Sega Dreamcast™

Kamui2 Function Reference

Kamui2 Function Reference TOC

1. Functions for Getting Version Information KMT–1

kmGetVersionInfo .... Obtains the Version Information. KMT–1

2. Functions for Initializing Devices KMT–3

kmInitDevice .... Initializes the hardware device. KMT–3
kmUnloadDevice .... Unloads HW device. KMT–4

3. Functions for Controlling Display KMT–5

kmAdjustDisplayCenter .... Adjusts the display position of the frame buffer on the screen. KMT–5
kmBlankScreen .... Stops the screen display of the frame buffer, and performs blanking. KMT–6
kmChangeDisplayAntialiasMode .... Sets the anti-aliasing filter. KMT–7
kmChangeDisplayDitherMode .... Sets up dither. KMT–8
kmGetCurrentDisplaySurface .... Returns the pointer to the currently displayed surface. KMT–9
kmGetCurrentScanline .... Reads the current H-Sync line. KMT–10
kmGetDisplayColorMode .... Returns the color mode. KMT–11
kmGetDisplayFilterMode .... Returns the filter mode. KMT–12
kmGetDisplayInfo .... Returns information about the display. KMT–13
kmGetDisplaySize .... Returns size of the screen. KMT–14
kmGetGunTriggerPos .... Returns the position where the trigger of Gun Peripheral is pressed. KMT–15
kmGetVBlankCount .... Returns the V-BLANK count from the beginning of display. KMT–16
kmSetDisplayMode .... Sets the display mode of the frame buffer. KMT–17
kmSetHSyncLine .... Specifies the display line on which an interrupt is caused. KMT–19
kmWaitVBlank .... Waits for V-Blank. KMT–20

4. Functions for Controlling Buffer KMT–21

kmSetSystemConfiguration .... Controls buffers (Vertex/Frame/Native) used in Kamui2. .... KMT–21

5. Functions for Global Setting KMT–29

kmConvertFogDensity ..... Converts fog coefficient. KMT–29
kmGenerateFogTable ..... Specifies a fog table and coefficient to table fog. KMT–30
kmGetSystemMetrics ..... KAMUI2 international information acquisition. KMT–34
kmResetRenderer ..... Resets the rendering pipeline through software. KMT–36
kmSetAutoSortMode ..... Sets auto-sort mode. KMT–37
kmSetBackGround ..... Specifies the vertex and CONTEXT for use in the background. KMT–38
kmSetBorderColor ..... Sets the border (outside of the display screen) color. KMT–39
kmSetCheapShadowMode ..... Sets the cheap shadow mode. KMT–40
kmSetColorClampMax ..... Specifies the color clamp maximum value. KMT–41
kmSetColorClampMaxValue ..... Specifies the color clamp maximum value. KMT–42
kmSetColorClampMin ..... Specifies the minimum color clamp value. KMT–43
kmSetColorClampMinValue ..... Specifies the minimum color clamp value. KMT–44
kmSetColorClampValue ..... Sets the color clamp value. KMT–45
kmSetCullingRegister ..... Specifies a threshold value for culling small polygons. KMT–46
kmSetFogDensity ..... Specifies a coefficient to table fog. KMT–47
kmSetFogTable ..... Sets the fog table. KMT–48
kmSetFogTableColor ..... Specifies a table fog color. KMT–49
kmSetFogVertexColor ..... Specifies a vertex fog color. KMT–50
kmSetGlobalClipping ..... Sets global clipping. KMT–51
kmSetPaletteBank ..... Changes entry of palette, in banks. KMT–52
kmSetPaletteBankData ..... Rewrites part of the on-chip palette data. KMT–53
kmSetPaletteData ..... Sets the on-chip palette data. KMT–54
kmSetPaletteMode ..... Sets the on-chip palette mode. KMT–57
kmSetPixelClipping ..... Specifies pixel-unit clipping. KMT–58
kmSetPunchThroughThreshold ..... Sets PunchThrough polygon (alpha) threshold. KMT–59
kmSetStrideWidth ..... Specifies the stride size for stride texture. KMT–60

(For compatibility purpose) KMT–61

kmSetBackGroundPlane ..... Sets the background plane. KMT–61
kmSetBackGroundRenderState ..... Registers the rendering parameter of the background plane. KMT–62
kmUseAnotherModifier ..... Sets modifier volume list. KMT–63

6. Vertex Definition Functions KMT–65

Functions for controlling rendering parameter KMT–65

kmGenerateStripHead ..... Generate and build Rendering Parameter (KMSTRIPHEAD). KMT–65
kmGenerateStripHead00 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType00. ..... KMT–67
kmGenerateStripHead01 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType01. ..... KMT–68
kmGenerateStripHead02 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType02. ..... KMT–69
kmGenerateStripHead03 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType03. ..... KMT–70
kmGenerateStripHead04 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType04. ..... KMT–72
kmGenerateStripHead05 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType05. ..... KMT–74
kmGenerateStripHead06 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType06. ..... KMT–76
kmGenerateStripHead07 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType07. ..... KMT–78
kmGenerateStripHead08 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType08. ..... KMT–80
kmGenerateStripHead09 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType09. ..... KMT–82
kmGenerateStripHead10 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType10. ..... KMT–84
kmGenerateStripHead11 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType11. ..... KMT–86
kmGenerateStripHead12 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType12. ..... KMT–88
kmGenerateStripHead13 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType13. ..... KMT–90
kmGenerateStripHead14 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType14. ..... KMT–92
kmGenerateStripHead15 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType15. ..... KMT–94
kmGenerateStripHead16 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType16. ..... KMT–95
kmGenerateStripHead17 ..... Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType17. ..... KMT–97
kmInitStripContext ..... Initializes StripContext (KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT). KMT–98
kmRegisterStripContext ..... Registers with the system KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT.(user-defined) KMT–101

(For compatibility purpose) KMT–102

kmProcessVertexRenderState .... Sets rendering parameters. KMT–102
kmRegisterDefaultContext .... Creates user-defined flags for initializing StripContext. KMT–103
kmRegisterDefaultStripContext .... Registers KMSTRIPCONTEXT in the system (User defined). KMT–104
kmSetModifierRenderState .... Registers with the system as the second rendering parameter. KMT–105
kmSetStripHead .... Registers with the system as parameters to be used for rendering. (KMSTRIPHEAD) KMT–106
kmSetVertexRenderState .... Registers with the system as parameters to be used for rendering. KMT–107

(Description of the structures to be used) KMT–108

KMSTRIPCONTEXT .... KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT Structure KMT–108
KMVERTEXCONTEXT .... KMVERTEXCONTEXT Structure (For compatibility purpose) KMT–117

7. Functions for Controlling Vertex Registration KMT–127

kmBeginPass .... Starts Pass. KMT–127
kmBeginScene .... Starts the Scene. KMT–128
kmContinuePass .... Continues Pass. KMT–129
kmEndPass .... Ends Pass. KMT–130
kmEndScene .... Ends the Scene. KMT–131
kmEndStrip .... Ends Strip (direct transfer). KMT–132
kmSetUserClipping .... Set UserClippingParameter. KMT–133
kmSetVertex .... Sends VertexParameter. KMT–135
kmStartStrip .... Performs start of Strip (direct transfer). KMT–136

(For compatibility purpose) KMT–137

kmEndVertexStrip .... Ends Strip. KMT–137
kmStartVertexStrip .... Starts Strip. KMT–138

8. Functions for Update KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT KMT–139

kmChangeStripBlendingMode .... Changes BlendingMode. KMT–139
kmChangeStripClampUV .... Changes ClampUV. KMT–141
kmChangeStripColorClamp .... Changes ColorClamp. KMT–142
kmChangeStripCullingMode .... Changes CullingMode. KMT–143
kmChangeStripDCalcControl .... Changes DCalcControl. KMT–144
kmChangeStripDepthCompareMode .... Changes DepthCompareMode. KMT–145
kmChangeStripDSTSelect .... Changes DSTSelect. KMT–146
kmChangeStripFaceColor .... Changes FaceColor. KMT–147
kmChangeStripFaceOffsetColor .... Changes FaceOffsetColor. KMT–148
kmChangeStripFilterMode .... Changes FilterMode. KMT–149
kmChangeStripFlipUV .... Changes FlipUV. KMT–150
kmChangeStripFogMode .... Changes FogMode. KMT–151
kmChangeStripGouraud .... Changes Gouraud. KMT–152
kmChangeStripIgnoreTextureAlpha .... Changes IgnoreTextureAlpha. KMT–153
kmChangeStripIgnoreTexureAlpha .... Changes IgnoreTexureAlpha. KMT–154
kmChangeStripIntensityMode .... Changes IntensityMode. KMT–155
kmChangeStripListType .... Changes ListType. KMT–156
kmChangeStripMipmapAdjust .... Changes MipmapAdjust. KMT–158
kmChangeStripModifierInstruction .... Changes ModifierInstruction. KMT–160
kmChangeStripOffset .... Changes Offset. KMT–161
kmChangeStripPaletteBank .... Changes PaletteBank. KMT–162
kmChangeStripShadowMode .... Changes ShadowMode. KMT–163
kmChangeStripSpriteBaseColor .... Changes SpriteBaseColor. KMT–164
kmChangeStripSpriteOffsetColor .... Changes SpriteOffsetColor. KMT–165
kmChangeStripSRCSelect .... Changes SRCSelect. KMT–166
kmChangeStripSuperSampleMode .... Changes SuperSampleMode. KMT–167
kmChangeStripTextureAddress .... Changes TextureAddress. KMT–168
kmChangeStripTextureShadingMode .... Changes TextureShadingMode. KMT–170
kmChangeStripTextureSurface .... Changes TextureSurface. KMT–171

kmChangeStripUseAlpha ....Changes UseAlpha. KMT–172
kmChangeStripUserClipMode..........................Changes UserClipMode. KMT–173
kmChangeStripZWriteDisable..........................Changes ZWriteDisable. KMT–174

9. Functions for Update Rendering Parameter Registration KMT–175

(For compatibility purpose) KMT–175

kmChangeContextBlendingMode ....Changes BlendingMode. KMT–175
kmChangeContextClampUV ....Changes ClampUV. KMT–177
kmChangeContextColorClamp.........................Changes ColorClamp. KMT–179
kmChangeContextCullingMode.......................Changes CullingMode. KMT–180
kmChangeContextDCalcControl......................Changes DCalcControl. KMT–181
kmChangeContextDepthCompareMode.........Changes DepthCompareMode. KMT–182
kmChangeContextDSTSelect............................Changes DSTSelect. KMT–183
kmChangeContextFaceColor............................Changes FaceColor. KMT–184
kmChangeContextFaceOffsetColor..................Changes FaceOffsetColor. KMT–185
kmChangeContextFilterMode...........................Changes FilterMode. KMT–186
kmChangeContextFlipUV ....Changes FlipUV. KMT–187
kmChangeContextFogMode ....Changes FogMode. KMT–188
kmChangeContextGouraud ....Changes Gouraud. KMT–189
kmChangeContextIgnoreTextureAlpha ....Changes IgnoreTextureAlpha. KMT–190
kmChangeContextIgnoreTexureAlpha ....Changes IgnoreTexureAlpha. KMT–191
kmChangeContextIntensityMode ....Changes IntensityMode. KMT–192
kmChangeContextListType...............................Changes ListType. KMT–193
kmChangeContextMipmapAdjust ....Changes MipmapAdjust. KMT–195
kmChangeContextModifierInstruction ....Changes ModifierInstruction. KMT–197
kmChangeContextOffset...................................Changes Offset. KMT–198
kmChangeContextPaletteBank ....Changes PaletteBank. KMT–199
kmChangeContextShadowMode......................Changes ShadowMode. KMT–200
kmChangeContextSpriteBaseColor..................Changes SpriteBaseColor. KMT–201
kmChangeContextSpriteOffsetColor ....Changes SpriteOffsetColor. KMT–202
kmChangeContextSRCSelect............................Changes SRCSelect. KMT–203
kmChangeContextSuperSampleMode.............Changes SuperSampleMode. KMT–204
kmChangeContextTextureAddress..................Changes TextureAddress. KMT–205
kmChangeContextTextureShadingMode........Changes TextureShadingMode. KMT–206
kmChangeContextTextureSurface....................Changes TextureSurface. KMT–207
kmChangeContextUseAlpha............................Changes UseAlpha. KMT–208
kmChangeContextUserClipMode ....Changes UserClipMode. KMT–209
kmChangeContextZWriteDisable.....................Changes ZWriteDisable. KMT–210

10. Functions for Controlling Rendering KMT–211

kmGetRenderStatus..Gets status of rendering that has been executed. KMT–211
kmRender..Starts rendering. KMT–212
kmRenderTexture ....Begins rendering for a texture mamory. KMT–213
kmQueryFinishLastTextureDMA.....................Checks for previous texture load DMA transfer end. KMT–214

11. Functions for Controlling Texture KMT–215

kmCreateCombinedTextureSurface.................Secures a VQ texture surface. KMT–215
kmCreateContiguousTextureSurface...............Secures two or more texture surfaces simultaneously. KMT–217
kmCreateFixedTextureArea ....Secures the fixed texture area. KMT–218
kmCreateTextureSurface...................................Secures a texture surface. KMT–219
kmFreeFixedTextureArea ....Releases a fixed texture area. KMT–221
kmFreeTexture..Releases a texture surface. KMT–222
kmGarbageCollectTexture.................................Performs garbage collection of frame buffer memory. KMT–223
kmGetCurrentTextureStatus ....Reads out information about the frame buffer memory management. KMT–224
kmGetFreeTextureMem ....Obtains the available texture memory space. KMT–225
kmGetTexture...Reads the texture in texture memory. KMT–226
kmLoadRectangleTexturePart...........................Writes the rectangular area (partial) of texture data in the main memory. KMT–227
kmLoadTexture ....Loads texture data. KMT–230

kmLoadTextureBlock.. Loads texture data blocks. KMT–231
kmLoadTexturePart.. Loads part of texture data. KMT–233
kmLoadVQCodebook.. Re-reads the code book portion of VQ texture. KMT–235
kmLoadYUVTexture.. Reads the YUV-format texture data. KMT–237
kmReLoadMipmap.. Overwrites the mipmap texture. KMT–240
kmSetStrideWidth.. Specifies the stride size for stride texture. KMT–244

12. Functions for Controlling Callback KMT–245

kmSetAssertCallback.. Registers assert callback function. KMT–245
kmSetEndOfVertexCallback.. Specifies the callback function to be called at the end of transfer of the data of one scene. KMT–247
kmSetEndOfYUVCallback.. Registers a callback function to use for the YUV termination interrupt. KMT–248
kmSetEORCallback.. Specifies a rendering end callback function. KMT–249
kmSetFatalErrorCallback.. Registers callback function for FatalError. KMT–250
kmSetHSyncCallback.. Specifies the callback function to be called at an entry into the horizontal flyback segment (Hsync). KMT–251
kmSetStripOverRunCallback.. 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 a strip buffer. KMT–252
kmSetTexOverflowCallback.. Registers the callback function to call when there is no texture memory left. KMT–253
kmSetVSyncCallback.. Specifies the callback function to be called at an entry into the vertical flyback segment (Vsync). KMT–254
kmSetWaitVsyncCallback.. Specifies a callback function to be called during a Vsync wait state. KMT–255

13. Functions for Controlling Utility-Related Tasks KMT–257

kmuCalculateKamuiWorkareaSize.. Calculate KAMUI work area size. KMT–257
kmuConvertFBtoBMP.. Converts rectangle format to Windows BMP format. KMT–259
kmuCreateTwiddledTexture.. Converts KAMUI bit map format to Twiddled format. KMT–260
kmuCreateTwiddledTextureEx.. Converts KAMUI bit map format to Twiddled format. KMT–262

(For compatibility purpose) KMT–264

kmuCheckPassTable.. Checks VERTEXCONTEXT. KMT–264
kmuConvertStripContext.. Converts to KMSTRIPCONTEXT structure. KMT–265
kmuGeneratePassTable.. Generates multipass VERTEXCONTEXT automatically. KMT–266

1. Functions for Getting Version Information

kmGetVersionInfo

Obtains the Version Information.

Format

```
KMSTATUS KMAPI
kmGetVersionInfo(
          OUT     PKMVERSIONINFO   pVersionInfo
          );
```

Description

This function obtains the version information of the library.

For the contents of the version information structure, see the structure list.

Parameters

`pVersionInfo`(output)	This parameter indicates a pointer to the `KMVERSIONINFO` structure allocated in advance.

Return Values

`KMSTATUS_SUCCESS`	Success

2. Functions for Initializing Devices

kmInitDevice

Initializes the hardware device.

Format

```
KMSTATUS KMAPI
kmInitDevice(
        IN  KMDEVICE   nDevice
);
```

Description

This function initializes the hardware. It outputs video signals to produce a blank screen.

Parameters

`nDevice`(input)	This parameter specifies a hardware mode by selecting one from: `KM_DREAMCAST`...DREAMCAST mode `KM_NAOMI` ...NAOMI mode

Return values

`KMSTATUS_SUCCESS`	Success

kmUnloadDevice

Unloads HW device.

Format

```
KMSTATUS KMAPI
kmUnloadDevice( KMVOID );
```

Description

This function unloads HW device.

Parameters

None

Return values

`KMSTATUS_SUCCESS`	Success

3. Functions for Controlling Display

kmAdjustDisplayCenter

Adjusts the display position of the frame buffer on the screen.

Format

```
KMSTATUS KMAPI
kmAdjustDisplayCenter(
                     IN  KMINT32    nXAdjust,
                     IN  KMINT32    nYAdjust
      );
```

Description

This function adjusts the display position of the frame buffer on the screen.

Parameters

`nXAdjust`(input)	Value to adjust the screen drawing position in the horizontal direction
`nYAdjust`(input)	Value to adjust the screen drawing position in the vertical direction

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_OUT_OF_RANGE`	A value outside the valid range is used.

kmBlankScreen

Stops the screen display of the frame buffer, and performs blanking.

FormatFormat

```
KMSTATUS KMAPI
kmBlankScreen(
                              IN   KMBOOLEAN       bBlanking
         );
```

Description

This function stops the screen display of the frame buffer, and performs blanking.
(For each V-Sync Callback, this function can only be called once.)

Parameters

`bBlanking`(input)

This parameter specifies whether to perform blanking of the screen, as follows:

`KM_TRUE`	Starts blanking.
`KM_FALSE`	Cancels blanking.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeDisplayAntialiasMode

Sets the anti-aliasing filter.

Format

```
KMSTATUS KMAPI
kmChangeDisplayAntialiasMode(
                IN  KMBOOLEAN       bEnable
);
```

Description

This function changes the enable/disable status of the anti-aliasing filter set at the `kmSetDisplayMode` function.

Parameters

`bEnable`(input)	This parameter specifies whether the anti-aliasing filter can be used or not. When the anti-aliasing filter is enabled, the operation speed may decrease.
`KM_TRUE`	Enables use of Anti Aliasing Filter.
`KM_FALSE`	Disables use of Anti Aliasing Filter.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_DISPLAY_MODE`	Invalid display mode. A display mode that does not match that specified during initialization was specified.

kmChangeDisplayDitherMode

Sets up dither.

Format

```
KMSTATUS KMAPI
kmChangeDisplayDitherMode(
                      IN  KMBOOLEAN      bEnable,
       );
```

Description

This function changes the enable/disable status of the dither set at the kmSetDisplayMode function.

Parameters

`bEnable`(input)	This parameter determines whether to use dither when the PowerVR writes the rendering result to the 16-bit frame buffer. If the frame buffer for rendering is RGB888 or ARGB8888, this flag will be ignored.
`KM_TRUE`	Uses dither.
`KM_FALSE`	Does not use dither.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_DISPLAY_MODE`	Invalid display mode. A display mode that does not match that specified during initialization was specified.

kmGetCurrentDisplaySurface

Returns the pointer to the currently displayed surface.

Format

```
KMSTATUS KMAPI
kmGetCurrentDisplaySurface(
                        OUT     PKMSURFACEDESC pDesc
       );
```

Description

This function returns the pointer to the currently displayed surface.

Parameters

`pDesc`(output)	This parameter is a pointer to the currently displayed surface.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetCurrentScanline

Reads the current H-Sync line.

Format

```
KMSTATUS KMAPI
kmGetCurrentScanline(
                        OUT PKMINT32     pScanline
       );
```

Description

This function reads the current H-Sync line.

Parameters

`pScanline`(output)	This parameter is a pointer to KMINT32 where the current H-Sync line is stored.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetDisplayColorMode

Returns the color mode.

Format

```
KMSTATUS KMAPI
kmGetDisplayColorMode(
                       OUT PKMDWORD        pDisplayColorMode,
                       IN  PKMSURFACEDESC  pSurfaceDesc
       );
```

Description

This function returns the color mode.

Parameters

`pDisplayColorMode`(output)	
`pSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetDisplayFilterMode

Returns the filter mode.

Format

```
KMSTATUS KMAPI
kmGetDisplayFilterMode(
                       OUT PKMDWORD       pDisplayFilterMode,
                       IN  PKMSURFACEDESC  pSurfaceDesc
       );
```

Description

This function returns the filter mode.

Parameters

`pDisplayFilterMode`(output)	
`pSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetDisplayInfo

Returns information about the display.

Format

```
KMSTATUS KMAPI
kmGetDisplayInfo(
                        OUT PKMDISPLAYINFO  pDisplayInfo,
                        IN  PKMSURFACEDESC  pSurfaceDesc
       );
```

Description

This function returns information about the display.

Parameters

`pDisplayInfo`(output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetDisplaySize

Returns size of the screen.

Format

```
KMSTATUS KMAPI
kmGetDisplaySize(
                        OUT     PKMINT32        pWidth,
                        OUT     PKMINT32        pHeight
                );
```

Description

This function returns size of the screen.

Parameters

`pWidth`(output)	`pWidth` pointer
`pHeight`(output)	`pHeight` pointer

Return values

`KMSTATUS_SUCCESS`	Success

kmGetGunTriggerPos

Returns the position where the trigger of Gun Peripheral is pressed.

Format

```
KMSTATUS KMAPI
kmGetGunTriggerPos(
                        OUT PKMDWORD     pHPos,
                        OUT PKMDWORD     pVPos
       );
```

Description

This function returns the position where the trigger of Gun Peripheral is pressed.

Parameters

`pHPos`(output)	Horizontal position from H-blank OUT
`pVPos`(output)	Vertical position from V-blank OUT

Return values

`KMSTATUS_SUCCESS`	Success

kmGetVBlankCount

Returns the V-BLANK count from the beginning of display.

Format

```
KMSTATUS KMAPI
kmGetVBlankCount(
                        OUT PKMDWORD  pVBlankCount
       );
```

Description

Returns the V-Blank count from the start of display

Parameters

`pBlankCount` (output)	Number of V-Blanks from the start of display

Return values

`KMSTATUS_SUCCESS`	Success

kmSetDisplayMode

Sets the display mode of the frame buffer.

Format

```
KMSTATUS KMAPI
kmSetDisplayMode(
                        IN  KMDISPLAYMODE   nDisplayMode,
                        IN  KMBPPMODE       nBpp,
                        IN  KMBOOLEAN       bDither,
                        IN  KMBOOLEAN       bAntiAlias
                );
```

Description

This function sets the display mode of the frame buffer.

Parameters

`nDisplayMode`(input) This parameter specifies a display mode.

Type	Mode	Size	scan	Freq	Remarks
VGA	KM_DSPMODE_VGA	640x480	Non-interlace	60Hz	
	KM_DSPMODE_VGA640x240	640x240	Non-interlace	60Hz	
	KM_DSPMODE_VGA320x480	320x480	Non-interlace	60Hz	
	KM_DSPMODE_VGA320x240	320x240	Non-interlace	60Hz	
NTSC	KM_DSPMODE_NTSCNI320x240	320x240	Non-interlace	60Hz	
	KM_DSPMODE_NTSCI320x240	320x240	Interlace	30Hz	
	KM_DSPMODE_NTSCNI320x480FF	320x240	Pseudo non-interlace	60Hz	
	KM_DSPMODE_NTSCNI320x480	320x240	Pseudo non-interlace	60Hz	Flicker-free
	KM_DSPMODE_NTSCI320x480	320x240	Interlace	30Hz	
	KM_DSPMODE_NTSCNI640x240	640x240	Non-interlace	60Hz	
	KM_DSPMODE_NTSCNI640x480	640x240	Interlace	30Hz	
	KM_DSPMODE_NTSCI640x240	640x480	Pseudo non-interlace	60Hz	
	KM_DSPMODE_NTSCNI640x480FF	640x480	Pseudo non-interlace	60Hz	Flicker-free
	KM_DSPMODE_NTSCI640x480	640x480	Interlace	30Hz	

Type	Mode	Size	scan	Freq	Remarks
PAL	KM_DSPMODE_PALNI320x240	320x240	Non-interlace	50Hz	
	KM_DSPMODE_PALI320x240	320x240	Interlace	50Hz	
	KM_DSPMODE_PALNI320x480	320x480	Pseudo non-interlace	50Hz	
	KM_DSPMODE_PALNI320x480FF	320x480	Pseudo non-interlace	50Hz	Flicker-free
	KM_DSPMODE_PALI320x480	320x480	Interlace	25Hz	
	KM_DSPMODE_PALNI640x240	640x240	Non-interlace	50Hz	
	KM_DSPMODE_PALI640x240	640x240	Interlace	25Hz	
	KM_DSPMODE_PALNI640x480	640x480	Pseudo non-interlace	50Hz	
	KM_DSPMODE_PALNI640x480FF	640x480	Pseudo non-interlace	50Hz	Flicker-free
	KM_DSPMODE_PALI640x480	640x480	Interlace	25Hz	

`nBpp`(input)	This parameter specifies a frame buffer color mode, using a predefined constant listed below.

Symbol	Color Mode	Bitdepth	Bit order
KM_DSPBPP_RGB565	RGB565	16	**** **** **** ****
KM_DSPBPP_RGB555	RGB555	16	**** **** **** ****
KM_DSPBPP_ARGB1555	ARGB1555	16	**** **** **** ****
KM_DSPBPP_RGB888	RGB888	24	**** **** **** **** **** ****
KM_DSPBPP_ARGB8888	ARGB8888	32	**** **** **** **** **** **** **** ****

`bDither`(input)	This parameter determines whether dithering is enabled or not when the PowerVR writes the results of rendering to the 16-bit frame buffer. This flag is ignored if the rendering destination frame buffer is 24 bits/32 bits (= RGB888 or ARGB8888).
`KM_TRUE`	Use dithering.
`KM_FALSE`	Do not use dithering.
`bAntiAlias`(input)	This parameter determines whether to use an antialiasing filter. Use of the antialiasing filter may reduce the operation speed.
`KM_TRUE`	Use Anti Aliasing Filter.
`KM_FALSE`	Do not use Anti Aliasing Filter.

Return Values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_DISPLAY_MODE`	Invalid display mode. A display mode that does not match that specified during initialization was specified.

kmSetHSyncLine

Specifies the display line on which an interrupt is caused.

Format

```
KMSTATUS KMAPI
kmSetHSyncLine(
                       IN  KMINT32     nInterruptLine
       );
```

Description

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

Parameters

`nInterruptLine`(input)	This parameter specifies the line on which an interrupt is caused. Specify a value within the range of 0 to 240/480.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_ILLEGAL_PARAMETER`	Invalid parameter

kmWaitVBlank

Waits for V-Blank.

Format

```
KMSTATUS KMAPI
kmWaitVBlank( KMVOID );
```

Description

This function waits for V-Blank.

Parameters

None

Return values

`KMSTATUS_SUCCESS`	Success

4. Functions for Controlling Buffer

kmSetSystemConfiguration

Controls buffers (Vertex/Frame/Native) used in Kamui2.

Format

```
KMSTATUS KMAPI
kmSetSystemConfiguration(
        IN OUT PKMSYSTEMCONFIGSTRUCT  pSystemConfigStruct
    );
```

Description

This function controls buffers (Vertex/Frame/Native) used in Kamui2.

This function sets the KAMUI system configuration according to the parameters specified in the `KMSYSTEMCONFIGSTRUCT` type structure. A native data buffer (double buffer) and display frame buffer are allocated in frame buffer memory. The capacity of the native data buffer area is obtained as follows:

Native data buffer capacity = whole frame buffer memory capacity - (size of the specified maximum texture + display frame buffer capacity)

However, the total capacity of the native data buffer (double buffer) cannot be more than half of the total capacity of the frame buffer memory.

If the result of the above calculation exceeds the requirement, only half of the total capacity of the frame buffer memory will be used as the native data buffer (double buffer).

Parameters

pSystemConfigStruct(input/output)

This parameter is a pointer to the KMSYSTEMCONFIGSTRUCT type structure, which is defined as follows:

```
[KMSYSTEMCONFIGSTRUCT]
typedef struct _tagKMSYSTEMCONFIGSTRUCT
    {
KMDWORD       dwSize;             /* Size Of KMSYSTEMCONFIGSTRUCT   */
KMDWORD       flags;             /* System Configuration Flags    */

      /* for Frame Buffer */
PPKMSURFACEDESC   ppSurfaceDescArray;       /* Array of SurfaceDesc Pointer   */
      union{
        KMUINT32    nNumOfFrameBuffer;       /* Number Of Frame Buffer      */
        KMUINT32    nStripBufferHeight;       /* Height of Strip Buffer      */
      }fb;

      /* for Texture Memory */
KMUINT32      nTextureMemorySize; * Texture Memory size             */
KMUINT32      nNumOfTextureStruct; * number of Texture Control Structure     */
KMUINT32      nNumOfSmallVQStruct; /* number of SmallVQ Texture Control Structure */
PKMDWORD      pTextureWork; /* Pointer to kamui work area         */

      /* for Vertex Buffer */
PKMVERTEXBUFFDESC  pBufferDesc;          /* pointer to KMVERTEXBUFFDESC   */
KMUINT32      nNumOfVertexBank;        /* Number of VertexBank       */
PKMDWORD      pVertexBuffer;         /* VertexBuffer Pointer       */
KMUINT32      nVertexBufferSize;       /* VertexBuffer Size        */
KMUINT32      nPassDepth;           /* Path Depth           */
KMPASSINFO     Pass[KM_MAX_DISPLAY_LIST_PASS]; /* Pass Information         */

      /* Reserve Area */
KMDWORD       reserved00;           /* reserved for future use     */
KMDWORD       reserved01;           /* reserved for future use     */
KMDWORD       reserved02;           /* reserved for future use     */
KMDWORD       reserved03;           /* reserved for future use     */
KMDWORD       reserved04;           /* reserved for future use     */
KMDWORD       reserved05;           /* reserved for future use     */
KMDWORD       reserved06;           /* reserved for future use     */
KMDWORD       reserved07;           /* reserved for future use     */

    } KMSYSTEMCONFIGSTRUCT, *PKMSYSTEMCONFIGSTRUCT;
```

The setting of each member is explained below:

[System configuration specification flag]

`dwSize`(input)	This parameter sets the size of the `KMSYSTEMCONFIGSTRUCT` structure in Sizeof(`KMSYSTEMCONFIGSTRUCT`).
`dwflags`(input)	This parameter specifies the types of data related to the system configuration. It is the result of ORing the following flags. If no flag is to be specified, the parameter shall be reset to zero.

`KM_CONFIGFLAG_ENABLE_CLEAR_FRAMEBUFFER`	This argument causes a frame buffer to be cleared when it is allocated.
`KM_CONFIGFLAG_ENABLE_STRIPBUFFER`	This argument causes a frame buffer to be created in StripBuffer format. The `nWidth` and `nHeight` members are enabled.
`KM_CONFIGFLAG_ENABLE_2V_LATENCY`	This argument causes KAMUI to operate in the 2V latency mode. If this argument is not entered, KAMUI operates in the 3V latency mode.
`KM_CONFIGFLAG_NOWAITVSYNC`	This argument causes a frame buffer surface to be displayed before a V-sync interrupt occurs. If this argument is not entered, the frame buffer surface is displayed after a V-sync interrupt has occurred.
`KM_CONFIGFLAG_SEPARATE_EACH_PASS`	VertexBuffer is allocated for each pass. The `KMPASSINFO` setting is valid.
`KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA`	This argument causes the texture load function to be terminated before DMA transfer of a texture to frame buffer memory, started by the function, ends. If this argument is entered, the `kmQueryFinishLastTextureDMA` function can be used to check whether DMA transfer has been completed. Avoid accessing memory from which a DMA transfer is under way.

[Parameters related to the frame buffer]

`ppSurfaceDescArray` (output)	This parameter specifies the pointer array for the `KMSURFACEDESC` structure for each frame buffer. Note that if `KMSTATUS_NOT_ENOUGH_MEMORY` is returned, the contents of this frame buffer structure are undefined. The application must set up the frame buffer structure area and the corresponding pointer array. `*E` In the case of non-strip buffer
`nNumOfFrameBuffer`(input)	This parameter specifies the number of frame buffer surfaces to be generated.

`ppSurfaceDescArray` should be specified using `nNumOfFrameBuffer`, as follows:

(Example) When there are two frame buffer surfaces

```
KMSURFACEDESC Surface1;
KMSURFACEDESC Surface2;
PKMSURFACEDESC ppSurfaceArray[nNumOfFrameBuffer];
ppSurfaceArray[0] = &Surface1;
ppSurfaceArray[1] = &Surface2;
ppSurfaceDescArray = ppSurfaceArray;
```

*EIn the case of strip buffer

`nStripBufferHeight`(input)	This parameter sets the height of the strip buffer. Be sure to use multiples of 32.

[Parameters related to the texture area]

`nTextureMemorySize`(input)	This parameter indicates the size of the largest texture. It is used to determine the capacity of the native data buffer. Note that the size must be multiples of 32 bytes.
`nNumOfTextureStruct`(input)	This parameter specifies the maximum number of texture management structures that Kamui will use for texture management. One texture/frame buffer surface and one empty area use up one management structure. Even if no textures will be used, a minimum of three structure areas are needed for the native data buffer, the frame buffer, and other empty areas. Note that this value can be calculated by using `kmuCalculateKamuiWorkareaSize`.

`nNumOfSmallVQStruct`(input)

This parameter specifies the maximum number of small VQ texture management structures that Kamui will use for texture management.

Small VQ textures use one Small VQ texture management structure and one texture management structure for each of the blocks indicated below.

SmallVQ size	Number of blocks
8x8	16
8x8 mipmap	16
16x16	16
16x16 mipmap	16
32x32	8
32x32 mipmap	4
64x64	2
64x64 mipmap	1

If SmallVQ texture is not used at all, the maximum number of SmallVQ texture control structure can be set to zero.

Also, the value can be determined by using the `kmuCalculateKamuiWorkareaSize` function.

`pTextureWork`(input)

This parameter gives the starting address of the Kamui work area.

It is necessary to prepare system memory space of a size that is determined according to the number of rendering passes and frame buffers/ textures used by the application.

Note that this size can be calculated by using `kmuCalculateKamuiWorkareaSize`.

[Vertex buffer-related parameters]

```
PKMVERTEXBUFFDESC  pBufferDesc;          /* pointer to KMVERTEXBUFFDESC   */
KMUINT32      nNumOfVertexBank;        /* Number of VertexBank       */
PKMDWORD      pVertexBuffer;         /* VertexBuffer Pointer       */
KMUINT32      nVertexBufferSize;       /* VertexBuffer Size        */
KMUINT32      nPassDepth;          /* Path Depth           */
KMPASSINFO     Pass[KM_MAX_DISPLAY_LIST_PASS]; /* Pass Information         */
```

`pBufferDesc`(input)

This parameter inputs the pointer for the `KMVERTEXBUFFDESC` vertex data buffer descriptor.

The application must set up the area for this structure.

This structure is referenced by `kmStartVertexStrip` and `kmSetVertex`.

`nNumOfVertexBank`(input)

This parameter specifies the number of banks of Vertex.

`pVertexBuffer`(input)	This parameter inputs a pointer to a vertex data buffer descriptor of `KMVERTEXBUFFDESC` type. The application program must prepare an area for this structure.
`nVertexBufferSize`(input)	This parameter specifies (in bytes) the size of the vertex data buffer allocated in system memory by the application. This size must be a multiple of 32 bytes.
`nPassDepth`(input)	This parameter specifies the number of bus splits of VertexBank.
Pass[`KM_MAX_PASS`](input)	This parameter sets the pass information.

```
typedef struct _tagKMPASSINFO{
KMDWORD     dwRegionArrayFlag;          /* Region Array Flag    */
PKMDWORD    pVertexBuffer;              /* VertexBuffer Pointer   */
KMUINT32    nVertexBufferSize;          /* VertexBuffer Size    */
KMUINT32    nDirectTransferList;        /* DirectTransfer List Type */
KMFLOAT     fBufferSize[KM_MAX_DISPLAY_LIST];  /* Buffer size in percent  */
KMDWORD     dwOPBSize[KM_MAX_DISPLAY_LIST];   /* Object Ponter Block Size */
}KMPASSINFO,*PKMPASSINFO;
```

`dwRegionArrayFlag`(input)	This parameter makes various specifications concerning `RegionArray`. Set this value by `ORing` the following flags:
`KM_PASSINFO_AUTOSORT`	Set this flag to set the translucent polygon sort mode to "AutoSort" for the current rendering processing.
`KM_PASSINFO_PRESORT`	Set this flag to set the translucent polygon sort mode to "Presort" for the current rendering processing.
`KM_PASSINFO_USE_ANOTHERLIST`	Set this flag to enable the `KM_PASSINFO_UA_XXXXX` flags described below.
`KM_PASSINFO_UA_TRMOD_AS_OPMOD`	Set this flag to use the same list as "`OpaqueModifier`" for "`TransModifier`." This flag is invalid if `KM_PASSINFO_USE_ANOTHERLIST` is not set.
`KM_PASSINFO_UA_OPMOD_AS_TRMOD`	Set this flag to use the same list as `TransModifier` for `OpaqueModifier`. This flag is invalid if `KM_PASSINFO_USE_ANOTHERLIST` is not set.
`KM_PASSINFO_UA_DISCADING_TRANSPOLY`	Set this flag to use "TransPolygon" as "PunchThroughPolygon." This flag is invalid if `KM_PASSINFO_USE_ANOTHERLIST` is not set.
`pVertexBuffer`(input)	This parameter sets the pointer for the vertex buffer that was allocated in system memory by the application. Kamui uses this value as the base address for the vertex buffer. (In order to avoid malloc within Kamui, the buffer must be set up by the application.) Note that this address must be aligned with a 32-byte boundary. This setting is valid when the flag `KM_CONFIGFLAG_SEPARATE_EACH_PASS` has been set.

`nVertexBufferSize`(input)	This parameter specifies, as a percentage the number of polygons used in one scene for each list types. These five list types are specified with a floating point value of between 0.0f and 100.0f. The total of the specified values must be 100.0f. If it exceeds 100.0f, KAMUI may not operate normally. KAMUI uses these values to assign a vertex data buffer to each polygon type.
`nDirectTransferList`(input)	This parameter specifies the ListType that is to be directly transferred in 2V_LATENCY mode.

This setting is invalid in any mode other than 2V_LATENCY mode.

`KM_OPAQUE_POLYGON`	`Directly transfer OpaquePolygon.`
`KM_OPAQUE_MODIFIER`	`Directly transfer OpaqueModifier.`
`KM_TRANS_POLYGON`	`Directly transfer TransPolygon.`
`KM_TRANS_MODIFIER`	`Directly transfer TransModifier.`
`KM_PUNCHTHROUGH_POLYGON`	`Directly transfer Punchthrough.`

`fBufferSize`[KM_MAX_DISPLAY_LIST](input)

This parameter specifies, as a percentage, the amount of polygons of each list type used in each scene.

For each list type, specify a floating-point value from 0.0f to 100.0f for each of the five types.

These five values must total to 100.0f. Operation is not guaranteed if the total value exceeds 100.0f.

Kamui allocates vertex data buffers for each polygon type on the basis of these values.

This setting is valid if the `KM_CONFIGFLAG_SEPARATE_EACH_PASS` flag has been set.

`dwOPBMode[KM_MAX_DISPLAY_LIST]`(input)

This parameter sets the OPB mode for each of the lists, as follows:

dwOPBMode[0]	OPB mode of OpaquePolygon
dwOPBMode[1]	OPB mode of OpaqueModifier
dwOPBMode[2]	OPB mode of TransPolygon
dwOPBMode[3]	OPB mode of TransModifier
dwOPBMode[4]	OPB mode of PunchThroughPolygon

`KM_OPB_ALLOCCTRL_NOLIST`	Use this setting when there is no list to register.
`KM_OPB_ALLOCCTRL_SMALL`	Use this setting when there is only a small number of lists to register.
`KM_OPB_ALLOCCTRL_NORMAL`	Normally use this setting when there are lists to register.
`KM_OPB_ALLOCCTRL_LARGE`	Use this setting when there are many lists to registered.
`reserved00 - reserved07`	These are reserved for future expansion. Their contents are undefined.

Return values

`KMSTATUS_SUCCESS`	System configuration set up successfully.
`KMSTATUS_NOT_ENOUGH_MEMORY`	Insufficient memory capacity for native data and frame buffers.

5. Functions for Global Setting

kmConvertFogDensity

Converts fog coefficient.

```
KMSTATUS KMAPI
kmConvertFogDensity(
IN KMFLOAT fFogDensity,
OUT KMDWORD *dwFogDensity
)
```

Description

This function coverts the floating point value to the 2-byte fog coefficient for use in setting the PowerVR hardware register.

The fog coefficient in floating point value obtained in the `kmGenerateFogTable` function is used by the `kmSetFogDensity` function to set the hardware.

Parameters

`fFogDensity`(input)	This parameter specifies the fog coefficient in floating point value obtained in the `kmGenerateFogTable` function.
*`dwFogDensity`(output)	This parameter returns the 2-byte value when the `kmSetFogDensity` function is setting the hardware.

Return values

KMSTATUS_SUCCESS	Success

kmGenerateFogTable

Specifies a fog table and coefficient to table fog.

```
KMSTATUS KMAPI
kmGenerateFogTable(
OUT PKMFLOAT pFogTable,
IN KMFLOAT fFrontBorder,
IN KMFLOAT fBackBorder,
IN KMFLOAT fFogDensity,
OUT KMFLOAT *fHWFogDensity,
IN KMDWORD dwFogType
)
```

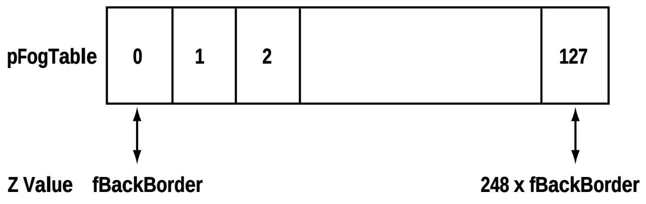
Description

This function generates the fog table automatically.

When a value is given to the depth of the front and back borders of the range where fog is valid, a fog table with smooth fog and fog coefficient are generated in the range.

At the fog table, the initial entry is created indicating the fog density of the value of the back border (`fBackBorder`) of the range (see the figure below). At the same time, when the generated fog coefficient (`*fHWFogDensity`) is set to the hardware by the `kmConvertFogDensity` and `kmSetFogDensity` functions, the leading entry of the generated fog table can now be referenced by the value of the back border (`fBackBorder`) of the range.

In this case, the last part of the fog table (front border) indicates the fog density at the position where depth `= 248 X fBackBorder`. In other words, the valid range of the fog table created by this function is from depth = `fBackBorder` (back border) to `248 *~ fBackBorder` (front border) (hardware specification). Note that depending on the value of `fBackBorder`, not all of the range specified by fFrontBorder can be covered.



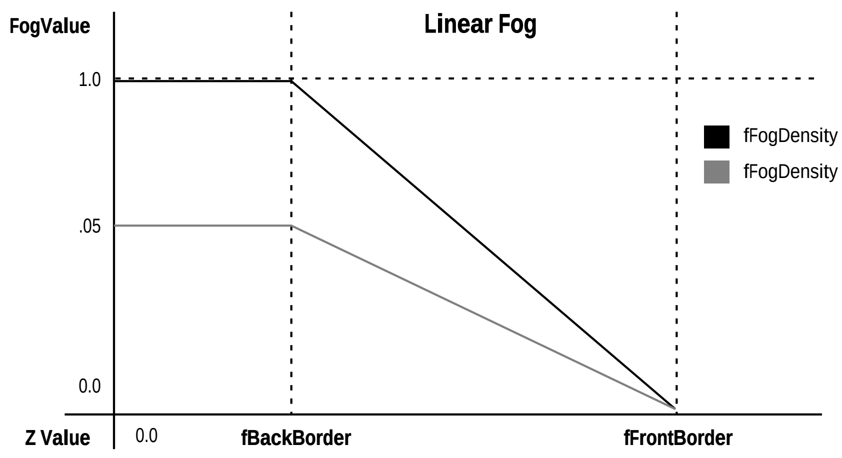
The fog table to create can be selected from one of the following four types (each with different density):

`KM_FOGTYPE_NONE`

Specifies to use no fog. All entries of the fog table will be set to zero.*B

`KM_FOGTYPE_LINEAR`

Specifies to use linear fog. The fog table is generated with changing line shape for depth value between `fFrontBorder` and `fBackBorder`. When `fFogDensity` is 1.0, the fog density is 1.0 (maximum density) at the fBackBorder position and 0.0 (no fog) at the fFrontBorder position (the red curve line in the figure below). By changing the value of `fFogDensity` between 1.0f and 0.0f, the steepness of fog density can be changed as well.



KM_FOGTYPE_EXPONENTIAL

Specifies to use exponential fog. The fog density of a depth (Z value) is determined by the following formula:

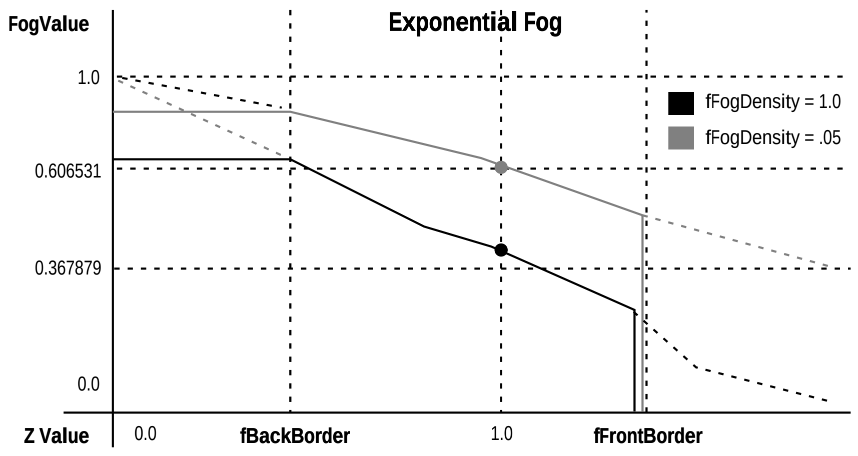
```
Fog density = e ^ - (fFogDensity x Z)
```

(`e` is the base of the natural log, and `^` indicates the exponent.)

This is equivalent to the exponential fog mode of Direct3D.

If `fFogDensity` is 1.0, when ,`y` value is 1.0 the exponential graph changes to have fog density being `0.367879...` (the red curve in the figure below). Also, in the interval where the `Z` value is smaller than `fBackBorder` (depth), the value for the position of `fBackBorder` is maintained. On the contrary, in the interval where the `Z` value is greater than `fFrontBorder` (the forward area), the fog density becomes zero.

As long as `fFogDensity` is greater than zero, any value can be specified. If `fFogDensity` is zero, the steepness of the fog density becomes flat (always maximum density). The bigger `fFogDensity` is, the steeper the fog density becomes.



`KM_FOGTYPE_EXPONENTIAL2`

Specifies to use exponential fog. The steepness of the fog density is bigger than that of `KM_FOGTYPE_EXPONENTIAL`. The fog density of a depth (Z value) is determined by the following formula:

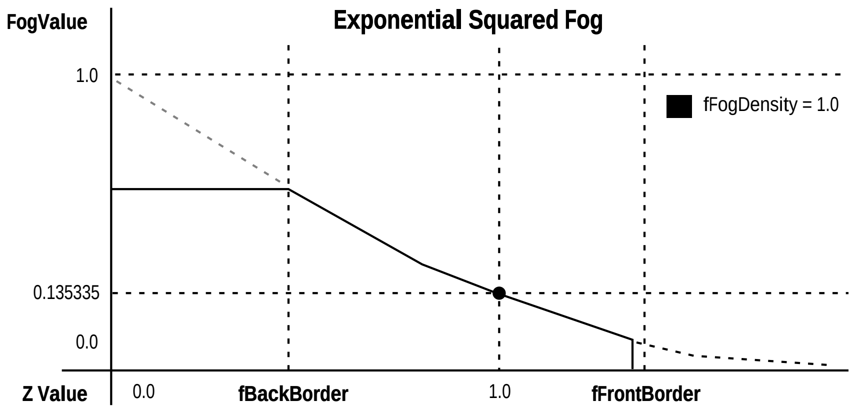
```
Fog density = (e ^ - (fFogDensity x Z)) ^ 2
```

(`e` is the base of the natural log, and `^` indicates the exponent.)

This is equivalent to the exponential squared fog mode of Direct3D.

If `fFogDensity` is 1.0, when ,`y` value is 1.0 the exponential graph changes to have fog density being `0.135335...` (the red curve in the figure below). Also, in the interval where the `Z` value is smaller than `fBackBorder` (depth), the value for the position of `fBackBorder` is maintained. On the contrary, in the interval where the `Z` value is greater than `fFrontBorder` (the forward area), the fog density becomes zero.

As long as `fFogDensity` is greater than zero, any value can be specified. If `fFogDensity` is zero, the steepness of the fog density becomes flat (always maximum density). The bigger `fFogDensity` is, the steeper the fog density becomes.



To generate fog table automatically, use the following Kamui APIs.

1) `kmGenerateFogTable`	(Generates fog table.)
2) `kmConvertFogDensity`	(Converts fog coefficient.)
3) `kmSetFogDensity`	(Sets fog coefficient.)
4) `kmSetFogTableColor`	(Sets fog color.)
5) `kmSetFogTable`	(Sets fog table.)

Parameters

`pFogTable`(output)	This parameter is the pointer to the one-dimensional array in the `MKFLOAT` format of 128 entries, for storing the generated fog table. Each element has clipping performed in the range from `0.0f` to `1.0f`. If this function returns an error, a fog table is also generated but it is different from the one specifically planned by the user and so the effect may be different. When `NULL` is specified for `pFogTable`, the fog table that was generated and `fHWFogDensity` are set immediately in the hardware.
`fFrontBorder`(input)	This parameter specifies the depth of the front border of the area where the fog is valid. The value must be greater than `0.0f` or `fBackBorder`.
fBackBorder(input)	This parameter specifies the depth of the back border of the area where the fog is valid. The value must be greater than `0.0f` or less than `fFrontBorder`.
`fFogDensity`(input)	This parameter specifies the fog density. The value must be floating point and greater than 0.0f. The value has the following meanings according to the type of fog.

If `dwFogType` == `KM_FOGTYPE_NONE`	This value carries no meaning.
If `dwFogType` == `KM_FOGTYPE_LINEAR`	Each of the elements of the generated table will be multiplied by this value. Control of the density of fog is possible. The valid value is between 0.0f and 1.0f.
If `dwFogType` == `KM_FOGTYPE_EXPONENTIAL`	The steepness of the density of the fog table can be changed. The bigger this value is, the steeper the density will become. When specifying the value, use floating point and make sure it is greater than 0.0f.
If `dwFogType` == `KM_FOGTYPE_EXPONENTIAL2`	The steepness of the density of the fog table can be changed. The bigger this value is, the steeper the density will become. When specifying the value, use floating point and make sure it is greater than 0.0f.

*`fHWFogDensity`(output)	This parameter returns the fog coefficient, in floating point, set at the PowerVR hardware by the `kmSetFogDensity` function. Be sure to use the `kmConvertFogDensity` function to convert it to `WORD` value and use the `kmSetFogDensity` function to set to the hardware.
`dwFogType`(input)	This parameter specifies the type of fog table. Depending on the fog type selected, the density steepness of the generated fog table will be different. One of the following can be selected.

Type Code	Description
KM_FOGTYPE_NONE	Specifies to use no fog. All entries in the fog table become zero.
KM_FOGTYPE_LINEAR	Sets linear fog. A fog table is generated with the line shape changing in the interval of depth value from `fFrontBorder` to `fBackBorder`.
KM_FOGTYPE_EXPONENTIAL	Sets exponential fog. This is equivalent to the exponential fog mode of Direct3D.
KM_FOGTYPE_EXPONENTIAL2	Sets exponential fog. This is equivalent to the exponential squared fog mode of Direct3D.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_OUT_OF_RANGE`	`fBackBorder` is less than 0.0f, or `fBackBorder` is greater than `fFrontBorder`, or `fFogDensity` is less than 0.0f.

kmGetSystemMetrics

KAMUI2 international information acquisition.

```
KMSTATUS KMAPI kmGetSystemMetrics(
IN OUT PKMSYSTEMMETRICS pSysMetrics
)
```

Description

Gets all internal information in KAMUI2. When this function is called, it writes KAMUI2 internal information to the structure specified in the parameter. However, to increase internal speed, only information specified in flags in the `KMSYSTEMMETRICS` structure can be gotten. Definitions that can be specified in flags are as follow.

Value	Description
KMSYSTEMMETRICS_VERTEXBUFFER_INFO	Gets vertex buffer information in system memory.
KMSYSTEMMETRICS_RENDERPERFORM_INFO	Gets information related to rendering performance.
KMSYSTEMMETRICS_TIMEOUT_INFO	Gets information for rendering time out, etc.
KMSYSTEMMETRICS_NATIVE_INFO	Gets information related to the native command buffer.

```
typedef struct _tagKMSYSTEMMETRICS
{
KMDWORD flags;

/* RENDER Performance */
KMDWORD nLastRenderTime[KM_MAX_RENDER_TIME]; Rendering time for last 8 times
KMDWORD nLastDMATime[KM_MAX_DISPLAY_LIST_PASS]; Time used for previous DMA transfer

/* TimeOut Setting Data */
KMDWORD nCurrentTimeOutCount; Current time out setting (VBLANK unit)
KMDWORD nDMATimeOutCount; Number of times DMA time out generated since system startup
KMDWORD nRenderTimeOutCount; Number of times rendering time out generated since system
startup
KMDWORD nOBJOverflowCount; Number of times object list overflowed
KMDWORD nParamOverflowCount; Number of times Parameter list overflowed

/* Native Buffer Information */
KMDWORD nParamCurrent; Amount of parameter size consumed by previous DMA [Byte]
KMDWORD nOBJCurrent; Amount of object list size consumed by previous DMA [Byte]
KMDWORD nOBJLimit; Limit value of ObjectList
KMDWORD nParamLimit; Limit value of ParameterList

/* VertexBuffer Information */
KMDWORD VertexBufferSize[KM_MAX_DISPLAY_LIST_PASS][KM_MAX_DISPLAY_LIST];
Actual size of vertex buffer allocated internally
KMDWORD MaxVertexSize[KM_MAX_DISPLAY_LIST_PASS][KM_MAX_DISPLAY_LIST];
Max value of vertex buffer size consumed from system startup to present
KMDWORD Reserved[16];
}KMSYSTEMMETRICS, *PKMSYSTEMMETRICS;
```

Parameters:

`pSysMetrics` (Input/output)	Specify the pointer to `KMSYSTEMMETRICS`

Return values:

`KMSTATUS_SUCCESS`	Success

kmResetRenderer

Resets the rendering pipeline through software.

Forced reset of renderer.

```
KMSTATUS KMAPI
kmResetRenderer( KMVOID )
```

Description

This function resets the rendering pipeline through software. It is used for forced reset if data in a strip cannot be fully drawn when a strip buffer is used.

Parameters

None

Return values

`KMSTATUS_SUCCESS`	Success

kmSetAutoSortMode

Sets auto-sort mode.

```
KMSTATUS KMAPI kmSetAutoSortMode(KMBOOLEAN bEnable)
```

Description

Controls turning AutoSort mode for translucent polygons ON/OFF.

For translucent polygon drawing, there are two types of modes: "`Auto-Sort Mode`" and "`Pre-Sort Mode`" and you can switch scene units using this function.

Auto-sort mode

The hardware automatically sorts polygons in units of pixels when drawing translucent polygons, and draws in order from the smallest pixel (deepest) in the Z coordinate, regardless of the order registered in KAMUI. In this situation, (alpha) blending is carried out when two translucent polygons intersect. Therefore, when many polygons overlap, processing speed is slowed down. In Auto-sort mode, the member `DepthMode` in `VERTEXCONTEXT` is ignored, and the Z coordinate is always compared by `KM_GREATEREQUAL`. Also, if the Z coordinates are exactly the same, drawing is done according to the order registered in KAMUI.

Pre-sort mode

Polygon drawing is carried out in the order registered in KAMUI. Therefore, it is necessary to sort on the application side according to the Z coordinate of polygons. Also, (alpha) blending for translucent polygons that intersect is not carried out correctly. However, processing speed is faster than with Auto-sort mode, so in cases in which Z sorting can be easily done on the application side, such as with 2D sprite, use Pre-sort mode.

Parameters

`bEnable` (input)	When `TRUE`, specifies auto sort mode for sorting of translucent scenes. When `FALSE`, emulates sorting by convention of the same software.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetBackGround

Specifies the vertex and CONTEXT for use in the background.

```
KMSTATUS KMAPI
kmSetBackGround(
        IN   PKMSTRIPHEAD  pStripHead,
        IN   KMVERTEXTYPE  VertexType,
        IN   PVOID      pVertex1,
        IN   PVOID      pVertex2,
        IN   PVOID      pVertex3
    );
```

Description

This function specifies the vertex and CONTEXT for use in the background.

Parameters

These parameters set up the background plane.

`pStripHead`(input)	This parameter sets the pointer for `KMSTRIPHEAD`.
`VertexType`(input)	This parameter sets the data type for the vertex data that is used in the background plane.

The setting for the first parameter is valid when using VertexType 09 to 14.

`pVertex1`(input)
`pVertex2`(input)
`pVertex3`(input)

This parameter sets the pointer for the vertex data structure that indicates the coordinates on the background plane.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetBorderColor

Sets the border (outside of the display screen) color.

```
KMSTATUS KMAPI
kmSetBorderColor(
IN KMPACKEDARGB BorderColor
)
```

Description

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

Parameters

`BorderColor`(input)	This parameter specifies a packed ARGB color.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetCheapShadowMode

Sets the cheap shadow mode.

```
KMSTATUS KMAPI
kmSetCheapShadowMode(
IN KMINT32 nIntensity
)
```

Description

This function selects the cheap (simple) shadow mode. The cheap shadow mode is intended to represent the shadow of polygons by lowering their luminance when they approach the modifier volume.

After cheap shadow mode has been set by this function, all the modifier volume are set in cheap shadow mode. Coexistence with two-parameter polygons in a scene is not allowed. To terminate cheap shadow mode, enter a negative number as the argument, then call this function.

To turn cheap shadow mode on and off when using `KMSTRIPCONTEXT`, set the `CheapShadow` mode effect in `nShadowMode`. When using `KMVERTEXCONTEXT`, enable cheap shadow mode by issuing this function before executing `kmSetVertexRenderState`. Once cheap shadow mode has been turned on, it is not necessary to execute `kmSetVertexRenderState` when changing only the intensity of shadows.

Similarly to the two-parameter volume, `KM_MODIFIER_A` is set in the `SelectModifier` member of the `VERTEXCONTEXT` for the polygons to be influenced by the cheap shadow mode.

The vertex data used consists of regular one-parameter polygons.

Parameters

`nIntensity`(input)	This parameter sets the luminance of a polygon in the modifier volume, using a value from 0 to 255. The hardware multiplies the base color and offset color for the polygon by the specified value after it is divided by 256. If the parameter specifies 128, the multiplier is 0.5 (= 128/256). If a negative value is input, the setting of cheap shadow mode is completed, and the normal 2-parameter polygon becomes valid from the scene.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_PARAMETER`	Invalid parameter

kmSetColorClampMax

Specifies the color clamp maximum value.

```
KMSTATUS KMAPI
kmSetColorClampMax(
IN KMPACKEDARGB Val
)
```

Description

This function specifies the color clamp maximum 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.

Parameters

`Val`(input)	This parameter specifies a maximum value for color clamping. It is a packed ARGB 32-bit color. If you want to specify the RGB color with a brightness of 128, enter 0x00808080.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetColorClampMaxValue

Specifies the color clamp maximum value.

```
KMSTATUS KMAPI kmSetColorClampMaxValue(
IN KMPACKEDARGB MaxVal
)
```

Description

This function specifies the color clamp maximum 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.

Parameters

`MaxVal`(input)	This parameter specifies a maximum value for color clamping. It is a packed ARGB 32-bit color. If you want to specify the RGB color with a brightness of 128, enter 0x00808080.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetColorClampMin

Specifies the minimum color clamp value.

```
KMSTATUS KMAPI
kmSetColorClampMin(
IN KMPACKEDARGB Val
)
```

Description

This function specifies the minimum 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.

Parameters

`Val`(input)	This parameter specifies a minimum value for color clamping. It is a packed ARGB 32-bit color. If you want to specify the RGB color with a brightness of 20, enter 0x00141414.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetColorClampMinValue

Specifies the minimum color clamp value.

```
KMSTATUS KMAPI kmSetColorClampMinValue(
IN KMPACKEDARGB MinVal
)
```

Description

This function specifies the minimum 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.

Parameters

`MinVal`(input)	This parameter specifies a minimum value for color clamping. It is a packed ARGB 32-bit color. If you want to specify the RGB color with a brightness of 20, enter 0x00141414.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetColorClampValue

Sets the color clamp value.

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

Description

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.

Parameters

`MaxVal`(input)	This parameter specifies a maximum value for color clamping. It is a packed ARGB 32-bit color. If you want to specify the RGB color with a brightness of 128, enter 0x00808080.
`MinVal`(input)	This parameter specifies a maximum value for color clamping. It is a packed ARGB 32-bit color. If you want to specify the RGB color with a brightness of 20, enter 0x00141414.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetCullingRegister

Specifies a threshold value for culling small polygons.

```
KMSTATUS KMAPI
kmSetCullingRegister(
IN KMFLOAT fCullVal
)
```

Description

This function specifies a threshold value for culling small polygons.

Parameters

`fCullVal`(input)	This parameter sets a determinant value for a plane parameter.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetFogDensity

Specifies a coefficient to table fog.

```
KMSTATUS KMAPI
kmSetFogDensity(
IN KMDWORD FogDensity
)
```

Description

This function assigns a coefficient (scale factor) to table fog. PowerVR has a fog table containing 128 levels from 0 to 127. `FogDensity` determines the depth range over which each of these 128 levels is effective. 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).

`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	= 0.0000001(b)	= 0.015625
0x8000	= 1.0(b)	= 1.0
0xFF00	= 1.1111111(b)	= 1.984375
0xFF01	= 11.111111(b)	= 3.96875
0xFF06	= 1111111.1(b)	= 128.5
0xFF07	= 11111111.0(b)	= 255
0xFF08	= 111111110.0(b)	= 510
0xFF09	= 1111111100.0(b)	= 1020
0xFF0A	= 2040	
0xFF0B	= 4080	
0xFF0C	= 8160	

The value of `FogDensity` as well as the content of the fog table can automatically be generated with the use of the `kmGenerateFogTable` function. For details, see the description of `kmGenerateFogTable`, `kmConvertFogDensity`, and `kmSetFogTable`.

Parameters

`FogDensity`(input)	This parameter is a coefficient of table fog (scale factor). Specify this parameter as "`kmSetFogDensity`(0xFF09)."

Return values:

`KMSTATUS_SUCCESS`	Success

kmSetFogTable

Sets the fog table.

```
KMSTATUS KMAPI
kmSetFogTable(
IN PKMFLOAT pfFogTable
)
```

Description

This function registers the fog table. 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. A fog table consists of 128 elements with indexes 0 to 127. The element of index of a fog table specifies fog density of the following position with a depth of (1/w value):

Depth = (pow(2.0, Index >> 4) * (float)((Index & 0x0F) + 16) / 16.0f) / FogDensity

An element of a fog table that is 0.0 has an attenuation rate 0 and an element that is 1.0 has the maximum attenuation rate. Therefore, specify the density starting from the most distant point, in sequence.

As indicated in the above formula, the value of depth for each element of the fog table changes according to the value of `FogDensity`. The value of `FogDensity` is set at the `kmSetFogDensity` function.

Content of the fog table can automatically be generated with the use of the `kmGenerateFogTable` function.

Parameters

`pfFogTable`(input)	This parameter is a pointer to the one-dimensional array in the KMFLOAT format of 128 entries where the fog table is stored.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetFogTableColor

Specifies a table fog color.

```
KMSTATUS KMAPI
kmSetFogTableColor(
IN KMPACKEDARGB FogTableColor
)
```

Description

This function specifies the fog color for the table fog when it is used.

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.

Parameters

`FogTableColor`(input)	This parameter specifies the packed 32-bit color to be used in `FogTable`.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetFogVertexColor

Specifies a vertex fog color.

```
KMSTATUS KMAPI
kmSetFogVertexColor(
IN KMPACKEDARGB FogVertexColor
)
```

Description

This function specifies a vertex fog color.

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.

Parameters

`FogVertexColor`(input)	This parameter specifies the packed 32-bit color to be used in `FogVertex`.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetGlobalClipping

Sets global clipping.

```
KMSTATUS KMAPI
kmSetGlobalClipping(
IN KMINT32 nWidth,
IN KMINT32 nHeight
)
```

Description

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

Parameters

`nWidth, nHeight`(input)	These parameters specify a global clipping area as a multiple of 32. To specify a 128 x 64 area, for example, set Width to 4 and Height to 2.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_PARAMETER`	`Setting failed`

kmSetPaletteBank

Changes entry of palette, in banks.

```
KMSTATUS KMAPI
kmSetPaletteBank(
IN KMINT32 Bank,
IN KMPALETTE_ENTRY_SIZE DataSize,
IN PKMDWORD pPaletteData
)
```

Description

This function rewrites a specified portion of the on-chip palette used by the palettized texture.

The value of `DataSize` determines the number of entries to update.

If `DataSize` is set to `KM_PALETTE_ENTRY_16`, 16 entries will be updated. 16 * 4 bytes of data from top of `pPaletteData` are set to specified bank. In this case, `pPaletteData` requires data for the 16 entries.

If `DataSize` is set to `KM_PALETTE_ENTRY_256`, 256 entries will be updated. 256 * 4 bytes of data from top of `pPaletteData` are set to specified bank. In this case, `pPaletteData` requires data for the 256 entries.

See the descriptions of `kmSetPaletteData` for the structure of the palette.

Caution: Palette data setting (`kmSetPaletteBank`/`kmSetPaletteData`/`kmSetPaletteBankData`) cannot precede palette mode setting (`kmSetPaletteMode`). If the palette mode type does not match the palette data type, invalid data will be set in the palette register.

Parameters

`Bank`(input)	This parameter specifies the bank number to update. The range is from 0 to 63.
	If `DataSize` is set to `KM_PALETTE_ENTRY_256`, the number can be 0, 16, 32, or 48. For other values, the lower 4 bits will be masked and it will be converted to 0, 16, 32, or 48 for use.
`DataSize`(input)	This parameter specifies the data size to update. One of the following can be selected.
`KM_PALETTE_ENTRY_16`	Handled as 4BPP palette. Data of 16 entries will be updated.
`KM_PALETTE_ENTRY_256`	Handled as 8BPP palette. Data of 256 entries will be updated.
`pPaletteData`(input)	This parameter is a pointer to a `DWORD` array. The number of elements constituting the palette data must be greater than or equal to the value specified in `DataSize`. If the number of elements is less than that value, normal operation is not guaranteed.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetPaletteBankData

Rewrites part of the on-chip palette data.

```
KMSTATUS KMAPI
kmSetPaletteBankData(
IN KMINT32 PaletteEntry,
IN KMINT32 DataSize,
IN PKMPALETTEDATA pPaletteTable
)
```

Description

This function rewrites a specified portion of the on-chip palette used by the palettized texture. See the descriptions of `kmSetPaletteData` for the structure of the palette.

The values that can be specified by `PaletteEntry` are `0` to `1,023` for both `4-` and `8-bpp` palette modes. The values need not be aligned with a bank boundary. They can start at any entry.

Data items in an area pointed to by `pPaletteTable` are sent to the palette between entries PaletteEntry and PaletteEntry + DataSize sequentially, starting at the beginning of the area. If `PaletteEnrty + DataSize > 1,024`, data for palette numbers greater than `1,023` is ignored. Put another way, data transfer ends at palette number `1,023`.

Caution: Palette data setting (`kmSetPaletteData`/`kmSetPaletteBank`/`kmSetPaletteBankData`) cannot precede palette mode setting (`kmSetPaletteMode`). If the palette mode type does not match the palette data type, invalid data will be set in the palette register.

Parameters

`PaletteEntry`(input)	This parameter specifies the first entry number of a palette where data is to be written, using a number between 0 and 1,023. A palette portion that begins with the specified entry number will be rewritten.
`DataSize`(input)	This parameter specifies the size of the data to be written (number of entries), using a number between 1 and 1,024.
`pPaletteTable`(input)	This parameter is a pointer to a palette setting array. The array is defined as follows:
`KMPALETTEDATA PaletteTable;`	The number of elements constituting the palette data must be greater than or equal to the value specified in PaletteEntry + DataSize. If the number of elements is less than that value, normal operation is not guaranteed.

Return values

`KMSTATUS_SUCCESS`	Success

Example

```
kmSetPaletteBankData( 32, 64, pPaletteTable);
```

This example coding rewrites 64 entries in the palette, starting at entry 32.

kmSetPaletteData

Sets the on-chip palette data.

```
KMSTATUS KMAPI
kmSetPaletteData(
IN PKMPALETTEDATA pPaletteTable
)
```

Description

This function sets the on-chip palette used by the palettized texture.

A palette has a total of 1,024 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 in 32-bpp mode (2 clocks/pixel).

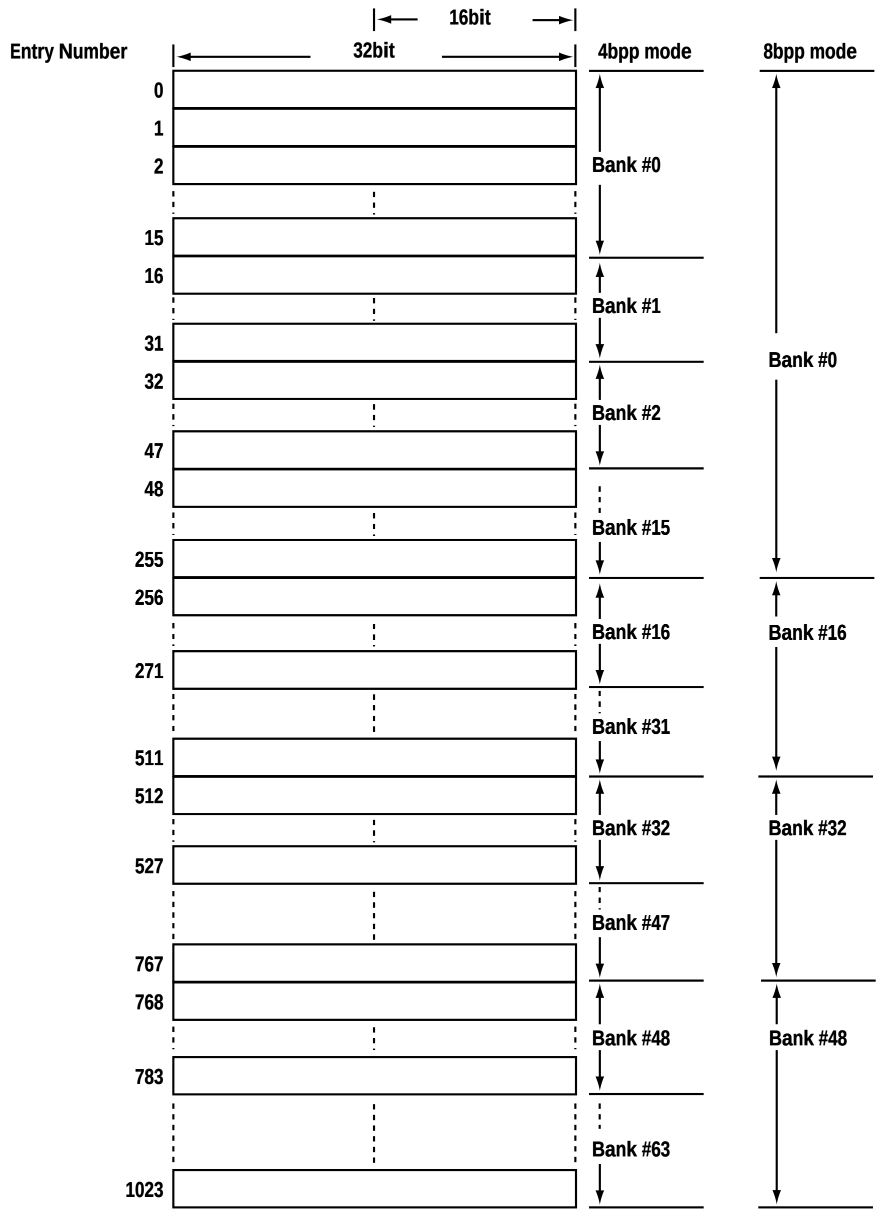
With Palettized-4bpp, the 1,024 entries are divided into 64 banks (1,024 entries/16 colors = 64 banks). With Palettized-8bpp, the 1,024 entries are divided into four banks (1,024 entries/256 colors = 4 banks). The banks are not separated physically, each bank being created by calculating pointers to the 1,024 entries.

The 4-bpp palette texture and 8-bpp palette texture can exist together in one scene, but the overlapping portion of the 1,024 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;
}
```

Palette Register Structure



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 for a `PaletteBank` value.

Caution: Palette data setting (`kmSetPaletteData`/`kmSetPaletteBank`/`kmSetPaletteBankData`) cannot precede palette mode setting (`kmSetPaletteMode`). If the palette mode type does not match the palette data type, invalid data will be set in the palette register.

Parameters

`pPaletteTable`(input)

This parameter specifies a pointer to a palette setting array. The array is defined as follows:
`KMPALETTEDATA` PaletteTable;

Example: The following coding is for setting 16-bpp data in the first 256 entries of a palette.

```
j = 0;
for(i = 0; i < 512; i+=2) {
    PaletteTable.dwPaletteData[j++]
        = (KMDWORD)((pdata[i+1]*256) + pdata[i]);
}
```

`kmSetPaletteData`(&PaletteTable);

The number of elements constituting the palette data must be 1,024. If there are no 1,024 elements, KAMUI may not operate normally.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetPaletteMode

Sets the on-chip palette mode.

```
KMSTATUS KMAPI
kmSetPaletteMode(
IN KMPALETTEMODE Palettemode
)
```

Description

This function specifies a mode of the on-chip palette used by the palettized texture. A palette has 1,024 entries. For details of how to set a palette, see the description of `kmSetPaletteData`.

Caution

Palette data setting (`kmSetPaletteData`/`kmSetPaletteBank`/`kmSetPaletteBankData`) cannot precede palette mode setting (`kmSetPaletteMode`). If the palette mode type does not match the palette data type, invalid data will be set in the palette register.

Parameters

`Palettemode`(input)

This parameter specifies the BPP mode of the palette. One of the following can be selected.

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

Return values

`KMSTATUS_SUCCESS`	Success

kmSetPixelClipping

Specifies pixel-unit clipping.

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

Description

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

Parameters

`Xmin`, `Ymin`, `Xmax`, `Ymax`(input)	These parameters 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`	Success
`KMSTATUS_INVALID_PARAMETER`	Invalid parameter

kmSetPunchThroughThreshold

Sets PunchThrough polygon (alpha) threshold.

```
KMSTATUS KMAPI
kmSetPunchThroughThreshold(
IN KMDWORD dwThreshold
);
```

Description

Sets PunchThrough Polygon (alpha) Threshold.

Decides whether to punch through by whether the alpha value of data loaded to the PunchThrough list exceeds the threshold.

```
0 < (alpha) < dwThreshold = 0 (punch through)
dwThreshold < (alpha) < 255 = 0xff
```

Parameters

`dwThreshold` (input)	PunchThrough Polygon (alpha) threshold (range: 0 to 255)

Return values

`KMSTATUS_SUCCESS`	Success

kmSetStrideWidth

Specifies the stride size for stride texture.

```
KMSTATUS KMAPI
kmSetStrideWidth(
IN KMINT32 nWidth
)
```

Description

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.

Parameters

`nWidth`(input)	This parameter sets the stride size.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_PARAMETER`	Invalid parameter

(For compatibility purpose)

kmSetBackGroundPlane

Sets the background plane.

Format

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

Description

This function registers a background plane.

`kmSetBackGroundRenderState` needs to be called before this function.

Parameters

`pVertex`[3](input)	This parameter is a pointer to a vertex data structure that indicates coordinates on a background plane.
`VertexType`(input)	This parameter indicates the data type of vertex data.
`StructSize`(input)	This parameter indicates the data type of vertex data.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetBackGroundRenderState Registers the rendering parameter of the background plane.

Format

```
KMSTATUS KMAPI
kmSetBackGroundRenderState(
    IN PKMVERTEXCONTEXT  pVertexContext
);
```

Description

This function registers the members (listed below) of the `KMVERTEXCONTEXT` structure that was set up by `kmProcessVertexRenderState` in the system as the rendering parameters for the background plane. For the BG plane settings that are specified by `kmSetBackGroundPlane()` subsequent to calling this function, these parameters and the value of `KMVERTEXCONTEXT` set by `pVertexContext` are valid.

Members that are referenced:

```
pVertexContext->GLOBALPARAMBUFFER
pVertexContext->ISPPARAMBUFFER
pVertexContext->TSPPARAMBUFFER
pVertexContext->TexturePARAMBUFFER
```

Parameters

`pVertexContext`(input)	This parameter is a pointer to `KMVERTEXCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success

kmUseAnotherModifier

Sets modifier volume list.

`KMSTATUS KMAPI` kmUseAnotherModifier(`KMLISTTYPE` kmModifierListType)

Description

Uses a modifier volume list specified by the input parameter kmModifierListType as the modifier volume list.

This function only rewrites the `OPAQUE-MODIFIER` or `TRANS-MODIFIER` pointers of the region arrays in native data. Be careful because when TRANS-MODIFIER object data specified on `KM_OPAQUE_MODIFIER` is registered, it overwrites `OPAQUE-MODIFIER` data. (The opposite is also true).

Parameters

`kmModifierListType` (input)	Specifies how to use the modifier volume list. One of the following is specified.
`KM_OPAQUE_MODIFIER`	Uses Opaque Modifier as Trans Modifier.
`KM_TRANS_MODIFIER`	Uses Trans Modifier as Opaque Modifier.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_LIST_TYPE`	Setting failure

6. Vertex Definition Functions

Functions for controlling rendering parameter

kmGenerateStripHead

Generate and build Rendering Parameter (KMSTRIPHEAD).

Format

```
KMSTATUS KMAPI
kmGenerateStripHead(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
        IN  KMVERTEXTYPE        nVertexType
);
```

Description

This function constructs the rendering parameters (`KMSTRIPHEAD`) from `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`).

`KMSTRIPHEAD` can be constructed by two methods, either `kmGenerateStripHead` or `kmGenerateStripHeadXX` (00 to 17).

Example: Constructing VertexType03 rendering parameters:

(1)`kmGenerateStripHead(pStripHead,pStripContext,KM_VERTEXTYPE_03);`

(2)`kmGenerateStripHead03`

Both (1) and (2) can be used to construct VertexType03 `KMSTRIPHEAD`.

The `pStripContext` setting depends on the VertexType to be used.

- When using VertextType00 to 08, 15, or 16:	Specify the pointer for `KMSTRIPCONTEXT`.
- When using VertextType09 to 14 (two parameters):	Specify the pointer for `KMTWOVOLUMESTRIPCONTEXT`.

In addition, it is necessary to specify the size of the structure that it to be used in `nSize` from `pStripContext` before calling `kmGenerateStripHead`. Operation is not guaranteed if the same size is not set.

For details on the settings within `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`), refer to the description of the `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`) structure.

Parameters

`pStripHead`(output)	This parameter is a pointer to `KMSTRIPHEAD`.
`pStripContext`(input)	This parameter is a pointer to `KMSTRIPCONTEXT`.
	When using `KMTWOVOLUMESTRIPCONTEXT` (two-parameter polygons), cast to the `PKMSTRIPCONTEXT` type. ex)

```
KMSTRIPHEAD                    StripHead;
KMTWOVOLUMESTRIPCONTEXT        TwoVolStripContext;
...
    TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
    kmGenerateStripHead( &StripHead,
    (PKMSTRIPCONTEXT)&TwoVolStripContext,KM_VERTEXTYPE_09 );
```

`nVertexType`(input)	Sets the VertexType.

KM_VERTEXTYPE_00
KM_VERTEXTYPE_01
KM_VERTEXTYPE_02
KM_VERTEXTYPE_03
KM_VERTEXTYPE_04
KM_VERTEXTYPE_05
KM_VERTEXTYPE_06
KM_VERTEXTYPE_07
KM_VERTEXTYPE_08
KM_VERTEXTYPE_09
KM_VERTEXTYPE_10
KM_VERTEXTYPE_11
KM_VERTEXTYPE_12
KM_VERTEXTYPE_13
KM_VERTEXTYPE_14
KM_VERTEXTYPE_15
KM_VERTEXTYPE_16
KM_VERTEXTYPE_17

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead00

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType00.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead00(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType00 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead01

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType01.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead01(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType01 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead02

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType02.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead02(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (KMSTRIPHEAD) for VertexType02 from KMSTRIPCONTEXT.

Set the parameters that are to be used in the members indicated below in KMSTRIPCONTEXT.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.nIntensityMode
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
```

The following members are required when:

```
pStripContext -> StripControl, nIntensityMode = KM_INTENSITY.
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fBlue
```

Parameters

pStripHead(output)	This specifies the pointer for KMSTRIPHEAD.
pStripContext(input)	This specifies the pointer for KMSTRIPCONTEXT.

Return values

KMSTATUS_SUCCESS	Success
KMSTATUS_INVALID_ADDRESS	pStripHead or pStripContext is invalid (NULL).
KMSTATUS_INVALID_SETTING	pStripContext size setting is invalid.

kmGenerateStripHead03

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType03.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead03(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (KMSTRIPHEAD) for VertexType03 from KMSTRIPCONTEXT.
Set the parameters that are to be used in the members indicated below in KMSTRIPCONTEXT.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead04

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType04.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead04(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType04 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead05

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType05.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead05(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType05 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead06

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType06.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead06(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType06 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
When using a palette texture, the following member settings are needed.
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return valuesExplanation

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead07

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType07.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead07(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType07 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.nIntensityMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
```

The following members are required when:

```
pStripContext -> StripControl, nIntensityMode = KM_INTENSITY.
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fBlue
```

The following members are required when:

```
pStripContext->StripControl.nIntensityMode = KM_INTENSITY,
pStripContext->StripControl.bOffset        = KM_TRUE.
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fBlue
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead08

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType08.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead08(
               OUT PKMSTRIPHEAD        pStripHead,
               IN  PKMSTRIPCONTEXT     pStripContext,
       );
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType08 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.nIntensityMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
```

The following members are required when:

```
pStripContext -> StripControl, nIntensityMode = KM_INTENSITY.
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fBlue
```

The following members are required when:

```
pStripContext->StripControl.nIntensityMode = KM_INTENSITY,
pStripContext->StripControl.bOffset        = KM_TRUE.
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_OFFSET].fBlue
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead09

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType09.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead09(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (KMSTRIPHEAD) for VertexType09 from KMTWOVOLUMESTRIPCONTEXT.

Set the parameters that are to be used in the members indicated below in KMTWOVOLUMESTRIPCONTEXT.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMTWOVOLUMESTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM2].bUseAlpha
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Cast to the `PKMSTRIPCONTEXT` type.

```
ex)
KMSTRIPHEAD              StripHead;
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
....
TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
kmGenerateStripHead09( &StripHead, (PKMSTRIPCONTEXT)&TwoVolStripContext );
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead10

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType10.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead10(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (KMSTRIPHEAD) for VertexType10 from KMTWOVOLUMESTRIPCONTEXT.

Set the parameters that are to be used in the members indicated below in KMTWOVOLUMESTRIPCONTEXT.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMTWOVOLUMESTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nIntensityMode
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM2].bUseAlpha
```

The following members are required when:

```
pStripContext->StripControl.nIntensityMode = KM_INTENSITY.
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fBlue
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fBlue
```

Parameters

pStripHead(output)	This specifies the pointer for KMSTRIPHEAD.
pStripContext(input)	
pStripContext(input)	This specifies the pointer for KMTWOVOLUMESTRIPCONTEXT.

Cast to the PKMSTRIPCONTEXT type.

```
ex)
KMSTRIPHEAD              StripHead;
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
....
TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
kmGenerateStripHead10( &StripHead, (PKMSTRIPCONTEXT)&TwoVolStripContext );
```

Return values

KMSTATUS_SUCCESS	Success
KMSTATUS_INVALID_ADDRESS	pStripHead or pStripContext is invalid (NULL).
KMSTATUS_INVALID_SETTING	pStripContext size setting is invalid.

kmGenerateStripHead11

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType11.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead11(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType11 from `KMTWOVOLUMESTRIPCONTEXT`.

Set the parameters that are to be used in the members indicated below in `KMTWOVOLUMESTRIPCONTEXT`.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMTWOVOLUMESTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
pStripContext->ImageControl[KM_IMAGE_PARAM2].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM2].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM2].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].pTextureSurfaceDesc
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMTWOVOLUMESTRIPCONTEXT`.

Cast to the `PKMSTRIPCONTEXT` type.

```
ex)
KMSTRIPHEAD              StripHead;
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
....
TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
kmGenerateStripHead11( &StripHead, (PKMSTRIPCONTEXT)&TwoVolStripContext );
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead12

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType12.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead12(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType12 from `KMTWOVOLUMESTRIPCONTEXT`.

Set the parameters that are to be used in the members indicated below in `KMTWOVOLUMESTRIPCONTEXT`.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMTWOVOLUMESTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
pStripContext->ImageControl[KM_IMAGE_PARAM2].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM2].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM2].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].pTextureSurfaceDesc
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMTWOVOLUMESTRIPCONTEXT`.

Cast to the `PKMSTRIPCONTEXT` type.

```
ex)
KMSTRIPHEAD              StripHead;
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
....
TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
kmGenerateStripHead12( &StripHead, (PKMSTRIPCONTEXT)&TwoVolStripContext );
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead13

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType13.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead13(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType13 from `KMTWOVOLUMESTRIPCONTEXT`.

Set the parameters that are to be used in the members indicated below in `KMTWOVOLUMESTRIPCONTEXT`.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMTWOVOLUMESTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nIntensityMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
pStripContext->ImageControl[KM_IMAGE_PARAM2].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM2].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM2].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].pTextureSurfaceDesc
```

The following members are required when:

```
pStripContext->StripControl.nIntensityMode = KM_INTENSITY.
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fBlue
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fBlue
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMTWOVOLUMESTRIPCONTEXT`.

Cast to the `PKMSTRIPCONTEXT` type.

```
ex)
KMSTRIPHEAD              StripHead;
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
....
TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
kmGenerateStripHead13( &StripHead, (PKMSTRIPCONTEXT)&TwoVolStripContext );
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead14

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType14.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead14(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType14 from `KMTWOVOLUMESTRIPCONTEXT`.

Set the parameters that are to be used in the members indicated below in `KMTWOVOLUMESTRIPCONTEXT`.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMTWOVOLUMESTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nIntensityMode
pStripContext->StripControl.bOffset
pStripContext->StripControl.bGouraud
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
pStripContext->ImageControl[KM_IMAGE_PARAM2].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM2].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM2].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM2].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM2].pTextureSurfaceDesc
```

The following members are required when:

```
pStripContext->StripControl.nIntensityMode = KM_INTENSITY.
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE].fBlue
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fAlpha
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fRed
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fGreen
pStripContext->type.intensity.Face[KM_INTENSITY_BASE_2ND].fBlue
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
pStripContext->ImageControl[KM_IMAGE_PARAM2].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	
`pStripContext`(input)	This specifies the pointer for `KMTWOVOLUMESTRIPCONTEXT`.

Cast to the `PKMSTRIPCONTEXT` type.

```
ex)
KMSTRIPHEAD             StripHead;
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
....
TwoVolStripContext.nSize = sizeof(KMTWOVOLUMESTRIPCONTEXT);
kmGenerateStripHead14( &StripHead, (PKMSTRIPCONTEXT)&TwoVolStripContext );
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead15

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType15.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead15(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext,
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType15 from `KMSTRIPCONTEXT`.

Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->type.splite.Base.dwPacked
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead16

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType16.

Format

```
KMSTATUS KMAPI
       kmGenerateStripHead16(
               OUT PKMSTRIPHEAD        pStripHead,
               IN  PKMSTRIPCONTEXT     pStripContext,
       );
```

Description

This function constructs rendering parameters (KMSTRIPHEAD) for VertexType16 from KMSTRIPCONTEXT.

Set the parameters that are to be used in the members indicated below in KMSTRIPCONTEXT.

Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->StripControl.nShadowMode
pStripContext->StripControl.bOffset
pStripContext->ObjectControl.nDepthCompare
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.bZWriteDisable
pStripContext->ObjectControl.bDCalcControl
pStripContext->ImageControl[KM_IMAGE_PARAM1].nSRCBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].nDSTBlendingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSRCSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].bDSTSelect
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFogMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bColorClamp
pStripContext->ImageControl[KM_IMAGE_PARAM1].bUseAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].bIgnoreTextureAlpha
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFlipUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nClampUV
pStripContext->ImageControl[KM_IMAGE_PARAM1].nFilterMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].bSuperSampleMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwMipmapAdjust
pStripContext->ImageControl[KM_IMAGE_PARAM1].nTextureShadingMode
pStripContext->ImageControl[KM_IMAGE_PARAM1].pTextureSurfaceDesc
pStripContext->type.splite.Base.dwPacked
pStripContext->type.splite.Offset.dwPacked
```

When using a palette texture, the following member settings are needed.

```
pStripContext->ImageControl[KM_IMAGE_PARAM1].dwPaletteBank
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmGenerateStripHead17

Generate and build Rendering Parameter(KMSTRIPHEAD) for VertexType17.

Format

```
KMSTATUS KMAPI
kmGenerateStripHead17(
        OUT PKMSTRIPHEAD        pStripHead,
        IN  PKMSTRIPCONTEXT     pStripContext
);
```

Description

This function constructs rendering parameters (`KMSTRIPHEAD`) for VertexType17 from `KMSTRIPCONTEXT`.
Set the parameters that are to be used in the members indicated below in `KMSTRIPCONTEXT`.
Operation is not guaranteed if all parameters are not set.

```
pStripContext->nSize            ( = sizeof(KMSTRIPCONTEXT) )
pStripContext->StripControl.nListType
pStripContext->StripControl.nUserClipMode
pStripContext->ObjectControl.nCullingMode
pStripContext->ObjectControl.dwModifierInstruction
```

Parameters

`pStripHead`(output)	This specifies the pointer for `KMSTRIPHEAD`.
`pStripContext`(input)	This specifies the pointer for `KMSTRIPCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripHead` or `pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmInitStripContext

Initializes StripContext (KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT).

Format

```
KMSTATUS KMAPI
kmInitStripContext(
                IN  KMDWORD             dwIndex,
                OUT PKMVOID             pStripContext
        );
```

Description

This function initializes `StripContext` (`KMSTRIPCONTEXT`).

There are two methods: one that uses the `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`) that was registered by the user through `kmRegisterDefaultStripContext`, and one that uses the basic `KMSTRIPCONTEXT`

(`KMTWOVOLUMESTRIPCONTEXT`) that is provided by the Kamui2 system.

The pStripContext setting depends on the VertexType to be used.

- When using VertextType00 to 08, 15, or 16: Specify the pointer for `KMSTRIPCONTEXT`.
- When using VertextType09 to 14 (two parameters): Specify the pointer for `KMTWOVOLUMESTRIPCONTEXT`.

Parameters

`dwIndex`(input)

This sets up the index (below) that is to be used.

- When using the basic `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`) that is provided by the system

For a polygon: Specify by ORing the following setting flags and the ListType.

`KM_STRIPCONTEXT_SYS_FLAT`

Basic setting for Flatshading

`KM_STRIPCONTEXT_SYS_GOURAUD`

Basic setting for `GouraudShading`

(Example)

`KM_STRIPCONTEXT_SYS_GOURAUD | KM_OPAQUE_POLYGON`

Use the basic setting for `GouraudShading` in `OpaquePolygon`.

For a modifier: Specify by ORing the following setting flags and the ListType.

`KM_STRIPCONTEXT_SYS_NORMAL_MODIFIER`

Basic setting for normal modifiers (except first and last)

`KM_STRIPCONTEXT_SYS_INCLUDE_FIRST_MODIFIER`

Basic setting for first INCLUDE modifier

`KM_STRIPCONTEXT_SYS_EXCLUDE_FIRST_MODIFIER`

Basic setting for first EXCLUDE modifier

`KM_STRIPCONTEXT_SYS_INCLUDE_LAST_MODIFIER`

Basic setting for last INCLUDE modifier

`KM_STRIPCONTEXT_SYS_EXCLUDE_LAST_MODIFIER`

Basic setting for last EXCLUDE modifier

(Example)

`KM_STRIPCONTEXT_SYS_INCLUDE_LAST_MODIFIER | KM_OPAQUE_MODIFIER`

Use the basic setting for the last INCLUDE modifier in OpaqueModifier.

- When using a user-defined `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`) that was registered through

`kmRegisterDefaultStripContext`:

```
KM_STRIPCONTEXT_USER00
KM_STRIPCONTEXT_USER01
KM_STRIPCONTEXT_USER02
KM_STRIPCONTEXT_USER03
KM_STRIPCONTEXT_USER04
KM_STRIPCONTEXT_USER05
KM_STRIPCONTEXT_USER06
KM_STRIPCONTEXT_USER07
KM_STRIPCONTEXT_USER08
KM_STRIPCONTEXT_USER09
```

`pStripContext`(output)

This specifies the pointer for the structure that is to be used (either KMSTRIPCONTEXT or `KMTWOVOLUMESTRIPCONTEXT`).

[Basic `KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`) settings]

```
The settings for the basic KMSTRIPCONTEXT (KMTWOVOLUMESTRIPCONTEXT) that is
provided by the system are shown below.
   pStripContext->StripControl.nListType              Specified ListType
   setting
   pStripContext->StripControl.nUserClipMode          KM_USERCLIP_DISABLE
   pStripContext->StripControl.nShadowMode            KM_NORMAL_POLYGON
   pStripContext->StripControl.nIntensityMode         KM_INTENSITY
   pStripContext->StripControl.bOffset                KM_FALSE
   pStripContext->StripControl.bGouraud               Specified ShadingMode
   setting
   pStripContext->ObjectControl.nDepthCompare         KM_GREATER
   pStripContext->ObjectControl.nCullingMode          KM_NOCULLING
   pStripContext->ObjectControl.bZWriteDisable        KM_FALSE
   pStripContext->ObjectControl.bDCalcControl         KM_FALSE
   pStripContext->ObjectControl.dwModifierInstruction Specified Modifier
   setting
- Common KM_INTENSITY_BASE/KM_INTENSITY_BASE_2ND/KM_INTENSITY_OFFSET
   pStripContext->type.intensity.Face[XX].fAlpha      1.0f
   pStripContext->type.intensity.Face[XX].fRed        1.0f
   pStripContext->type.intensity.Face[XX].fGreen       1.0f
   pStripContext->type.intensity.Face[XX].fBlue       1.0f
- Common KM_IMAGE_PARAM1/KM_IMAGE_PARAM2
   pStripContext->ImageControl[XX].nSRCBlendingMode   KM_ONE
(KM_SRCALPHA when using TransPolygon)
```

```
   pStripContext->ImageControl[XX].nDSTBlendingMode   KM_ZERO
(KM_INVSRCCOLOR when using TransPolygon)
   pStripContext->ImageControl[XX].bSRCSelect         KM_FALSE
   pStripContext->ImageControl[XX].bDSTSelect         KM_FALSE
   pStripContext->ImageControl[XX].nFogMode           KM_NOFOG
   pStripContext->ImageControl[XX].bColorClamp        KM_FALSE
   pStripContext->ImageControl[XX].bUseAlpha          KM_FALSE
(KM_TRUE when using TransPolygon)
   pStripContext->ImageControl[XX].bIgnoreTexureAlpha KM_FALSE
   pStripContext->ImageControl[XX].nFlipUV            KM_NOFLIP
   pStripContext->ImageControl[XX].nClampUV           KM_NOCLAMP
   pStripContext->ImageControl[XX].nFilterMode        KM_POINT_SAMPLE
   pStripContext->ImageControl[XX].bSuperSampleMode   KM_FALSE
   pStripContext->ImageControl[XX].dwMipmapAdjust     KM_MIPMAP_D_ADJUST_1_00
   pStripContext->ImageControl[XX].nTextureShadingModeKM_MODULATE
(KM_MODULATE_ALPHA when using TransPolygon)
   pStripContext->ImageControl[XX].dwPaletteBank      0
   pStripContext->ImageControl[XX].pTextureSurfaceDescNULL
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`pStripContext` is invalid (NULL).
`KMSTATUS_INVALID_SETTING`	`pStripContext` size setting is invalid.

kmRegisterStripContext Registers with the system KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT.(user-defined)

Format

```
KMSTATUS KMAPI
kmRegisterStripContext(
                IN  KMDWORD            dwIndex,
                IN  PKMSTRIPCONTEXT    pStripContext
        );
```

Description

This function registers a user-defined KMSTRIPCONTEXT (KMTWOVOLUMESTRIPCONTEXT) in the Kamui2 system.

Ten KMSTRIPCONTEXT (KMTWOVOLUMESTRIPCONTEXT) entries can be registered in Kamui2.

A KMSTRIPCONTEXT (KMTWOVOLUMESTRIPCONTEXT) registered by this function can be used in kmInitStripContext.

Parameters

dwIndex(input)

This sets one of the following indexes as the index where KMSTRIPCONTEXT (KMTWOVOLUMESTRIPCONTEXT) is to be registered.

```
KM_STRIPCONTEXT_USER00
KM_STRIPCONTEXT_USER01
KM_STRIPCONTEXT_USER02
KM_STRIPCONTEXT_USER03
KM_STRIPCONTEXT_USER04
KM_STRIPCONTEXT_USER05
KM_STRIPCONTEXT_USER06
KM_STRIPCONTEXT_USER07
KM_STRIPCONTEXT_USER08
KM_STRIPCONTEXT_USER09
```

pStripContext(input)	This specifies the pointer for KMSTRIPCONTEXT (KMTWOVOLUMESTRIPCONTEXT). When using KMTWOVOLUMESTRIPCONTEXT (two-parameter polygon), cast to the PKMSTRIPCONTEXT type.

```
ex)
KMTWOVOLUMESTRIPCONTEXT  TwoVolStripContext;
...
kmRegisterStripContext((PKMSTRIPCONTEXT)&TwoVolStripContext,KM_STRIPCONTEXT_USER01);
```

Return values

KMSTATUS_SUCCESS	Success
KMSTATUS_OUT_OF_RANGE	dwIndex setting is invalid.
KMSTATUS_INVALID_ADDRESS	pStripContext is invalid (NULL).
KMSTATUS_INVALID_SETTING	>pStripContext size setting is invalid.

(For compatibility purpose)

kmProcessVertexRenderState

Sets rendering parameters.

Format

```
KMSTATUS KMAPI
kmProcessVertexRenderState(
        IN OUT  PKMVERTEXCONTEXT    pVertexContext
);
```

Description

This function sets rendering parameters.

For details on the settings within `KMVERTEXCONTEXT`, refer to the description of the `KMVERTEXCONTEXT` structure.

Parameters

`pVertexContext`(input/output)	This parameter is a pointer to `KMVERTEXCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success

kmRegisterDefaultContext

Creates user-defined flags for initializing StripContext.

Format

```
KMSTATUS KMAPI
kmRegisterDefaultContext(
                OUT KMDWORD             dwPresetFlag,
                IN  PKMSTRIPCONTEXT     pStripContext
);
```

Description

Creates user-defined flags for initializing `StripContext`.

Creates PresetFlag that can automatically set the following members from `KMSTRIPCONTEXT` settings specified in `pStripContext`.

`ListType`,`UserClipMode`,`ShadowMode`,`IntensityMode`,`Offset`
`Gouraud`,`DepthMode`,`CullingMode`,`ZWrite`,`DCalcExact`,`ModifierInstruction`

Parameters

`dwPresetFlag` (output)	Creates user-defined flags for `StripContext` initialization
`pStripContext` (input)	Pointer to `KMSTRIPCONTEXT`

Return values

`KMSTATUS_SUCCESS`	Success

kmRegisterDefaultStripContext

Registers KMSTRIPCONTEXT in the system (User defined).

Format

```
KMSTATUS KMAPI
kmRegisterDefaultStripContext(
                IN  PKMSTRIPCONTEXT     pStripContext,
                IN  KMDWORD             dwIndex
        );
```

Description

Internally registers the user-defined `KMSTRIPCONTEXT` in KAMUI2.

10 entries of `KMSTRIPCONTEXT` can be internally registered in KAMUI2.

Here, the registered `KMSTRIPCONTEXT` can be used with `kmInitStripContext`.

Parameters

`pStripContext` (input)

Specifies the pointer to `KMSTRIPCONTEXT`.

`dwIndex` (input)

Specifies the following indexes to register.

```
KM_STRIPCONTEXT_USER00
KM_STRIPCONTEXT_USER01
KM_STRIPCONTEXT_USER02
KM_STRIPCONTEXT_USER03
KM_STRIPCONTEXT_USER04
KM_STRIPCONTEXT_USER05
KM_STRIPCONTEXT_USER06
KM_STRIPCONTEXT_USER07
KM_STRIPCONTEXT_USER08
KM_STRIPCONTEXT_USER09
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_OUT_OF_RANGE`	`dwIndex` setting is invalid.
`KMSTATUS_INVALID_ADDRESS`	`pStripContext` is invalid (NULL).

kmSetModifierRenderState

Registers with the system as the second rendering parameter.

Format

```
KMSTATUS KMAPI
kmSetModifierRenderState(
        OUT PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN  PKMVERTEXCONTEXT    pVertexContext
);
```

Description

This function registers with the system as the second rendering parameter.
(`pVertexBuffDesc`->pGlobalParam)
When starting a vertex data strip with the data registered here, use `kmStartVertexStrip`.

Parameters

`pVertexBuffDesc`(output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pVertexContext`(input)	This parameter is a pointer to `KMVERTEXCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetStripHead

Registers with the system as parameters to be used for rendering.(KMSTRIPHEAD)

Format

```
KMSTATUS KMAPI
kmSetStripHead(
        OUT PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN  PKMSTRIPHEAD        pStripHead
);
```

Description

This function registers the rendering parameters (`KMSTRIPHEAD`) in `pGlobalParam` from `pVertexBuffDesc`.

When starting a vertex data strip with the data registered here, use `kmStartVertexStrip`.

Parameters

`pVertexBuffDesc`(output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pStripHead`(input)	This parameter is a pointer to `KMSTRIPHEAD`.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetVertexRenderState

Registers with the system as parameters to be used for rendering.

Format

```
KMSTATUS KMAPI
kmSetVertexRenderState(
        OUT PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN  PKMVERTEXCONTEXT    pVertexContext
);
```

Description

This function registers with the system as parameters to be used for rendering.
    `(VertexBuffDesc->pGlobalParam)`
When starting a vertex data strip with the data registered here, use `kmStartVertexStrip`.

Parameters

`pVertexBuffDesc`(output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pVertexContext`(input)	This parameter is a pointer to `KMVERTEXCONTEXT`.

Return values

`KMSTATUS_SUCCESS`	Success

(Description of the structures to be used)

KMSTRIPCONTEXT

KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT Structure

In the previous version of Kamui, the KMVERTEXCONTEXT structure was used for the rendering parameter settings. Although Kamui2 permits the use of the KMVERTEXCONTEXT structure for compatibility, Kamui2 also provides the KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT structures for faster and more efficient processing.

In Kamui2, we strongly recommend using the KMSTRIPCONTEXT/ KMTWOVOLUMESTRIPCONTEXT structure.

In Kamui2, the rendering parameters that can be set for each vertex (strip) are centralized

in the KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT structure.

Which structure is used depends on the VertexType:

- When using VertextType00 to 08, 15 to 17: Use KMSTRIPCONTEXT structure.
- When using VertextType09 to 14: Use the KMTWOVOLUMESTRIPCONTEXT structure.

Initially, the application allocates the KMSTRIPHEAD structure and the KMSTRIPCONTEXT/ KMTWOVOLUMESTRIPCONTEXT structure, and sets the values for the necessary members in the KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT structure. Which members are necessary depends on the VertexType that is being used.

Next, complete KMSTRIPHEAD with kmGenerateStripHeadXX (00 to 17: VertexType to be used).

The kmChangeStripXxxxxx API can be used to change some members in a KMSTRIPHEAD structure that has already been completed.

The members that are to be set are indicated below.

```
[KMSTRIPCONTROL]
typedef struct _tagKMSTRIPCONTROL
{
KMLISTTYPE          nListType;
KMUSERCLIPMODE      nUserClipMode;
KMSHADOWMODE        nShadowMode;
KMCOLORTYPE         nIntensityMode;
KMBOOLEAN           bOffset;
KMBOOLEAN           bGouraud;
}KMSTRIPCONTROL,*PKMSTRIPCONTROL;
```

- nListType (corresponding VertexType: 00 to 17)

  This sets the list type in which the vertex data is to be stored.

- In the case of VertexType 00 to 16, select the PolygonType as indicated below.

KM_OPAQUE_POLYGON	Sets opaque polygon for the ListType.
KM_TRANS_POLYGON	Sets translucent polygon for the ListType.
KM_PUNCHTHROUGH_POLYGON	Sets punch-through polygon for the ListType.

- In the case of VertexType 17, select the ModifierType as indicated below.

KM_OPAQUE_MODIFIER	Sets opaque modifier volume for the ListType.
KM_TRANS_MODIFIER	Sets translucent modifier for the ListType.

- nUserClipMode (corresponding VertexType: 00 to 17)

  This sets the effect of the clipping area set by kmSetUserClipping.

`KM_USERCLIP_DISABLE`	Disables user clipping. (User clipping has no effect.)
`KM_USERCLIP_INSIDE`	Enables the inside of the specified clipping area. (The outside is clipped.)
`KM_USERCLIP_OUTSIDE`	Enables the outside of the specified clipping area. (The inside is clipped.)

- `nShadowMode` (corresponding VertextType: 00 to 08, 15, 16)

This sets the effect of kmSetCheapShadowMode.

`KM_NORMAL_POLYGON`	CheapShadow has no effect.
`KM_CHEAPSHADOW_POLYGON`	CheapShadow has effect.

- `nIntensityMode` (corresponding VertexType: 02, 07, 08, 10, 13, 14)

This sets IntensityMode.

`KM_INTENSITY`	Sets FaceColor and uses IntensityColor.
`KM_INTENSITY_PREV_FACE_COL`	Uses the last FaceColor that was specified in the last Intensity format that was registered. When using this type of polygon, a `KM_INTENSITY` type polygon must have been used at least once previously within the same scene.

- `bOffset` (compatible VertexType: 03 to 08, 11 to 14, 16)

This sets the offset color.

When using BUMP mapping, set bOffset to `KM_TRUE`.

`KM_TRUE`	Uses offset color.
`KM_FALSE`	Does not use offset color.

- `bGouraud` (compatible VertexType: 00 to 14)

This sets the shading mode.

When `KM_FALSE` has been set, the color data for the first and second vertices in the vertex strip become invalid. (The data for the third vertex is valid.)

`KM_TRUE`	Uses gouraud shading.
`KM_FALSE`	Does not use gouraud shading.

```
[KMOBJECTCONTROL]
typedef struct _tagKMOBJECTCONTROL
{
    KMDEPTHMODE          nDepthCompare;
    KMCULLINGMODE        nCullingMode;
    KMBOOLEAN            bZWriteDisable;
    KMBOOLEAN            bDCalcControl;
    KMDWORD              dwModifierInstruction;
}KMOBJECTCONTROL,*PKMOBJECTCONTROL;
```

- `nDepthCompare` (compatible VertexType: 00 to 16)

  This sets the Z value comparison mode.

  ```
  KM_IGNORE
  KM_LESS
  KM_EQUAL
  KM_LESSEQUAL
  KM_GREATER
  KM_NOTEQUAL
  KM_GREATEREQUAL
  KM_ALWAYS
  ```

- `nCullingMode` (corresponding VertexType: 00 to 17)

  This sets the culling mode.

  In order to enable culling, it is necessary to execute `kmSetCullingRegister` in the global settings. If `KM_CULLCCW` or `KM_CULLCW` are specified, small polygon culling is also performed simultaneously.

`KM_NOCULLING`	No culling is performed.
`KM_CULLSMALL`	Performs small polygon culling.
`KM_CULLCCW`	Performs culling in the counterclockwise direction.
`KM_CULLCW`	Performs culling in the clockwise direction.

- `bZWriteDisable` (corresponding VertexType: 00 to 16)

  This sets the Z value operation.

`KM_TRUE`	Disables Z value updating.
`KM_FALSE`	Enables Z value updating.

- `bDCalcExact` (corresponding VertexType: 03 to 08, 11 to 14, 16)

  This sets the calculation precision for the D parameter that is used in mipmapping.

  If `KM_TRUE` is set, D parameter calculations are performed with precision.

  However, this calculation consumes more time, so the speed of operation may slow down.

`KM_TRUE`	Calculates the D parameter with precision.
`KM_FALSE`	Does not calculate the D parameter with precision.

- ModifierInstruction (corresponding VertexType: 17)

  When registering a modifier volume, this sets the type of polygon data that is to be registered.

`KM_MODIFIER_INCLUDE_FIRST_POLY`	Indicates that the data is for the first polygon of an Inclusion modifier volume.
`KM_MODIFIER_EXCLUDE_FIRST_POLY`	Indicates that the data is for the first polygon of an Exclusion modifier volume.
`KM_MODIFIER_INCLUDE_LAST_POLY`	Indicates that the data is for the last polygon of an Inclusion modifier volume.
`KM_MODIFIER_EXCLUDE_LAST_POLY`	Indicates that the data is for the last polygon of an Exclusion modifier volume.
`KM_MODIFIER_NORMAL_POLY`	Indicates that the data is for a polygon that is neither the first nor the last polygon of a modifier volume.

```
[KMIMAGECONTROL]
typedef struct _tagKMIMAGECONTROL
{
    KMBLENDINGMODE          nSRCBlendingMode;
    KMBLENDINGMODE          nDSTBlendingMode;
    KMBOOLEAN               bSRCSelect;
    KMBOOLEAN               bDSTSelect;
    KMFOGMODE               nFogMode;
    KMBOOLEAN               bColorClamp;
    KMBOOLEAN               bUseAlpha;
    KMBOOLEAN               bIgnoreTextureAlpha;
    KMFLIPMODE              nFlipUV;
    KMCLAMPMODE             nClampUV;
    KMFILTERMODE            nFilterMode;
    KMBOOLEAN               bSuperSampleMode;
    KMDWORD                 dwMipmapAdjust;
    KMTEXTURESHADINGMODE    nTextureShadingMode;
    KMDWORD                 dwPaletteBank;
    PKMSURFACEDESC          pTextureSurfaceDesc;
}KMIMAGECONTROL,*PKMIMAGECONTROL;
```

- `nSRCBlendingMode` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)
- `nDSTBlendingMode` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)

This sets the Blending mode.

`KM_BOTHINVSRCALPHA`	Uses (1-[alpha]s, 1-[alpha]s, 1-[alpha]s, 1-[alpha]s) as the source blending parameters and ([alpha]s, [alpha]s, [alpha]s, [alpha]s) as the destination blending parameters. When SRCBlendingMode has been set, the `DSTBlendingMode` is overridden, and vice versa.
`KM_BOTHSRCALPHA`	Uses ([alpha]s, [alpha]s, [alpha]s, [alpha]s) as the source blending parameters and (1-[alpha]s, 1-[alpha]s, 1-[alpha]s, 1-[alpha]s) as the destination blending parameters. When SRCBlendingMode has been set, the DSTBlendingMode is overridden, and vice versa.
`KM_DESTALPHA`	Uses ([alpha]d, [alpha]d, [alpha]d, [alpha]d) as the blending parameters.
`KM_DESTCOLOR`	Uses ([alpha]d, Rd, Gd, Bd) as the blending parameters.
`KM_INVDESTALPHA`	Uses (1-[alpha]d, 1-[alpha]d, 1-[alpha]d, 1-[alpha]d) as the blending parameters.
`KM_INVDESTCOLOR`	Uses (1-[alpha]d, 1-Rd, 1-Gd, 1-Bd) as the blending parameters.
`KM_INVSRCALPHA`	Uses (1-[alpha]s, 1-[alpha]s, 1-[alpha]s, 1-[alpha]s) as the blending parameters.
`KM_INVSRCCOLOR`	Uses (1-[alpha]s, 1-Rs, 1-Gs, 1-Bs) as the blending parameters.
`KM_SRCALPHA`	Uses ([alpha]s, [alpha]s, [alpha]s, [alpha]s) as the blending parameters.
`KM_SRCCOLOR`	Uses ([alpha]s, Rs, Gs, Bs) as the blending parameters.
`KM_ONE`	Uses (1, 1, 1, 1) as the blending parameters.
`KM_ZERO`	Uses (0, 0, 0, 0) as the blending parameters.

Note: ([alpha]s, Rs, Gs, Bs) indicate the source colors, and ([alpha]d, Rd, Gd, Bd) indicate the destination colors. In `KMSTRIPCONTEXT` for the background plane, set `KM_ZERO` for `DSTBlendingMode`. (Refer to the description of `kmSetBackGroundRenderState`.)

- `bSRCSel` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)

This selects usage for the source color.

`KM_TRUE`	Uses the second accumulation buffer as the source color.
`KM_FALSE`	Does not use the second accumulation buffer as the source color.

- `bDSTSel` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)

This selects usage for the destination color.

`KM_TRUE`	Uses the second accumulation buffer as the destination color.
`KM_FALSE`	Does not use the second accumulation buffer as the destination color.

- `nFogMode` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)

This sets the fog mode.

`KM_FOGTABLE`	Generates the fog [alpha] value through linear interpolation based on the table data corresponding to the depth value.
`KM_FOGTABLE_2`	Substitutes the polygon color for the fog color, and the polygon [alpha] value for the fog [alpha] value.
`KM_FOGVERTEX`	Uses the [alpha] value of OffsetColor as the Fog [alpha] value. Enable `OffsetColor` when using this setting. If `OffsetColor` is disabled, this setting has the same effect as `KM_NOFOG`.
`KM_NOFOG`	Does not perform fog processing.

- `bColorClamp` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)

This sets the color clamp.

The hardware clamps the pixel color to the clamp value set by kmSetColorClampValue.

`KM_TRUE`	Enables the color clamp.
`KM_FALSE`	Disables the color clamp.

- `bUseAlpha` (corresponding `VertexType`: [When `KM_IMAGE_PARAM1` is specified] 00 to 16, [When `KM_IMAGE_PARAM2` is specified] 09 to 14)

This sets vertex [alpha].

Even in the case of TransPolygon, an object does not become translucent if this setting is `KM_FALSE`.

`KM_TRUE`	Enables vertex [alpha].
`KM_FALSE`	Disables vertex [alpha] (opaque).

- `bIgnoreTextureAlpha` (corresponding `VertexType`: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16, [When `KM_IMAGE_PARAM2` is specified] 11 to 14)

  This sets texture [alpha].

  If `KM_TRUE` is specified, the [alpha] bit in texture data is ignored. The hardware ignores transparency information that is included in textures.

`KM_TRUE`	Ignores the [alpha] bit within texture data.
`KM_FALSE`	Enables the [alpha] bit within texture data.

- `nFlipUV` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16, [When `KM_IMAGE_PARAM2` is specified] 11 to 14)

  This sets texture flipping.

`KM_NOFLIP`	No flipping.
`KM_FLIP_V`	Flips in the V coordinate direction.
`KM_FLIP_U`	Flips in the U coordinate direction.
`KM_FLIP_UV`	Flips in the U and V coordinate directions.

- `nClampUV` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16, [When `KM_IMAGE_PARAM2` is specified] 11 to 14)

  This sets texture clamping.

  When ClampUV is enabled, FlipUV setting becomes invalid.

`KM_NOCLAMP`	No clamping.
`KM_CLAMP_V`	Clamps in the V coordinate direction.
`KM_CLAMP_U`	Clamps in the U coordinate direction.
`KM_CLAMP_UV`	Clamps in the U and V coordinate directions.

- `FilterMode` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16, [When `KM_IMAGE_PARAM2` is specified] 11 to 14)

  This sets the texture filter mode.

`KM_POINT_SAMPLE`	Uses the point sampling filter.
`KM_BILINEAR`	Uses the bilinear filter.
`KM_TRILINEAR_A`	Uses the trilinear filter (1st pass).
`KM_TRILINEAR_B`	Uses the trilinear filter (2nd and later passes).

- `bSuperSample` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16, [When `KM_IMAGE_PARAM2` is specified] 11 to 14)

  This sets the 4x super sampling filter (anisotoropic filter).

  Using this setting improves the quality of texture mapping, but at a some loss of performance.

`KM_TRUE`	Enables the 4x super sampling filter (anisotoropic filter).
`KM_FALSE`	Disables the 4x super sampling filter (anisotoropic filter).

- dwMipMapAdjust (corresponding VertexType: [When KM_IMAGE_PARAM1 is specified] 03 to 08, 11 to 14, 16,

[When KM_IMAGE_PARAM2 is specified] 11 to 14)

This sets the coefficient for calculating the D parameter for mipmap level selection.

Normally, set KM_MIPMAP_D_ADJUST_1_00 (D = 1.0).

KM_MIPMAP_D_ADJUST_0_25	Sets the coefficient to D = 0.25.
KM_MIPMAP_D_ADJUST_0_50	Sets the coefficient to D = 0.50.
KM_MIPMAP_D_ADJUST_0_75	Sets the coefficient to D = 0.75.
KM_MIPMAP_D_ADJUST_1_00	Sets the coefficient to D = 1.00.
KM_MIPMAP_D_ADJUST_1_25	Sets the coefficient to D = 1.25.
KM_MIPMAP_D_ADJUST_1_50	Sets the coefficient to D = 1.50.
KM_MIPMAP_D_ADJUST_1_75	Sets the coefficient to D = 1.75.
KM_MIPMAP_D_ADJUST_2_00	Sets the coefficient to D = 2.00.
KM_MIPMAP_D_ADJUST_2_25	Sets the coefficient to D = 2.25.
KM_MIPMAP_D_ADJUST_2_50	Sets the coefficient to D = 2.50.
KM_MIPMAP_D_ADJUST_2_75	Sets the coefficient to D = 2.75.
KM_MIPMAP_D_ADJUST_3_00	Sets the coefficient to D = 3.00.
KM_MIPMAP_D_ADJUST_3_25	Sets the coefficient to D = 3.25.
KM_MIPMAP_D_ADJUST_3_50	Sets the coefficient to D = 3.50.
KM_MIPMAP_D_ADJUST_3_75	Sets the coefficient to D = 3.75.

- nTextureShadingMode (corresponding VertexType: [When KM_IMAGE_PARAM1 is specified] 03 to 08, 11 to 14, 16, [When KM_IMAGE_PARAM2 is specified] 11 to 14)

This sets the texture blending mode.

Specify one of the values shown below.

KM_DECAL	Adds the offset value to the texture color. Uses the [alpha] value of the texture as is. Pixel Color = TextureRGB + OffsetRGB Pixel [alpha] = Texture [alpha]
KM_MODULATE	Applies the shading effect color to the texture color. Replaces the texture [alpha] value with the shading color [alpha] value. Pixel Color = ShadingRGB x TextureRGB + OffsetRGB Pixel [alpha] = Texture [alpha]
KM_DECAL_ALPHA	Blends the shading color with the texture color according to the texture [alpha] value. Pixel Color = (TextureRGB x Texturef¿) + {ShadingRGB x (1-Texture [alpha])} + OffsetRGB Pixel [alpha] = Shading [alpha]
KM_MODULATE_ALPHA	Applies the shading color to the texture color. Applies the shading color [alpha] value to the texture [alpha] value. Pixel Color = (TextureRGB x ShadingRGB) + OffsetRGB Pixel [alpha] = Shading [alpha] x Texture [alpha]

- `dwPaletteBank` (corresponding `VertexType`: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16,

[When `KM_IMAGE_PARAM2` is specified] 11 to 14)

This sets the palette bank number.

This value is valid only when a palettized texture has been specified as the texture.

If a palettized texture is not used, this value does not need to be set even if a corresponding VertextType is being used. In palettized 4bpp mode, the range of values that can be set is from 0 to 63. In palettized 8bpp mode, the values also range from 0 to 63, but since only the upper two bits of the 6 bits are valid, the four values that can actually be used are 0 (0 to 15), 16 (16 to 31), 32 (32 to 47), and 48 (48 to 63).

(For details, refer to kmSetPaletteData.)

- `pTextureSurfaceDesc` (corresponding VertexType: [When `KM_IMAGE_PARAM1` is specified] 03 to 08, 11 to 14, 16, [When `KM_IMAGE_PARAM2` is specified] 11 to 14)

This sets the pointer for the texture surface Surface structure.

```
[KMSTRIPCONTEXT]
       typedef struct _tagKMSTRIPCONTEXT
       {
           KMSTRIPCONTROL  StripControl;           /* StripControl */
           KMOBJECTCONTROL ObjectControl;          /* ObjectControl*/
           union
           {
               struct {
                   KMFLOATCOLOR    Face[2];
               }intensity;
               struct {
                   KMPACKEDARGB    Base;
                   KMPACKEDARGB    Offset;
               }splite;
           }type;
           KMIMAGECONTROL  ImageControl[2];        /* ImageControl */
       }KMSTRIPCONTEXT,*PKMSTRIPCONTEXT;
       [KMTWOVOLUMESTRIPCONTEXT]
       typedef struct _tagKMTWOVOLUMESTRIPCONTEXT
       {
               KMINT32                 nSize;
               KMSTRIPCONTROL  StripControl;                   /* StripControl */
               KMOBJECTCONTROL ObjectControl;                  /* ObjectControl*/
               union
               {
                       struct {
                               KMFLOATCOLOR    Face[2];
                       }intensity;
                       struct {
                           KMPACKEDARGB        Base;
                           KMPACKEDARGB        Offset;
                       }splite;
               }type;
               KMIMAGECONTROL  ImageControl[2];                /* ImageControl */
       }KMTWOVOLUMESTRIPCONTEXT,*PKMTWOVOLUMESTRIPCONTEXT;
```

- Face (corresponding VertexType: see below)
    This sets FaceColor.
      Face[`KM_INTENSITY_BASE`] (corresponding VertexType: 02, 07, 08, 10, 13, 14)
      Face[`KM_INTENSITY_OFFSET`] (corresponding VertexType: 07, 08)
      Face[`KM_INTENSITY_BASE_2ND`] (corresponding VertexType: 10, 13, 14)
- Base (corresponding VertexType: 15, 16)
    This sets the Sprite BaseColor.
- Offset (corresponding VertexType: 16)
    This sets the Sprite OffsetColor.

KMVERTEXCONTEXT

KMVERTEXCONTEXT Structure (For compatibility purpose)

KMVERTEXCONTEXT is provided for the sake of compatibility with the previous version of Kamui.

In Kamui2, KMSTRIPCONTEXT is newly defined, and makes faster and more efficient processing possible.

Using KMSTRIPCONTEXT in Kamui2 is recommended.

In Kamui, the rendering parameters that can be set for each vertex (strip) are centralized in the KMVERTEXCONTEXT structure. An application may have several of these structures, and can switch among them. First, the application allocates a KMVERTEXCONTEXT structure, and sets the value for each member. Next, the application completes KMVERTEXCONTEXT by executing

kmProcessVertexRenderState. Finally, the application registers the structure in the system by executing kmSetVertexRenderState. Switching among completed structures is possible only by executing kmSetVertexRenderState. To change some of the members in a previously completed KMVERTEXCONTEXT structure, it is necessary to execute kmProcessVertexRenderState and kmSetVertexRenderState again.

For a (two-parameter) polygon that is affected by modifier volumes, two KMVERTEXCONTEXT structures are required, one for the inside of modifier volumes and one for the outside of modifier volumes.

The application allocates these two KMVERTEXCONTEXT structures and then sets the parameters for the inside of modifier volumes and for the outside of modifier volumes, respectively.

The parameters for the outside of modifier volumes are registered in the system through kmProcessVertexRenderState and kmSetVertexRenderState. The parameters for the inside of modifier volumes are registered in the system through kmProcessVertexRenderState and kmSetModifierRenderState.

When first using VERTEXCONTEXT in kmProcessVertexRenderState, it is necessary to specify all members.

Set all of the flags in RenderState, and define all parameters. Operation is not guaranteed if there are any undefined bits.

[KMVERTEXCONTEXT structure]

This structure contains all of the parameters that can be set for each vertex (strip), and 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;              /* DepthMode                  */
KMCULLINGMODE           CullingMode;            /* Culling Mode               */
 KMDWORD                  reserved00;
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;        /* Desitination Blending Mode */
KMBOOLEAN               bSRCSel;                /* Source Select              */
KMBOOLEAN               bDSTSel;                /* Distination Select         */
KMFOGMODE               FogMode;                /* Fogging                    */
KMBOOLEAN               bUseSpecular;           /* Specular Highlight         */
KMBOOLEAN               bUseAlpha;              /* Alpha                      */
KMBOOLEAN               bIgnoreTextureAlpha;    /* Ignore Texture Alpha       */
KMCLAMPMODE             ClampUV;                /* Clamp                      */
KMFLIPMODE              FlipUV;                 /* Flip                       */
KMFILTERMODE            FilterMode;             /* Texture Filter             */
KMBOOLEAN               bSuperSample;           /* Anisotoropic Filter        */
KMDWORD                 MipMapAdjust;           /* Mipmap D Adjust            */
KMTEXTURESHADINGMODE    TextureShadingMode;     /* Texture Shading Mode       */
KMBOOLEAN               bColorClamp;            /* ColorClamp Mode            */
KMDWORD                 PaletteBank;            /* Bank of Palette            */

 /*
  * for Texture Control Bit/Address
  */
PKMSURFACEDESC          pTextureSurfaceDesc;    /* Texture Handle             */

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

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

```
        /*
         * Internal use values.
         */
        KMDWORD               GLOBALPARAMBUFFER;      /*  Grobal Parameter Buffer     */
        KMDWORD               ISPPARAMBUFFER;         /*  ISP Parameter Buffer        */
        KMDWORD               TSPPARAMBUFFER;         /*  TSP Parameter Buffer        */
        KMDWORD               TexturePARAMBUFFER;     /*  Texture Parameter Buffer    */

        /*
         * for ModifierInstruction
         */
        KMDWORD               ModifierInstruction;    /* ModifierInstruction          */
        KMFLOAT                  reserved01;
        KMFLOAT                  reserved02;
        KMFLOAT                  reserved03;
        KMFLOAT                  reserved04;

        KMBOOLEAN             bDCalcExact;            /* D Calc Exact                 */
        KMDWORD                  reserved05;
        KMUSERCLIPMODE        UserClipMode;           /* UserClip Mode                */

    } KMVERTEXCONTEXT, *PKMVERTEXCONTEXT, **PPKMVERTEXCONTEXT;
```

In order to change some or all of the members, use the following status flags to set the new type of member that is to be written in RenderState in the structure before executing `kmProcessVertexRenderState`.

The following status flags can be set in RenderState.

```
KM_PARAMTYPE
KM_LISTTYPE
KM_USERCLIPMODE
KM_COLORTYPE
KM_UVFORMAT
KM_DEPTHMODE
KM_CULLINGMODE
KM_SHADINGMODE
KM_MODIFIER
KM_ZWRITEDISABLE
KM_SRCBLENDINGMODE
KM_DSTBLENDINGMODE
KM_SRCSELECT
KM_DSTSELECT
KM_FOGMODE
KM_USESPECULAR
KM_USEALPHA
KM_IGNORETEXTUREALPHA
KM_FLIPUV
KM_CLAMPUV
KM_FILTERMODE
KM_SUPERSAMPLE
KM_MIPMAPDADJUST
KM_TEXTURESHADINGMODE
KM_COLORCLAMP
KM_PALETTEBANK
KM_DCALCEXACT
```

The following values can be set for each member.

`[ParamType]`

This specifies the vertex data type.

One of the following types can be selected.

`KM_POLYGON`	Normal polygon
`KM_MODIFIERVOLUME`	ModifierVolume
`KM_SPRITE`	Sprite

`[ListType]`

This specifies the type of list in which vertex data is to be stored.

`KM_OPAQUE_POLYGON`	Opaque polygon
`KM_OPAQUE_MODIFIER`	Opaque modifier volume
`KM_TRANS_POLYGON`	Translucent polygon
`KM_TRANS_MODIFIER`	Translucent modifier
`KM_PUNCHTHROUGH_POLYGON`	Punch-through polygon

`[UserClipMode]`

This specifies the effect of the clipping area set by kmSetUserClipping.

`KM_USERCLIP_DISABLE`	Disables user clipping.
`KM_USERCLIP_INSIDE`	Enables the inside of the specified clipping area.
`KM_USERCLIP_OUTSIDE`	Enables the outside of the specified clipping area.

`[ColorType]`

This specifies the vertex color format.

`KM_PACKEDCOLOR`	32bit ARGB packed color format
`KM_FLOATINGCOLOR`	32bit x 4 floating color format
`KM_INTENSITY`	Intensity format
`KM_INTENSITY_PREV_FACE_COL`	Intensity format (See below.)

`KM_INTENSITY_PREV_FACE_COL`

Uses the last Intensity format that was registered for FaceColor. When using this type of polygon, a `KM_INTENSITY` type polygon must have been used at least once previously within the same scene. In the case of a sprite polygon, specify `KM_PACKEDCOLOR` for ColorType.

`[UVFormat]`

This specifies the format for the U and V coordinate parameters that are included in the vertex data.

In 32-bit UV format, the U and V coordinates are expressed in IEEE754 32-bit floating point format.

In 16-bit UV format, the lower 16 bits of the 32-bit UV format are deleted, resulting in lower accuracy.

`KM_32BITUV`	`32bit KMFLOAT format`
`KM_16BITUV`	`16bit KMFLOAT format`

In the case of a sprite polygon, specify KM_16BITUV for UVFormat.

[DepthMode]

This specifies the Z value comparison mode.

```
KMDEPTHMODE
KM_IGNORE
KM_LESS
KM_EQUAL
KM_LESSEQUAL
KM_GREATER
KM_NOTEQUAL
KM_GREATEREQUAL
KM_ALWAYS
```

[CullingMode]

This permits selection of one of four parameters: no culling, clockwise culling, counterclockwise culling, and small polygon culling.

KM_NOCULLING	No culling
KM_CULLSMALL	Small polygon culling
KM_CULLCCW	Counterclockwise culling
KM_CULLCW	Clockwise culling

If KM_CULLSMALL is set, it is necessary to execute kmSetCullingRegister in the global settings.

If KM_CULLCCW or KM_CULLCW are specified, small polygon culling is also performed simultaneously.

[bDCalcExact]

KM_TRUE	Calculates the D parameter with precision.
KM_FALSE	Does not calculate the D parameter with precision.

If KM_TRUE is set, D parameter calculations are performed with precision. However, this calculation consumes more time, so the speed of operation may slow down.

[ShadingMode]

This selects the shading mode. There are four possible combinations of texture/no texture, and flat gouraud.

KM_NOTEXTUREFLAT	No texture, flat shading
KM_NOTEXTUREGOURAUD	No texture, gouraud shading
KM_TEXTUREFLAT	Texture, flat shading
KM_TEXTUREGOURAUD	Texture, gouraud shading

When KM_TEXTUREFLAT has been set, the color data for the first and second vertices in the vertex strip become invalid. (The data for the third vertex is valid.)

In the case of a sprite polygon, specify either KM_NOTEXTUREFLAT or KM_TEXTUREFLAT.

[SelectModifier]

This specifies whether a polygon is to be affected by modifier volumes (for example, a two-parameter polygon or a polygon to which cheap shadow effects are to be applied) or not. If modifier volume effects are enabled, the vertex data structure for two-parameter polygons must be used.

However, in the case of cheap shadow mode (when kmSetCheapShadowMode has been executed), use the vertex data structure for one-parameter polygons.

KM_NOMODIFIER	Does not use modifier volumes.
KM_MODIFIER_A	Modifier volumes have effect.

[`bZWriteDisable`]

`KM_TRUE`	Disables Z value updating.
`KM_FALSE`	Enables Z value updating.

[`SRCBlendingMode`, `DSTBlendingMode`]

This specifies the Blending mode.

`KM_BOTHINVSRCALPHA`	Uses (1-[alpha]s, 1-[alpha]s, 1-[alpha]s, 1-[alpha]s) as the source blending parameters and ([alpha]s, [alpha]s, [alpha]s, [alpha]s) as the destination blending parameters. When SRCBlendingMode has been set, the DSTBlendingMode is overridden, and vice versa.
`KM_BOTHSRCALPHA`	Uses ([alpha]s, [alpha]s, [alpha]s, [alpha]s) as the source blending parameters and (1-[alpha]s, 1-[alpha]s, 1-[alpha]s, 1-[alpha]s) as the destination blending parameters. When SRCBlendingMode has been set, the DSTBlendingMode is overridden, and vice versa.
`KM_DESTALPHA`	Uses ([alpha]d, [alpha]d, [alpha]d, [alpha]d) as the blending parameters.
`KM_DESTCOLOR`	Uses ([alpha]d, Rd, Gd, Bd) as the blending parameters.
`KM_INVDESTALPHA`	Uses (1-[alpha]d, 1-[alpha]d, 1-[alpha]d, 1-[alpha]d) as the blending parameters.
`KM_INVDESTCOLOR`	Uses (1-[alpha]d, 1-Rd, 1-Gd, 1-Bd) as the blending parameters.
`KM_INVSRCALPHA`	Uses (1-[alpha]s, 1-[alpha]s, 1-[alpha]s, 1-[alpha]s) as the blending parameters.
`KM_INVSRCCOLOR`	Uses (1-[alpha]s, 1-Rs, 1-Gs, 1-Bs) as the blending parameters.
`KM_SRCALPHA`	Uses ([alpha]s, [alpha]s, [alpha]s, [alpha]s) as the blending parameters.
`KM_SRCCOLOR`	Uses ([alpha]s, Rs, Gs, Bs) as the blending parameters.
`KM_ONE`	Uses (1, 1, 1, 1) as the blending parameters.
`KM_ZERO`	Uses (0, 0, 0, 0) as the blending parameters.

Note: ([alpha]s, Rs, Gs, Bs) indicate the source colors, and ([alpha]d, Rd, Gd, Bd) indicate the destination colors. In `VERTEXCONTEXT` for the background plane, set zero for `DSTBlendingMode`. (Refer to the description of `kmSetBackGroundRenderState`.)

[`bSRCSel`]

`KM_TRUE`	Uses the contents of the second accumulation buffer as the source color.
`KM_FALSE`	Does not use the contents of the second accumulation buffer as the source color.

[`bDSTSel`]

`KM_TRUE`	Uses the contents of the second accumulation buffer as the destination color.
`KM_FALSE`	Does not use the contents of the second accumulation buffer as the destination color.

[`FogMode`]

This sets the fog mode.

`KM_FOGTABLE`	Generates the fog [alpha] value through linear interpolation based on the table data corresponding to the depth value.

`KM_FOGTABLE_2`	Substitutes the polygon color for the fog color, and the polygon [alpha] value for the fog [alpha] value.
`KM_FOGVERTEX`	Uses the OffsetColor [alpha] value as the Fog [alpha] value.
`KM_NOFOG`	Does not perform fog processing.

`[bUseSpecular]`

Specifies whether to use Specular Highlight (offset color).

`KM_TRUE`	Uses the offset color.
`KM_FALSE`	Does not use the offset color.

When using BUMP mapping, specify KM_TRUE for bUseSpecular.

`[bUseAlpha]`

KM_TRUE	Enables the [alpha] bit within shading colors.
KM_FALSE	Disables the [alpha] bit within shading colors.

`[bIgnoreTextureAlpha]`

`KM_TRUE`	Ignores the [alpha] bit within texture data.
`KM_FALSE`	Enables the [alpha] bit within texture data.

If `KM_TRUE` is specified, the [alpha] bit in texture data is ignored. The hardware ignores transparency information that is included in textures.

`[FlipUV]`

This specifies whether or not to flip patterns when repeatedly mapping textures.

One of the following values can be specified.

`KM_NOFLIP`	No flipping.
`KM_FLIP_V`	Flips in the V coordinate direction.
`KM_FLIP_U`	Flips in the U coordinate direction.
`KM_FLIP_UV`	Flips in the U and V coordinate directions.

`[ClampUV]`

This specifies texture clamping.

One of the following values can be specified. If ClampUV is enabled, the FlipUV specification is ignored.

`KM_NOCLAMP`	No clamping.
`KM_CLAMP_V`	Clamps in the V coordinate direction.
`KM_CLAMP_U`	Clamps in the U coordinate direction.
`KM_CLAMP_UV`	Clamps in the U and V coordinate directions.

`[FilterMode]`

This sets the texture filter mode.

`KM_POINT_SAMPLE`	Point sampling filter
`KM_BILINEAR`	Bilinear filter
`KM_TRILINEAR_A`	Trilinear filter (1st pass)
`KM_TRILINEAR_B`	Trilinear filter (2nd and later passes)

`[bSuperSample]`

`KM_TRUE`	Enables the 4x super sampling filter (anisotoropic filter).
`KM_FALSE`	Disables the 4x super sampling filter (anisotoropic filter).

Using this setting improves the quality of texture mapping, but at a some loss of performance.

[MipMapAdjust]

This sets the coefficient for calculating the D parameter for mipmap level selection. The aliasing adjustment can be made through this parameter.

The lower four bits of these values are valid; the data is fixed-point data, with two bits representing the integer portion, and two bits representing the decimal portion.

```
KM_MIPMAP_D_ADJUST_0_25     /* D=0.25 */
KM_MIPMAP_D_ADJUST_0_50     /* D=0.50 */
KM_MIPMAP_D_ADJUST_0_75     /* D=0.75 */
KM_MIPMAP_D_ADJUST_1_00     /* D=1.00 */
KM_MIPMAP_D_ADJUST_1_25     /* D=1.25 */
KM_MIPMAP_D_ADJUST_1_50     /* D=1.50 */
KM_MIPMAP_D_ADJUST_1_75     /* D=1.75 */
KM_MIPMAP_D_ADJUST_2_00     /* D=2.00 */
KM_MIPMAP_D_ADJUST_2_25     /* D=2.25 */
KM_MIPMAP_D_ADJUST_2_50     /* D=2.50 */
KM_MIPMAP_D_ADJUST_2_75     /* D=2.75 */
KM_MIPMAP_D_ADJUST_3_00     /* D=3.00 */
KM_MIPMAP_D_ADJUST_3_25     /* D=3.25 */
KM_MIPMAP_D_ADJUST_3_50     /* D=3.50 */
KM_MIPMAP_D_ADJUST_3_75     /* D=3.75 */
```

Normally, set `KM_MIPMAP_D_ADJUST_1_00` (1.0).

TextureShadingMode

This specifies the texture blending mode.

Specify one of the values shown below.

KM_DECAL	Adds the offset value to the texture color. Uses the [alpha] value of the texture as is. Pixel Color = TextureRGB + OffsetRGB Pixel [alpha] = Texture [alpha]
KM_MODULATE	Applies the shading effect color to the texture color. Replaces the texture [alpha] value with the shading color [alpha] value. Pixel Color = ShadingRGB x TextureRGB + OffsetRGB Pixel [alpha] = Texture [alpha]
KM_DECAL_ALPHA	Blends the shading color with the texture color according to the texture [alpha] value. Pixel Color = (TextureRGB x Texture [alpha]) + {ShadingRGB x (1- Texture [alpha])} + OffsetRGB Pixel [alpha] = Shading [alpha]

`KM_MODULATE_ALPHA`	Applies the shading color to the texture color. Applies the shading color [alpha] value to the texture [alpha] value. Pixel Color = (TextureRGB x ShadingRGB) + OffsetRGB Pixel [alpha] = Shading [alpha] x Texture [alpha]

[`bColorClamp`]

This specifies whether to use color clamping or not.

The hardware clamps the pixel color to the clamp value set by `kmSetColorClampValue`.

`KM_TRUE`	Enables the color clamp.
`KM_FALSE`	Disables the color clamp.

[`PaletteBank`]

This value is valid only when a palettized texture has been specified as the texture. In palettized 4bpp mode, the range of values that can be set is from 0 to 63. In palettized 8bpp mode, the values also range from 0 to 63, but since only the upper two bits of the 6 bits are valid, the four values that can actually be used are 0 (0 to 15), 16 (16 to 31), 32 (32 to 47), and 48 (48 to 63).

(For details, refer to `kmSetPaletteData`.)

[`pTextureSurfaceDesc`]

This specifies the pointer for the texture surface Surface structure.

To change just a texture without changing the other parameters in `KMVERTEXCONTEXT`, change just the `pTextureSurfaceDesc` member and then call `kmProcessVertexRenderState` and `kmSetVertexRenderstate`.

If `KM_TEXTUREFLAT` or `KMTEXTUREGOURAUD` is specified for `ShadingMode`, Kamui will load the information in the `pTextureSurfaceDesc` member into the system each time that `kmProcessVertexRenderState` is called.

In addition, if NULL is specified for the `pTextureSurfaceDesc` member, then Kamui will not load the texture address even if `KM_TEXTUREFLAT` or `KMTEXTUREGOURAUD` is specified for ShadingMode.

If the texture is undefined and you simply want to set a different parameter first, specify NULL for the `pTextureSurfaceDesc` member.

[`ModifierInstruction`]

When registering a modifier volume, this specifies the type of polygon data that is to be registered.

`KM_MODIFIER_INCLUDE_FIRST_POLY`	Indicates that the data is for the first polygon of an Inclusion modifier volume.
`KM_MODIFIER_EXCLUDE_FIRST_POLY`	Indicates that the data is for the first polygon of an Exclusion modifier volume.
`KM_MODIFIER_INCLUDE_LAST_POLY`	Indicates that the data is for the last polygon of an Inclusion modifier volume.
`KM_MODIFIER_EXCLUDE_LAST_POLY`	Indicates that the data is for the last polygon of an Exclusion modifier volume.
`KM_MODIFIER_NORMAL_POLY`	Indicates that the data is for a polygon that is neither the first nor the last polygon of a modifier volume.

This member does not need to be specified except for a modifier volume.

7. Functions for Controlling Vertex Registration

kmBeginPass

Starts Pass.

Format

```
KMSTATUS KMAPI
kmBeginPass(
                IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc
);
```

Description

This function starts Pass. Operation is not guaranteed if this function is not called at the start of the pass.
(Example)

```
kmBeginScene();
  kmBeginPass();
        ...
        ...
        ...
  kmEndPass();
kmEndScene();
```

Parameters

`pVertexBuffDesc`(input)

This parameter is a pointer to `KMVERTEXBUFFDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmBeginScene

Starts the Scene.

Format

```
KMSTATUS KMAPI
kmBeginScene(
                IN OUT  PKMSYSTEMCONFIGSTRUCT   pSystemConfigStruct
);
```

Description

This function starts the scene.

Operation is not guaranteed if this function is not called at the start of the scene.

(Example)

```
kmBeginScene();
  kmBeginPass();
        ...
        ...
        ...
  kmEndPass();
kmEndScene();
```

Parameters

`pSystemConfigStruct`(input)	This parameter is a pointer to `KMSYSTEMCONFIGSTRUCT`.

Return values

`KMSTATUS_SUCCESS`	Success

kmContinuePass

Continues Pass.

Format

```
KMSTATUS KMAPI
kmContinuePass(
                IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc
);
```

Description

This functions continues Pass.

In a multi-pass situation, `kmEndPass`/`kmBeginPass` can also be used for pass continuation.

The code for a multi-pass situation can be written in the two ways shown below.

(Example) In the case of three passes

```
(1)                            (2)
    kmBeginScene();                 kmBeginScene();
        kmBeginPass();                  kmBeginPass();
            ...                             ...
          (Pass1)                         (Pass1)
            ...                             ...
        kmContinuePass                  kmEndPass();
            ...                         kmBeginPass();
          (Pass2)                           ...
            ...                           (Pass2)
        kmContinuePass                      ...
            ...                         kmEndPass();
          (Pass3)                       kmBeginPass();
            ...                             ...
        kmEndPass();                      (Pass3)
    kmEndScene();                           ...
                                        kmEndPass();
                                    kmEndScene();
```

Parameters

`pVertexBuffDesc`(input)	This parameter is a pointer to `KMVERTEXBUFFDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmEndPass

Ends Pass.

Format

```
KMSTATUS KMAPI
kmEndPass(
                IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc
);
```

Description

This function ends Pass.

In a multi-pass situation, `kmContinuePass` can also be used for pass continuation. The code for a multi-pass situation can be written in the two ways shown below.

(Example) In the case of three passes

```
(1)                            (2)
    kmBeginScene();                 kmBeginScene();
        kmBeginPass();                  kmBeginPass();
            ...                             ...
          (Pass1)                         (Pass1)
            ...                             ...
        kmContinuePass                  kmEndPass();
            ...                         kmBeginPass();
          (Pass2)                           ...
            ...                           (Pass2)
        kmContinuePass                      ...
            ...                         kmEndPass();
          (Pass3)                       kmBeginPass();
            ...                             ...
        kmEndPass();                      (Pass3)
    kmEndScene();                           ...
                                        kmEndPass();
                                    kmEndScene();
```

Parameters

`pVertexBuffDesc`(input)	This parameter is a pointer to `KMVERTEXBUFFDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmEndScene

Ends the Scene.

Format

```
KMSTATUS KMAPI
kmEndScene(
                IN OUT  PKMSYSTEMCONFIGSTRUCT   pSystemConfigStruct
);
```

Description

This function ends the scene.

This API is executed after all passes that are to be used have been registered and kmRender has been executed.

Operation is not guaranteed if this function is not called at the end of the scene.

(Example)

```
kmBeginScene();
  kmBeginPass();
        ...
        ...
        ...
  kmEndPass();
  kmRender();
kmEndScene();
```

Parameters

`pSystemConfigStruct`	(input)	This parameter is a pointer to `KMSYSTEMCONFIGSTRUCT`.

Return values

`KMSTATUS_SUCCESS`	Success

kmEndStrip

Ends Strip (direct transfer).

Format

```
KMSTATUS KMAPI
kmEndStrip(
                IN  PKMVERTEXBUFFDESC   pVertexBuffDesc,
);
```

Description

This function reports the end of a vertex data strip.

(Example)

```
        ...
kmStartStrip();
kmSetVertex();
kmSetVertex();
kmSetVertex();
kmSetVertex();
kmEndStrip();
        ...
```

Parameters

`pVertexBuffDesc`(input)	This parameter is a pointer to `KMVERTEXBUFFDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetUserClipping

Set UserClippingParameter.

Format

```
KMSTATUS KMAPI
kmSetUserClipping(
                IN  PKMVERTEXBUFFDESC   pVertexBuffDesc,
                IN  KMUSERCLIPMODE      nPrevUserClipMode,
                IN  KMLISTTYPE          nListType,
                IN  PKMRECT             pRect
);
```

Description

This function sets up the user clipping area.

The user clipping area that is specified here is valid for polygons for which `KM_USERCLIP_INSIDE` or `KM_USERCLIP_OUTSIDE` was specified in the `KMSTRIPCONTEXT` structure's `StripControl.nUserClipMode`

member and the `KMVERTEXCONTEXT` structure's `UserClipMode` member.

Note that this area is valid for individual ListTypes.

Note: It is not possible to clip only a portion of a strip when registering a vertex strip. Specifically, `kmSetUserClipping` must not be issued from the point when a vertex strip started by `kmStartVertexStrip` is registered until `KMVERTEXPARAM_ENDOFSTRIP` is registered by `kmSetVertex`.

Parameters

`pVertexBuffDesc`(input)

This parameter is a pointer to `KMVERTEXBUFFDESC`.

`nPrevUserClipMode`(input)

This sets UserClipMode before switching.

```
KM_USERCLIP_DISABLE
KM_USERCLIP_INSIDE
KM_USERCLIP_OUTSIDE
```

When specifying this at the start of each pass, set `KM_USERCLIP_DISABLE`.

`nListType`(input)

This sets the ListType for which the `UserClipping` area is to be set up.

```
KM_OPAQUE_POLYGON
KM_OPAQUE_MODIFIER
KM_TRANS_POLYGON
KM_TRANS_MODIFIER
KM_PUNCHTHROUGH_POLYGON
```

`pRect`(input)

This specifies the pointer to `KMRECT`.

This sets the upper left and lower right coordinates for a user clipping area in `KMRECT`.

The values that are specified here are given in units of tiles. (1 = 32 pixels)

pRect->nXmin	(only lower 6 bits are valid)
pRect->nYmin	(only lower 4 bits are valid)
pRect->nXmax	(only lower 6 bits are valid)
pRect->nYmax	(only lower 4 bits are valid)

Return values

`KMSTATUS_SUCCESS`	Success

kmSetVertex

Sends VertexParameter.

Format

```
KMSTATUS KMAPI
kmSetVertex(
                IN  PKMVERTEXBUFFDESC   pVertexBuffDesc,
                IN  PVOID               pVertex,
                IN  KMVERTEXTYPE        nVertexType,
                IN  KMUINT32            nStructSize
);
```

Description

This writes the vertex data that is indicated by `pVertex` to the vertex buffer that is set by `pVertexBuffDesc` -> `pCurrentListState` -> `ListType`.

Operation is not guaranteed if `ParamControlWord` in the vertex data at the end of the strip is not `KM_VERTEXPARAM_ENDOFSTRIP`.

Parameters

`pVertexBuffDesc`(input)

This parameter is a pointer to `KMVERTEXBUFFDESC`.

`pVertex`(input)

This parameter is a pointer to the vertex data structure.

`nVertexType`(input)

VertexType setting:

```
KM_VERTEXTYPE_00
KM_VERTEXTYPE_01
KM_VERTEXTYPE_02
KM_VERTEXTYPE_03
KM_VERTEXTYPE_04
KM_VERTEXTYPE_05
KM_VERTEXTYPE_06
KM_VERTEXTYPE_07
KM_VERTEXTYPE_08
KM_VERTEXTYPE_09
KM_VERTEXTYPE_10
KM_VERTEXTYPE_11
KM_VERTEXTYPE_12
KM_VERTEXTYPE_13
KM_VERTEXTYPE_14
KM_VERTEXTYPE_15
KM_VERTEXTYPE_16
KM_VERTEXTYPE_17
```

`nStructSize`(input)	This parameter specifies the vertex data size. Specify in accordance with the type to be used for the vertex data, such as sizeof(`KMVERTEX_01`).

Return values

`KMSTATUS_SUCCESS`	Success

kmStartStrip

Performs start of Strip (direct transfer).

Format

```
KMSTATUS KMAPI
kmStartStrip(
                IN  PKMVERTEXBUFFDESC   pVertexBuffDesc,
                IN  PKMSTRIPHEAD        pStripHead
);
```

Description

This function performs start of `Strip`(direct transfer).

This function writes the rendering parameters that were constructed in `KMSTRIPHEAD` to the vertex buffer indicated by ListType in the internal data.

This API does not set `pVertexBuffDesc`->`pGlobalParam`, but performs direct transfer.

Note: For the vertex type of the rendering parameter that is written here, use the same vertex type that is written in `kmSetVertex` subsequently to this function. Operation is not guaranteed if these vertex types are different.

Parameters

`pVertexBuffDesc`(input)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pStripHead`(input)	This parameter is a pointer to `KMSTRIPHEAD`.

Return values

`KMSTATUS_SUCCESS`	Success

(For compatibility purpose)

kmEndVertexStrip

Ends Strip.

Format

```
KMSTATUS KMAPI
kmEndVertexStrip(
                IN  PKMVERTEXBUFFDESC   pVertexBuffDesc
);
```

Description

This function reports the end of a vertex data strip.

(Example)

```
                ...
        kmStartVertexStrip();
        kmSetVertex();
        kmSetVertex();
        kmSetVertex();
        kmSetVertex();
        kmEndVertexStrip();
                ...
```

Parameters

`pVertexBuffDesc`(input)	This parameter is a pointer to `KMVERTEXBUFFDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmStartVertexStrip

Starts Strip.

Format

```
KMSTATUS KMAPI
kmStartVertexStrip(
                IN  PKMVERTEXBUFFDESC   pVertexBuffDesc
);
```

Description

This function starts Strip.

This function writes the rendering parameters indicated by `pVertexBufferDesc` -> `pGlobalParam` that was set by `kmSetStripHead`/`kmSetVertexRenderState`/`kmSetModifierRenderState` to the vertex buffer of the set ListType.

Note: Operation is not guaranteed if these vertex types are different if the combination of the vertex type of the rendering parameter that is written here and the vertex type that is written in `kmSetVertex` subsequently to this function is incorrect.

Parameters

`pVertexBuffDesc`(input)	This parameter is a pointer to `KMVERTEXBUFFDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

8. Functions for Update

KMSTRIPCONTEXT/KMTWOVOLUMESTRIPCONTEXT

kmChangeStripBlendingMode

Changes BlendingMode.

Format

```
KMSTATUS KMAPI
kmChangeStripBlendingMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBLENDINGMODE  nSRCBlendingMode,
        IN      KMBLENDINGMODE  nDSTBlendingMode
);
```

Description

This function changes BlendingMode in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)

This parameter is a pointer to `KMSTRIPHEAD`.

`nParam`(input)

This parameter specifies the parameter for update.

`KM_IMAGE_PARAM1`	parameter 1
`KM_IMAGE_PARAM2`	parameter 2

`nSRCBlendingMode`(input)

`nDSTBlendingMode`(input)

This parameter specifies the BlendingMode setting.

```
KM_BOTHINVSRCALPHA
KM_BOTHSRCALPHA
KM_DESTALPHA
KM_DESTCOLOR
KM_INVDESTALPHA
KM_INVDESTCOLOR
KM_INVSRCALPHA
KM_INVSRCCOLOR
KM_SRCALPHA
KM_SRCCOLOR
KM_ONE
KM_ZERO
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current `VertexType`.

kmChangeStripClampUV

Changes ClampUV.

Format

```
KMSTATUS KMAPI
kmChangeStripClampUV(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMCLAMPMODE     nClampUV
);
```

Description

This function changes ClampUV in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`nClampUV`(input)	This parameter specifies the ClampUV setting.
`KM_NOCLAMP`	
`KM_CLAMP_V`	
`KM_CLAMP_U`	
`KM_CLAMP_UV`	

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current `VertexType`.

kmChangeStripColorClamp

Changes ColorClamp.

Format

```
KMSTATUS KMAPI
kmChangeStripColorClamp(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bColorClamp
);
```

Description

This function changes `ColorClamp` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`dwColorClamp`(input)	This parameter specifies the ColorClamp setting.

```
KM_TRUE    :   ColorClamp is valid.
KM_FALSE   :   ColorClamp is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current `VertexType`.

kmChangeStripCullingMode

Changes CullingMode.

Format

```
KMSTATUS KMAPI
kmChangeStripCullingMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMCULLINGMODE   nCullingMode
);
```

Description

This function changes `CullingMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 17).

The following is a list of supported VertexType:

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	◎

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nCullingMode`(input)	This parameter specifies the `CullingMode` setting.

```
KM_NOCULLING
KM_CULLSMALL
KM_CULLCCW
KM_CULLCW
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeStripDCalcControl

Changes DCalcControl.

Format

```
KMSTATUS KMAPI
kmChangeStripDCalcControl(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bDCalcControl
);
```

Description

This function changes `DCalcControl` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	◎	VertexType12	◎
VertexType01	X	VertexType07	◎	VertexType13	◎
VertexType02	X	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	X	VertexType15	X
VertexType04	◎	VertexType10	X	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`bDCalcControl`(input)	This parameter specifies the `DCalcControl` setting.
`KM_TRUE`	`: DCalcControl is valid.`
`KM_FALSE`	`: DCalcControl is not valid.`

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current `VertexType`.

kmChangeStripDepthCompareMode

Changes DepthCompareMode.

Format

```
KMSTATUS KMAPI
kmChangeStripDepthCompareMode(
        IN OUT  PKMSTRIPHEAD       pStripHead,
        IN      KMDEPTHMODE        nDepthCompareMode
);
```

Description

This function changes `DepthCompareMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported `VertexType`:

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nDepthCompareMode`(input)	This parameter specifies the `DepthCompareMode` setting.

```
KM_IGNORE
KM_LESS
KM_EQUAL
KM_LESSEQUAL
KM_GREATER
KM_NOTEQUAL
KM_GREATEREQUAL
KM_ALWAYS
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripDSTSelect

Changes DSTSelect.

Format

```
KMSTATUS KMAPI
kmChangeStripDSTSelect(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bDSTSelect
);
```

Description

This function changes `DSTSelect` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`bDSTSelect`(input)	This parameter specifies the `DSTSelect` setting.

```
KM_TRUE    :  DSTSelect is valid.
KM_FALSE :  DSTSelect is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripFaceColor

Changes FaceColor.

Format

```
KMSTATUS KMAPI
kmChangeStripFaceColor(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      PKMFLOATCOLOR   pFaceColor
);
```

Description

This function changes `FaceOffsetColor` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (07, 08).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	◎	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`pFaceColor`(input)	This parameter is a pointer to `KMFLOATCOLOR`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripFaceOffsetColor

Changes FaceOffsetColor.

Format

```
KMSTATUS KMAPI
kmChangeStripFaceOffsetColor(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      PKMFLOATCOLOR   pFaceOffsetColor
);
```

Description

This function changes `FaceOffsetColor` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (07, 08).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	◎	VertexType13	X
VertexType02	X	VertexType08	◎	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	X	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`pFaceOffsetColor`(input)	This parameter is a pointer to `KMFLOATCOLOR`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripFilterMode

Changes FilterMode.

Format

```
KMSTATUS KMAPI
kmChangeStripFilterMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMFILTERMODE    nFilterMode
);
```

Description

This function changes `FilterMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
KM_IMAGE_PARAM1	`parameter 1`
KM_IMAGE_PARAM2	`parameter 2`
`nFilterMode`(input)	This parameter specifies the `FilterMode` setting.
KM_POINT_SAMPLE	
KM_TRILINEAR_A	
KM_BILINEAR	
KM_TRILINEAR_B	

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripFlipUV

Changes FlipUV.

Format

```
KMSTATUS KMAPI
kmChangeStripFlipUV(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMFLIPMODE      nFlipUV
);
```

Description

This function changes FlipUV in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
nFlipUV(input)	This parameter specifies the FlipUV setting.
`KM_NOFLIP`	
`KM_FLIP_V`	
`KM_FLIP_U`	
`KM_FLIP_UV`	

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripFogMode

Changes FogMode.

Format

```
KMSTATUS KMAPI
kmChangeStripFogMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMFOGMODE       nFogMode
);
```

Description

This function changes `FogMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`nFogMode`(input)	This parameter specifies the `FogMode` setting.
`KM_FOGTABLE`	
`KM_FOGVERTEX`	
`KM_NOFOG`	
`KM_FOGTABLE_2`	

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripGouraud

Changes Gouraud.

Format

```
KMSTATUS KMAPI
kmChangeStripGouraud(
        IN OUT  PKMSTRIPHEAD      pStripHead,
        IN      KMBOOLEAN         bGouraud
);
```

Description

This function changes Gouraud in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 14).

The following is a list of supported VertexType:

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	X
VertexType04	◎	VertexType10	◎	VertexType16	X
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`bGouraud`(input)	This parameter specifies the Gouraud setting.

```
KM_TRUE    :   Gouraud is valid.
KM_FALSE   :   Gouraud is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripIgnoreTextureAlpha

Changes IgnoreTextureAlpha.

Format

```
KMSTATUS KMAPI
kmChangeStripIgnoreTextureAlpha(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bIgnoreTextureAlpha
);
```

Description

This function changes `IgnoreTextureAlpha` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bIgnoreTextureAlpha`(input) This parameter specifies the IgnoreTextureAlpha setting.

```
KM_TRUE    :   TextureAlpha is not valid.
KM_FALSE   :   TextureAlpha is valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripIgnoreTexureAlpha

Changes IgnoreTexureAlpha.

Format

```
KMSTATUS KMAPI
kmChangeStripIgnoreTexureAlpha(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bIgnoreTexureAlpha
);
```

Description

This function changes `IgnoreTexureAlpha`.

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bIgnoreTexureAlpha`(input)	This parameter specifies the `IgnoreTexureAlpha` setting.

```
KM_TRUE    :   TexureAlpha is not valid.
KM_FALSE   :   TexureAlpha is valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripIntensityMode

Changes IntensityMode.

Format

```
KMSTATUS KMAPI
kmChangeStripIntensityMode(
        IN OUT  PKMSTRIPHEAD        pStripHead,
        IN      KMCOLORTYPE         nIntensityMode
);
```

Description

This function changes IntensityMode in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (02, 07, 08, 10, 13, 14).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	◎	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nIntensityMode`(input)	This parameter specifies the IntensityMode setting.

```
KM_INTENSITY
KM_INTENSITY_PREV_FACE_COL
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripListType

Changes ListType.

Format

```
KMSTATUS KMAPI
kmChangeStripListType(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMLISTTYPE      nListType
);
```

Description

This function changes ListType in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 17).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	★
VertexType01	★	VertexType07	★	VertexType13	★
VertexType02	★	VertexType08	★	VertexType14	★
VertexType03	★	VertexType09	★	VertexType15	★
VertexType04	★	VertexType10	★	VertexType16	★
VertexType05	★	VertexType11	★	VertexType17	★

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

The ListType that can be changed is determined by VertexType.

VertexType0*`16 Can be one of the following:

```
KM_OPAQUE_POLYGON
KM_TRANS_POLYGON
KM_PUNCHTHROUGH_POLYGON
```

VertexType17 Can be either one of the following:

```
KM_OPAQUE_MODIFIER
KM_TRANS_MODIFIER
```

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nListType`(input)	This parameter specifies the ListType setting.

```
KM_OPAQUE_POLYGON
KM_OPAQUE_MODIFIER
KM_TRANS_POLYGON
KM_TRANS_MODIFIER
KM_PUNCHTHROUGH_POLYGON
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripMipmapAdjust

Changes MipmapAdjust.

Format

```
KMSTATUS KMAPI
kmChangeStripMipmapAdjust(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMDWORD         dwMipmapAdjust
);
```

Description

This function changes `MipmapAdjust` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`dwMipmapAdjust`(input)	This parameter specifies the `MipmapAdjust` setting.

```
KM_MIPMAP_D_ADJUST_0_25
KM_MIPMAP_D_ADJUST_0_50
KM_MIPMAP_D_ADJUST_0_75
KM_MIPMAP_D_ADJUST_1_00
KM_MIPMAP_D_ADJUST_1_25
KM_MIPMAP_D_ADJUST_1_50
KM_MIPMAP_D_ADJUST_1_75
KM_MIPMAP_D_ADJUST_2_00
KM_MIPMAP_D_ADJUST_2_25
KM_MIPMAP_D_ADJUST_2_50
KM_MIPMAP_D_ADJUST_2_75
KM_MIPMAP_D_ADJUST_3_00
KM_MIPMAP_D_ADJUST_3_25
KM_MIPMAP_D_ADJUST_3_50
KM_MIPMAP_D_ADJUST_3_75
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripModifierInstruction

Changes ModifierInstruction.

Format

```
KMSTATUS KMAPI
kmChangeStripModifierInstruction(
        IN OUT  PKMSTRIPHEAD            pStripHead,
        IN      KMDWORD                 dwModifierInstruction
);
```

Description

This function changes `ModifierInstruction` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (17).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	X	VertexType13	X
VertexType02	X	VertexType08	X	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	X	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	◎

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`dwModifierInstruction`(input)	This parameter specifies the ModifierInstruction setting.

```
KM_MODIFIER_NORMAL_POLY
KM_MODIFIER_INCLUDE_FIRST_POLY
KM_MODIFIER_EXCLUDE_FIRST_POLY
KM_MODIFIER_INCLUDE_LAST_POLY
KM_MODIFIER_EXCLUDE_LAST_POLY
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripOffset

Changes Offset.

Format

```
KMSTATUS KMAPI
kmChangeStripOffset(
        IN OUT  PKMSTRIPHEAD       pStripHead,
        IN      KMBOOLEAN          bOffset
);
```

Description

This function changes Offset in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	◎	VertexType12	◎
VertexType01	X	VertexType07	◎	VertexType13	◎
VertexType02	X	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	X	VertexType15	X
VertexType04	◎	VertexType10	X	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`bOffset`(input)	This parameter specifies the Offset setting.

```
KM_TRUE    :   Offset is valid.
KM_FALSE   :   Offset is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripPaletteBank

Changes PaletteBank.

Format

```
KMSTATUS KMAPI
kmChangeStripPaletteBank(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMDWORD         dwPaletteBank
);
```

Description

This function changes PaletteBank in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

In addition, because the `PaletteBank` information is initialized by executing `kmChangeStripTextureSurface`, if it is desired to change `TextureSurface` and `PaletteBank`, execute `kmChangeStripTextureSurface` -> `kmChangeStripPaletteBank`, in that order.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
KM_IMAGE_PARAM1	`parameter 1`
KM_IMAGE_PARAM2	`parameter 2`
`dwPaletteBank`(input)	This parameter specifies the `PaletteBank` setting.(0-63)

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripShadowMode

Changes ShadowMode.

Format

```
KMSTATUS KMAPI
kmChangeStripShadowMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMSHADOWMODE    nShadowMode
);
```

Description

This function changes `ShadowMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 08, 15, 16).

The following is a list of supported VertexType:

VertexType00	◎	VertexType06	◎	VertexType12	X
VertexType01	◎	VertexType07	◎	VertexType13	X
VertexType02	◎	VertexType08	◎	VertexType14	X
VertexType03	◎	VertexType09	X	VertexType15	◎
VertexType04	◎	VertexType10	X	VertexType16	◎
VertexType05	◎	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nShadowMode`(input)	This parameter specifies the `ShadowMode` setting.

`KM_NORMAL_POLYGON`	`normal polygon`
`KM_CHEAPSHADOW_POLYGON`	`simple shadow polygon`

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripSpriteBaseColor

Changes SpriteBaseColor.

Format

```
KMSTATUS KMAPI
kmChangeStripSpriteBaseColor(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMPACKEDARGB    dwBaseColor
);
```

Description

This function changes Sprite `BaseColor` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (15, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	X	VertexType13	X
VertexType02	X	VertexType08	X	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	◎
VertexType04	X	VertexType10	X	VertexType16	◎
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`dwBaseColor`(input)	This parameter specifies the Sprite `BaseColor` setting.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripSpriteOffsetColor

Changes SpriteOffsetColor.

Format

```
KMSTATUS KMAPI
kmChangeStripSpriteOffsetColor(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMPACKEDARGB    dwOffsetColor
);
```

Description

This function changes Sprite `OffsetColor` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	X	VertexType13	X
VertexType02	X	VertexType08	X	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	X	VertexType16	◎
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`dwOffsetColor`(input)	This parameter specifies the Sprite `OffsetColor` setting.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripSRCSelect

Changes SRCSelect.

Format

```
KMSTATUS KMAPI
kmChangeStripSRCSelect(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bSRCSelect
);
```

Description

This function changes `SRCSelect` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`bSRCSelect`(input)	This parameter specifies the `SRCSelect` setting.
`KM_TRUE`	`: SRCSelect is valid.`
`KM_FALSE`	`: SRCSelect is not valid.`

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripSuperSampleMode

Changes SuperSampleMode.

Format

```
KMSTATUS KMAPI
kmChangeStripSuperSampleMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      KMBOOLEAN       bSuperSampleMode
);
```

Description

This function changes `SuperSampleMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1    parameter 1
KM_IMAGE_PARAM2    parameter 2
```

`bSuperSampleMode`(input)	This parameter specifies the `SuperSampleMode` setting.

```
KM_TRUE    :   SuperSampleMode is valid.
KM_FALSE   :   SuperSampleMode is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripTextureAddress

Changes TextureAddress.

Format:

```
KMSTATUS KMAPI
kmChangeStripTextureAddress(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMINT32         nParam,
        IN      PKMSURFACEDESC  pTextureSurfaceDesc
);
```

Description

This function changes `TextureAddress` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16). Because this API does not change anything other than `TextureAddress`, it can be used when there are no changes to the following members.

```
pTextureSurfaceDesc->PixelFormat
pTextureSurfaceDesc->u0.USize
pTextureSurfaceDesc->u1.VSize
pTextureSurfaceDesc->fSurfaceFlags
```

If there are any changes to the above members, use `kmChangeStripTextureSurface`, not `kmChangeStripTextureAddress`.

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`pTextureSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripTextureShadingMode

Changes TextureShadingMode.

Format

```
KMSTATUS KMAPI
kmChangeStripTextureShadingMode(
        IN OUT  PKMSTRIPHEAD            pStripHead,
        IN      KMINT32                 nParam,
        IN      KMTEXTURESHADINGMODE    nTextureShadingMode
);
```

Description

This function changes `TextureShadingMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`nTextureShadingMode`(input)	This parameter specifies the `TextureShadingMode` setting.

`KM_DECAL`
`KM_MODULATE`
`KM_DECAL_ALPHA`
`KM_MODULATE_ALPHA`

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripTextureSurface

Changes TextureSurface.

Format

```
KMSTATUS KMAPI
kmChangeStripTextureSurface(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      PKMSURFACEDESC  pTextureSurfaceDesc
);
```

Description

This function changes `TextureSurface` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (03 to 08, 11 to 14, 16).

The following is a list of supported VertexType:

In addition, because the `PaletteBank` information is initialized as a result of executing this API, set the `PaletteBank` information through `kmChangeStripPaletteBank` if necessary.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`pTextureSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripUseAlpha

Changes UseAlpha.

Format

```
KMSTATUS KMAPI
       kmChangeStripUseAlpha(
               IN OUT  PKMSTRIPHEAD    pStripHead,
               IN      KMINT32         nParam,
               IN      KMBOOLEAN       bUseAlpha
       );
```

Description

This function changes `UseAlpha` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)

This parameter is a pointer to `KMSTRIPHEAD`.

`nParam`(input)

This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bUseAlpha`(input) This parameter specifies the `UseAlpha` setting.

```
KM_TRUE    :ALPHA is valid.
KM_FALSE   :ALPHA is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

kmChangeStripUserClipMode

Changes UserClipMode.

Format

```
KMSTATUS KMAPI
kmChangeStripUserClipMode(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMUSERCLIPMODE  nUserClipMode
);
```

Description

This function changes `UserClipMode` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 17).

The following is a list of supported VertexType:

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	◎

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nUserClipMode`(input)	This parameter specifies the `UserClipMode` setting.

```
KM_USERCLIP_DISABLE
KM_USERCLIP_INSIDE
KM_USERCLIP_OUTSIDE
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeStripZWriteDisable

Changes ZWriteDisable.

Format

```
KMSTATUS KMAPI
kmChangeStripZWriteDisable(
        IN OUT  PKMSTRIPHEAD    pStripHead,
        IN      KMBOOLEAN       bZWriteDisable
);
```

Description

This function changes `ZWriteDisable` in a `KMSTRIPHEAD` structure that was constructed by `kmGenerateStripHead`/`kmGenerateStripHeadXX` (00 to 16).

The following is a list of supported VertexType:

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`bZWriteDisable`(input)	This parameter specifies the ZWriteDisable setting.

```
KM_TRUE    :   ZWriteDisable is not valid.
KM_FALSE   :   ZWriteDisable is valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEX_TYPE`	This setting is invalid at the current VertexType.

9. Functions for Update Rendering Parameter Registration

(For compatibility purpose)

kmChangeContextBlendingMode

Changes BlendingMode.

Format

```
KMSTATUS KMAPI
kmChangeContextBlendingMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBLENDINGMODE      nSRCBlendingMode,
        IN      KMBLENDINGMODE      nDSTBlendingMode
);
```

Description

This function changes the rendering parameter BlendingMode that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output) This parameter is a pointer to `KMVERTEXBUFFDESC`.

`nParam`(input) This parameter specifies the parameter for update.

KM_IMAGE_PARAM1	parameter 1
KM_IMAGE_PARAM2	parameter 2

`nSRCBlendingMode`(input)

`nDSTBlendingMode`(input) This parameter specifies the `BlendingMode` setting.

KM_BOTHINVSRCALPHA
KM_BOTHSRCALPHA
KM_DESTALPHA
KM_DESTCOLOR
KM_INVDESTALPHA
KM_INVDESTCOLOR
KM_INVSRCALPHA
KM_INVSRCCOLOR
KM_SRCALPHA
KM_SRCCOLOR
KM_ONE
KM_ZERO

Return values

KMSTATUS_SUCCESS	Success

kmChangeContextClampUV

Changes ClampUV.

Format

```
KMSTATUS KMAPI
kmChangeContextClampUV(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMCLAMPMODE         nClampUV
);
```

Description

This function changes the rendering parameter ClampUV that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`nClampUV`(input)	This parameter specifies the ClampUV setting.
`KM_NOCLAMP`	
`KM_CLAMP_V`	
`KM_CLAMP_U`	
`KM_CLAMP_UV`	

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextColorClamp

Changes ColorClamp.

Format

```
KMSTATUS KMAPI
kmChangeContextColorClamp(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bColorClamp
);
```

Description

This function changes the rendering parameter ColorClamp that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)This parameter is a pointer to `KMVERTEXBUFFDESC`.

`nParam`(input) This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`dwColorClamp`(input) This parameter specifies the ColorClamp setting.

```
KM_TRUE    :   ColorClamp is valid.
KM_FALSE   :   ColorClamp is not valid.
```

Return values

KMSTATUS_SUCCESS	Success

kmChangeContextCullingMode

Changes CullingMode.

Format

```
KMSTATUS KMAPI
kmChangeContextCullingMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMCULLINGMODE       nCullingMode
);
```

Description

This function changes the rendering parameter `CullingMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	◎

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters:

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nCullingMode`(input)	This parameter specifies the CullingMode setting.

```
KM_NOCULLING
KM_CULLSMALL
KM_CULLCCW
KM_CULLCW
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextDCalcControl

Changes DCalcControl.

Format

```
KMSTATUS KMAPI
kmChangeContextDCalcControl(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bDCalcControl
);
```

Description

This function changes the rendering parameter `DCalcControl` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from pVertexBuffDesc to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	◎	VertexType12	◎
VertexType01	X	VertexType07	◎	VertexType13	◎
VertexType02	X	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	X	VertexType15	X
VertexType04	◎	VertexType10	X	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output) This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input) This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bDCalcControl`(input) This parameter specifies the `DCalcControl` setting.

```
KM_TRUE    :   DCalcControl is valid.
KM_FALSE   :   DCalcControl is not valid.
```

Return values

KMSTATUS_SUCCESS	Success

kmChangeContextDepthCompareMode

Changes DepthCompareMode.

Format

```
KMSTATUS KMAPI
kmChangeContextDepthCompareMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMDEPTHMODE         nDepthCompareMode
);
```

Description

This function changes the rendering parameter `DepthCompareMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nDepthCompareMode`(input)	This parameter specifies the `DepthCompareMode` setting.

KM_IGNORE
KM_LESS
KM_EQUAL
KM_LESSEQUAL
KM_GREATER
KM_NOTEQUAL
KM_GREATEREQUAL
KM_ALWAYS

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextDSTSelect

Changes DSTSelect.

Format

```
KMSTATUS KMAPI
kmChangeContextDSTSelect(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bDSTSelect
);
```

Description

This function changes the rendering parameter `DSTSelect` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pStripHead`(input/output)	This parameter is a pointer to `KMSTRIPHEAD`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`bDSTSelect`(input)	This parameter specifies the DSTSelect setting.
`KM_TRUE`	`: DSTSelect is valid.`
`KM_FALSE`	`: DSTSelect is not valid.`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextFaceColor

Changes FaceColor.

Format

```
KMSTATUS KMAPI
kmChangeContextFaceColor(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      PKMFLOATCOLOR       pFaceColor
);
```

Description

This function changes the rendering parameter FaceColor that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	◎	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`pFaceColor`(input)	This parameter is a pointer to `KMFLOATCOLOR`.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextFaceOffsetColor

Changes FaceOffsetColor.

Format

```
KMSTATUS KMAPI
kmChangeContextFaceOffsetColor(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      PKMFLOATCOLOR       pFaceOffsetColor
);
```

Description

This function changes the rendering parameter `FaceOffsetColor` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	◎	VertexType13	X
VertexType02	X	VertexType08	◎	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	X	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pFaceOffsetColor`(input)	This parameter is a pointer to `KMFLOATCOLOR`.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextFilterMode

Changes FilterMode.

Format

```
KMSTATUS KMAPI
kmChangeContextFilterMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMFILTERMODE        nFilterMode
);
```

Description

This function changes the rendering parameter FilterMode that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`nFilterMode`(input)	This parameter specifies the FilterMode setting.
`KM_POINT_SAMPLE`	
`KM_TRILINEAR_A`	
`KM_BILINEAR`	
`KM_TRILINEAR_B`	

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextFlipUV

Changes FlipUV.

Format

```
KMSTATUS KMAPI
kmChangeContextFlipUV(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMFLIPMODE          nFlipUV
);
```

Description

This function changes the rendering parameter FlipUV that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.
The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.

`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`

`nFlipUV`(input) This parameter specifies the FlipUV setting:

`KM_NOFLIP`
`KM_FLIP_V`
`KM_FLIP_U`
`KM_FLIP_UV`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextFogMode

Changes FogMode.

Format

```
KMSTATUS KMAPI
kmChangeContextFogMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMFOGMODE           nFogMode
);
```

Description

This function changes the rendering parameter `FogMode` that was registered in pGlobalParam from pVertexBuffDesc by kmSetVertexRenderState/kmSetStripHead.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to KMVERTEXBUFFDESC.
`nParam`(input)	This parameter specifies the parameter for update.

KM_IMAGE_PARAM1	parameter 1
KM_IMAGE_PARAM2	parameter 2

`nFogMode`(input)	This parameter specifies the FogMode setting.

KM_FOGTABLE
KM_FOGVERTEX
KM_NOFOG
KM_FOGTABLE_2

Return values

KMSTATUS_SUCCESS	Success

kmChangeContextGouraud

Changes Gouraud.

Format

```
KMSTATUS KMAPI
kmChangeContextGouraud(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMBOOLEAN           bGouraud
);
```

Description

This function changes the rendering parameter Gouraud that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	X
VertexType04	◎	VertexType10	◎	VertexType16	X
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`bGouraud`(input)	This parameter specifies the Gouraud setting.

```
KM_TRUE    :   Gouraud is valid.
KM_FALSE   :   Gouraud is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextIgnoreTextureAlpha

Changes IgnoreTextureAlpha.

Format

```
KMSTATUS KMAPI
kmChangeContextIgnoreTextureAlpha(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bIgnoreTextureAlpha
);
```

Description

This function changes the rendering parameter `IgnoreTextureAlpha` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bIgnoreTextureAlpha`(input)

This parameter specifies the IgnoreTextureAlpha setting.

```
KM_TRUE    :   TextureAlpha is not valid.
KM_FALSE   :   TextureAlpha is valid.
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextIgnoreTexureAlpha

Changes IgnoreTexureAlpha.

Format

```
KMSTATUS KMAPI
kmChangeContextIgnoreTexureAlpha(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bIgnoreTexureAlpha
);
```

Description

This function changes `IgnoreTexureAlpha`.

The following is a list of supported `VertexType`:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bIgnoreTexureAlpha`(input)	This parameter specifies the IgnoreTexureAlpha setting.

```
KM_TRUE    :   TextureAlpha is not valid.
KM_FALSE   :   TextureAlpha is valid.
```

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_VERTEXTYPE`	This setting is invalid at the current `VertexType`.

kmChangeContextIntensityMode

Changes IntensityMode.

Format

```
KMSTATUS KMAPI
kmChangeContextIntensityMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMCOLORTYPE         nIntensityMode
);
```

Description

This function changes the rendering parameter `IntensityMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	◎	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nIntensityMode`(input)	This parameter specifies the IntensityMode setting.

`KM_INTENSITY`
`KM_INTENSITY_PREV_FACE_COL`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextListType

Changes ListType.

Format

```
KMSTATUS KMAPI
kmChangeContextListType(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMLISTTYPE          nListType
);
```

Description

This function changes the rendering parameter ListType that was registered in pGlobalParam from `pVertexBuffDesc` by kmSetVertexRenderState/kmSetStripHead.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	★
VertexType01	★	VertexType07	★	VertexType13	★
VertexType02	★	VertexType08	★	VertexType14	★
VertexType03	★	VertexType09	★	VertexType15	★
VertexType04	★	VertexType10	★	VertexType16	★
VertexType05	★	VertexType11	★	VertexType17	★

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

The ListType that can be changed is determined by `VertexType`.

`VertexType0*`16` Can be one of the following:

KM_OPAQUE_POLYGON
KM_TRANS_POLYGON
KM_PUNCHTHROUGH_POLYGON

`VertexType17` Can be one of the following:

KM_OPAQUE_MODIFIER
KM_TRANS_MODIFIER

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nListType`(input)	This parameter specifies the ListType setting.

```
KM_OPAQUE_POLYGON
KM_OPAQUE_MODIFIER
KM_TRANS_POLYGON
KM_TRANS_MODIFIER
KM_PUNCHTHROUGH_POLYGON
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextMipmapAdjust

Changes MipmapAdjust.

Format

```
KMSTATUS KMAPI
kmChangeContextMipmapAdjust(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMDWORD             dwMipmapAdjust
);
```

Description

This function changes the rendering parameter `MipmapAdjust` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`dwMipmapAdjust`(input)	This parameter specifies the MipmapAdjust setting.

```
KM_MIPMAP_D_ADJUST_0_25
KM_MIPMAP_D_ADJUST_0_50
KM_MIPMAP_D_ADJUST_0_75
KM_MIPMAP_D_ADJUST_1_00
KM_MIPMAP_D_ADJUST_1_25
KM_MIPMAP_D_ADJUST_1_50
KM_MIPMAP_D_ADJUST_1_75
KM_MIPMAP_D_ADJUST_2_00

KM_MIPMAP_D_ADJUST_2_25
KM_MIPMAP_D_ADJUST_2_50
KM_MIPMAP_D_ADJUST_2_75
KM_MIPMAP_D_ADJUST_3_00
KM_MIPMAP_D_ADJUST_3_25
KM_MIPMAP_D_ADJUST_3_50
KM_MIPMAP_D_ADJUST_3_75
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextModifierInstruction

Changes ModifierInstruction.

Format

```
KMSTATUS KMAPI
kmChangeContextModifierInstruction(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMDWORD             dwModifierInstruction
);
```

Description

This function changes the rendering parameter ModifierInstruction that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	X	VertexType13	X
VertexType02	X	VertexType08	X	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	X	VertexType16	X
VertexType05	X	VertexType11	X	VertexType17	◎

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`dwModifierInstruction`(input)	This parameter specifies the ModifierInstruction setting.

```
KM_MODIFIER_NORMAL_POLY
KM_MODIFIER_INCLUDE_FIRST_POLY
KM_MODIFIER_EXCLUDE_FIRST_POLY
KM_MODIFIER_INCLUDE_LAST_POLY
KM_MODIFIER_EXCLUDE_LAST_POLY
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextOffset

Changes Offset.

Format

```
KMSTATUS KMAPI
kmChangeContextOffset(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMBOOLEAN           bOffset
);
```

Description

This function changes the rendering parameter Offset that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	◎	VertexType12	◎
VertexType01	X	VertexType07	◎	VertexType13	◎
VertexType02	X	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	X	VertexType15	X
VertexType04	◎	VertexType10	X	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`bOffset`(input)	This parameter specifies the Offset setting.

```
KM_TRUE    :   Offset is valid.
KM_FALSE   :   Offset is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextPaletteBank

Changes PaletteBank.

Format

```
KMSTATUS KMAPI
kmChangeContextPaletteBank(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMDWORD             dwPaletteBank
);
```

Description

This function changes the rendering parameter `PaletteBank` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

In addition, because the PaletteBank information is initialized by executing `kmChangeContextTextureSurface`, if it is desired to change `TextureSurface` and `PaletteBank`, execute `kmChangeContextTextureSurface` -> `kmChangeContextPaletteBank`, in that order.

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`dwPaletteBank`(input)	This parameter specifies the PaletteBank setting.(0-63)

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextShadowMode

Changes ShadowMode.

Format

```
KMSTATUS KMAPI
kmChangeContextShadowMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMSHADOWMODE        nShadowMode
);
```

Description

This function changes the rendering parameter `ShadowMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	◎	VertexType06	◎	VertexType12	X
VertexType01	◎	VertexType07	◎	VertexType13	X
VertexType02	◎	VertexType08	◎	VertexType14	X
VertexType03	◎	VertexType09	X	VertexType15	◎
VertexType04	◎	VertexType10	X	VertexType16	◎
VertexType05	◎	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nShadowMode`(input)	This parameter specifies the ShadowMode setting.

`KM_NORMAL_POLYGON`	`normal polygon`
`KM_CHEAPSHADOW_POLYGON`	`simple shadow polygon`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextSpriteBaseColor

Changes SpriteBaseColor.

Format

```
KMSTATUS KMAPI
        kmChangeContextSpriteBaseColor(
                 IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
                 IN      KMPACKEDARGB        dwBaseColor
        );
```

Description

This function changes the rendering parameter Sprite BaseColor that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	X	VertexType13	X
VertexType02	X	VertexType08	X	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	◎
VertexType04	X	VertexType10	X	VertexType16	◎
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`dwBaseColor`(input)	This parameter specifies the sprite base color setting.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextSpriteOffsetColor

Changes SpriteOffsetColor.

Format

```
KMSTATUS KMAPI
kmChangeContextSpriteOffsetColor(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMPACKEDARGB        dwOffsetColor
);
```

Description

This function changes the rendering parameter Sprite `OffsetColor` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	X	VertexType12	X
VertexType01	X	VertexType07	X	VertexType13	X
VertexType02	X	VertexType08	X	VertexType14	X
VertexType03	X	VertexType09	X	VertexType15	X
VertexType04	X	VertexType10	X	VertexType16	◎
VertexType05	X	VertexType11	X	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`dwOffsetColor`(input)	This parameter specifies the sprite offset color setting.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextSRCSelect

Changes SRCSelect.

Format

```
KMSTATUS KMAPI
kmChangeContextSRCSelect(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bSRCSelect
);
```

Description

This function changes the rendering parameter `SRCSelect` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.

```
KM_IMAGE_PARAM1     parameter 1
KM_IMAGE_PARAM2     parameter 2
```

`bSRCSelect`(input)	This parameter specifies the SRCSelect setting.

```
KM_TRUE    :   SRCSelect is valid.
KM_FALSE   :   SRCSelect is not valid.
```

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextSuperSampleMode

Changes SuperSampleMode.

Format

```
KMSTATUS KMAPI
kmChangeContextSuperSampleMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bSuperSampleMode
);
```

Description

This function changes the rendering parameter `SuperSampleMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.

KM_IMAGE_PARAM1	parameter 1
KM_IMAGE_PARAM2	parameter 2

`bSuperSampleMode`(input)	This parameter specifies the `SuperSampleMode` setting.

```
KM_TRUE    :   SuperSampleMode is valid.
KM_FALSE   :   SuperSampleMode is not valid.
```

Return values

KMSTATUS_SUCCESS	Success

kmChangeContextTextureAddress

Changes TextureAddress.

Format

```
KMSTATUS KMAPI
kmChangeContextTextureAddress(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      PKMSURFACEDESC      pTextureSurfaceDesc
);
```

Description

This function changes the rendering parameter `TextureAddress` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

Because it does not make any changes other than in `TextureAddress`, this API can be used when the following members are not being changed.

```
pTextureSurfaceDesc->PixelFormat
pTextureSurfaceDesc->u0.USize
pTextureSurfaceDesc->u1.VSize
pTextureSurfaceDesc->fSurfaceFlags
```

If there are any changes in the above members, use kmChangeContextTextureSurface, not `kmChangeContextTextureAddress`.

The following is a list of supported VertexType:

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`pTextureSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextTextureShadingMode

Changes TextureShadingMode.

Format

```
KMSTATUS KMAPI
kmChangeContextTextureShadingMode(
        IN OUT  PKMVERTEXBUFFDESC       pVertexBuffDesc,
        IN      KMINT32                 nParam,
        IN      KMTEXTURESHADINGMODE    nTextureShadingMode
);
```

Description

This function changes the rendering parameter `TextureShadingMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.
`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`
`nTextureShadingMode`(input)	This parameter specifies the `TextureShadingMode` setting.

`KM_DECAL`
`KM_MODULATE`
`KM_DECAL_ALPHA`
`KM_MODULATE_ALPHA`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextTextureSurface

Changes TextureSurface.

Format

```
KMSTATUS KMAPI
kmChangeContextTextureSurface(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      PKMSURFACEDESC      pTextureSurfaceDesc
);
```

Description

This function changes the rendering parameter `TextureSurface` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState/kmSetStripHead`.

The following is a list of supported `VertexType`:

In addition, because the `PaletteBank` information is initialized as a result of executing this API, set the `PaletteBank` information through `kmChangeContextPaletteBank` if necessary.

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	X	VertexType06	★	VertexType12	◎
VertexType01	X	VertexType07	★	VertexType13	◎
VertexType02	X	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	X	VertexType15	X
VertexType04	★	VertexType10	X	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`pTextureSurfaceDesc`(input)	This parameter is a pointer to `KMSURFACEDESC`.

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextUseAlpha

Changes UseAlpha.

Format

```
KMSTATUS KMAPI
kmChangeContextUseAlpha(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMINT32             nParam,
        IN      KMBOOLEAN           bUseAlpha
);
```

Description

This function changes the rendering parameter `UseAlpha` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	★	VertexType06	★	VertexType12	◎
VertexType01	★	VertexType07	★	VertexType13	◎
VertexType02	★	VertexType08	★	VertexType14	◎
VertexType03	★	VertexType09	◎	VertexType15	★
VertexType04	★	VertexType10	◎	VertexType16	★
VertexType05	★	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nParam`(input)	This parameter specifies the parameter for update.

`KM_IMAGE_PARAM1`	`parameter 1`
`KM_IMAGE_PARAM2`	`parameter 2`

`bUseAlpha`(input) This parameter specifies the UseAlpha setting:

`KM_TRUE`	`:ALPHA is valid.`
`KM_FALSE`	`:ALPHA is not valid.`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextUserClipMode

Changes UserClipMode.

Format

```
KMSTATUS KMAPI
kmChangeContextUserClipMode(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMUSERCLIPMODE      nUserClipMode
);
```

Description

This function changes the rendering parameter `UserClipMode` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported `VertexType`:This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	◎

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.
★: Only `KM_IMAGE_PARAM1` can be used.
X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`nUserClipMode`(input)	This parameter specifies the `UserClipMode` setting.

`KM_USERCLIP_DISABLE`
`KM_USERCLIP_INSIDE`
`KM_USERCLIP_OUTSIDE`

Return values

`KMSTATUS_SUCCESS`	Success

kmChangeContextZWriteDisable

Changes ZWriteDisable.

Format

```
KMSTATUS KMAPI
kmChangeContextZWriteDisable(
        IN OUT  PKMVERTEXBUFFDESC   pVertexBuffDesc,
        IN      KMBOOLEAN           bZWriteDisable
);
```

Description

This function changes the rendering parameter `ZWriteDisable` that was registered in `pGlobalParam` from `pVertexBuffDesc` by `kmSetVertexRenderState`/`kmSetStripHead`.

The following is a list of supported VertexType:

Note: This function changes some of `pGlobalParam` from `pVertexBuffDesc` before starting a strip in response to `kmStartVertexStrip`. Operation is not guaranteed if the rendering parameters are not registered from `pVertexBuffDesc` to `pGlobalParam` within the same pass beforehand.

VertexType00	◎	VertexType06	◎	VertexType12	◎
VertexType01	◎	VertexType07	◎	VertexType13	◎
VertexType02	◎	VertexType08	◎	VertexType14	◎
VertexType03	◎	VertexType09	◎	VertexType15	◎
VertexType04	◎	VertexType10	◎	VertexType16	◎
VertexType05	◎	VertexType11	◎	VertexType17	X

Table Key:

◎: `KM_IMAGE_PARAM1` / `KM_IMAGE_PARAM2` can be used.

★: Only `KM_IMAGE_PARAM1` can be used.

X: None can be used.

Parameters

`pVertexBuffDesc`(input/output)	This parameter is a pointer to `KMVERTEXBUFFDESC`.
`bZWriteDisable`(input)	This parameter specifies the `ZWriteDisable` setting:

```
KM_TRUE    :   ZWriteDisable is not valid.
KM_FALSE   :   ZWriteDisable is valid.
```

Return values

`KMSTATUS_SUCCESS`	Success

10. Functions for Controlling Rendering

kmGetRenderStatus

Gets status of rendering that has been executed.

Format

```
KMSTATUS
kmGetRenderStatus(
                IN   KMINT32    nRenderID
        );
```

Description

This function checks the current status of a rendering operation that was executed.

Parameters

`nRenderID`(input)

Input the return value when `kmRender` or `kmRenderTexture` has been executed.

Note: If NO_FLIP was specified for `kmRenderTexture` or `kmRender`, `KMSTATUS_UNDER_DISPLAY` will not be a return value for this function.

Return values

`KMSTATUS_UNDER_DMA`	The rendering operation with the specified ID is currently transferring vertex data by means of DMA transfer.
`KMSTATUS_FINISH_DMA`	The rendering operation with the specified ID has currently finished DMA transfer.
`KMSTATUS_UNDER_RENDER`	The rendering operation with the specified ID is currently engaged in the rendering process.
`KMSTATUS_FINISH_RENDER`	The rendering operation with the specified ID has currently finished the rendering process.
`KMSTATUS_UNDER_DISPLAY`	The rendering operation with the specified ID is currently being displayed.
`KMSTATUS_FINISH_ALL_SEQUENCE`	The rendering operation with the specified ID has currently finished all processing.

kmRender

Starts rendering.

Format

```
KMINT32
kmRender(
                IN   KMDWORD    dwFlipMode
);
```

Description

This function notifies Kamui when registration of one page of vertex data in relation to the tile accelerator is completed. When the vertex data rendering is completed, it starts to apply rendering to the back buffer. Issue `kmEndScene` (the end of the scene) after this function.

Parameters

`dwFlipMode`(input)	This parameter specifies the flip mode, as follows:

```
KM_RENDER_FLIP
KM_RENDER_NOFLIP
```

Return values

0 < (return)	Success
0 > (return)	Failure

kmRenderTexture

Begins rendering for a texture mamory.

Format

```
KMINT32
kmRenderTexture(
                OUT  PKMSURFACEDESC pTextureSurface
                IN   KMDWORD        dwDitherMode
);
```

Description

This function notifies the tile accelerator that all vertex data of a single scene has been written. The renderer begins rendering for a texture surface specified after vertex data expansion is completed. Also, the texture for drawing must be in the Rectangle/Stride format.

Parameters

`pTextureSurface`(output)	Texture of which rendering result is stored
`dwDitherMode`(input)	This parameter specifies the rendering flag, as follows:

```
KM_RENDER_DITHER
KM_RENDER_NODITHER
```

Return values

`0 <` (return)	Success
`0 >` (return)	Failure

kmQueryFinishLastTextureDMA Checks for previous texture load DMA transfer end.

Format

```
KMSTATUS KMAPI
kmQueryFinishLastTextureDMA(KMVOID);
```

Description

This function checks whether a DMA transfer started by the previous texture load function has ended.

The function is valid only if kmSetSystemConfiguration sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag. Otherwise, `KMSTATUS_SUCCESS` is returned.

Parameters

None

Return values

`KMSTATUS_SUCCESS`	Previous texture load DMA transfer ended.
`KMSTATUS_NOT_FINISH_DMA`	Previous texture load DMA transfer not ended.

11. Functions for Controlling Texture

kmCreateCombinedTextureSurface

Secures a VQ texture surface.

```
KMSTATUS KMAPI
kmCreateCombinedTextureSurface(
OUT PKMSURFACEDESC pSurfaceDesc1,
OUT PKMSURFACEDESC pSurfaceDesc2,
IN KMINT32 nWidth,
IN KMINT32 nHeight,
IN KMTEXTURETYPE nTextureType
)
```

Description

This function secures a texture surface in texture memory. It secures two texture surfaces of the same size and format. Use the kmFreeTexture function to release the area secured by this function.

This API is kept here for compatibility with ARC1. Therefore, try as much as possible not to use this API.

Like `kmCreateTextureSurface`, this API can allocate texture surfaces in all formats. KAMUI aligns the first texture surface address and size with a 32-byte boundary.

Caution: This function must be executed after `kmSetSystemConfiguration` is called.

Parameters

`pSurfaceDesc1`(output)	This parameter is a pointer (No. 1) to `KMSURFACEDESC`-type structure. Surface information is returned to the structure using the pointer. It becomes undefined if, for `KMSTATUS`, `KMSTATUS_NOT_ENOUGH_MEMORY` is returned.
`pSurfaceDesc2`(output)	This parameter is a pointer (No. 2) to `KMSURFACEDESC`-type structure. Surface information is returned to the structure using the pointer. It becomes undefined if, for `KMSTATUS`, `KMSTATUS_NOT_ENOUGH_MEMORY` is returned.
`nWidth, nHeight`(input)	These parameters specify the horizontal and vertical texture sizes. If MIPMAP is used, the top-level texture size must be specified. For the square texture, texture size can be between 8x8 and 1024x1024, but the value specified for nWidth or nHeight must be 8, 16, 32, 64, 128, 256, 512, or 1,024.
`nTextureType`(input)	This parameter specifies a texture format. See `KmCreateTextureSurface`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified
`KMSTATUS_NOT_ENOUGH_MEMORY`	Insufficient memory

kmCreateContiguousTextureSurface Secures two or more texture surfaces simultaneously.

```
KMSTATUS KMAPI
kmCreateContiguousTextureSurface(
OUT PKMSURFACEDESC pSurfaceDesc,
IN KMINT32 nTexture,
IN KMINT32 nWidth,
IN KMINT32 nHeight,
IN KMTEXTURETYPE nTextureType
)
```

Description

This function simultaneously secures two or more texture surfaces at contiguous addresses in the frame buffer. Use the `kmFreeTexture` function to release the area secured by this function.

It is used to read two or more textures of YUV422 type in succession, by using the YUV converter of tiling accelerator of the PowerVR (see kmLoadYUVTexture).

This API, however, can also allocate texture surfaces in all formats, excluding "small VQ compression format" and "small VQ compression format with a mipmap."

Caution: This function must be executed after `kmSetSystemConfiguration` is called.

Parameters

`ppSurfaceDesc`(output)	This parameter is a pointer to a `KMSURFACEDESC` structure. Texture surface information is returned to the structure. It becomes undefined if, for `KMSTATUS`, `KMSTATUS_NOT_ENOUGH_MEMORY` is returned.
`nTexture`(input)	This parameter specifies the number of texture surfaces to be secured in succession.
`nWidth`, `nHeight`(input)	These parameters 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 1,024.
`nTextureType`(input)	This parameter specifies a texture format. See `KmCreateTextureSurface`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified
`KMSTATUS_NOT_ENOUGH_MEMORY`	Insufficient memory

kmCreateFixedTextureArea

Secures the fixed texture area.

```
KMSTATUS KMAPI
kmCreateFixedTextureArea(
OUT PKMSURFACEDESC pSurfaceDesc,
IN KMINT32 nWidth,
IN KMINT32 nHeight,
IN KMTEXTURETYPE nTextureType
)
```

Description

This function secures the fixed texture area which will not be lost even when the `kmSetSystemConfiguration` function is issued. The fixed texture area can be used to hold the font texture, etc, for use through out the game. The area secured here can only be released by using the `kmFreeFixedTextureArea` function. In fact, when this function is used, all fixed texture areas are released at the same time. It is not possible to select and release an individual fixed texture area.

Actually, the fixed texture area is secured when the `kmSetSystemConfiguration` function is issued. After this function is called, the `kmSetSystemConfiguration` function must also be called.

Also, data write to the fixed texture area must be performed after the `kmSetSystemConfiguration` function.

In other words, the order in calling the functions are as follows:

(1) Run the `kmCreateFixedTextureArea` function to secure the fixed texture area. (It can be run multiple times.)

(2) Run the `kmSetSystemConfiguration` to actually secure the fixed texture areas.

(3) Run the `kmLoadTexture` function, etc. to read texture data into the fixed texture areas.

Before issuing the `kmSetSystemConfiguration` function, if the `kmFreeFixedTextureArea` function has already been issued, the `kmCreateFixedTextureArea` function will become ignored if it has been issued. In this case, the fixed texture area is completely not secured.

Also, the area secured here does not become the target for garbage collection, and the total capacity of the fixed texture area must not exceed 4MB.

Parameters

`pSurfaceDesc`(output)	This parameter is a pointer to the `KMSURFACEDESC`-type structure. Kamui returns the surface information to this structure. If `KMSTATUS_NOT_ENOUGH_MEMORY` is returned to `KMSTATUS`, then it is not definite.
`nWidth`, `nHeight`(input)	This parameter specifies the horizontal and vertical sizes of the texture. In the case that MIPMAP is used, it specifies the top-level texture size. While square texture with size ranging from 8x8 to 1024x1024 can be used, nWidth and nHeight are limited to 8, 16, 32, 64, 128, 256, 512, or 1024.
`nTextureType`(input)	This parameter specifies a texture format. See `KmCreateTextureSurface`.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_NOT_ENOUGH_MEMORY`	The total capacity of the fixed texture area has exceeded 4MB.

kmCreateTextureSurface

Secures a texture surface.

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

Description

This function secures a texture surface in texture memory. This API can allocate texture surfaces in all formats. Use the `kmFreeTexture` function to release the area secured by this function.

Whenever possible, KAMUI aligns the texture address with a 2 KB boundary so as to quicken memory access. Therefore, no texture surface may be created even if the total of the free frame buffer capacities is larger than the capacity to be allocated to the texture. In this case, it is necessary to perform garbage collection, using `kmGarbageCollectTexture`. KAMUI aligns the first texture surface address and size with a 32-byte boundary.

Caution: This function must be executed after `kmSetSystemConfiguration` is called.
Efficient use of texture memory
Secure/release as many texture surfaces as possible at the same time.
Call the creation of a frame buffer area or native data area before the creation of a texture surface, and avoid releasing and re-creating these until AP ends.

Parameters

`pSurfaceDesc`(output)	This parameter is a pointer to the `KMSURFACEDESC`-type structure. Surface information is returned to the structure using the pointer. It becomes undefined if `KMSTATUS` is responded with `KMSTATUS_NOT_ENOUGH_MEMORY`.
`nWidth`, `nHeight`(input)	These parameters specify the horizontal and vertical texture sizes. If MIPMAP is used, the top-level texture size must be specified. The value specified for nWidth or nHeight must be 8, 16, 32, 64, 128, 256, 512, or 1,024.
`nTextureType`(input)	This parameter 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	Meaning
`KM_TEXTURE_TWIDDLED`	Twiddled format
`KM_TEXTURE_TWIDDLED_RECTANGLE`	Rectangular Twiddled format
`KM_TEXTURE_TWIDDLED_MM`	Twiddled format with a mipmap
`KM_TEXTURE_VQ`	VQ compression format
`KM_TEXTURE_VQ_MM`	VQ compression format with a mipmap
`KM_TEXTURE_SMALLVQ`	Small VQ compression format
`KM_TEXTURE_SMALLVQ_MM`	Small VQ compression format with a mipmap
`KM_TEXTURE_PALETTIZE4`	4-bpp palette format
`KM_TEXTURE_PALETTIZE4_MM`	4-bpp palette format with a mipmap
`KM_TEXTURE_PALETTIZE8`	8-bpp palette format
`KM_TEXTURE_PALETTIZE8_MM`	8-bpp palette format with a mipmap
`KM_TEXTURE_RECTANGLE`	Rectangle
`KM_TEXTURE_STRIDE`	Rectangle (stride specification)

Pixel format codes	Meaning
`KM_TEXTURE_1555`	ARGB-1555 format
`KM_TEXTURE_565`	RGB-565 format
`KM_TEXTURE_4444`	ARGB-4444 format
`KM_TEXTURE_YUV422`	YUV-422 format
`KM_TEXTURE_BUMP`	Bump map

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`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified
`KMSTATUS_NOT_ENOUGH_MEMORY`	Insufficient memory

kmFreeFixedTextureArea

Releases a fixed texture area.

```
KMSTATUS KMAPI
kmFreeFixedTextureArea( KMVOID );
```

Description

This function releases all fixed texture surface allocated by `kmCreateFixedTextureArea`.

When this function is used, all fixed texture areas are released at the same time. It is not possible to select and release an individual fixed texture area.

Actually, the fixed texture area is released when the `kmSetSystemConfiguration` function is issued. After this function is called, the kmSetSystemConfiguration function must also be called.

This function cannot release the area secured by the `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/`kmCreateContiguousTextureSurface`.

Parameters

None

Return values

`KMSTATUS_SUCCESS`	Success

kmFreeTexture

Releases a texture surface.

```
KMSTATUS KMAPI
kmFreeTexture(
IN PKMSURFACEDESC pSurfaceDesc
)
```

Description

This function releases a texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/`kmCreateContiguousTextureSurface`.

This function cannot release the area secured by the `kmCreateFixedTextureArea` function.

Parameters

`pSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`. This parameter is a pointer to `KMSURFACEDESC`-type structure.

Return values

`KMSTATUS_SUCCESS`	Released successfully
`KMSTATUS_INVALID_ADDRESS`	Specified area (Surface) is not allocated.

kmGarbageCollectTexture

Performs garbage collection of frame buffer memory.

```
KMSTATUS KMAPI
kmGarbageCollectTexture( KMVOID );
```

Description

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 function 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 after this function is used.

Note that the frame buffer area for display and native data buffer are not subject to garbage collection.

Parameters

None

Return values

`KMSTATUS_SUCCESS`	Garbage collection successful
`KMSTATUS_NOT_ENOUGH_MEMORY`	Insufficient memory

kmGetCurrentTextureStatus

Reads out information about the frame buffer memory management.

```
KMSTATUS KMAPI
kmGetCurrentTextureStatus(
OUT PKMFBSTATUS pFBStatus
)
```

Description

This function reads out information about the frame buffer memory management. When the user prepares the following structure and passes the pointer to it, the function then sets the content of each member.

FB management information

typedef struct `tagKMFBSTATUS`

```
{
    /* Size Of KMFBSTATUS */
KMDWORD dwSize; /* The size of this structure                                */
KMDWORD dwNumberOfFreeBlocks; /* Number of free texture management structures currently being used
*/
KMDWORD dwNumberOfUsedBlocks; /* Number of used texture management structures currently being used
*/
KMDWORD dwNumberOfMaxBlocks; /* Number of texture management structures that can be used
KMDWORD dwNumberOfVQBlocks; /* Number of VQ texture management structures currently being used */
KMDWORD dwNumberOfMaxVQBlocks; /* Number of VQ texture management structures that can be used */
KMDWORD dwSizeOfFixedArea; /* The size of the fixed area currently being secured (bytes)*/
KMDWORD dwSizeOfFrameBuffer; /* Size of the frame buffer memory (byte) */
} KMFBSTATUS, PKMFBSTATUS;
```

Parameters

`pFBStatus`(output)	This parameter is a pointer to the FB management information structure.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetFreeTextureMem

Obtains the available texture memory space.

```
KMSTATUS KMAPI
kmGetFreeTextureMem(
OUT PKMUINT32 pSizeOfTexture,
OUT PKMUINT32 pMaxBlockSizeOfTexture
)
```

Description

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`, `kmCreateCombinedTextureSurface`, or `kmCreateContiguousTextureSurface`).

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

Parameters

`pSizeOfTexture`(output)	This parameter is a pointer to the `KMDWORD` area to which the available texture memory space is returned.
`pMaxBlockSizeOfTexture`(output)	This parameter is a pointer to the `KMDWORD` area to which the largest vacant block in the texture memory is to be returned.

Return values

`KMSTATUS_SUCCESS`	Success

kmGetTexture

Reads the texture in texture memory.

```
KMSTATUS KMAPI
kmGetTexture(
OUT PKMDWORD pTexture,
IN PKMSURFACEDESC pSurfaceDesc
)
```

Description

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.

If the start address of texture data in system memory is on a 32-byte boundary, and its size is a multiple of 32 bytes, the DMA mode is used to transfer the texture data to texture memory, so that high-speed transfer becomes possible.

Parameters

`pTexture`(output)	This parameter is a pointer indicating the area in main memory where the texture is to be saved. Secure a multiple of 32 bytes, aligned with a 32-byte boundary. (Read destination)
`pSurfaceDesc`(input)	Texture surface to which the texture is saved. This parameter is a pointer to `KMSURFACEDESC`-type structure. (Read source)

Return values

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

kmLoadRectangleTexturePart

Writes the rectangular area (partial) of texture data in the main memory.

```
KMSTATUS KMAPI
kmLoadRectangleTexturePart(
IN PKMSURFACEDESC pSurfaceDesc,
IN PKMDWORD pTexture,
IN KMUINT32 width,
IN KMUINT32 height,
IN KMUINT32 dst_X,
IN KMUINT32 dst_Y,
IN KMUINT32 src_X,
IN KMUINT32 src_Y,
IN KMUINT32 src_U
)
```

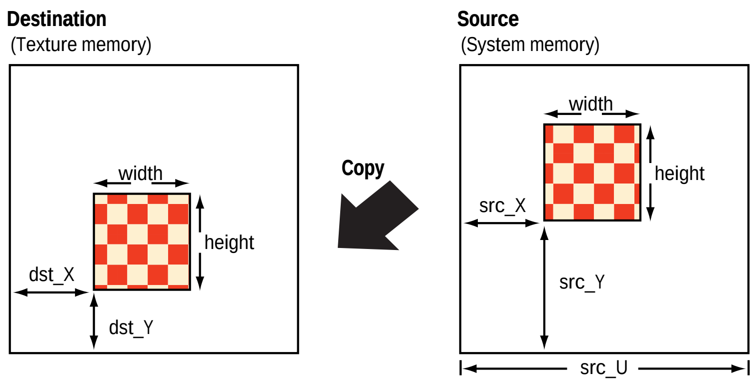
Description

This function writes the rectangular area (partial) of texture data in the main memory specified by `pTexture`, to the rectangular texture area in the Rectangle/Stride format in the frame buffer memory secured by `kmCreateTextureSurface`/`kmCreateCombinedTextureSurface`/`kmCreateContiguousTextureSurface`/`kmCreateFixedTextureArea`. This allows part of the texture in the Rectangle/Stride format to be updated.

The rectangular area of the transfer destination is (dst_X, dst_Y)-(dst_X + width, dst_Y + height).

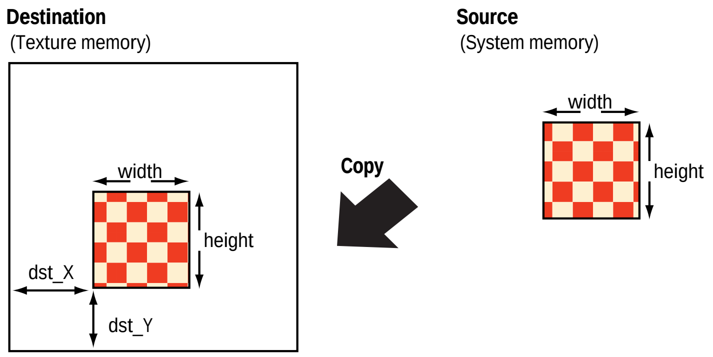
The rectangular area of the transfer source is (src_X, src_Y)-(src_X + width, src_Y + height).

(See the figure below.)



If src_U is set to zero, the size of the texture of the transfer source is regarded as same as the width and height of the rectangular area. In this case, src_X and src_Y will be ignored.

(See the figure below.)



If the address of the left texel of the rectangular area data in the system memory has a 32-byte alignment and its size is multiple of 32 bytes, the data transfer will employ DMA. High-speed transfer will become possible.

If the DMA mode is used to transfer texture data, it is possible to select whether to wait until the transfer ends. If kmSetSystemConfiguration sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag, the function ends without waiting for the completion of a DMA transfer. In this case, the `kmQueryFinishLastTextureDMA` function can be used to check for the end of DMA transfer.

If the CPU directly rewrites texture data into main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

This function supports only Rectangle/Stride format texture. If `pSurfaceDesc` of any other format is specified, `KMSTATUS_INVALID_TEXTURE_TYPE` is returned.

The rectangular area must fall completely within the texture area of both transfer destination and transfer source. If this is not the case, texture data including the one in the transfer destination may become corrupted. Note that Kamui does not check for this problem.

Parameters

`pSurfaceDesc`(input)	Texture surface in the Rectangle/Stride format secured by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure.
`pTexture`(input)	This parameter is a pointer to the pixel data portion of the texture in main memory.
`width`, `height`(input)	These parameters specify the width and height of the rectangular area to transfer, in number of texels.
`dst_X`, `dst_Y`(input)	The X- and Y-coordinates of the rectangular area of the texture in the transfer destination. They are specified using relative number of texels from the lower left (leading) texel of the texture in the transfer destination.
`src_X`, `src_Y`(input)	The X- and Y-coordinates of the rectangular area of the texture in the transfer source. They are specified using relative number of texels from the lower left (leading) texel of the texture in the transfer source.
`src_U`(input)	Specifies the width of the texture in the transfer source, in number of texels. If it is zero, the size of the texture of the transfer source is regarded as same as the width and height of the rectangular area. In this case, src_X and src_Y will be ignored (regarded as zero).

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture specified.

kmLoadTexture

Loads texture data.

```
KMSTATUS KMAPI
kmLoadTexture(
IN PKMSURFACEDESC pSurfaceDesc,
IN PKMDWORD pTexture
)
```

Description

This function loads the texture on main memory specified by `pTexture` into the texture memory area allocated by `kmCreateTextureSurface`/`kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. 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.

If the start address of texture data in system memory is aligned with a 32-byte boundary, and its size is a multiple of 32 bytes, the DMA mode is used to transfer the texture data to texture memory, so that high-speed transfer becomes possible.

If the DMA mode is used to transfer texture data, it is possible to select whether to wait until the transfer ends. If `kmSetSystemConfiguration` sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag, the function ends without waiting for the completion of a DMA transfer. In this case, the `kmQueryFinishLastTextureDMA` function can be used to check for the end of DMA transfer.

If the CPU directly rewrites texture data in main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

Parameters

`pSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure.
`pTexture`(input)	This parameter is a 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 32-byte boundary (16 bytes of the header portion of KAMUI are skipped). If this address is not on a 32-byte boundary, DMA transfer cannot be used, resulting in the processing being slow.

Return values

`KMSTATUS_SUCCESS`	Read successfully.
`KMSTATUS_INVALID_ADDRESS`	The specified area (Surface) is not allocated.
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified.

kmLoadTextureBlock

Loads texture data blocks.

```
KMSTATUS KMAPI
kmLoadTextureBlock(
IN PKMSURFACEDESC pSurfaceDesc,
IN PKMDWORD pTexture,
IN KMUINT32 nBlockNum,
IN KMUINT32 nBlockSize
)
```

Description

This function loads texture blocks from a main memory area specified by `pTexture` into a texture memory area allocated using `kmCreateTextureSurface`/`kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/`kmCreateFixedTextureArea`.

Texture data is divided into blocks before it is loaded. It makes it possible to load large texture data without allocating a large work area in main memory.

To load texture data by dividing it in BUFFSIZE*32 byte units, for example, code the following:

```
i = 0;
Load the first BUFFSIZE*32 byte block into pTexture;
while(KMSTATUS_SUCCESS == kmLoadTextureBlock(
        &TexSurfaceDesc,
        pTexture,
        i++,
        BUFFSIZE
        )) {
                Load the next BUFFSIZE*32 byte block into pTexture;
        }
```

Even if the size of the whole texture data is not an integer multiple of the block size, loading is performed normally. It is impossible to change the `BUFFSIZE` value in a loop in which one set of texture blocks is being loaded. If the value is changed, the texture display becomes illegal.

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

If the start address of texture data in system memory is on a 32-byte boundary, and its size is a multiple of 32 bytes, the DMA mode is used to transfer the texture data to texture memory, so that high-speed transfer becomes possible.

If the DMA mode is used to transfer texture data, it is possible to select whether to wait until the transfer ends. If kmSetSystemConfiguration sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag, the function ends without waiting for the completion of DMA transfer. In this case, the `kmQueryFinishLastTextureDMA` function can be used to check for the end of DMA transfer.

If the CPU directly rewrites texture data into main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

This function does not support texture data of Small VQ format. If `pSurfaceDesc` of Small VQ format is specified, `KMSTATUS_INVALID_TEXTURE_TYPE` is returned.

Parameters

`pSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure.
`pTexture`(input)	This parameter is a 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 32-byte boundary (16 bytes of the header portion of KAMUI are skipped).
`nBlockNum`(input)	Specify a texture block number from 0 to n (n varies with the format and size).
`nBlockSize`(input)	Specify the size of a texture block in 32-byte units, that is, an actual block size (in bytes) divided by 32. Even if the size of the entire texture block is not an integer multiple of the block size, loading is performed normally.

Return values

`KMSTATUS_SUCCESS`	Read successfully.
`KMSTATUS_INVALID_BLOCKNUMBER`	Illegal block number.
`KMSTATUS_INVALID_ADDRESS`	Specified area (Surface) not allocated.
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified.

kmLoadTexturePart

Loads part of texture data.

```
KMSTATUS KMAPI
kmLoadTexturePart(
IN PKMSURFACEDESC pSurfaceDesc,
IN PKMDWORD pTexture,
IN KMUINT32 nOffset,
IN KMUINT32 nSize
)
```

Description

This function loads texture portions from a main memory area specified by `pTexture` into a texture memory area allocated using `kmCreateTextureSurface`/`kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/`kmCreateFixedTextureArea`.

Texture data is divided into portions before it is loaded. This makes it possible to load a large amount of texture data without allocating a large work area in main memory.

Unlike `kmLoadTextureBlock`, `kmLoadTexturePart` can load one texture data item by dividing it into portions of different sizes. The user is responsible for managing the size (offset from the beginning of the texture data) of each texture portion that has already been loaded.

Example

```
nOffset = 0;
nSize   = ***;
Load the first nSize byte portion of texture data into an area specified by pTexture;
while(KMSTATUS_SUCCESS == kmLoadTexturePart(...)) {
        nOffset = nOffset + nSize;
        nSize   = ????;
Load the next nSize byte portion into an area specified by pTexture;
}
```

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

If the start address of texture data in system memory is on a 32-byte boundary, and its size is a multiple of 32 bytes, the DMA mode is used to transfer the texture data to texture memory, so that high-speed transfer becomes possible.

If the DMA mode is used to transfer texture data, it is possible to select whether to wait until the transfer ends. If `kmSetSystemConfiguration` sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag, the function ends without waiting for the completion of a DMA transfer. In this case, the `kmQueryFinishLastTextureDMA` function can be used to check for the end of DMA transfer.

If the CPU directly rewrites texture data into main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

This function does not support texture data of Small VQ format. If `pSurfaceDesc` of Small VQ format is specified, `KMSTATUS_INVALID_TEXTURE_TYPE` is returned.

Parameters

`pSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure.
`pTexture`(input)	This parameter is a 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 32-byte boundary (16 bytes of the header portion of KAMUI are skipped).
`nOffset`(input)	Specify the size of the texture data portion that has already been loaded (the offset from the beginning of the entire texture data) in byte units. This size must be an integer multiple of 4, because it is used to obtain the address of the transfer destination texture area in frame buffer memory.
`nSize`(input)	Specify the size of the texture data portion to be loaded, in byte units. This size must be an integer multiple of 4. If nSize is greater than the size of the remaining texture portion (= texture size - nOffset), texture data loading is completed by loading only the rest of the texture data.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	`nOffset` greater than the texture size.
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified.

kmLoadVQCodebook

Re-reads the code book portion of VQ texture.

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

Description

This function reads only the code book portion of the VQ or small VQ texture in main memory as specified by `pTexture` to the VQ or small VQ texture surface specified by `pSurfaceDesc`.

It is used to rewrite only the code book of the VQ or small VQ texture already loaded and use the color palette effect.

If the start address of texture data in system memory is on a 32-byte boundary, and its size is a multiple of 32 bytes, the DMA mode is used to transfer the texture data to texture memory, so that high-speed transfer becomes possible.

If the DMA mode is used to transfer texture data, it is possible to select whether to wait until the transfer ends. If `kmSetSystemConfiguration` sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag, the function ends without waiting for the completion of DMA transfer. In this case, the `kmQueryFinishLastTextureDMA` function can be used to check for the end of DMA transfer.

If the CPU directly rewrites texture data into main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

Parameters

`pSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure. The category of this surface must be one of the following types: `KM_TEXTURE_VQ` `KM_TEXTURE_VQ_MM` `KM_TEXTURE_SMALLVQ` `KM_TEXTURE_SMALLVQ_MM`
`pTexture`(input)	This parameter is a pointer indicating a texture (code book) in main memory. Specify an address aligned with a 32-byte boundary. This does not have to be in the complete VQ or small VQ texture format, but a code book (its size in bytes is indicated below) must be included in the beginning.

The following table lists the relationships between the texture size and code book size.

Texture Type/Size	Codebook Size (byte)
VQ / VQ mipmap	0x800
16x16 small VQ	0x80
16x16 small VQ mipmap	0x80
32x32 small VQ	0x100
32x32 small VQ mipmap	0x200
64x64 small VQ	0x400

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture surface specified

kmLoadYUVTexture

Reads the YUV-format texture data.

```
KMSTATUS KMAPI
kmLoadYUVTexture(
IN PKMSURFACEDESC pSurfaceDesc,
IN PKMDWORD pTexture,
IN KMUINT32 nWidth,
IN KMUINT32 nHeight,
IN KMUINT32 nFormat,
IN KMBOOLEAN bWaitEndOfDMA
)
```

Description

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`/ `kmCreateFixedTextureArea`.

In doing so, the YUV-data converter built into tiling accelerator of the CLX1/2 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 (when addressing mode of nFormat is `KM_TEXTURE_YUV_MULTI`), 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 function. 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` function.

To read in one set of YUV-data (when the addressing mode of nFormat is `KM_TEXTURE_YUV_SINGLE`), nHeight must be 16, 32, 64, 128, 256, 512, or 1024. Also, nWidth must be multiples of 32 and between 32 and 992. However, if the texture surface to read in is `KM_TEXTURE_RECTANGLE`, nWidth must also be 16, 32, 64, 128, 256, 512, or 1024.

If the CPU directly rewrites texture data into main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

Parameters

`ppSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure.
`pTexture`(input)	This parameter is a pointer indicating YUV420-data/YUV422-data in main memory. Specify an address aligned with a 32-byte boundary. If the address is not on a 32-byte boundary, the YUV converter cannot operate because of hardware constraints. In this case, `KMSTATUS_INVALID_ADDRESS` is returned.
`nWidth`, `nHeight`(input)	When the addressing mode of nFormat is `KM_TEXTURE_YUV_MULTI`: This parameter specifies the amount of texture to read in continuously, in number of micro blocks. The valid value of nWidth and nheight can be from 1 to 64. One micro block equals 16 x 16 texels. When the addressing mode of `nFormat` is `KM_TEXTURE_YUV_SINGLE`: This parameter specifies the size of texture to read in texel. -If output is in the Rectangle format: Both `nWidth` and `nHeight` must be 16, 32, 64, 128, 256, 512, or 1024. -If output is in the Stride format: `nHeight` must be 16, 32, 64, 128, 256, 512, or 1024. `nWidth` must be multiples of 32 and between 32 and 992.
`nFormat`(input)	This parameter specifies the format of the data to read in. One from each of the following categories can be selected and combined logically.

Color mode

Color mode	Description
`KM_TEXTURE_YUV420`	Indicates that the input data is YUV420-data.
`KM_TEXTURE_YUV422`	Indicates that the input data is YUV422-data.

Addressing Mode

Addressing Mode	Description
`KM_TEXTURE_YUV_MULTI`	The input data consists of multiple 16x16 YUV-data.
`KM_TEXTURE_YUV_SINGLE`	The input data consists of one nWidth x nHeight YUV-data.

`bWaitEndOfDMA`(input)	If `TRUE` is specified, the function waits until the DMA transfer of data to the YUV converter has been completed. This API does not end until DMA transfer ends. If `FALSE` is specified, the function does not wait until DMA transfer ends. To detect the end of DMA transfer, use the `kmSetEndOfYUVCallback` function.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture specified.
`KMSTATUS_INVALID_ADDRESS`	`pTexture` not on a 32-byte boundary.

kmReLoadMipmap

Overwrites the mipmap texture.

```
KMSTATUS KMAPI
kmReLoadMipmap(
IN PKMSURFACEDESC pSurfaceDesc,
IN PKMVOID pTexture
IN KMINT32 nMipmapCount
)
```

Description

This function overwrites the mipmap texture on main memory specified by `pTexture` and loads it into the texture memory area allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/`kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. The type of the texture (surface) that can be specified is one of the following types:

```
KM_TEXTURE_TWIDDLED_MM
KM_TEXTURE_VQ_MM
KM_TEXTURE_PALETTIZE4_MM
KM_TEXTURE_PALETTIZE8_MM
KM_TEXTURE_SMALLVQ_MM
```

The format and size of the texture to be read are identified by the surface descriptor specified by `pSurfaceDesc`.

If the start address of texture data in system memory is on a 32-byte boundary, and its size is a multiple of 32 bytes, the DMA mode is used to transfer the texture data to texture memory, so that high-speed transfer becomes possible.

If the DMA mode is used to transfer texture data, it is possible to select whether to wait until the transfer ends. If `kmSetSystemConfiguration` sets the `KM_CONFIGFLAG_NOWAIT_FINISH_TEXTUREDMA` flag, the function ends without waiting for the completion of DMA transfer. In this case, the `kmQueryFinishLastTextureDMA` function can be used to check for the end of DMA transfer.

If the CPU directly rewrites texture data into main memory before it is loaded, it is necessary to purge the cache before executing the load function in order to maintain cache coherency.

(Specifically, execute the SH4 ocbwb instruction.)

Since no DMA transfer can be used, this function is slower than `kmLoadTexture`. This is because, in texture data transfer by the function, the address of the data transfer source and destination areas is not necessarily on a 32-byte boundary.

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 1 x 1 Mipmap is specified when reloading VQ-Mipmap.

The correct picture is not displayed if the texture palette data at the reloading destination and that at the reloading source coincide when reloading Palettized-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 section)

SIZE	OFFSET	BYTES
1x1	6	2
2x2	8	8
4x4	16(10h)	32(20h)
8x8	48(30h)	128(80h)
16x16	176(B0h)	512(200h)
32x32	688(2B0h)	2048(800h)
64x64	2736(AB0h)	8192(2000h)
128x128	10928(2AB0h)	32768(8000h)
256x256	43696(AAB0h)	131072(20000h)
512x512	174768(2AAB0h)	524288(80000h)
1024x1024	699056(AAAB0h)	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 section)

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)

Offset from the beginning of the palettized 4-bpp mipmap file in KAMUI texture format to each mipmap level and the number of bytes of each mipmap level (except 16 bytes of the header section)

SIZE	OFFSET	BYTES
1x1	1	0.5
2x2	2	2
4x4	4	8
8x8	12(0Ch)	32(20h)
16x16	44(2Ch)	128(80h)
32x32	172(ACh)	512(200h)
64x64	684(2ACh)	2048(800h)
128x128	2732(AACh)	8192(2000h)
256x256	10924(2AACh)	32768(8000h)
512x512	43692(AAACh)	131072(20000h)
1024x1024	174764(2AAACh)	524288(80000h)

Offset from the beginning of the palettized 8-bpp mipmap file in KAMUI texture format to each mipmap level and the number of bytes of each mipmap level (except 16 bytes of the header section)

SIZE	OFFSET	BYTES
1x1	3	1
2x2	4	4
4x4	8	16(10h)
8x8	24(18h)	64(40h)
16x16	88(58h)	256(100h)
32x32	344(158h)	1024(400h)
64x64	1368(558h)	4096(1000h)
128x128	5464(1558h)	16384(4000h)
256x256	21848(5558h)	65536(10000h)
512x512	87384(15558h)	262144(40000h)
1024x1024	349528(55558h)	1048576(100000h)

Parameters

`pSurfaceDesc`(input)	Texture surface allocated by `kmCreateTextureSurface`/ `kmCreateCombinedTextureSurface`/ `kmCreateContiguousTextureSurface`/ `kmCreateFixedTextureArea`. This parameter is a pointer to `KMSURFACEDESC`-type structure. (Reload destination)
`pTexture`(input)	This parameter is a 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.

nMipmapCount	Texture Size
`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_TYPE`	Invalid texture specified

kmSetStrideWidth

Specifies the stride size for stride texture.

```
KMSTATUS KMAPI
kmSetStrideWidth(
IN KMINT32 nWidth
)
```

Description

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.

Parameters

`nWidth`(input)	This parameter sets the stride size.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_PARAMETER`	Invalid parameter

12. Functions for Controlling Callback

kmSetAssertCallback

Registers assert callback function.

Format

```
KMSTATUS KMAPI
kmSetAssertCallback(
                IN      PKMYCALLBACKFUNC pCallback
        );
```

Description

Registers the callback function for when Assert is generated.

This API is used for debugging. When the debugging library is linked, internal information of the library is displayed when an error is generated. Register the line callback functions for when an error is generated as follows:

These values are not returned in the release version of the library.

```
ex.)
KMVOID AssertCallbackFunc(PKMASSERTIONFAIL pInfo)
{
        DPF(pInfo->pszErrMsg);          /* Error message */
        DPF(pInfo->pszFormula);         /* Formula during error generation */
        DPF(pInfo->pszFile);            /* File where error is generated */
        DPF(pInfo->nLine);              /* Line where error is generated */
        while(1);
}
```

Note: In this example, it is assumed that DPF is the error message display function. The user must create the display function.
Also, when the message displayed is incomprehensible, contact support.

Parameters

`pCallback` (input)	Specifies the pointer to the callback function when `Assert` is generated.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetEndOfVertexCallback

Specifies the callback function to be called at the end of transfer of the data of one scene.

Format

```
KMSTATUS KMAPI
kmSetEndOfVertexCallback(
                IN   PKMCALLBACKFUNC    pEndOfVertexCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

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

Parameters

`pEndOfVertexCallback`(input)	This parameter is a pointer to the callback function
`pCallbackArguments`(input)	This parameter is a pointer to parameter to be passed to the function called at callback

Return values

`KMSTATUS_SUCCESS`	Success

kmSetEndOfYUVCallback

Registers a callback function to use for the YUV termination interrupt.

Format

```
KMSTATUS KMAPI
kmSetEndOfYUVCallback(
                IN   PKMCALLBACKFUNC    pEndOfYUVCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

This function registers a callback function to use for the YUV termination interrupt issued when the YUV converter finishes the conversion process.

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

Parameters

`pEndOfYUVCallback`(input)	This parameter specifies a pointer to a callback function to use when the YUV converter finishes its task.
`pCallbackArguments`(input)	This parameter is a pointer to an argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetEORCallback

Specifies a rendering end callback function.

Format

```
KMSTATUS KMAPI
kmSetEORCallback(
                IN   PKMCALLBACKFUNC    pEORCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

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

Parameters

`pEORCallback`(input)	This parameter is a pointer to the function to be called at the end of rendering.
`pCallbackArguments`(input)	This parameter is a pointer to argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetFatalErrorCallback

Registers callback function for FatalError.

Format

```
KMSTATUS KMAPI
kmSetFatalErrorCallback(
                IN   PKMCALLBACKFUNC    pFatalErrorCallback,
);
```

Description

When a fatal error occurs, the PVOID argument of the callback function is set to one of the following values:

`KMI_FATAL_ERR_NATIVEOVERFLOW`	(0x01)
`KMI_FATAL_ERR_VTXOVERFLOW`	(0x02)
`KMI_FATAL_ERR_UNKNOWN`	(0xFF)

VOID `FatalErrorCallbackFunc`(PVOID `pCallbackArguments`);

`pCallbackArguments`(input): Pointer to the parameter set at the specification

Parameters

`pFatalErrorCallback`(input)	This parameter is a pointer to a fatal error callback function.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetHSyncCallback

Specifies the callback function to be called at an entry into the horizontal flyback segment (Hsync).

Format

```
KMSTATUS KMAPI
kmSetHSyncCallback(
                IN   PKMCALLBACKFUNC    pHSyncCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

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
```

Parameters

`pHSyncCallback`(input)	This parameter is a pointer to the function to be called at an entry into Hsync.
`pCallbackArguments`(input)	This parameter is a pointer to argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetStripOverRunCallback

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 a strip buffer.

Format

```
KMSTATUS KMAPI
kmSetStripOverRunCallback(
                IN   PKMCALLBACKFUNC    pStripOverRunCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

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 a strip buffer. Code the callback function in the following format:

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

Parameters

`pStripOverRunCallback`(input)	This parameter is a pointer to the callback function.
`pCallbackArguments`(input)	This parameter is a pointer to argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetTexOverflowCallback

Registers the callback function to call when there is no texture memory left.

Format

```
KMSTATUS KMAPI
kmSetTexOverflowCallback(
                IN   PKMCALLBACKFUNC    pTexOverflowCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

This function registers the callback function to call when an attempt is being made to register a texture while there is no texture memory left. Code the callback function in the following format:

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

Parameters

`pTexOverflowCallback`(input)	This parameter is a pointer to the callback function to be called at texture overflow.
`pCallbackArguments`(input)	This parameter is a pointer to argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetVSyncCallback

Specifies the callback function to be called at an entry into the vertical flyback segment (Vsync).

Format

```
KMSTATUS KMAPI
kmSetVSyncCallback(
                IN   PKMCALLBACKFUNC    pVSyncCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

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
```

Parameters

`pVSyncCallback`(input)	This parameter is a pointer to the function to be called at an entry into `Vsync`.
`pCallbackArguments`(input)	This parameter is a pointer to argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

kmSetWaitVsyncCallback

Specifies a callback function to be called during a Vsync wait state.

Format

```
KMSTATUS KMAPI
kmSetWaitVsyncCallback(
                IN   PKMCALLBACKFUNC    pWaitVsyncCallback,
                IN   PVOID              pCallbackArguments
);
```

Description

This function specifies a callback function to be called during a Vsync wait state.

It is used when reading from CD-ROM is carried out in the background or when processing is performed asynchronously with other V periods.

Do not try to call too large function or to use an endless loop in a callback.

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

Parameters

`pWaitVsyncCallback`(input)	This parameter is a pointer to the function in a Vsync wait state.
`pCallbackArguments`(input)	This parameter is a pointer to an argument to be passed to the function called at callback.

Return values

`KMSTATUS_SUCCESS`	Success

13. Functions for Controlling Utility-Related Tasks

kmuCalculateKamuiWorkareaSize

Calculate KAMUI work area size.

```
KMSTATUS KMAPI
kmuCalculateKamuiWorkareaSize(
IN OUT PKMWORKAREASIZE pWorkareaSize,
)
```

Description

This function calculates the size of the work area that is specified by the contents of the `dwTextureStructNum` and `dwSmallVQStructNum` members of the `SystemConfigStruct` that is set by `kmSetSystemConfiguration`, and the `pAddress` member according to the parameters specified in the `KMWORKAREASIZE` structure.

Set the values described below for each member of the `KMWORKAREASIZE` structure.

typedef struct `tagKMWORKAREASIZE`

```
{
/* Inputs by user */
```

`KMDWORD dwNumberOfFameBuffes;`

This specifies the number of frame buffers. For example, in the case of a double buffer, specify "2."

`KMDWORD dwNumberOfNativeBuffers;`

This specifies the number of native command buffers. For example, in the case of a double buffer, specify "2."

`KMDWORD dwNumberOfTextures;`

This specifies the total number of non-Small VQ textures to be used at one time.

`KMDWORD dwNumberOf8x8SmallVQ;`

This specifies the total number of 8x8 small VQ textures to be used at one time.

`KMDWORD dwNumberOf8x8SmallVQmm;`

This specifies the total number of 8x8 small VQ mipmap textures to be used at one time.

`KMDWORD dwNumberOf16x16SmallVQ;`

This specifies the total number of 16x16 small VQ textures to be used at one time.

`KMDWORD dwNumberOf16x16SmallVQmm;`
: This specifies the total number of 16x16 small VQ mipmap textures to be used at one time.

`KMDWORD dwNumberOf32x32SmallVQ;`
: This specifies the total number of 32x32 small VQ textures to be used at one time.

`KMDWORD dwNumberOf32x32SmallVQmm;`
: This specifies the total number of 32x32 small VQ mipmap textures to be used at one time.

`KMDWORD dwNumberOf64x64SmallVQ;`
: This specifies the total number of 64x64 small VQ textures to be used at one time.

`KMDWORD dwNumberOf64x64SmallVQmm;`
: This specifies the total number of 64x64 small VQ mipmap textures to be used at one time.

`/* Outputs for SystemConfigStruct */`

`KMDWORD dwTextureStructNum;`
: This returns the maximum number of texture management structures that Kamui requires in order to manage the number of textures/buffers specified above. This is the value that is

set in the `dwTextureStructNum` member of `SystemConfigStruct`.

`KMDWORD dwSmallVQStructNum;`
: This returns the maximum number of Small VQ texture management structures that Kamui requires in order to manage the number of Small VQ textures specified above. This is

the value that is set in the `dwSmallVQStructNum` member of `SystemConfigStruct`.

`KMDWORD dwKamuiWorkareaSize;`
: This returns the maximum work area capacity that is required in order to store the number of texture management structures specified above. This is the size of the work area that is specified in the `pAddress` member of `SystemConfigStruct`.

`} KMWORKAREASIZE, *PKMWORKAREASIZE;`

Parameters

`pWorkareaSize`(input/output)	This is the structure in which the input parameters and results are saved.

Return values

`KMSTATUS_SUCCESS`	Success

kmuConvertFBtoBMP

Converts rectangle format to Windows BMP format.

```
KMSTATUS KMAPI
kmuConvertFBtoBMP(
OUT PKMDWORD pOutputData,
IN PKMDWORD pInputData,
IN KMINT32 nWidth,
IN KMINT32 nHeight,
IN KMBPPMODE nBpp
)
```

Description

This function converts the contents of the frame buffer (rectangle format) 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 function does not create the 54 bytes of the header in Windows BMP format.

Parameters

`pOutputData`(output)	Address of main memory into which the converted pixel data is to be written
`pInputData`(input)	This parameter is a 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)	This parameter specifies the screen size of the read frame buffer.
`nBpp`(input)	This parameter specifies the pixel format of the read frame buffer. One of the following can be specified.

KM_DSPBPP_RGB565	RGB-565 format
KM_DSPBPP_RGB555	RGB-555 format
KM_DSPBPP_ARGB4444	ARGB-4444 format
KM_DSPBPP_ARGB1555	ARGB-1555 format

Return value

KMSTATUS_SUCCESS	Success

kmuCreateTwiddledTexture

Converts KAMUI bit map format to Twiddled format.

```
KMSTATUS KMAPI
kmuCreateTwiddledTexture(
OUT PKMDWORD pOutputTexture,
IN PKMDWORD pInputTexture,
IN KMBOOLEAN bAutoMipMap,
IN KMBOOLEAN bDither,
IN KMINT32 USize,
IN KMTEXTURETYPE nTextureType
)
```

Description

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.

Parameters

`pOutputTexture`(output)	Address in main memory to which converted texture data is to be written
`pInputTexture`(input)	This parameter is a pointer indicating an input texture in `KM_TEXTURE_BMP` format.
`bAutoMipMap`(input)	This parameter specifies whether a mipmap is created automatically. If `TRUE` is specified, a mipmap is automatically created (the output is in `KM_TEXTURE_TWIDDLED_MM` format). If FALSE is specified, a mipmap is not created (output is in `KM_TEXTURE_TWIDDLED` format).
`bDither`(input)	This parameter specifies whether dither is effected. If `TRUE` is specified, dither is effected.
`USize`(input)	This parameter specifies the number of texels per side of texture. Select one of the following:

`KM_MAPSIZE_8`	`8x8 texels`
`KM_MAPSIZE_16`	`16x16 texels`
`KM_MAPSIZE_32`	`32x32 texels`
`KM_MAPSIZE_64`	`64x64 texels`
`KM_MAPSIZE_128`	`128x128 texels`
`KM_MAPSIZE_256`	`256x256 texels`
`KM_MAPSIZE_512`	`512x512 texels`
`KM_MAPSIZE_1024`	`1024x1024 texels`

`nTextureType`(input) This parameter specifies the pixel format of the converted texture. Select one of the following:

`KM_TEXTURE_ARGB1555`	`ARGB-1555 format`
`KM_TEXTURE_RGB565`	`RGB-565 format`
`KM_TEXTURE_ARGB4444`	`ARGB-4444 format`

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified.

kmuCreateTwiddledTextureEx

Converts KAMUI bit map format to Twiddled format.

```
KMSTATUS KMAPI
kmuCreateTwiddledTextureEx(
OUT PKMDWORD pOutputTexture,
IN PKMDWORD pInputTexture,
IN KMBOOLEAN bAutoMipMap,
IN KMBOOLEAN bDither,
IN KMINT32 USize,
IN KMINT32 VSize,
IN KMTEXTURETYPE nTextureType,
IN PKMDWORD pWorkarea
)
```

Description

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.

If different values are specified for `USize` and `VSize`, a Twiddled-Rectangle format texture is created. Note that in this case no mipmap is created. The `bAutoMipMap` flag is ignored.

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

Parameters

`pOutputTexture`(output) — Address in main memory to which converted texture data is to be written.

`pInputTexture`(input) — This parameter is a pointer indicating an input texture in `KM_TEXTURE_BMP` format.

`bAutoMipMap`(input) — This parameter specifies whether a mipmap is created automatically. If `TRUE` is specified, a mipmap is automatically created (the output is in `KM_TEXTURE_TWIDDLED_MM` format).

If `FALSE` is specified, a mipmap is not created (output is in `KM_TEXTURE_TWIDDLED` format).

If different values are specified for `USize` and `VSize`, no mipmap is created. In this case, this flag is ignored.

`bDither`(input) — This parameter specifies whether dither is effected. If `TRUE` is specified, dither is effected.

`USize`,`VSize`(input) — This parameter specifies width / height of texture by number of texels. Select one of the following:

`KM_MAPSIZE_8`	`8x8 texels`
`KM_MAPSIZE_16`	`16x16 texels`
`KM_MAPSIZE_32`	`32x32 texels`
`KM_MAPSIZE_64`	`64x64 texels`
`KM_MAPSIZE_128`	`128x128 texels`
`KM_MAPSIZE_256`	`256x256 texels`
`KM_MAPSIZE_512`	`512x512 texels`
`KM_MAPSIZE_1024`	`1024x1024 texels`

`nTextureType`(input)	This parameter specifies the pixel format of the converted texture. Select one of the following:
	`KM_TEXTURE_ARGB1555` `ARGB-1555 format`
	`KM_TEXTURE_RGB565` `RGB-565 format`
	`KM_TEXTURE_ARGB4444` `ARGB-4444 format`

`pWorkarea`(input)	This parameter specifies the starting address for a newly prepared work area of the same size as the input data. This work area is needed when converting a texture that is wider than it is tall into the Twiddled-Rectangle format. In other cases, the work area is not needed. In such a case, specify `NULL` for this parameter.

Return values

`KMSTATUS_SUCCESS`	`Success`
`KMSTATUS_INVALID_TEXTURE_TYPE`	Invalid texture type specified.
`KMSTATUS_INVALID_ADDRESS`	`pWorkarea` == `NULL` while converting a texture that is wider than it is tall.

(For compatibility purpose)

kmuCheckPassTable

Checks VERTEXCONTEXT.

```
KMUPASSSTATUS KMAPI
kmuCheckPassTable(
IN PPKMVERTEXCONTEXT ppVertexContextTable,
IN KMUINT32 nNumContext,
OUT PKMUINT32 pPass
)
```

Description

This function checks whether the content of each context in the specified `VERTEXCONTEXT` table is correct. The function is intended mainly for debugging when multiple passes are used.

Parameters

`ppVertexContextTable`(input)	This parameter specifies a pointer to an array of pointers to the prepared `VERTEXCONTEXT`.
`nNumContext`(input)	This parameter specifies the entries (passes) in the prepared ppVertexContextTable.
`pPass`(output)	If an error is detected, `KAMUI` sets the invalid `VERTEXCONTEXT` in `pPass`. (If `KMU_PASS_OK` is returned, the contents of `pPass` will be undefined.)

Return values

`KMU_PASS_OK`	Success
`KMU_PASS_ERROR_VERTEXCONTEXT`	Invalid (`NULL`) `VertexContext`
`KMU_PASS_ERROR_VERTEXCONTEXT_PASS`	Invalid `nNumContext` (less than 1)
`KMU_PASS_ERROR_PARAMTYPE`	Invalid parameter type
`KMU_PASS_ERROR_LISTTYPE`	Invalid list type
`KMU_PASS_ERROR_MIPMAP_D_ADJUST`	Invalid `Mipmap_D_Adjust`
`KMU_PASS_ERROR_FOGMODE`	Invalid fog mode (for ARC1)
`KMU_PASS_ERROR_FILTERMODE`	Invalid filter mode (for ARC1)
`KMU_PASS_ERROR_TEXTURESHADINGMODE`	Invalid texture shading mode (for ARC1)
`KMU_PASS_ERROR_COLORTYPE`	Invalid color type
`KMU_PASS_ERROR_SHADINGMODE`	Invalid shading mode (for ARC1)
`KMU_PASS_ERROR_USERCLIPMODE`	Invalid user clip mode
`KMU_PASS_ERROR_TRILINEAR_SETTING`	Invalid combination of trilinear settings. Either `ListType, ShadingMode, FilterMode, SRCBlendingMode, DSTBlendingMode, bSRCSel, or bDSTSel` is set so that it cannot be used at `Trilinear.`
`KMU_PASS_ERROR_SPRITE_SETTING`	Invalid combination of sprite settings. Either `ShadingMode`, `ColorType`, or `UVFormat` is set so that it cannot be used at `Sprite`.
`KMU_PASS_ERROR_BLENDINGMODE_SETTING`	Invalid combination of blending settings. When opaque polygon is used, (`ListType` =`KM_OPAQUE_POLYGON`), a combination is specified so that it cannot be set at `SRCBlendingMode` and `DSTBlendingMode`.
`KMU_PASS_ERROR_MODIFIER_SETTING`	Invalid combination of modifier settings. When modifier volume is used, (`ParamType` =`KM_MODIFIERVOLUME`), the `ModifierInstruction` setting is incorrect.

kmuConvertStripContext

Converts to KMSTRIPCONTEXT structure.

```
KMSTATUS KMAPI
kmuConvertStripContext(
OUT PKMVOID pStripContext,
IN PKMVERTEXCONTEXT pVertexContext1,
IN PKMVERTEXCONTEXT pVertexContext2
)
```

Description

Converts the internal setting value from the previous compatible `KMVERTEXCONTEXT` structure to `KAMUI2 KMSTRIPCONTEXT` (`KMTWOVOLUMESTRIPCONTEXT`).

Parameters

`pStripContext` (output)	Specifies the pointer to `KMSTRIPCONTEXT` or `KMTWOVOLUMESTRIPCONTEXT` structure. When using `Param2`, use the pointer to `KMTWOVOLUMESTRIPCONTEXT`.
`pVertexContext1` (input)	Specifies the pointer to the `KMVERTEXCONTEXT` structure used by `Param1`.
`pVertexContext2` (input)	Specifies the pointer to the `KMVERTEXCONTEXT` structure used by `Param2`. When `Param2` is not used, specify `NULL` in this parameter.

Return values

`KMSTATUS_SUCCESS`	Success
`KMSTATUS_INVALID_ADDRESS`	Invalid (`NULL`) parameter
`KMSTATUS_INVALID_SETTING`	Invalid `pStripContext` size setting

kmuGeneratePassTable

Generates multipass VERTEXCONTEXT automatically.

```
KMUPASSSTATUS KMAPI
kmuGeneratePassTable(
IN PKMVERTEXCONTEXT pVertexContext,
IN KMUINT32 nNumContext,
OUT PPKMVERTEXCONTEXT ppVertexContextTable,
OUT PKMUINT32 pPass
)
```

Description

This function generates the context for each pass of the multipass process (Trilinear) according to the rendering specification (context) set by the user.

A multipass process requires that `VERTEXCONTEXT` be set (when a trilinear filter is used). This function automatically generates `VERTEXCONTEXT` to relieve the user from the task of setting it for individual passes.

When a trilinear filter is used in `pVertexContext`, specifying a `VERTEXCONTEXT` value for pass 1 generates the `VERTEXCONTEXT` required for each pass according to the specified value. (The opaque polygon uses a two-pass process, while the transparent polygon uses a three-pass process.) When a trilinear filter is used for the transparent polygon, the blending mode for pass 3 can be set to any value. However, this function sets the blending mode as follows:

```
SRCBlendingMode = KM_SRCALPHA
DSTBlendingMode = KM_INVSRCALPHA
```

If `NULL` is specified in `ppVertexContextTable`, only the required number of passes is returned to `pPass`.

Parameters

`pVertexContext`(input)	This parameter is a pointer to the context for specifying rendering conditions.
`nNumContext`(input)	This parameter specifies the number of entries (passes) in the prepared `pVertexContextTable`. If the specified value is smaller than the number of actually required passes, `KMU_PASS_ERROR_VERTEXCONTEXT_PASS` is returned. In this case, the function ends only by setting the number of required passes in `pPass`.
`ppVertexContextTable`(output)	This parameter specifies a pointer to an array of pointers to `VERTEXCONTEXT` where the generated multipass context is to be received. If `NULL` is specified in this argument, only the number of required passes is returned to `pPass`.
`pPass`(output)	KAMUI returns the number of multipasses required in the specified rendering to this parameter.

Return values

`KMU_PASS_OK`	Success
`KMU_PASS_ERROR_VERTEXCONTEXT`	Invalid (NULL) `VERTEXCONTEXT`
`KMU_PASS_ERROR_VERTEXCONTEXT_PASS`	The number of specified passes is insufficient.