified texture.

Usually, when 640 x 480 rendering is performed, a 1024 x 512 texture surface is specified as the rendering destination. As a result, the result of rendering is written to the UV coordinates (0.0f, 0.0f)-(0.625f, 0.9375f) of this texture (the top line on the screen is written to the bottom line of the texture). If a texture surface less than the screen resolution is specified as the rendering destination surface, the upper-left part of the screen is rendered to the texture. For example, if rendering is performed to a 256 x 256 texture surface where the screen resolution is 640 x 480, the upper-left part (0,0)-(255, 255) of the screen is written to the texture.

When using this API, make sure that BPP of the frame buffer is the same as BPP of the texture at the rendering destination by using `kmSetDisplayMode`. Otherwise, the performance will drop. Note that the texture surface specified here must be of `RECTANGLE / STRIDE` type.

Argument:

pTextureSurface (output)

Texture of which rendering result is stored

Return values:

KMSTATUS_SUCCESS Successful notification

KMSTATUS_INVALID_TEXTURE Invalid texture specified

3.6.17 Specifying a Modifier Volume List

```
KMSTATUS kmUseAnotherModifier(  
                                KMLISTTYPE kmModifierListType)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function uses the Modifier specified by input parameter `kmModifierListType` as another Modifier.

This API only rewrites the pointer to OPAQUE-MODIFIER and TRANS-MODIFIER of the region array. If `KM_OPAQUE_MODIFIER` is specified and TRANS-MODIFIER object data is registered, the data of OPAQUE-MODIFIER is overwritten (the converse holds true).

Argument:

kmModifierListType (input)

This argument specifies the usage of the modifier volume list. Specify it in either of the following ways:

KM_OPAQUE_MODIFIER

Also uses Opaque Modifier as Trans Modifier.

KM_TRANS_MODIFIER

Also uses Trans Modifier as Opaque Modifier.

Return values:

<code>KMSTATUS_SUCCESS</code>	Set successfully
<code>KMSTATUS_INVALID_LIST_TYPE</code>	Setting failed

3.6.18 Obtaining Current Writing Position of VertexBuffer

```
KMDWORD kmGetCurrentVertexOffset ( KMLISTTYPE ListType )
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function returns the number of 32-bit words to which vertex data is stored in VertexBuffer of the currently specified list type.

By using the value obtained by this function in combination with kmChangeVertexOffset or kmChangeVertexPCW, the value of VertexBuffer can be changed.

Argument:

ListType (input)

```
KM_OPAQUE_POLYGON      = 0      // Opaque polygon
KM_OPAQUE_MODIFIER      = 1      // Opaque modifier volume
KM_TRANS_POLYGON        = 2      // Translucent/transparent polygon
KM_TRANS_MODIFIER        = 3      // Translucent/transparent modifier volume
```

Return value:

The current writing position of the specified list type is returned. The value is a 32-bit word offset from the first pointer of VertexBuffer of the specified list type (i.e., currently used capacity of VertexBuffer of specified list type).

3.6.19 Changing Current Writing Position of VertexBuffer

```
KMSTATUS kmChangeVertexOffset (
                                KMLISTTYPE ListType,
                                KMDWORD VertexOffset )
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function changes the current writing position of VertexBuffer of the specified list type.

Arguments:

ListType (input)

```
KM_OPAQUE_POLYGON      = 0      // Opaque polygon
KM_OPAQUE_MODIFIER      = 1      // Opaque modifier volume
KM_TRANS_POLYGON        = 2      // Translucent/transparent polygon
KM_TRANS_MODIFIER       = 3      // Translucent/transparent modifier volume
```

VertexOffset (input)

This argument specifies the writing position of the specified list type. The value obtained by KmGetCurrentVertexOffset is used.

Return values:

```
KMSTATUS_SUCCESS        Set successfully
KMSTATUS_INVALID_VALUE  A value that must not be specified is specified
```

3.6.20 Direct Rewriting of Vertex Control Word

```
KMSTATUS kmChangeVertexPCW (
    KMLISTTYPE ListType,
    KMDWORD VertexPCW,
    KMDWORD IncPtr)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function writes a value specified by VertexPCW to the current writing position of VertexBuffer of the specified list type. In this case, the pointer of VertexBuffer is incremented by IncPtr. It is used to abort a strip after strip definition of a triangle is finished. Exercise care when using this function because it may destroy matching in VertexBuffer. This function must not be used when a Direct function is used for Vertex definition.

Arguments:

ListType (input)

```
KM_OPAQUE_POLYGON      = 0          // Opaque polygon
KM_OPAQUE_MODIFIER      = 1          // Opaque modifier volume
KM_TRANS_POLYGON        = 2          // Translucent/transparent polygon
KM_TRANS_MODIFIER        = 3          // Translucent/transparent modifier volume
```

VertexPCW (input)

This argument specifies ParameterControlWord for vertex definition.

```
KM_VERTEXPARAM_NORMAL      = 0xE0000000    // Normal vertex data
KM_VERTEXPARAM_ENDOFSTRIP   = 0xF0000000    // Vertex data at end of strip
```

IncPtr (input)

This argument specifies how much the pointer of VertexBuffer advances after the parameter specified by VertexPCW has been written. Usually, specify the size of VertexType because it is used like `sizeof(...)/4`.

Return value:

```
KMSTATUS_SUCCESS          Set successfully
```

3.6.21 Flushing Opaque VertexBuffer

KMSTATUS kmFlushVertexBuffer (KMLISTTYPE ListType)

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Partially Implemented	Implemented

Note: COSMOS+ARC1 only supports Opaque.

Explanation: This function flushes `VertexBuffer` of the specified list type. Internally, DMA transfer of the specified list is started. This function is used to save the capacity of `VertexBuffer` used for Opaque list by using the fact that the list is sent from Opaque with the CLX1 and by sending the list little by little. This function must not be used with a `VertexDirect` function. If the previous DMA transfer has not been completed when this function is called, waiting for the completion of DMA takes place. Make sure that waiting does not occur when using this function.

Argument:

ListType (input)

KM_OPAQUE_POLYGON	= 0	// Opaque polygon
KM_OPAQUE_MODIFIER	= 1	// Opaque modifier volume
KM_TRANS_POLYGON	= 2	// Translucent/transparent polygon
KM_TRANS_MODIFIER	= 3	// Translucent/transparent modifier volume _

Return value:

KMSTATUS_SUCCESS	Set successfully
------------------	------------------

3.7 CALLBACK FUNCTIONS AND CALLBACK AUXILIARY FUNCTIONS

KAMUI can specify callback functions that modify rendering conditions and do other operation at a particular timing. The functions are called at particular events (end of rendering, for example), irrespective of the normal processing flow



Caution: If you want to use Callback Functions, please contact SEGA Tech Support Hotline at 650/802-1719. There are some limitations in keeping the console stable when using callback functions (processing time, register limitations, etc.).

3.7.1 Specifying a Rendering End Callback Function

```
KMSTATUS kmSetEORCallback(PKMCALLBACKFUNC pEORCallback,
                          PVOID pCallbackArguments)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies the callback function to be called at the end of rendering.

Code the callback function in the following format:

```
VOID EORCallbackFunc(PVOID pCallbackArguments);
pCallbackArguments (input): Pointer to the parameter set at the specification
```

Arguments:

pEORCallback (input)

Pointer to the function to be called at the end of rendering

If NULL is specified, the callback function is canceled.

pCallbackArguments (input)

Pointer to argument to be passed to the function called at callback

Return value:

```
KMSTATUS_SUCCESS                      Successful specification
```

3.7.2 Specifying a V-Sync Callback Function

```
KMSTATUS kmSetVSyncCallback(PKMCALLBACKFUNC pVSyncCallback,  
                             PVOID pCallbackArguments)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies the callback function to be called at an entry into the vertical flyback segment (Vsync).

Code the callback function in the following format:

```
VOID VSyncCallbackFunc(PVOID pCallbackArguments);  
    pCallbackArguments (input): Pointer to the parameter set at the specification
```

Arguments:

pVSyncCallback (input)

Pointer to the function to be called at an entry into Vsync

If NULL is specified, the callback function is canceled.

pCallbackArguments (input)

Pointer to argument to be passed to the function called at callback

Return value:

KMSTATUS_SUCCESS	Successful specification
------------------	--------------------------

3.7.3 Specifying an H-Sync Interrupt Callback Function

```
KMSTATUS kmSetHSyncCallback(PKMCALLBACKFUNC pHSyncCallback,  
                             PVOID pCallbackArguments)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies the callback function to be called at an entry into the horizontal flyback segment (Hsync). Code the callback function in the following format:

```
VOID HSyncCallbackFunc(PVOID pCallbackArguments);  
    pCallbackArguments (input): Pointer to the parameter set at the specification
```

Arguments:

pHSyncCallback (input)

Pointer to the function to be called at an entry into Hsync

If NULL is specified, the callback function is cancelled.

pCallbackArguments (input)

Pointer to argument to be passed to the function called at callback. The line number specified by KmSetHSyncLine is not passed. Hold the value specified by KmSetHSyncLine in the area specified by this pointer. Alternatively, obtain the current scanline count using KmGetCurrentScanline().

Return value:

```
KMSTATUS_SUCCESS                Successful specification
```

3.7.4 Setting the H-Sync Interrupt Line

```
KMRESULT kmSetHSyncLine(KMINT32 nInterruptLine)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies the display line on which an interrupt is caused.

Argument:

nInterruptLine (input)

Specify the line on which an interrupt is caused. Specify a value within the range of 0 to 240/480.

Return values:

KMSTATUS_SUCCESS Set successfully

3.7.5 Reading the Current H-Sync Line

```
KMRESULT kmGetCurrentScanline(PKMINT32 pScanline)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function reads the current H-Sync line. Zero is returned for the first line of the display area and a negative value is returned for a line of an area other than the display area. The negative value is counted up to -1 which indicates one line before the first line of the display area.

Argument:

pScanline (input)

Pointer to KMINT32 where the current H-Sync line is stored

Return value:

KMSTATUS_SUCCESS Set successfully

3.7.6 Specifying a Texture Memory Overflow Callback Function

```
KMSTATUS kmSetTexOverflowCallback(  
    PKMCALLBACKFUNC pTexOverflowCallback,  
    PVOID pCallbackArguments)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function registers a callback function that is called when a texture surface is to be allocated by `kmCreateTextureSurface` or `kmCreateCombinedTextureSurface`.

Code the callback function in the following format:

```
VOID TexOverflowCallbackFunc(PVOID pCallbackArguments);  
    pCallbackArguments (input): Pointer to the parameter set at the specification
```

Arguments:

pTexOverflowCallback (input)

Pointer to the callback function to be called at texture overflow

If `NULL` is specified, the callback function is canceled.

pCallbackArguments (input)

Pointer to argument to be passed to the function called at callback

Return value:

`KMSTATUS_SUCCESS` Successful specification

3.7.7 Specifying a Strip Buffer Overrun Callback Function

```
KMSTATUS kmSetStripOverRunCallback (  
PKMCALLBACKFUNC pStripOverRunCallback,  
PVOID pCallbackArguments)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function specifies the callback function to be called when the rendering of the next strip is not completed during the display period of the vertical dimension of StripBuffer.

Code the callback function in the following format:

```
VOID StripOverRunCallbackFunc(PVOID pCallbackArguments);
```

pCallbackArguments (input):

Pointer to the parameter set at the specification

Arguments:

pStripOverRunCallback (input)

Pointer to the callback function

If NULL is specified, the callback function is canceled.

pCallbackArguments (input)

Pointer to argument to be passed to the function called at callback

Return value:

KMSTATUS_SUCCESS	Successful specification
------------------	--------------------------

3.7.8 Specifying a Vertex Data Transfer End Callback Function

```
KMSTATUS kmSetEndOfVertexCallback (
    PCALLBACKFUNC pEndOfVertexCallback,
    PVOID pCallbackArguments)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function specifies the callback function to be called at the end of transfer of the data of one scene from KAMUI to the rendering hardware.

Code the callback function in the following format:

```
VOID EndOfVertexCallbackFunc(PVOID pCallbackArguments);
```

pCallbackArguments (input):

Pointer to the parameter set at the specification

Arguments:

pEndOfVertexCallback (input)

Pointer to the callback function

If NULL is specified, the callback function is canceled.

pCallbackArguments (input)

Pointer to argument to be passed to the function called at callback

Return value:

KMSTATUS_SUCCESS Successful specification

3.8 OTHER FUNCTIONS

3.8.1 Stopping the Frame Buffer Display

KMSTATUS kmStopDisplayFrameBuffer (VOID)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function stops the frame buffer display. The function just causes the CRT controller to stop display and does not change the frame buffer status.

Argument:

None

Return value:

KMSTATUS_SUCCESS Success

3.8.2 Obtaining the Version Information

KMSTATUS kmGetVersionInfo(PKMVERSIONINFO pVersionInfo)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function obtains the version information of the library. For the contents of the version information structure, see the structure list.

Argument:

pVersionInfo (output)

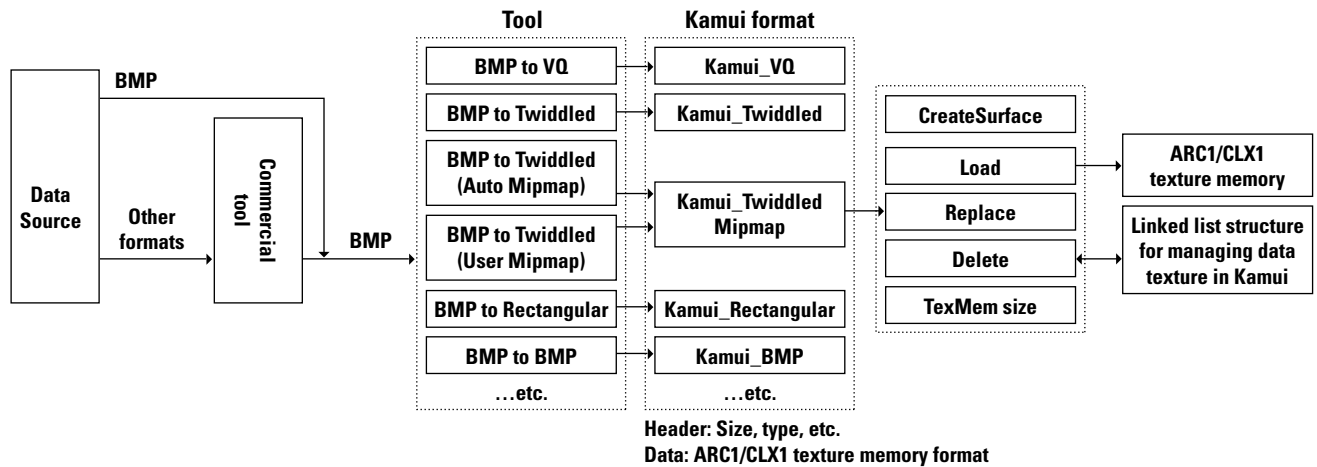
Indicates the pointer to the KMVERSIONINFO structure allocated in advance.

Return value:

KMSTATUS_SUCCESS Success

3.9 TEXTURE HANDLING FUNCTIONS OF KAMUI

The figure below shows the texture flow with KAMUI.



The texture control function of KAMUI transfers texture in the image format on texture memory. The texture must be converted beforehand by a tool into the KAMUI texture format having the image on texture memory.

For the details of the texture format, see Chapter 6.

3.9.1 Loading Texture Data

```
KMSTATUS kmLoadTexture(PKMSURFACEDESC pSurfaceDesc,  
                        PKMDWORD pTexture,  
                        KMBOOLEAN bAutoMipMap,  
                        KMBOOLEAN bDither)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function loads the texture on main memory specified by `pTexture` into the texture memory area allocated by `kmCreateTextureSurface`.

The format and size of the texture to be read are identified by the surface descriptor specified by `pSurfaceDesc`. If the actual format and size of the texture are different from the contents of the surface descriptor specified by `pSurfaceDesc`, the display is illegal.

Arguments:

pSurfaceDesc (input)

Texture surface allocated by `kmCreateTextureSurface`/`kmCreateCombinedTextureSurface`

pTexture (input)

Pointer to the pixel data portion of the texture in main memory. The address specified for this pointer is the first address of the texture file of KAMUI texture format + 16. Specify an address aligned with a four-byte boundary (16 bytes of the header portion of KAMUI are skipped).

bAutoMipMap (input)

Specify whether MIPMAP is automatically generated. When `TRUE` is specified, MIPMAP is automatically generated. If `TRUE` is specified, the pixel data must have `KM_TEXTURE_BMP` as a category code of texture type and be a square texture of 512 x 512 texels or less.

In this case, the read texture is converted into `KM_TEXTURE_TWIDDLED_MM` format. Therefore, the type of the surface specified by `pSurfaceDesc` must be `KM_TEXTURE_TWIDDLED_MM`.

If `TRUE` is specified as `bAutoMipMap`, a work area of 512 KB is allocated in the stack. The operation is not guaranteed unless sufficient memory is allocated.

bDither (input)

Specifies whether dither is applied to the texture to be read. If `TRUE` is specified, dither is applied. If `TRUE` is specified, the pixel data must have `KM_TEXTURE_BMP` as a category code of texture type and be a square texture of 512 x 512 texels.

In this case, the read texture is converted into `KM_TEXTURE_TWIDDLED`/`KM_TEXTURE_TWIDDLED_MM` format. Therefore, the type of the surface specified by `pSurfaceDesc` must be `KM_TEXTURE_TWIDDLED`/`KM_TEXTURE_TWIDDLED_MM`.

If TRUE is specified for `bdither`, a work area of 512 KB is allocated in the stack.

The operation is not guaranteed unless sufficient memory is allocated.

Return values:

KMSTATUS_SUCCESS	Read successfully
KMSTATUS_INVALID_TEXTURE_TYPE	Invalid texture type specified
KMSTATUS_INVALID_MIPMAPED_TEXTURE	Use of AutoMIPMAP / AutoDither is attempted for a texture for which use of MIPMAP / Dither is inhibited

3.9.2 Re-reading the Code Book Portion of VQ Texture

```
KMSTATUS kmLoadVQCodebook(          PKMSURFACEDESC pSurfaceDesc,  
                                   PKMDWORD pTexture)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function reads only the code book portion of the VQ texture in main memory as specified by `pTexture` to the VQ texture surface specified by `pSurfaceDesc`. It is used to rewrite only the code book (800h bytes) of the VQ texture already loaded and use the color palette effect.

Arguments:

pSurfaceDesc (input)

Texture surface allocated by `kmCreateTextureSurface` / `kmCreateCombinedTextureSurface`. The category of this surface must be either `KM_TEXTURE_VQ` or `KM_TEXTURE_VQ_MM`.

pTexture (input)

Pointer indicating a texture (code book) in main memory. This does not have to be in the complete VQ texture format, but a code book must be included in the first 800h bytes.

Return values:

KMSTATUS_SUCCESS	Read successfully
KMSTATUS_INVALID_TEXTURE_TYPE	Invalid texture surface specified

3.9.3 Reloading a Particular Mipmap Texture

```
KMSTATUS kmReloadMipmap(PKMSURFACEDESC pSurfaceDesc,
                        PKMVOID pTexture,
                        KMINT32 nMipmapCount)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function overwrites the mipmap texture on main memory specified by pTexture and loads it into the texture memory area allocated by kmCreateTextureSurface.

The type of the texture (surface) that can be specified is either KM_TEXTURE_TWIDDLED_MM or KM_TEXTURE_VQ_MM (the format and size of the texture to be read are identified by the surface descriptor specified by pSurfaceDesc).

The correct picture is not displayed if the code book at the reloading destination and that at the reloading source coincide when reloading VQ-Mipmap. Nothing is performed if 1x1 Mipmap is specified when reloading VQ-Mipmap.

[Reference]

Offset from the beginning of Twiddled mipmap file in KAMUI texture format to each mipmap level and the number of bytes of each mipmap level (except 16 bytes of Header)

SIZE	OFFSET	BYTES
1x1	2	2
2x2	4	8
4x4	12(Ch)	32(20h)
8x8	44(2Ch)	128(80h)
16x16	172(ACh)	512(200h)
32x32	684(2ACh)	2048(800h)
64x64	2732(AACh)	8192(2000h)
128x128	10924(2AACh)	32768(8000h)
256x256	43692(AACh)	131072(20000h)
512x512	174764(2AACh)	524288(80000h)
1024x1024	699052(AAACh)	2097152(200000h)

Offset from the beginning of VQ mipmap file in KAMUI texture format to each mipmap level and the number of bytes of each mipmap level (except 16 bytes of Header)

SIZE	OFFSET	BYTES
1x1	--	--
2x2	2048 + 1	1
4x4	2048 + 2	4
8x8	2048 + 6	16(10h)
16x16	2048 + 22(16h)	64(40h)
32x32	2048 + 86(56h)	256(100h)
64x64	2048 + 342(156h)	1024(400h)
128x128	2048 + 1366(556h)	4096(1000h)
256x256	2048 + 5462(1556h)	16384(4000h)
512x512	2048 + 21846(5556h)	65536(10000h)
1024x1024	2048 + 87382(15556h)	262144(40000h)

Arguments:

pSurfaceDesc (input)

Texture surface allocated by kmCreateTextureSurface/kmCreateCombinedTextureSurface

pTexture (input)

Pointer indicating the pixel data portion of the texture in main memory. Indicates the beginning of the texture data of the mipmap level specified by nMipmapCount.

<Reload source>

nMipmapCount (input)

Specify the level of the mipmap texture to be read. One of the following enum values can be specified.

<u>nMipmapCount</u>	<u>Texture Size</u>
KM_MAPSIZE_1	1x1
KM_MAPSIZE_2	2x2
KM_MAPSIZE_4	4x4
KM_MAPSIZE_8	8x8
KM_MAPSIZE_16	16x16
KM_MAPSIZE_32	32x32
KM_MAPSIZE_64	64x64
KM_MAPSIZE_128	128x128
KM_MAPSIZE_256	256x256
KM_MAPSIZE_512	512x512
KM_MAPSIZE_1024	1024x1024

Return values:

KMSTATUS_SUCCESS	Success
KMSTATUS_INVALID_PARAMETER	Invalid parameter
KMSTATUS_INVALID_TEXTURE	Invalid texture specified

3.9.4 Reading the YUV-Format Texture Data

```
KMSTATUS kmLoadYUVTexture(PPKMSURFACEDESC ppSurfaceDesc,
                           PKMDWORD pTexture,
                           KMINT32 nTexture,
                           KMINT32 nFormat)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Does not work	Does not work	Implemented

Explanation: This function converts the YUV420-data/YUV422-data in main memory specified by `pTexture` into Non-Twiddled YUV422 texture and reads it into a texture memory area allocated by `kmCreateTextureSurface/kmCreateCombinedTextureSurface/kmCreateContiguousTextureSurface`. In doing so, the YUV-data converter built into Tile Accelerator of the CLX1 is used. Because the output of the YUV-data converter is Non-Twiddled, the Texture Surface at the read destination specified by this API must be in either of the following formats:

```
KM_TEXTURE_RECTANGLE | KM_TEXTURE_YUV422    // Rectangular
KM_TEXTURE_STRIDE    | KM_TEXTURE_YUV422    // Rectangular (with stride specification)
```

If two or more YUV-data are read successively at one time (`nTexture > 1`), the size of each texture must be 16 x 16 texels. Exercise care in specifying the size of Texture Surface at the read destination specified by this API. In this case, Texture Surface at read destination must be allocated to contiguous addresses in the frame buffer. Specify the Texture Surface allocated by "kmCreateContiguousTextureSurface" API.

Arguments:

ppSurfaceDesc (input)

Pointer of pointer array to KMSURFACEDESC structure indicating the texture surface that has already been allocated.

pTexture (input)

Pointer indicating YUV420-data/YUV422-data in main memory

nTexture (input)

Specifies the number of Textures to be read successively. A value of 1 to 64 can be specified.

nFormat (input)

Specifies the format of the data to be read. Specify either of the following:

```
KM_TEXTURE_YUV420    // Indicates that the input data is YUV420-data.
KM_TEXTURE_YUV422    // Indicates that the input data is YUV422-data.
```

Return values:

```
KMSTATUS_SUCCESS      Success
KMSTATUS_INVALID_TEXTURE Invalid texture specified
```

3.9.5 Deleting Texture Data

KMSTATUS kmFreeTexture(PKMSURFACEDESC pSurfaceDesc)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function releases a specified texture surface.

Argument:

PSurfaceDesc (I/O)

Texture surface allocated by kmCreateTextureSurface

Return values:

KMSTATUS_SUCCESS	Successful release
KMSTATUS_INVALID_PARAMETER	Release ended in failure. The specified texture is illegal.
KMSTATUS_INVALID_ADDRESS	Specified Area (Surface) is not allocated.

3.9.6 Obtaining the Available Texture Memory Space

```
KMSTATUS kmGetFreeTextureMem(PKMUINT32 pSizeOfTexture
                             PKMUINT32 pMaxBlockSizeOfTexture)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function returns the unused capacity of the texture memory.

The texture memory is managed in block units. If it is repeatedly allocated and released, the texture memory is divided into many blocks. This API can check the total size (`pSizeOfTexture`) of all the vacant blocks of texture memory and the size of the largest vacant block (`pMaxBlockSizeOfTexture`).

Even if the total size of the vacant blocks is sufficient, if the size of the largest vacant block is not sufficient, `KMSTATUS_NOT_ENOUGH_MEMORY` (insufficient memory) is returned when a texture surface is allocated (`kmCreateTextureSurface` or `kmCreateCombinedTextureSurface`).

With ARC1, the VQ and mipmap textures are interleaved in memory. Therefore, these surfaces may not be secured even if `pMaxBlockSizeOfTexture` is sufficient.

To use the texture memory efficiently, secure and release as many texture surfaces as possible.

Argument:

pSizeOfTexture (input)

Pointer to the `KMDWORD` area to which the available texture memory space is returned

pMaxBlockSizeOfTexture (input)

Pointer to the `KMDWORD` area to which the largest vacant block in the texture memory is to be returned.

Return value:

<code>KMSTATUS_SUCCESS</code>	Success
-------------------------------	---------

3.9.7 Reading the Texture in Texture Memory

```
KMSTATUS kmGetTexture(      PKMDWORD pTexture,
                           PKMSURFACEDESC pSurfaceDesc)
```

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function reads the texture in texture memory specified by pSurfaceDesc to the main memory specified by pTexture. Only the texture pixel data of the KAMUI texture format is output. No header is appended. If SurfaceDesc of the frame buffer is specified for pSurfaceDesc, the contents of the specified frame buffer can be read into main memory.

pTexture (output)

Pointer indicating the area in main memory where the texture is to be saved. Secure a multiple of four bytes, aligned with a four-byte boundary.

<Read destination>

pSurfaceDesc (input)

Texture surface to which the texture is saved.

<Read source>

Return values:

- KMSTATUS_SUCCESS Read successfully
- KMSTATUS_INVALID_ADDRESS Specified texture surface is not allocated.

3.9.8 Garbage Collection of Texture Memory

KMSTATUS kmGarbageCollectTexture(VOID)

IRIS+ARC1	COSMOS+ARC1	Holly
Implemented	Implemented	Implemented

Explanation: This function performs garbage collection for the frame buffer memory. If there is a vacant area at addresses lower than the already allocated texture surface, the texture is moved and aligned with the lower addresses.

The address of the texture is changed after this API has been called (the contents of pSurface of the KMSURFACEDESC structure are rewritten). Consequently, kmProcessVertexRenderState and kmSetVertexRenderState must be re-executed for all the KMVERTEXCONTEXT structures using texture.

Note that the frame buffer area, TA output data area, and VQ/Mipmap texture area of ARC1 are not subject to garbage collection. To use the memory efficiently, call the functions that create a frame buffer area and TA output data area (kmCreateFrameBufferSurface and kmCreateVertexBuffer API) before creating the texture surface and, whenever possible, release and re-create these areas before the end of AP. With ARC1, allocate or release VQ/Mipmap texture areas at the same time and in combination.

Argument:

None

Return value:

KMSTATUS_SUCCESS

Garbage collection successful



4. KAMUI UTILITY LIBRARY

This chapter explains the utility library.

Those functions that do not access the hardware but which are closely related to the hardware are supplied as a utility library. To use these functions, include `kmutil.h` in the source code of the application, and link `kmutil.lib`.

The names of all the functions included in this library start with `kmu`.

4.1 TEXTURE-RELATED FUNCTIONS

4.1.1 Conversion from KAMUI Bit Map Format to Twiddled Format

```

KMSTATUS      kmuCreateTwiddledTexture (
                                PKMDWORD      pOutputTexture,
                                PKMDWORD      pInputTexture,
                                KMBOOLEAN     bAutoMipMap,
                                KMBOOLEAN     bDither,
                                KMINT32       USize,
                                KMTEXTURETYPE nTextureType
                                );

```

Explanation: This function converts a texture in KM_TEXTURE_BMP format (ABGR8888) in main memory into a texture in Twiddled/Twiddled Mipmap format. If TRUE is specified for bAutoMipMap, a mipmap is created automatically. If TRUE is specified for bDither, dither is effected.



Caution: The contents of the input texture data are destroyed if mipmap or dither is specified.

Arguments:

pOutputTexture (output)

Address in main memory to which converted texture data is to be written.

pInputTexture (input)

Pointer indicating an input texture in KM_TEXTURE_BMP format.

bAutoMipMap (input)

Specifies whether MIPMAP is created automatically. If TRUE is specified, MIPMAP is automatically created (the output is in KM_TEXTURE_TWIDDLED_MM format). If FALSE is specified, MIPMAP is not created (output is in KM_TEXTURE_TWIDDLED format).

bDither (input)

Specifies whether dither is effected. If TRUE is specified, dither is effected.

USize (input)

Specifies the number of texels per side of texture. Select one of the following:

```

KM_MAPSIZE_8
KM_MAPSIZE_16
KM_MAPSIZE_32
KM_MAPSIZE_64
KM_MAPSIZE_128
KM_MAPSIZE_256
KM_MAPSIZE_512
KM_MAPSIZE_1024

```

nTextureType (input)

Specifies the pixel format of the converted texture. Select one of the following:

```
KM_TEXTURE_ARGB1555
KM_TEXTURE_RGB565
KM_TEXTURE_ARGB4444
```

Return values:

KMSTATUS_SUCCESS	Converted successfully
KMSTATUS_INVALID_TEXTURE_TYPE	Invalid texture type specified

4.1.2 Conversion from Frame Buffer Format (Rectangle) to Windows BMP Format

```
KMSTATUS          kmuConvertFBtoBMP (
                    PKMDWORD      pOutputData,
                    PKMDWORD      pInputData,
                    KMINT32        nWidth,
                    KMINT32        nHeight,
                    KMBPPMODE      nBpp
                    );
```

Explanation: This function converts the contents of the frame buffer read into main memory by `kmGetTexture` into pixel data in Windows full-color BMP format (BGR888) and writes it into memory. This is a debug function in that it saves the contents of the frame buffer in Windows BMP format.

This API does not create the 54 bytes of the header in Windows BMP format.

Arguments:

pOutputData (output)

Address of main memory into which the converted pixel data is to be written.

pInputData (input)

Pointer indicating the contents of the frame buffer. Pointer to the pixel data of the frame buffer read by specifying a descriptor of the frame buffer surface by using `kmGetTexture`.

nWidth, nHeight (input)

Specifies the screen size of the read frame buffer.

nBpp (input)

Specifies the pixel format of the read frame buffer. One of the following can be specified.

```
KM_DSPBPP_RGB565
KM_DSPBPP_RGB555
KM_DSPBPP_ARGB4444
KM_DSPBPP_ARGB1555
```

Return value:

KMSTATUS_SUCCESS	Converted successfully
------------------	------------------------



5. STRUCTURES

5.1 FRAME BUFFER/TEXTURE SURFACE STRUCTURE

```
typedef struct tagKMSURFACEDESC
{
    KMDWORD SurfaceType;        // 0... FrameBuffer 1...Texture
    //    KM_SURFACETYPE_FRAMEBUFFER
    //    KM_SURFACETYPE_TEXTURE
    // Note    The higher 16 bits of this member are reserved for Ninja
    KMDWORD BitDepth;           // Indicates number of bits per pixel (e.g., 16 for 16 bpp)
    //    KM_BITDEPTH_16
    //    KM_BITDEPTH_24
    //    KM_BITDEPTH_32
    KMDWORD PixelFormat;        // 1555, 4444, etc.
    //    KM_PIXELFORMAT_ARGB1555
    //    KM_PIXELFORMAT_RGB565
    //    KM_PIXELFORMAT_ARGB4444
    //    KM_PIXELFORMAT_YUV422
    //    KM_PIXELFORMAT_BUMP
    //    KM_PIXELFORMAT_PALETTIZED_4BPP
    //    KM_PIXELFORMAT_PALETTIZED_8BPP
    union{
        KMDWORD USize;          // USize 8 - 1024
        //    KM_MAPSIZE_8x8
        //    KM_MAPSIZE_16x16
        //    KM_MAPSIZE_32x32
        //    KM_MAPSIZE_64x64
        //    KM_MAPSIZE_128x128
        //    KM_MAPSIZE_256x256
        //    KM_MAPSIZE_512x512
        //    KM_MAPSIZE_1024x1024
        KMDWORD Width;           // For Frame Buffer, Horizontal Size
    };
    union{
        KMDWORD VSize;          // VSize 8 - 1024
        //    KM_MAPSIZE_8x8
    };
};
```

```

        //      KM_MAPSIZE_16x16
        //      KM_MAPSIZE_32x32
        //      KM_MAPSIZE_64x64
        //      KM_MAPSIZE_128x128
        //      KM_MAPSIZE_256x256
        //      KM_MAPSIZE_512x512
        //      KM_MAPSIZE_1024x1024
    KMDWORD Height;        // For Frame Buffer, Vertical Size
};
union {
    KMDWORD dwTextureSize; // Texture Size (byte)
    KMDWORD dwFrameBufferSize; // FrameBuffer Size
};
    KMDWORD fSurfaceFlags; // Surface Flags
    KMDWORD *pSurface;     // Pointer to Texture Instance
}KMSURFACEDESC, *PKMSURFACEDESC;

```

fSurfaceFlags

---- For texture

bit 0	0... Non MipMap	1... MipMapped
bit 1	0... Chromakey Fix OFF	1... ChromaKey Fix ON!
bit 2	0... Rectangle	1... Twiddled
bit 3	0... NonVQ	1... VQed Texture
bit 4	0... NonStride	1... Stride Select
bit 5	0... Non Palettized	1...Palettized

Specify the OR of the following flags:

KM_SURFACEFLAGS_MIPMAPED	0x001
KM_SURFACEFLAGS_CHROMAKEYFIX	0x002
KM_SURFACEFLAGS_TWIDDLED	0x004
KM_SURFACEFLAGS_VQ	0x008
KM_SURFACEFLAGS_PALETTIZED	0x010

---- For frame buffer

bit 0	0... Full-Screen Buffer	1... StripBuffer
-------	-------------------------	------------------

5.2 VERSION INFORMATION STRUCTURE

```

typedef struct KMVERSIONINFO
{
    KMDWORD kmMajorVersion;        // 1 for IRIS, 2 for COSMOS, 3 for HOLLY
    KMDWORD kmLocalVersion;
    KMDWORD kmFrameBufferSize;    // Total Size of Texture and Frame Buffer
} KMVERSIONINFO, *PKMVERSIONINFO;

```


5.3 VERTEX CONTEXT

```

typedef struct tagKMVERTEXCONTEXT
{
    KMDWORD                RenderState;                // Render Context

    // for Global Parameter
    KMPARAMTYPE             ParamType                  // Parameter Type
    KMLISTTYPE              ListType                   // List Type
    KMCOLORTYPE             ColorType                  // Color Type
    KMUVFORMAT              UVFormat                   // UV format

    // for ISP/TSP Instruction Word
    KMDEPTHMODE             DepthMode;                 // Depth Mode Specification
    KMCULLINGMODE           CullingMode;               // Culling Mode
    KMSCREENCOORDINATION    ScreenCoordination;        // Screen Coordination
    KMSHADINGMODE           ShadingMode;               // Shading Mode
    KMMODIFIER              SelectModifier;            // Modifier Volume Valiant
    KMBOOLEAN               bZWriteDisable;            // Z Write Disable

    // for TSP Control Word
    KMBLENDINGMODE          SRCBlendingMode;           // Source Blending Mode
    KMBLENDINGMODE          DSTBlendingMode;           // Destination Blending Mode
    KMBOOLEAN               bSRCSel;                   // Source Select
    KMBOOLEAN               bDSTSel;                   // Destination Select
    KMFOGMODE               FogMode;                   // Fogging
    KMBOOLEAN               bUseSpecular;              // Specular Highlight
    KMBOOLEAN               bUseAlpha;                 // Alpha
    KMBOOLEAN               bIgnoreTextureAlpha;       // Ignore Texture Alpha
    KMFLIPMODE              FlipUV;                    // Flip U,V
    KMCLAMPMODE             ClampUV;                   // Clamp U,V
    KMFILTERMODE            FilterMode;                 // Texture Filter
    KMBOOLEAN               bSuperSample;              // Anisotropic Filter
    KMDWORD                 MipMapAdjust;              // Mipmap D Adjust
    KMTEXTURESHADINGMODE    TextureShadingMode;        // Texture Shading Mode
    KMBOOLEAN               bColorClamp;               // Color Clamp
    KMDWORD                 PaletteBank;               // Palette Bank

    // for Texture Control Bits/Address
    PKMSURFACEDESC          pTextureSurfaceDesc;       // Texture DESC Pointer

    // FaceColor Setting for Intensity
    KMFLOAT                 fFaceColorAlpha;           // Face Color Alpha
    KMFLOAT                 fFaceColorRed;             // Face Color Red
    KMFLOAT                 fFaceColorGreen;           // Face Color Green
    KMFLOAT                 fFaceColorBlue;            // Face Color Blue

```

```
// Specular Highlight Specification for Intensity
KMFLOAT          fOffsetColorAlpha;      // Specular Alpha
KMFLOAT          fOffsetColorRed;        // Specular Red
KMFLOAT          fOffsetColorGreen;      // Specular Green
KMFLOAT          fOffsetColorBlue;       // Specular Blue

/* Internal Variables */
KMDWORD          GLOBALPARAMBUFFER;      // Global Parameter Buffer
KMDWORD          ISPPARAMBUFFER;         // ISP Parameter Buffer
KMDWORD          TSPPARAMBUFFER;         // TSP Parameter Buffer
KMDWORD          TexturePARAMBUFFER;     // TextureParameter Buffer
} KMVERTEXCONTEXT, *PKMVERTEXCONTEXT;

/* for ModifierInstruction */

KMDWORD          ModifierInstruction;     /* ModifierInstruction */
KMFLOAT          fBoundingBoxXmin;       /* BoundingBoxXmin(ShadowVolume)*/
KMFLOAT          fBoundingBoxYmin;       /* BoundingBoxYmin(ShadowVolume)*/
KMFLOAT          fBoundingBoxXmax;       /* BoundingBoxXmax(ShadowVolume)*/
KMFLOAT          fBoundingBoxYmax;       /* BoundingBoxYmax(ShadowVolume)*/
```

5.4 PACKED 32-BIT COLORS

```
typedef union _tagKMPACKEDARGB
{
    KMDWORD      dwPacked;
    struct {
        BYTE     bBlue;
        BYTE     bGreen;
        BYTE     bRed;
        BYTE     bAlpha;
    }byte;
}KMPACKEDARGB, *PKMPACKEDARGB;
```

5.5 PALETTE DEFINITION STRUCTURE

```
typedef union _tagKMPALETTE_DATA
{
    KMDWORD wPaletteData[KM_PALETTE_ENTRY2];
    KMDWORD dwPaletteData[KM_PALETTE_ENTRY];
}KMPALETTE_DATA, *PKMPALETTE_DATA;
```

5.6 VERTEX DATA BUFFER STRUCTURE

```
typedef struct _tagKMVERTEXBUFFERDESC
{
    PKMDWORD pOpaquePolygonBuffer;
    PKMDWORD pOpaqueModifierBuffer;
    PKMDWORD pTransPolygonBuffer;
```

```
    PKMDWORD pTransModifierBuffer;  
    PKMDWORD pTAOutputBuffer;  
}KMVERTEXBUFFDESC,*PKMVERTEXBUFFDESC;
```




6. TEXTURE FORMAT

6.1 TEXTURE FORMATS SUPPORTED BY ARC1/CLX1

The ARC1/CLX1 supports the following textures. ARC1 in the figure indicates the texture supported by both ARC1 and CLX1. CLX1 indicates the texture supported only by CLX1. X indicates a format that is not supported.

Texture Format			ARGB 1555	RGB 565	ARGB 4444	YUV 422	Bump	ARGB 8888
Twiddled	Non VQ	Square	ARC1	ARC1	ARC1	CLX1	CLX1	X
		Square Mipmap	ARC1	ARC1	ARC1	CLX1	CLX1	X
		Rectangle	CLX1	CLX1	CLX1	CLX1	CLX1	X
		Palettized(4,8bpp)	CLX1	CLX1	CLX1	X	X	CLX1
		Palettized(4,8bpp) Mipmap	CLX1	CLX1	CLX1	X	X	CLX1
	VQ	Square	ARC1	ARC1	ARC1	X	X	X
		Square Mipmap	ARC1	ARC1	ARC1	X	X	X
Scan Order	Rectangle		ARC1	ARC1	ARC1	CLX1	CLX1	X
	Stride		ARC1	ARC1	ARC1	CLX1	CLX1	X

The ARGB8888 color uses 32 bpp (4 bytes) for each texel. The other colors use 16 bpp (2 bytes) for each texel.

The number of bits of the index for Palettized 4 bpp is 4.

The number of bits of the index for Palettized 8 bpp is 8.

The number of bits of the index in VQ format is 8.

6.2 ARC1/CLX1 TEXTURE FORMAT

6.2.1 Texture Format of KAMUI

The texture format of KAMUI is the pixel data of a texture with headers (4 x 32 bits) indicating size and type appended. The headers are not referenced by KAMUI. Instead, KAMUI identifies the format of a texture based on information on the texture surface descriptor at the load destination specified by a load function. The headers are appended data for high-level applications such as Ninja.

When specifying the address of a texture by using a load function of KAMUI, skip the headers and specify the first address of pixel data.

"PVRT"(4byte)		KAMUI ID
Texture Data Size(4byte)		Size of Pixel Data +8 (+8 means nTexture Type and Width and Height)
nTextureType(4byte)		Type of texture
nWidth(2byte)	nHeight(2byte)	Width and Height (Num of Texel)
Texture Data		Pixel Data portion
:		
:		
:		

"PVRT"

The KAMUI texture starts with four half-width characters "PVRT", which constitute an identifier indicating a KAMUI texture.

Texture Data Size

Saves the number of bytes of pixel data of the texture + 8. If two or more texture files are successively synthesized into one file, the first position of the next texture can be checked by using this value.

nTextureType

Specifies the format of the texture by using a category code and pixel format. The following enum values are ORed and specified (see the description of `kmCreateTextureSurface`).

Note that the higher 16 bits of this field are reserved for Ninja.

Category codes

KM_TEXTURE_TWIDDLED(0x00000100)

Texture in Twiddled-Non VQ-Square format

Texture of special pixel array used for PCX1/PCX2

KM_TEXTURE_TWIDDLED_MM(0x00000200)

Twiddled-Non VQ-Square format with mipmap

KM_TEXTURE_TWIDDLED_RECTANGLE(0x00000D00)

Twiddled-Non VQ-Rectangle format. Rectangular texture
Can be used with CLX1 only.

KM_TEXTURE_VQ(0x00000300)

Texture in Twiddled-VQ-Square format. Texture on which VQ (Vector Quantization) compression is performed. The first half of the texture data includes a code book table.

KM_TEXTURE_VQ_MM(0x00000400)

In Twiddled-VQ-Square format with mipmap

KM_TEXTURE_PALETTIZE4(0x00000500)

KM_TEXTURE_PALETTIZE4_MM(0x00000600)

Texture in Twiddled-NonVQ-Palettized format

Palettized texture with index of 4 bpp (16 colors). Because only one palette can be used in a system, texture data does not include palette information. To set a palette, use `kmSetPaletteMode` or `kmSetPaletteData`.

KM_TEXTURE_PALETTIZE8(0x00000700)

KM_TEXTURE_PALETTIZE8_MM(0x00000800)

Palettized texture with an index of 8 bpp (256 colors) in Twiddled-Non VQ-Palettized format

KM_TEXTURE_RECTANGLE(0x00000900)

Rectangular texture in Scan Order-Rectangle format. The arrangement of the pixel data conforms to Windows BMP format.

KM_TEXTURE_STRIDE(0x00000B00)

Rectangular texture in Scan Order-Stride format. The arrangement of the pixel data conforms to Windows BMP format.

KM_TEXTURE_BMP(0x00000E00)

Input-dedicated format for MIPMAP automatic creation/dithering of `kmLoadTexture`. The texture data is ABGR8888 conforming to the pixel data in Windows BMP format (24 bpp). In this case, specifying the pixel format in a header is not necessary.

The texture in this format is Twiddled (or Twiddled Mipmap) when read by `kmLoadTexture`, and the pixel format is RGB565, ARGB1555, or ARGB4444. Specify the pixel format by `kmCreateTextureSurface`.

Pixel format codes

KM_TEXTURE_ARGB1555 (0x00000000)

Format consisting of one bit of alpha value and five bits of RGB values

The alpha value indicates transparent when it is 0 and opaque when it is 1.

KM_TEXTURE_RGB565 (0x00000001)

Format without alpha value and consisting of five bits of RB values and six bits of G value

KM_TEXTURE_ARGB4444 (0x00000002)

Format consisting of four bits of alpha value and four bits of RGB values.

The alpha value indicates completely transparent when it is 0x0 and completely opaque when it is 0xF.

KM_TEXTURE_YUV422 (0x00000003)

YUV422 format

KM_TEXTURE_BUMP (0x00000004)

Specifies a texture for bump mapping

ARGB8888 color can be specified in Palettized format only. In this case, it is set for a palette by `kmPaletteMode` API instead of `nTextureType`.

nWidth, nHeight

Specifies the horizontal and vertical sizes of a texture. The horizontal and vertical sizes of a texture must be 8, 16, 32, 64, 128, 256, 512, or 1024.

For details, see the description of `kmCreateTextureSurface`.

Texture Data

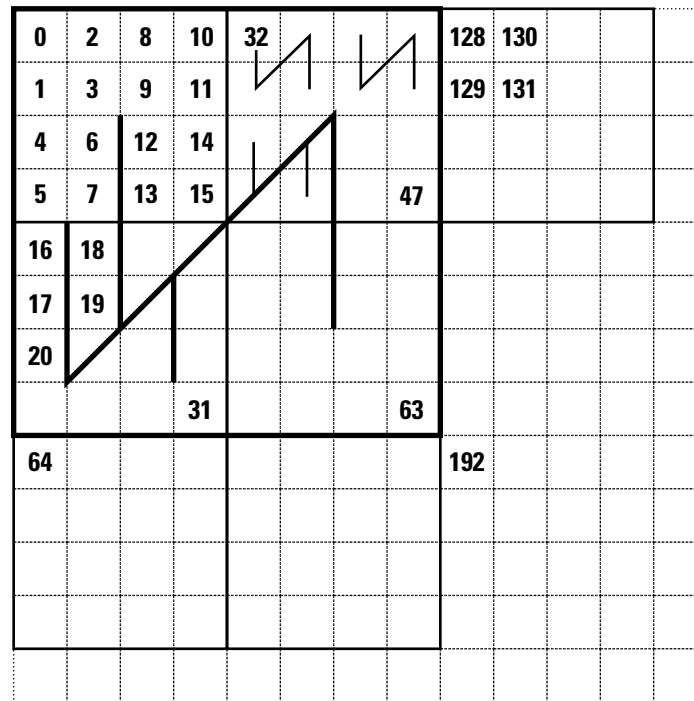
Pixel data of a texture. Basically, conforms to the image in the texture memory of ARC1/CLX1.

The address of the texture-specified parameter `pTexture` of the texture-related API of KAMUI is the first address of this portion. Note that the header may be skipped.

For details of the pixel data, see the following chapters.

6.2.2 Twiddled Format and Twiddled Mipmap Format

Twiddled-NonVQ-Square and Twiddled-NonVQ-Square Mipmap formats are usually called Twiddled and Twiddled Mipmap formats. Twiddled format is the basic format of the PowerVR texture. Because this format executes texture filtering such as Bilinear filter at high speeds, addressing is optimized by hardware. Therefore, pixels in Twiddled format must not be arranged in raster order. The pixel order in Twiddled format is as shown below. Because Twiddled format is 16 bpp, it must be doubled to obtain the actual byte addresses.



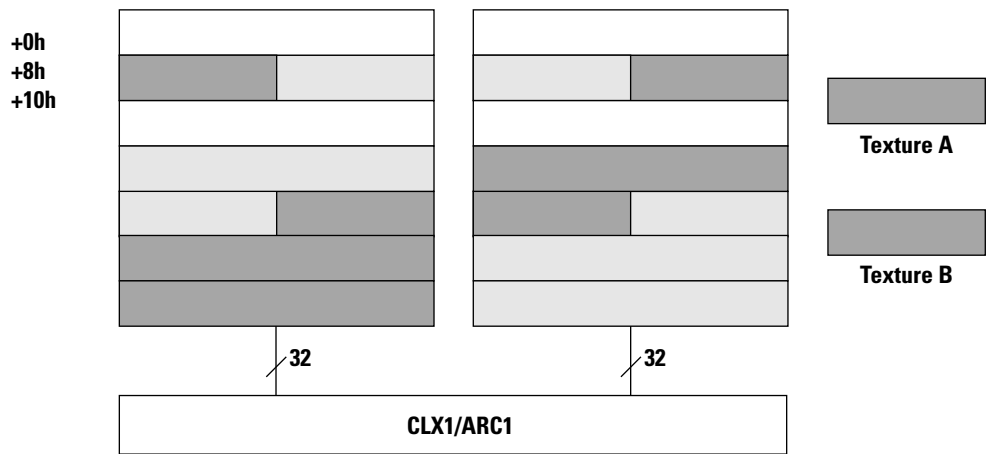
Twiddled format is as follows in KAMUI texture format:

+00h	KAMUI Texture Header (10h bytes)
+10h	Texture Pixel Data

The size of ARC1/CLX1 twiddled mipmap format is as follows, In ARC1, Twiddled format should always be square.

W,H size	Pixel Data Size
8x8	80h bytes
16x16	200h bytes
32x32	800h bytes
64x64	2000h bytes
128x128	8000h bytes
256x256	20000h bytes
512x512	80000h bytes
1024x1024	200000h bytes

With ARC1, the mipmap texture in Twiddled format is specially arranged in memory. ARC1 has a 64-bit texture data bus. This bus is divided into two 32-bit banks and arranged so that each mipmap level is interleaved. To enhance the efficiency of the memory, two sets of textures are usually interleaved and registered as one set. The level of a mipmap is 1x1, i.e., the lowest level is registered first. An example of interleaving is shown below.



The byte address image in ARC1 viewed from the PCI/SH4 is as follows:

Bank 0		Bank 1	
+00h	Dummy Zero(6h bytes)	+00h	Dummy Zero(6h bytes)
+06h	Texture A 1x1 map(2h bytes)	+06h	Texture B 1x1 map(2h bytes)
+08h	Texture B 2x2 map(8h bytes)	+08h	Texture A 2x2 map(8h bytes)
+10h	Texture A 4x4 map(20h bytes)	+10h	Texture B 4x4 map(20h bytes)
+30h	Texture B 8x8 map(80h bytes)	+30h	Texture A 8x8 map(80h bytes)
	:		:

In the above example, two textures, A and B, are interleaved and located. With KAMUI, the start address of a texture is always aligned with an eight-byte boundary.

With KAMUI, the bank size is set to 4 MB. Where the start address of a 1x1 mipmap of texture A written into Bank0 is 00000006h, the start address of a 1x1 mipmap of texture B is 00400006h.

The Twiddled mipmap is interleaved and located only with ARC1. Because KAMUI absorbs the differences between the ARC1 and CLX1, interleaving is unnecessary in KAMUI texture format. To maintain compatibility with SGL, two Dummy bytes are provided.

In KAMUI texture format, the Twiddled mipmap format is as follows (Twiddled mipmap format must always be a square).

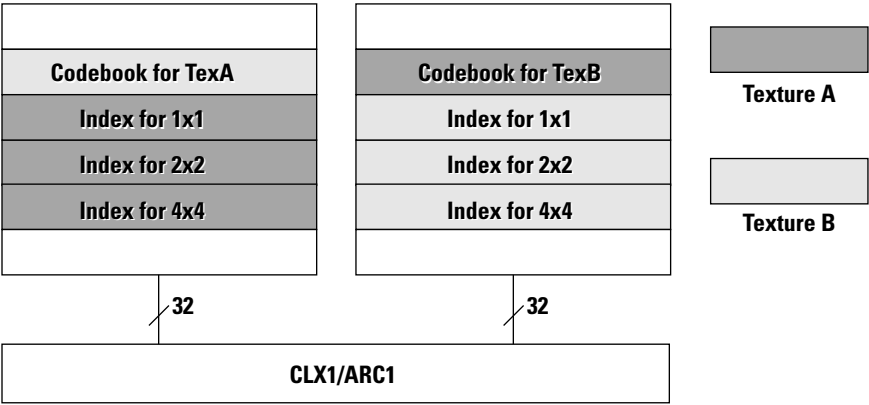
+00h	KAMUI Texture Header (10h bytes)
+10h	Dummy Zero (2h bytes)
+12h	1x1 mipmap texture (2h bytes)
+14h	2x2 mipmap texture (8h bytes)
+1Ch	4x4 mipmap texture (20h bytes)
+3Ch	8x8 mipmap texture (80h bytes)
+BCh	16x16 mipmap texture (200h bytes)
+2BCh	32x32 mipmap texture (800h bytes)
+ABCh	64x64 mipmap texture (2000h bytes)
+2ABCh	128x128 mipmap texture (8000h bytes)
+AABCh	256x256 mipmap texture (20000h bytes)
+2AABCh	512x512 mipmap texture (80000h bytes)
+AAABCh	1024x1024 mipmap texture (200000h bytes)

6.2.3 VQ Format and VQ Mipmap Format

Twiddled-VQ-Square and Twiddled-VQ-Square-Mipmap formats are usually called VQ/VQ Mipmap formats.

VQ (Vector Quantization) format is a compressed texture format with a high compression rate. VQ format largely consists of two areas. One is a color table called a code book, while the other is Index data that indicates the position of this code book. The VQ format structure is very close to the palette texture in that it expands Index data by a code book and creates an image. However, this code book can be set for each texture.

In VQ format, each pixel index is arranged in Twiddled format. With the ARC1, the code book and index are interleaved in memory to enable high-speed access. The memory mapping if mipmap is effected is shown below (if mipmap is not effected, only the Index of the registered size is registered).



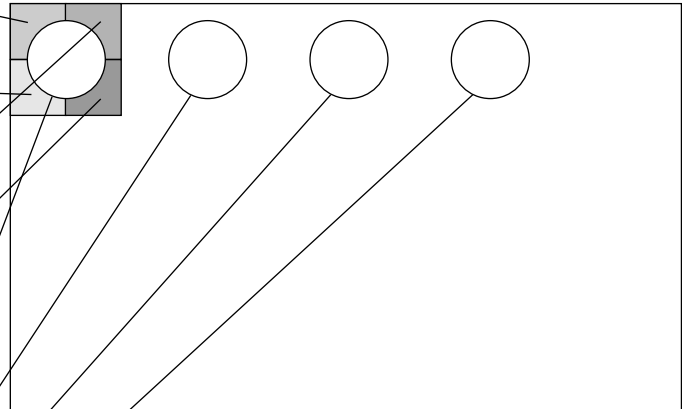
A texture of VQ format consists of a code book and index data. The size of a code book is always 256 entries. Each entry contains data for four texels.

Codebook

+0	Pixel A Low
+1	Pixel A High
+2	Pixel B Low
+3	Pixel B High
+4	Pixel C Low
+5	Pixel C High
+7	Pixel D Low
+8	Pixel D High
.....
+7FE	Codebook end

Index (Bank opposite to codebook)

+0	Index#0
+1	Index#1
+2	Index#2
+3	Index#3
+4	
+5	
+7	
+8	
.....



If a texture of 256 x 256 is compressed by VQ, the size is reduced to about 1/7, as follows:

$$(256 \times 256 \times 16) / ((128 \times 128 \times 8) + (256 \times 16 \times 4)) = 7.1 : 1$$

↑ Index

↑ Code book

The byte address image viewed from the PCI/SH4 is as follows:

Bank 0		Bank 1	
+00h	Texture A CodeBook (800h bytes)	+00h	Texture B CodeBook (800h bytes)
+800h	Texture B Index	+800h	Texture A Index
[VQ MipMap]			
+00h	Texture A CodeBook (800h bytes)	+00h	Texture B CodeBook (800h bytes)
+800h	Dummy Zero (1h byte)	+00h	Dummy Zero (1h byte)
+801h	Texture B 2x2 mipmap Index (1h byte)	+801h	Texture A 2x2 mipmap Index (1h byte)
+802h	Texture B 4x4 mipmap Index (4h bytes)	+802h	Texture A 4x4 mipmap Index (4h bytes)
+806h	Texture B 8x8 mipmap Index (10h bytes)	+806h	Texture A 8x8 mipmap Index (10h bytes)
+816h	Texture B 16x16 mipmap Index (40h bytes) :	+816h	Texture A 16x16 mipmap Index (40h bytes) :
+???h	Dummy Zero (Ah byte)	+???h	Dummy Zero (Ah byte)

Note that, unlike in Twiddled format, the mipmap of each level is not interleaved. In VQ, the index of a 1x1 mipmap does not exist.

With KAMUI, the bank size is set to 4 MB. If the start address of a 2x2 mipmap of texture B written into Bank0 is 00000801h, the start address of a 2x2 mipmap of texture A is 00400801h.

The code book is interleaved only with ARC1.

In KAMUI texture format, it is possible to safely ignore VQ interleaving.

The VQ format of KAMUI is as follows (VQ format must be always a square).

+00h	KAMUI Texture Header (10h bytes)
+10h	Code Book (800h bytes)
+810h	Index

The size of the index of a texture ind is as follows:

W.H size	Size of Index
8x8	10h bytes
16x16	40h bytes
32x32	100h bytes
64x64	400h bytes
128x128	1000h bytes
256x256	4000h bytes
512x512	10000h bytes
1024x1024	40000h bytes

The VQ mipmap format of KAMUI is as follows (VQ mipmap format must always be a square).

+00h	KAMUI Texture Header (10h bytes)
+10h	Code Book (800h bytes)
+810h	2x2 mipmap Index (1h bytes)
+811h	4x4 mipmap Index (4h bytes)
+815h	8x8 mipmap Index (10h bytes)
+825h	16x16 mipmap Index (40h bytes)
+865h	32x32 mipmap Index (100h bytes)
+965h	64x64 mipmap Index (400h bytes)
+D65h	128x128 mipmap Index (1000h bytes)
+1D65h	256x256 mipmap Index (4000h bytes)
+5D65h	512x512 mipmap Index (10000h bytes)
+15D65h	1024x1024 mipmap Index (40000h bytes)

6.2.4 Palettized 4-bpp/8-bpp Format

Twiddled-NonVQ-Palettized format is usually called Palettized format.

There are two types of Palettized format: 4 bpp and 8 bpp. A system has only one palette with a total of 1024 entries. For Palettized-4 bpp, the 1024 entries are divided into 64 banks (1024 entries / 16 colors = 64 banks). For Palettized-8 bpp, the 1024 entries are divided into 4 banks (1024 entries / 256 colors = 4 banks). Each bank is not physically separated, but is created by obtaining a pointer to the 1024 entries through calculation. The 4 bpp texture and 8 bpp texture can exist together in one scene, but the overlapping entries of the 1024 entries are shared. Therefore, changing the contents of the palette affects both the 4 bpp and 8 bpp textures.

The bank of the palette can be specified in VERTEX (polygon) units. A bank number is specified by PaletteBank member of KMVERTEXCONTEXT. The entry that can actually be used is selected as follows using the palette bank number (PaletteBank) and index value (index_data) of each texel of a texture.

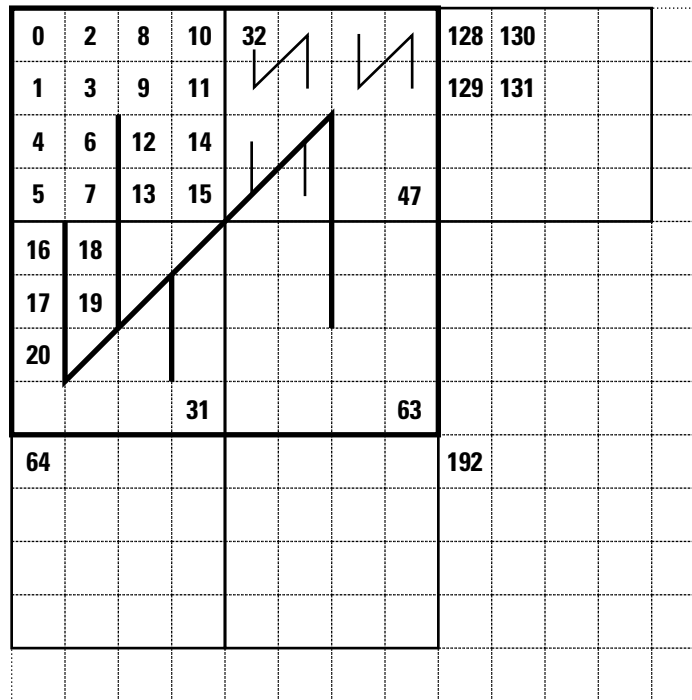
```
if (PixelFormat == 8BPP)
{
    palette_entry = (PaletteBank << 4) & 0x300 + index_data;
}

if (PixelFormat == 4BPP)
{
    palette_entry = (PaletteBank << 4) + index_data;
}
```

A value of 0 to 63 can be specified as PaletteBank in the case of 4 bpp. It is also 0 to 63 for 8 bpp. In this case, however, only the higher 2 bits of the 6 bits are valid, and only four types of values can be used: 0 (0 to 15), 16 (16 to 31), 32 (32 to 47), and 48 (48 to 63).

The format in memory is almost the same as the twiddled format. However, four dots constitute 1 (16-bit) word for 4 bpp, and four dots constitute 2 (16-bit) words for 8 bpp.

The format of a texture is as follows:



The address sequence dose not differ from that of Twiddled. This is packed in little Endian, in the order of (U=0, V=0), (U=0, V=1), (U=1, V=0), (U=1, V=1). Therefore, the address sequence is as follows in the case of 4 bpp and 8 bpp (Palettized / Palettized mipmap format must always be a square).

Texel arrangement for 4 bpp

Bits 15-12	Bits 11-8	Bits 7-4	Bits 3-0
Texel U=1 V=1	Texel U=1 V=0	Texel U=0 V=1	Texel U=0 V=0

Texel arrangement for 8 bpp

Bits 15-8	Bits 23-16
Texel U=0 V=1	Texel U=0 V=0

Palettized format in KAMUI texture format is as follows:

+00h	KAMUI Texture Header (10h bytes)
+10h	Texture Pixel Data

The size of the pixel data of a texture is as follows:

Vertical/Horizontal Size	Pixel Data Size	
	4bpp	8bpp
8x8	20h bytes	40h
16x16	80h bytes	100h
32x32	200h bytes	400h
64x64	800h bytes	1000h
128x128	2000h bytes	4000h
256x256	8000h bytes	10000h
512x512	20000h bytes	40000h
1024x1024	80000h bytes	100000h

The Palettized mipmap format in KAMUI texture format is as follows:

4bpp			8bpp	
Offset	Length	Data	Length	Offset
+00h	10h	KAMUI Texture Header	10h	+00h
+10h	1h	Dummy Zero	3h	+10h
+11h	1h	1x1 mipmap Index	1h	+13h
+12h	2h	2x2 mipmap Index	4h	+14h
+14h	8h	4x4 mipmap Index	10h	+18h
+1Ch	20h	8x8 mipmap Index	40h	+28h
+3Ch	80h	16x16 mipmap Index	100h	+68h
+BCh	200h	32x32 mipmap Index	400h	+168h
+2BCh	800h	64x64 mipmap Index	1000h	+568h
+ABCh	2000h	128x128 mipmap Index	4000h	+1568h
+2ABCh	8000h	256x256 mipmap Index	10000h	+5568h
+AABCh	20000h	512x512 mipmap Index	40000h	+15568h
+2AABCh	80000h	1024x1024 mipmap Index	100000h	+55568h
+AAABCh	4h	Dummy Zero	none	none

Four bytes of dummy data are necessary at the end of the data for mipmap of Palettized-4 bpp.

With ARC1, Palettized texture cannot be used.

6.2.5 Rectangle Format

Scan Order-Rectangle format is usually called Rectangle format. Note that, with the CLX1, Twiddled-NonVQ-Rectangle format texture also exists.

Rectangle format is a texture that can set different values for the U and V size. When this format is used, mipmap cannot be effected and the performance drops relative to that in Twiddled format.

In Rectangle format, however, access is extremely easy because pixel data are arranged in raster order (complicated addressing does not have to be performed, unlike in Twiddled format). Rectangle format can be subject to rendering and, therefore, can be used for environment mapping.

The vertical / horizontal size of the texture that can be specified in Rectangle format is 8, 16, 32, 64, 128, 256, 512, or 1024.

Rectangle format in KAMUI texture format is as follows:

+00h	KAMUI Texture Header (10h bytes)
+10h	Texture Pixel Data

Like the location of pixel data in the Windows BMP (24 bpp), 2-byte pixel data of (U=0, V=0) (lower left of texture) is first located, followed by (U=1, V=0), (U=2, V=0), ... (U=Umax, V=0), (U=0, V=1) and so on.

The size of the pixel data of a texture can be calculated as follows:

Size =

Number of horizontal texels (U-size) x Number of vertical texels (V-size) x 2 (bytes per texel)

6.2.6 Stride Format

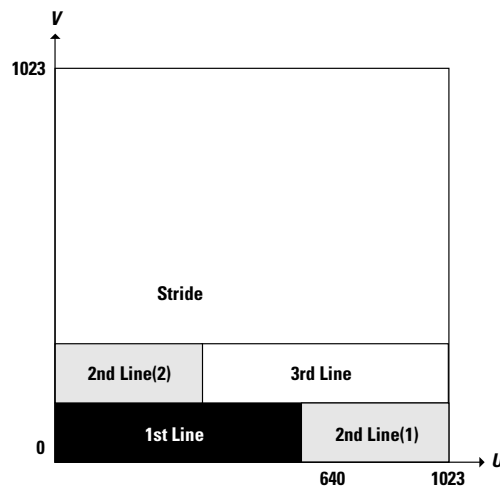
Scan Order-Stride format is usually called Stride format.

Stride format is a special type of Rectangle format. First, a global Stride value is set, after which texel is determined by using the following addressing.

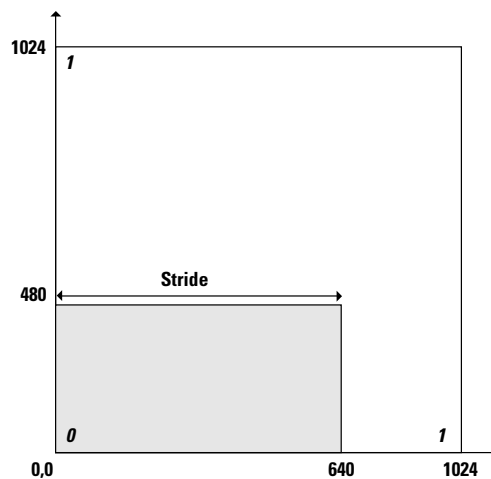
$$\text{Addr} = u + v * \text{stride}$$

Therefore, the number of horizontal texels of a texture can be varied by the Stride value. Note, however, that a Stride value must be a multiple of 32 (see the description of `kmSetStrideWidth`).

For example, when creating 640 x 480 areas in a texture of 1024 x 1024 when environment mapping is used, specify 640 as the Stride value. When rendering is performed on this texture surface by using `KmRenderTexture`, the portion on the first one line ((x,y)=(0,0)-(639,0)) on the screen is written to (U,V)=(0,0)-(639,0) of the texture surface, and the portion on the second line ((x,y)=(0,1)-(639,1)) is written to (U,V)=(640,0)-(1023,0) and (0,1)-(255,1) of the texture surface.



When performing texture mapping by using this texture, the entire screen can be pasted as a texture where (U,V)=(0.0f,0.0f)-(0.625f,0.46875f).



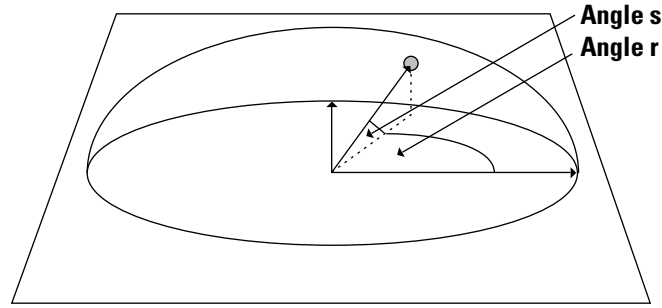
The Stride format of KAMUI is exactly the same as Rectangle format. The only difference is the flag of the texture type that is specified for nTextureType of a header.

6.2.7 BUMP-Mapping Format

With PVR2, BUMP-Mapping is implemented by the following technique. BUMP-Mapping information expresses projections and recesses on a surface by storing a normal vector as texture data, instead of pixel data of a normal texture. In addition, light source data is set to a bit of offset color, and the inner product of each dot is calculated based on this light source data. Therefore, BUMP-Mapping can be used only with a polygon with texture offset.

Within PVR2, a polar coordinate system is used for calculating BUMP-Mapping. The normal vector stored in a texture is expressed as follows:

$$\begin{aligned} \chi_s &= \cos(s')\cos(r') \\ \gamma_s &= \sin(s') \\ Z_s &= \cos(s')\sin(r') \end{aligned} \quad \text{where} \quad \begin{aligned} s' &= \pi/2 \frac{s}{256} \\ r' &= \pi \frac{r}{256} \end{aligned}$$



s and r are stored in the texture. In this case, s and r are 8-bit data and indicate an angle of elevation and azimuth angle of the BUMP data in the texture. Since the texture of the PVR2 is 16 bits long, one set of s and r exists per texel.

The light source vector for the polygon to which this texture is pasted is also expressed by using polar coordinates.

$$\begin{aligned} \chi_l &= \cos(t')\cos(q') \\ \gamma_l &= \sin(t') \\ Z_l &= \cos(t')\sin(q') \end{aligned}$$

In this case, t and q are expressed as follows, like the normal vector in a texture:

$$\begin{aligned} t' &= \pi/2 \frac{t}{256} \\ q' &= \pi \frac{q}{256} \end{aligned}$$

In introducing this light source vector to a polygon, a Scale Factor is introduced. This Scale Factor is hereafter called Strength because it indicates the intensity of the light source.

By using Strength, the following K1, K2, and K3 are calculated.

$$\begin{aligned}k_1 &= 1 - strength \\k_2 &= strength.\sin(t') \\k_3 &= strength.\cos(t')\end{aligned}$$

Strength and K value are 8-bit values normalized to 1.

To give this light source information to a polygon, the following 32-bit offset color data is used.

Base Color: xRGB			
K ₁	K ₂	K ₃	q'

To use a Floating Color VertexType such as VertexType5, the above data is normalized to 1 and input. To use Packed Color VertexType such as VertexType4, it must be converted into 8-bit data and packed.

The ultimate brightness of each texel is calculated from the above texture normal vector and light source normal vector by the inner product.

$$\text{DotProduct} = \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} \cdot \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = x_s x_l + y_s y_l + z_s z_l$$

No further textures can be pasted to a polygon on which BUMP-Mapping has been performed. To paste a texture, the fact that translucent auto sort is GreaterEqual with the CLX1 is used and translucent polygon at the same coordinates as the BUMP-Mapped polygon are registered in duplicate. In this way, a texture can be pasted to the BUMP-Mapped polygon.

6.2.8 Kamui Bit Map Format

Because KAMUI creates a texture on which mipmap and dither have automatically been performed when a texture is read by using `kmLoadTexture` API, it defines a texture in KAMUI bit map format (KAMUI-BMP)" as its input format. The pixel format of this texture is `ABGR8888` (note that the location is not `ARGB`) in conformance with pixel data in Windows BMP format (24 bpp).

The KAMUI bit map format in KAMUI texture format is as follows:

+00h	KAMUI Texture Header (10h bytes)
+10h	U=0,V=0 Pixel Data (Alpha) (1h byte)
+11h	U=0,V=0 Pixel Data (Blue) (1h byte)
+12h	U=0,V=0 Pixel Data (Green) (1h byte)
+13h	U=0,V=0 Pixel Data (Red) (1h byte)
+14h	U=1,V=0 Pixel Data (Alpha)(1h byte)
	:
	:

In the same manner as the Windows BMP (24 bpp) format, pixel data of 4 bytes of (U=0, V=0) (lower left of texture) are located first, followed by (U=1, V=0), (U=2, V=0), ... (U=Umax, V=0), (U=0, V=1), and so on.

For automatic creation of mipmap and dither, a work area of the same capacity as in the Twiddled format is necessary in stack. Therefore, the vertical and horizontal size of a texture in bit map format handled by KAMUI is up to 512 texels

KAMUI bit map texture is converted into Twiddled/Twiddled Mipmap when it is read by `kmLoadTexture`. The pixel format at this time is converted into `KM_TEXTURE_ARGB1555`, `KM_TEXTURE_RGB565`, or `KM_TEXTURE_ARGB4444` in accordance with the setting of the surface at the load destination. When creating the load destination texture surface of KAMUI bit map texture by using `kmCreateTextureSurface`, specify `KM_TEXTURE_TWIDDLED` or `KM_TEXTURE_TWIDDLED_MM` as a category code, and `KM_TEXTURE_ARGB1555`, `KM_TEXTURE_RGB565`, or `KM_TEXTURE_ARGB4444` as a pixel format.



KAMUI

Index

B

bColorClamp	KAM-44
bDCalcExact	KAM-40
bDSTSel	KAM-42
bIgnoreTextureAlpha	KAM-42
bSRCSel	KAM-42
bSuperSample	KAM-43
bUseAlpha	KAM-42
bUseSpecular	KAM-42
bZWriteDisable	KAM-41

C

ClampUV	KAM-43
ColorType	KAM-39
CullingMode	KAM-40

D

DepthMode	KAM-39
DSTBlendingMode	KAM-41

F

fboundingBoxXmax	KAM-45
fboundingBoxXmin	KAM-45
fBoundingBoxYmax	KAM-45
fBoundingBoxYmi	KAM-45
FilterMode	KAM-43
FlipUV	KAM-43
FogMode	KAM-42

K

KM_DECAL_ALPHA	KAM-44
KM_MODULATE	KAM-44
KM_MODULATE_ALPHA	KAM-44
kmActivateFrameBuffer	KAM-20
kmChangeContextClampUV	KAM-55
kmChangeContextColorClamp	KAM-55
kmChangeContextColorType	KAM-54
kmChangeContextCullingMode	KAM-54
kmChangeContextDepthMode	KAM-54
kmChangeContextDSTBlendMode	KAM-54
kmChangeContextFilterMode	KAM-55
kmChangeContextFlipUV	KAM-54
kmChangeContextFogMode	KAM-54
kmChangeContextPaletteBank	KAM-55
kmChangeContextSRCBlendMode	KAM-54
kmChangeContextSuperSample	KAM-55
kmChangeContextTextureShadingMode	KAM-55
kmChangeContextZWriteDisable	KAM-54
kmChangeDisplayFilterMode	KAM-9
kmChangeVertexOffset	KAM-78
kmChangeVertexPCW	KAM-79
kmCreateCombinedTextureSurface	KAM-14
kmCreateContiguousTextureSurface	KAM-15
kmCreateFrameBufferSurface	KAM-11, KAM-12, KAM-20, KAM-99
kmCreateTABuffer	KAM-57, KAM-65, KAM-66
kmCreateTextureSurface	KAM-12, KAM-14, KAM-15, KAM-85, KAM-90, KAM-91, KAM-92, KAM-93, KAM-95, KAM-110, KAM-111, KAM-112, KAM-127
kmCreateVertexBuffer	KAM-65
kmDiscardVertexBuffer	KAM-66
kmFlipFrameBuffer	KAM-21
kmFlushVertexBuffer	KAM-80
kmFreeTexture	KAM-96
kmGarbageCollectTexture	KAM-99
kmGetCurrentScanline	KAM-84
kmGetCurrentVertexOffset	KAM-77
kmGetFreeTextureMem	KAM-97
kmGetTexture	KAM-98
kmGetVersionInfo	KAM-88
kmInitDevice	KAM-8
kmLoadTexture	KAM-90
kmLoadVQCodebook	KAM-91
kmLoadYUVTexture	KAM-95
kmProcessVertexRenderState	KAM-36
kmReloadMipmap	KAM-92

kmRender	KAM-73
kmRenderDirect	KAM-74
kmRenderTexture	KAM-74
kmRenderTextureDirect	KAM-75
kmResetRenderer	KAM-35
kmSetAlphaThreshold	KAM-19
kmSetAutoSortMode	KAM-31
kmSetBackGroundPlane	KAM-30
kmSetBackGroundRenderState	KAM-29
kmSetBorderColor	KAM-29
kmSetCheapShadowMode	KAM-33
kmSetColorClampValue	KAM-23
kmSetCullingRegister	KAM-22
kmSetDisplayMode	KAM-9
kmSetEndOfListDirect	KAM-73
kmSetEndOfVertexCallback	KAM-87
kmSetEORCallback	KAM-81
kmSetFogDensity	KAM-25
kmSetFogTable	KAM-26
kmSetFogTableColor	KAM-24
kmSetFogVertexColor	KAM-24
kmSetFramebufferTexture	KAM-17
kmSetGlobalClipping	KAM-49
kmSetHSyncCallback	KAM-83
kmSetModifierRenderState	KAM-48
kmSetPaletteData	KAM-28
kmSetPaletteMode	KAM-27
kmSetPixelClipping	KAM-32
kmSetStrideWidth	KAM-17
kmSetStripLength	KAM-50
kmSetStripOverRunCallback	KAM-86
kmSetTexOverflowCallback	KAM-85
kmSetUserClipping	KAM-51
kmSetUserClippingDirect	KAM-53
kmSetVertex	KAM-69
kmSetVertexDirect	KAM-71
kmSetVertexRenderState	KAM-47
kmSetVSyncCallback	KAM-82
kmSetWaitVsyncCount	KAM-34
kmStartVertexStrip	KAM-67
kmStartVertexStripDirect	KAM-68
kmStopDisplayFrameBuffer	KAM-88
kmuCreateTwiddledTexture	KAM-102
kmUseAnotherModifier	KAM-76

L

ListTypeKAM-39, KAM-50

M

MipMapAdjust KAM-44

ModifierInstruction KAM-45

P

PaletteBank KAM-45

ParamType KAM-38

pTextureSurfaceDesc KAM-45

pVertexContext KAM-46

S

ScreenCoordination KAM-40

SelectModifier KAM-41

ShadingMode KAM-40

SRCBleedingMode KAM-41

T

TextureShadingMode KAM-44

U

UVFormat KAM-39