PVRTexTool

Reference Manual

\\Gb1\PowerVR\DevRel\SDK\Internal\PVRTexTools

Filename : PVRTexTool.Reference Manual.mht

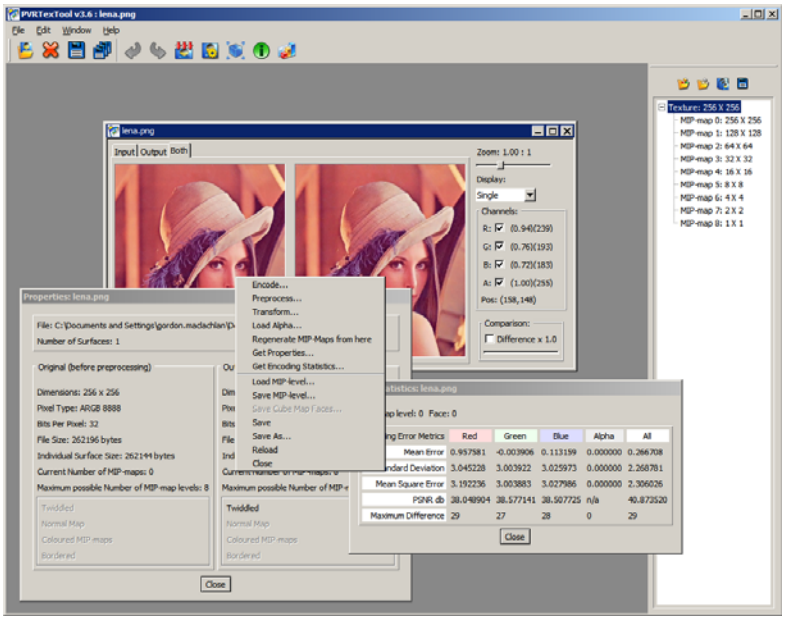
Version : 1.3f (PowerVR SDK 2.03.23.1159)

Issue Date : 28 Aug 2008

Author : PowerVR

Contents

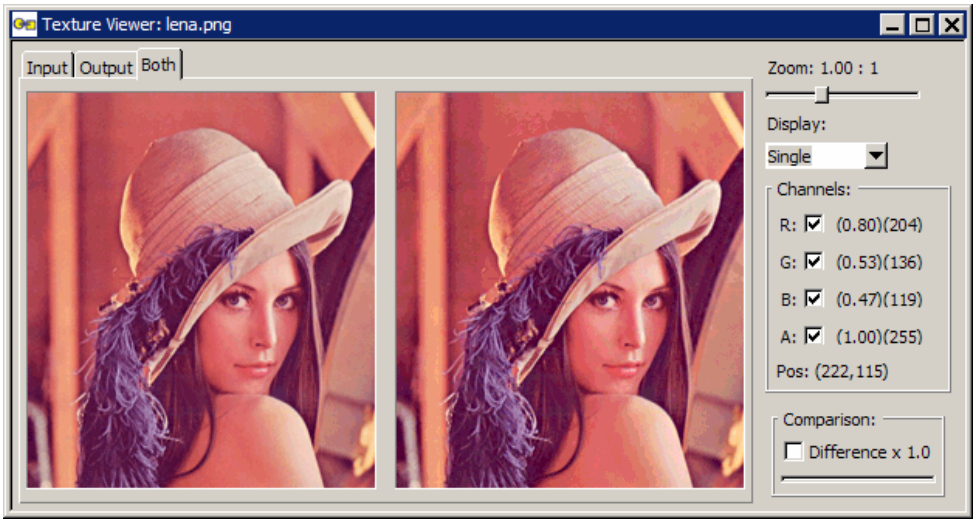
1. PVRTexTool GUI



PVRTexTool GUI is used to convert image files (BMP, TGA, GIF, PCX, JPG, PNG) files into hardware-friendly texture files. This is a Graphical User Interface program, available for Windows and Linux. Only the PVRTexTool GUI executable is required to run the program.

1.1. Texture Viewer

This window can display the currently selected MIP-map level of a texture in a number of different ways, along with some diagnostic information.



1. View Panels – Show the current image data before and after encoding, on a chequered background. If the image has transparency, it will be alpha blended with the background. The image can be moved around by clicking and dragging.

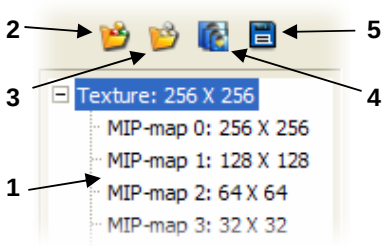
2. Zoom – Use this slider to zoom in and out of the image. The zoom feature can also be operated using the mouse wheel on either of the view panels.

3. Display mode – Selects how the image should be displayed. If the file is a normal 2D texture, you can choose between a single and a tiled view. If you are viewing a cube map, you can choose between a single view of the currently selected face and a 3D preview.

4. Channels – Un-checking each of these tick boxes will suppress the corresponding channel in the view panels. For example, if "A" is unchecked, alpha blending will be turned off. This pane also gives numerical values for each of the channels at the current cursor position, and the position on the texture.

5. Difference – Activate the checkbox to display the difference between the original image data, and the image after it has been encoded, to highlight compression artefacts. Ticking the box will enable a slider which controls the intensity of this image, with a scale factor from 1 to 9.

1.2. Image Browser



1. MIP-map browser – Shows a list of all MIP-map levels in the texture. Click on a MIP-map to view it in the texture viewer.

2. Load level – Load an image into the currently selected MIP-map level.

3. Load alpha – Load an image as the alpha for the currently selected MIP-map level.

4. Save level – Saves an image of the current MIP-map level once it has been encoded in the specified format, then decoded.

5. Generate MIP-maps – Uses the currently selected MIP-map to generate all smaller MIP-maps.

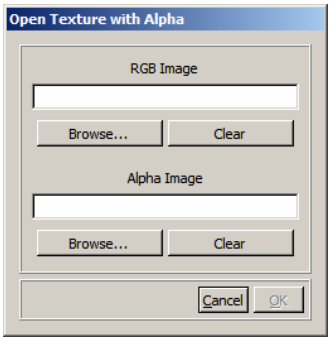
1.3. File Menu

1.3.1. Open…

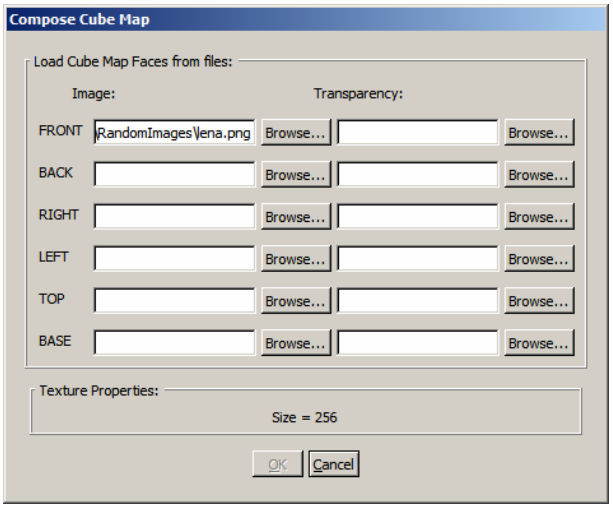
Use File | Open, to open a previously created texture file, or to open a standard image file as a texture. PVRTexTool supports POWERVR texture files (*.PVR), and Microsoft Direct Draw Surface files (*. DDS). It can also read the following image formats: BMP, TGA, GIF, PCX, JPG and PNG.

1.3.2. Open with Alpha…

Use File | Alpha to bring up a dialog for combining two image files to produce a texture with an alpha channel as well as RGB data. Specifying only an alpha image will produce a white texture with only the alpha channel filled.



1.3.3. Compose Cube Map…



Use this option to create a cube map texture by combining existing images. It allows you to specify image files from which to load each of the six faces of the cube and an optional transparency image for each one. All of these source images must be of identical, square dimensions.

1.3.4. Reload From File

This option reloads the current texture from disk reverting any pre-processing already carried out upon it. If the file has been updated in some way by another program since being opened this also allows the texture in memory to be updated to what is currently stored on disk.

1.3.5. Save/Save As

The user can save the texture in different formats:

- .PVR files. The data consists of a 12 bytes header (described in section 4) followed by the raw texture data.
- .H files which are like .PVR file but in a format that allows including it in a C project. The default type for the data is currently 'unsigned long' although the legacy mode ('unsigned char') is still supported as an option. We recommend using the default mode to avoid alignment problems in some platforms enhancing portability of these textures files.
- .DDS files. Files ready for use in Microsoft DirectX. See the DirectX documentation for a detailed description.
- .NGT files. Files ready for use with Nokia's NGAGE 2 SDK. See the NGAGE 2 documentation for a detailed description.

The user will be prompted to select a texture encoding format, if the texture hasn't already been encoded through the usual encoding dialogue (described below).

1.3.6. Close/Close All

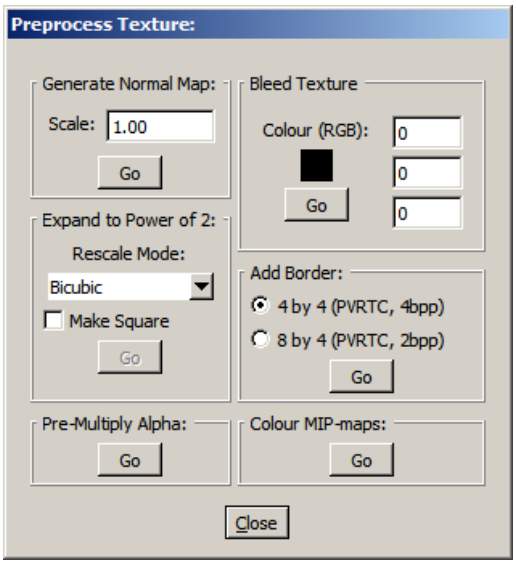
These options variously close the active document window or all open windows.

1.3.7. Quit

Exit PVRTexTool.

1.4. Edit Menu

1.4.1. Preprocess…



This option brings up a pre-processing dialog box showing all the procedures which can be performed on the texture prior to encoding.

Options

Generate Normal Map

A normal map is a texture that stores normal vectors instead of colours. The X component of the normal vector is stored in the red channel of the texture, the Y component in the green channel and the Z component in the blue channel. Normal maps are commonly used with Dot3 bump mapping. The normal map is calculated from a grey scale height map that indicates the roughness of the surface. Large values in the map mean taller heights, while small values mean lower heights. The values come from the red channel of the input texture.

A normal vector is calculated per pixel using the difference between the intensity of adjacent pixels and then is compressed into the format selected. Note that internally, the format of normal maps is automatically raised to 32-bit floating point precision.

Once the top MIP-map level has been converted to a normal map, lower MIP-map levels are regenerated from this image using the standard 2 by 2 averaging algorithm. This will give physically correct results for per pixel lighting on matte surfaces.

Expand to Power of 2

This is a quick function that will expand any size of texture so that its dimensions have values of powers of two although not necessarily equal unless the Make Square option is chosen. Nearest neighbour, Bilinear and Bicubic scaling algorithms are available for this operation.

Pre-Multiply Alpha

For alpha-blended rendering it is sometimes useful to have the other, opaque channels of a texture encoded with their value pre-multiplied by the alpha value for each pixel. Clicking Go will carry this out.

Bleed Texture

When mapping certain parts of a texture on an object, the texture may contain an invisible void between useful parts. This void creates discontinuities around the textured parts that can impair the texture compression with certain formats. Uncompressed textures can also benefit from this bleeding operation as texture filtering (bilinear and/or MIP-mapping) can cause undesired adjacent texels to contribute to the final colour being sampled. To avoid this issue, the bleeding process transforms these frontiers by filling the void with a mix from nearby pixels. You can enter this background colour in hexadecimal via the text field, or by clicking on the swatch to the left of the text field and selecting the colour from the image.

Add Border

PVRTC texture data is assumed to be continuous across texture edges, which is a common case in graphic applications where various material textures like rock, brick, grass etc. are tiled together to represent high texture detail for a larger area. This can occasionally result in minor compression artefacts along the edges of texture data that does not tile.

Whenever results are deemed unsatisfying, this issue can easily be resolved by adding a border around the original texture data. This border will then absorb any possible artefacts related to tiling.

This option has two possible functions depending on the width and height of the texture:

- Width and height both a power of 2. In this case the user can select one of two options. The algorithm will scale the texture down by 8 pixels in the vertical direction (4 on each side) and either 8 or 16 pixels in the horizontal direction, and then add mirrored borders.
- Width and/or height not a power of 2. In this other case, the border is directly added, so as to expand the image dimensions to their nearest respected powers of 2. This way the graphics are not changed by a downsize operation.

Skyboxes: Skybox textures are a collection of six textures which are mapped onto a cube to represent the sky or background of a 3D scene. These textures are a special type of non-tiling textures with a very specific tiling behaviour from one texture to another. When filling in the border pixels this special behaviour needs to be taken into account to obtain the best possible end-result in combination with PVRTC. The specific tiling behaviour is illustrated in the PVRTC Texture Compression Usage Guide document from the POWERVR SDK. Skyboxes have six textures, specifically two textures for each major axis: X+ and X-, Y+ and Y-, Z+ and Z-.

To remove potential tiling or texture filtering artefacts the border regions of each texture face need to be filled with the correct data from the neighbouring textures. For instance the borders of the Z+ texture get the top data from the Y+ texture, the bottom border gets data from the Y- texture, the left border gets data from the X- texture and finally the right border gets pixels from the X+ texture.

Saving an image which this has been applied to as a .pvr or .h file will set a flag in the output file's header (see table 2.1.1). For more information, please refer to the PVRTC Texture Compression Usage Guide document that's part of the POWERVR SDK.

A texture pre-processed in this way will have an edge of 4x4 pixels for the PVRTC 4bpp case and an edge of 8x4 for the PVRTC 2bpp case. This texture has to be remapped correctly to restore it to its original size. This remapping can be calculated as follows:

(Note that 1 is added to the border size to avoid bilinear bleeding. ResX and ResY are the original texture resolution)

PVRTC 4bpp:

u = ((4+1)/ResX)+u*(1-(2*(4+1)/ResX)

v = ((4+1)/ResY)+v*(1-(2*(4+1)/ResY)

PVRTC 2bpp:

u = ((8+1)/ResX)+u*(1-(2*(8+1)/ResX)

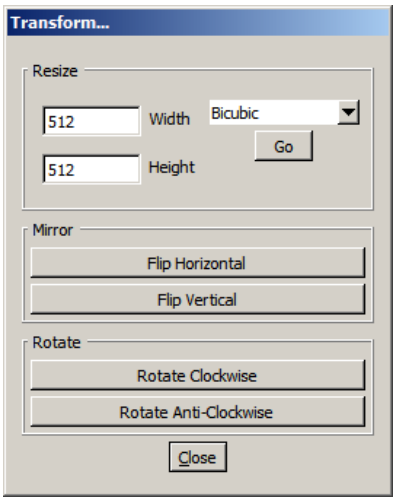
v = ((4+1)/ResY)+v*(1-(2*(4+1)/ResY)

Colour MIP-maps

Sometimes, for testing purposes, it's useful to mark each MIP-map in the chain by giving it a colour. This option goes through the existing MIP-map chain and colours the levels green, red, yellow, pink, cyan, blue in descending order. It leaves the top MIP-map level unchanged.

1.4.2. Transform…

The transform dialog allows basic geometric transformations to be applied to the current texture.



Resize

Enter the desired dimensions, in pixels, into the Width and Height boxes, choose the scaling algorithm you desire and press go. Nearest, Bilinear and Bicubic scaling is available.

Mirror

Basic reversal in the horizontal and vertical axes is available through this section of the Transform Dialog.

Rotate

Click the option in this section to rotate the current texture by 90 degrees in a clockwise or anti-clockwise direction.

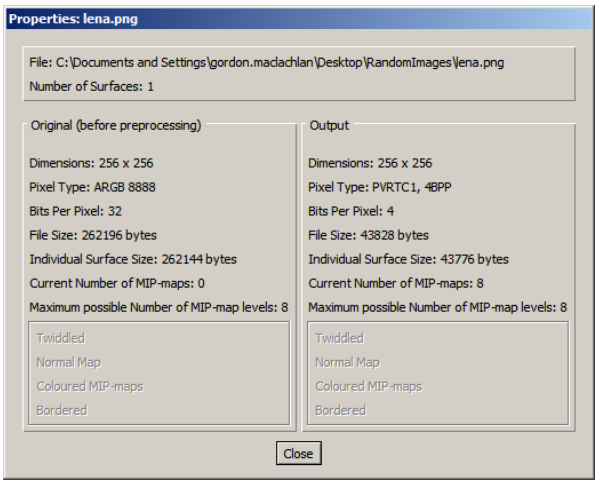
For cube map textures, only the mirror and rotate options are available and these may be applied to the currently selected surface in the Surface Browser or all surfaces at once.

1.4.3. Regenerate MIP-maps

This option regenerates the entire MIP-map chain down to the smallest possible size, from the currently selected level, by successively performing 2 by 2 averaging.

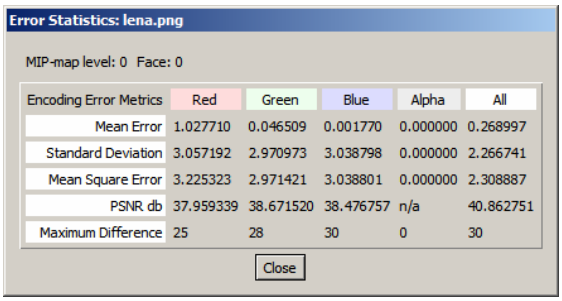
1.4.4. Get Properties…

Displays some properties of the currently active texture in a floating dialog:



1.4.5. Get Error Statistics…

Displays some error statistics concerning the difference between input and output images of the current texture:



Encoding Error Metrics	Red	Green	Blue	Alpha	All
Mean Error	1.027710	0.046509	0.001770	0.000000	0.268997
Standard Deviation	3.057192	2.970973	3.038798	0.000000	2.266741
Mean Square Error	3.225323	2.971421	3.038801	0.000000	2.308887
PSNR db	37.959339	38.671520	38.476757	n/a	40.862751
Maximum Difference	25	28	30	0	30

1.4.6. Load MIP-Level/Load MIP-Level Alpha/Save MIP-Level

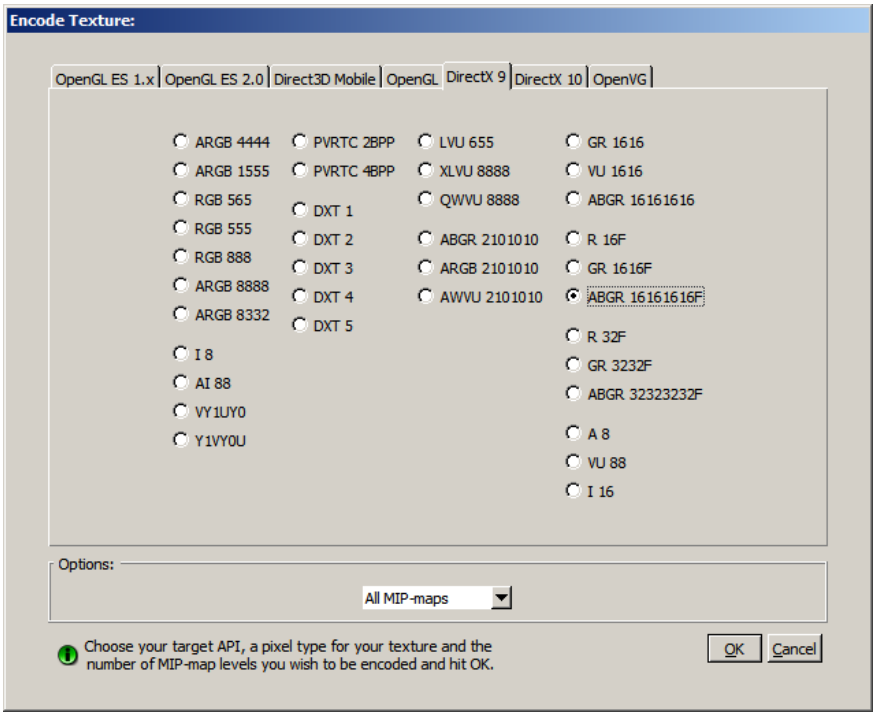
These options allow you to load an image into the currently selected MIP-map level, load an image for its output channel, or save an image of the current level after it has been encoded.

1.4.7. Save Cube Map Faces

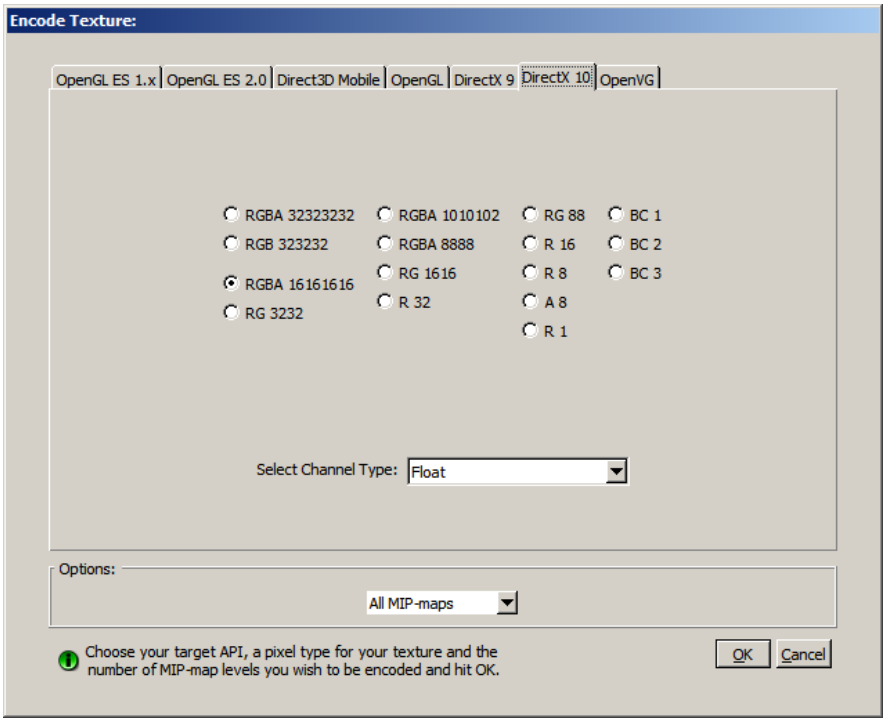
This option allows the faces of a cube map to be saved in PNG, JPG, GIF, TGA, PCX or BMP format. The filenames will be suffixed by FRONT, BACK, LEFT etc. corresponding to each face.

1.4.8. Encode…

This menu displays a dialog box that allows the user to specify the format of texture data produced.



First, choose your target API from the row of tabs at the top. Then choose the texture format that you require from the list in the tab. For the DX10 tab you will need to choose a channel type from the drop down menu as well.



Finally, choose the number of MIP-map levels to encode in addition to the actual image or just to encode this top image with the MIP-map selector and click OK.

2. PVR file format description

The PVR file format is composed of a header followed by texture data. The header is 13 DWORDs (52 bytes) long with the following format:

2.1.1. File Header description:

DWORD Offset	Content	Description
0	HEADER SIZE (BYTES)	52
1	Height	Vertical height of the texture
2	Width	Horizontal width of the texture
3	MIP-map Levels	The number of MIP-map levels present in this file.
4	0x000000XX	Pixel format identifier – please refer to the table at the end of this document for possible values.
	Format flags:	
	0x00000100	File has MIP-maps.
	0x00000200	Twiddled.
	0x00000400	Normal Map.
	0x00000800	Added border.
	0x00001000	File is a cube map.
	0x00002000	False colour MIP-maps.
	0x00004000	Volume Texture.
	0x00008000	Alpha in texture.
5	Surface Data Size (bytes).	Size of entire texture, or one cube map face if this is a cube map.
6	Bits Per Pixel	The number of bits of data describing a single pixel.
7	Red mask	Mask for the red channel.
8	Green Mask	Mask for the green channel.
9	Blue mask	Mask for the blue channel.
10	Alpha Mask	Mask for the alpha channel.
11	PVR Id	‘P’ ’V’ ’R’ ’!’
12	Number of Surfaces	The number of slices for volume textures or sky boxes

2.1.2. File Header structure

```
typedef struct PVR_TEXTURE_HEADER_TAG{
    unsigned int  dwHeaderSize;     /* size of the structure */
    unsigned int  dwHeight;         /* height of surface to be created */
    unsigned int  dwWidth;          /* width of input surface */
    unsigned int  dwMipMapCount;    /* number of MIP-map levels requested */
    unsigned int  dwpfFlags;        /* pixel format flags */
    unsigned int  dwDataSize;       /* Size of the compress data */
    unsigned int  dwBitCount;       /* number of bits per pixel  */
    unsigned int  dwRBitMask;       /* mask for red bit */
    unsigned int  dwGBitMask;       /* mask for green bits */
    unsigned int  dwBBitMask;       /* mask for blue bits */
    unsigned int  dwAlphaBitMask;   /* mask for alpha channel */
    unsigned int  dwPVR;            /* should be 'P' 'V' 'R' '!' */
    unsigned int  dwNumSurfs;       /* number of slices for volume textures or skyboxes */
} PVR_TEXTURE_HEADER;
```

Notes:

dwMipMapCount is the number of MIP-map levels present in addition to the top level. 0 means that there is only the top level, 1 means the top level plus an extra MIP-map level, etc…

PVRTC4, PVRTC2 and formats that use more than 32 bits per pixel do not have **dwRBitMask, dwGBitMask** and **dwBBitMask** defined. For PVRTC4 and PVRTC2 **dwAlphaBitMask** will be 1 or 0 depending on whether the texture has alpha information or not.

3. PVRTexTool Command-line version

3.1. Description

This is the command-line version of PVRTexTool, and can be found inside the “CL” folder. It exists for Windows and Linux platforms. Only the PVRTexTool executable (PVRTexTool.exe for Windows and PVRTexTool for Linux) is required to run the program.

3.2. Usage

PVRTexTool can be used from the command-line to be able to process and compress textures using a batch file. The syntax for the command-line is as follows:

PVRTexTool [-m] [-b] [-c<scaler>] [-nt] [-h] [-u8] [-x<width>] [-y<height>] -f(format) -iSrcImage [-aSrcAlpha[.bmp]] [-oOutputStem]

Options

-**m**	Automatically generate all MIP-Map levels.
-b[factor]	Bump/Normal map. This option calculates the normal-map from a height-map passed as input. [factor] is a multiplication factor for the normal map. Default value is 2.0.
-**nt**	No twiddle. (Twiddle is enabled by default except for .dds files)
-ngt	Output Nokia NGAGE2 NGT file. This requires use of a PVRTC 4BPP format.
-p	Create binary PVR file with texture data. This is the default setting if the output type is not specified.
-h	Create include file with texture data.
-d	Create output file(s) with decompressed texture data.
-**f**	Output file format. For example –fPVRTC4 4444, 1555, 565, 555, 888, 8888, 8332, 88, 8, VY1UY0, Y1VY0U, PVRTC2, PVRTC4, ETC, OGL4444, OGL5551, OGL8888, OGLBGRA8888, OGL565, OGL555, OGL888, OGL88, OGL8, OGLPVRTC2, OGLPVRTC4, 1_BPP, DXT1, DXT3, DXT5, 332, 44, LVU655, XLVU8888, QWVU8888, ABGR2101010, ARGB2101010, AWVU2101010, GR1616, VU1616, ABGR16161616, R16F, GR1616F, ABGR16161616F, R32F, GR3232F, ABGR32323232F. Please see the table at the end of this document for further format options.
-i	Input file name (BMP, JPG, PNG, GIF or TGA file).
-a	Input alpha file name (also BMP, JPG, PNG, GIF or TGA file).
-o	OutputFile name. If this option is not used the output file name will be the same as the input file name with the extension .pvr or .h
-t1	Preprocess texture for tiling with a mirrored border of 4 pixels at top and bottom and 8 pixels at the sides – for use specifically with PVRTC2 compression.
-**t2**	Preprocess texture for tiling with a mirrored border of 4 pixels on all sides – for use specifically with PVRTC4 compression.
-s	Compress a skybox. Files must be named XXXXXn where n=1-6.
-x	Define a new width
-y	Define a new height
-r	Choose a resizing algorithm. 1 = nearest, 2 = bilinear, 3 = bicubic. Default is bicubic.
-q	quality mode for ETC compression. 0 = Fast, 1 = Medium, 2 = Slow, 3 = Fast Perceptual, 4 = Medium Perceptual, 5 = Slow Perceptual. Default is 3.
-lRRGGBB	Apply a bleed filter to the texture and its MIP-maps with RRGGBB supplied as a 32-bit hexadecimal colour value.

Examples

To encode the file Example.bmp as an header include file with MIP-maps in ARGB 1555 format.

```
PVRTexTool -h -m -f1555 -iExample.bmp
```

To encode a sky box from files named skyboxn.bmp in the same format:

```
PVRTexTool -h –s -m -f1555 –iskybox1.bmp
```

4. PVRTexTool Plug-ins

4.1. Adobe Photoshop CS/CS2/CS3

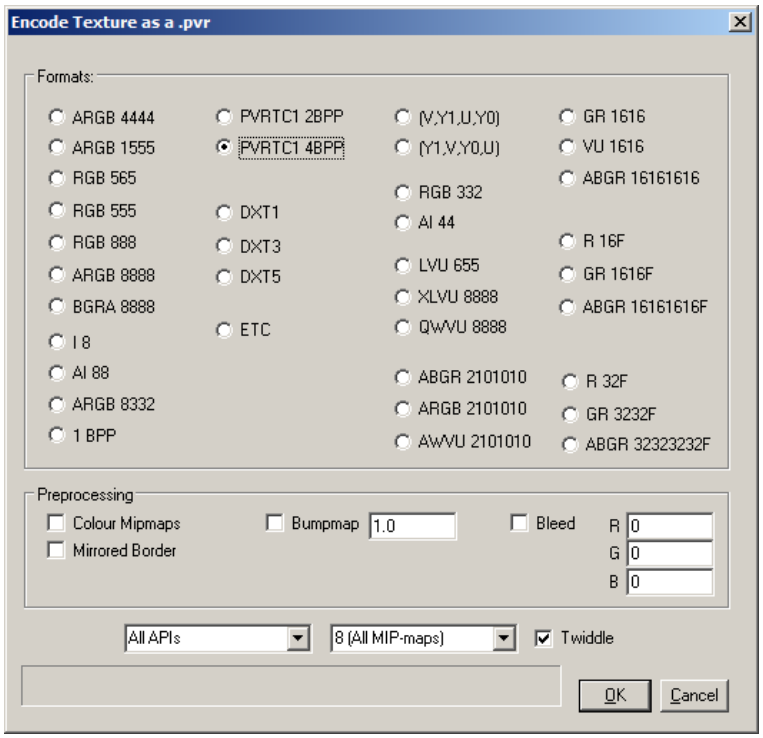
This plug-in for Photoshop allows full PVRTexTool capabilities from within this professional image manipulation package.

To install it just copy the file supplied into the folder

\Program Files\Adobe\Adobe Photoshop CS3\Plug-Ins\File Formats

Or similar.

Once you start Photoshop, PVR will be available as one of the formats supported for loading and also for saving images:



4.2. Autodesk 3DStudioMAX v6, 7, 8, 9, 2008, 2009

This plug-in will allow 3DSMax to load the PVR image format. When applying a material, this format will be available in the supported list and it will be displayed properly in the view-ports and final rendered images.

It is possible to save a rendered image in PVR format, but the output format will be limited to 32 bits per pixel to keep quality.

To install this plug-in just copy the file supplied into the folder

\Program Files\Autodesk\3dsMax<version>\plugins

4.3. Autodesk Maya

This plug-in, like the previous one, will allow to load a PVR file as a texture for a material. Equally, when saving an image, the format will be limited to 32 bits per pixel..

To install this plug-in just copy the file supplied into the folder
\Program Files\Alias\Maya<version>\bin\plug-ins\image

5. Twiddle format description

How it works:

The U and V indices are interleaved in memory as follows: -

U3V3U2V2U1V1U0V0

The subscript denotes the bit position in U or V, which gives an order of texel storage shown below.

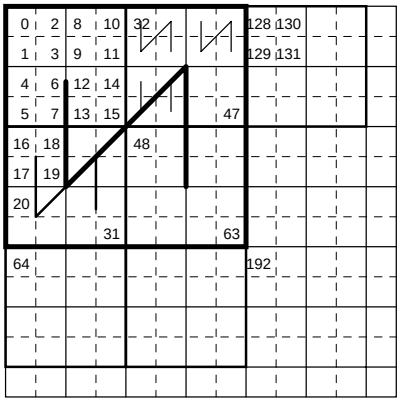


Figure 1. Twiddled Textures

The grid represents the position in the bitmap, and the number represents the location that it is stored in memory. In the case of rectangular twiddling, the common least significant bits are twiddled, and the left over most significant bits are appended to the top. For example, take a 4 bit U co-ordinate and a 6-bit V co-ordinate. Twiddle the 4 least significant bits of U and V, then append the 2 most significant bits of V onto the top of the result of the twiddling thus: -

V5V4U3V3U2V2U1V1U0V0

Under MGL, source texture data can be twiddled into hardware-friendly format by using the `mglx_texture_twiddle()` function.

When creating a texture with the OpenGL ES API, for instance, the texture source data must *not* be twiddled, as the drivers will perform the twiddling internally prior to uploading the texture to the hardware. PVRTexTool should handle the toggling of twiddling automatically for most cases.

6. Texture Format Reference

Although some of the formats below are for specific colour spaces PVRTexTool is not colour space aware and it is up to the user to ensure that data from the correct colour space is used with PVRTexTool.

Please note that greyed out formats, whilst present in the PixelType enum, are not supported by PVRTexTool at this time.

Format	Description	Command Line Identifier eg –f4444	Identifer Enum	Enum Value
ARGB 4444	Good 16-bit format when smooth translucency is needed.	4444	MGLPT_ARGB_4444	0x0
ARGB 1555	Punch-through 16-bit translucent format.	1555	MGLPT_ARGB_1555	0x1
RGB 565	Best quality 16-bit opaque format.	565	MGLPT_RGB_565	0x2
RGB 555	As 1555 format but alpha is ignored. Good channel balance.	555	MGLPT_RGB_555	0x3
RGB 888	24-bit opaque format with 8 bits for each colour channel.	888	MGLPT_RGB_888	0x4
ARGB 8888	Best quality 32-bit format, but size and performance are worse than 16-bit formats.	8888	MGLPT_ARGB_8888	0x5
ARGB 8332	High quality translucency 16-bit format.	8332	MGLPT_ARGB_8332	0x6
I 8	8-bit intensity only format.	8	MGLPT_I_8	0x7
AI 88	16-bit alpha and intensity format.	88	MGLPT_AI_88	0x8
1BPP	One bit per pixel.	1_BPP	MGLPT_1_BPP	0x9
(V,Y1,U,Y0)	YUV 16-bit format. Used for streaming movies. Good for photographic quality textures.	VY1UY0	MGLPT_VY1UY0	0xA
(Y1,V,Y0,U)	YUV format.	Y1VY0U	MGLPT_Y1VY0U	0xB
PVRTC2	PVRTC compression format. 2-bit per pixel.	PVRTC2	MGLPT_PVRTC2	0xC
PVRTC4	PVRTC compression format. 4-bit per pixel.	PVRTC4	MGLPT_PVRTC4	0xD
OpenGL ARGB 4444	Good 16-bit format when smooth translucency is needed.	OGL4444	OGL_RGBA_4444	0x10
OpenGL ARGB 1555	Punch-through 16-bit translucent format.	OGL1555	OGL_RGBA_5551	0x11

OpenGL ARGB 8888	Best quality 32-bit format, but size and performance are worse than 16-bit formats.	OGL8888	OGL_RGBA_8888	0x12
OpenGL RGB 565	Best quality 16-bit opaque format.	OGL565	OGL_RGB_565	0x13
OpenGL RGB 555	As 1555 format but alpha is ignored. Good channel balance.	OGL555	OGL_RGB_555	0x14
OpenGL RGB 888	24-bit opaque format with 8 bits for each colour channel.	OGL888	OGL_RGB_888	0x15
OpenGL I 8	8-bit intensity only format.	OGL8	MGLPT_I_8	0x16
OpenGL AI 88	16-bit alpha and intensity format.	OGL88	MGLPT_AI_88	0x17
OpenGL PVRTC2	PVRTC compression format. 2-bit per pixel.	OGLPVRTC2	MGLPT_PVRTC2	0x18
OpenGL PVRTC4	PVRTC compression format. 4-bit per pixel.	OGLPVRTC4	MGLPT_PVRTC4	0x19
OpenGL BGRA 8888	An OpenGL|ES extension-only format offering the same quality as ARGB 8888 in what may be a more desirable channel order.	OGLBGRA8888	OGL_BGRA_8888	0x1A
DXT1	Microsoft S3TC format, 4 bits per pixel with no alpha information.	DXT1	D3D_DXT1	0x20
DXT2	Microsoft S3TC format, 8 bits per pixel. Good for sharp alpha transitions. Alpha is considered premultiplied.	DXT2	D3D_DXT2	0x21
DXT3	Microsoft S3TC format, 8 bits per pixel. Good for sharp alpha transitions.	DXT3	D3D_DXT3	0x22
DXT4	Microsoft S3TC format, 8 bits per pixel. Good for gradient alpha transitions. Alpha is considered premultiplied.	DXT4	D3D_DXT4	0x23
DXT5	Microsoft S3TC format, 8 bits per pixel. Good for gradient alpha transitions.	DXT5	D3D_DXT5	0x24
RGB 332	8-bit opaque format.	332	D3D_RGB_332	0x25
AI 44	8-bit alpha and intensity format.	44	D3D_AI_44	0x26
LVU 655	YUV format.	LVU655	D3D_LVU_655	0x27
XLVU 8888	YUV format.	XLVU8888	D3D_XLVU_8888	0x28

QWVU 8888	Signed 8bit format designed for bump mapping.	QWVU8888	D3D_QWVU_8888	0x29
ABGR 2101010	10-bit precision format with 2 bits for alpha.	ABGR2101010	D3D_ABGR_2101010	0x2A
ARGB 2101010	Another 10-bit precision format with 2 bits for alpha.	ARGB2101010	D3D_ARGB_2101010	0x2B
AWVU 2101010	10-bit precision signed format with 2 bits for alpha.	AWVU2101010	D3D_AWVU_2101010	0x2C
GR 1616	2-channel 16-bit per channel format.	GR1616	D3D_GR_1616	0x2D
VU 1616	2-channel 16-bit per channel format.	VU1616	D3D_VU_1616	0x2E
ABGR 16161616	64-bit format with transparency.	ABGR16161616	D3D_ABGR_16161616	0x2F
R 16F	Single channel 16-bit floating point format.	R16F	D3D_R16F	0x30
GR 1616F	2-channel 16-bit floating point format.	GR1616F	D3D_GR_1616F	0x31
ABGR 16161616F	64-bit floating point format with transparency.	ABGR16161616F	D3D_ABGR_16161616F	0x32
R 32F	Single channel 32-bit floating point format.	R32F	D3D_R32F	0x33
GR 3232F	2-channel 32-bit floating point format.	GR3232F	D3D_GR_3232F	0x34
ABGR 32323232F	128-bit floating point format with transparency.	ABGR32323232F	D3D_ABGR_32323232F	0x35
ETC	Ericsson Texture Compression, 4 bits per pixel with no alpha information.	ETC	ETC_RGB_4BPP	0x36
A 8	8-bit alpha texture format.	DX9 A 8	D3D_A8	0x39
VU 88	2 channel 16-bit format.	DX9 VU 88	D3D_V8U8	0x3A
I 16	Single channel 16-bit format.	DX9 I 16	D3D_I16	0x3B

6.1. DirectX 10 Formats

Format	Channel Type	Description	Command Line Identifier	Identifer Enum	Enum Value
RGBA 32323232	float	High precision formats with alpha support	DX10_R32G32B32A32_FLOAT	DX10_R32G32B32A32_FLOAT	0x50

RGBA 32323232	unsigned int		DX10_R32G32B32A32_UINT	DX10_R32G32B32A32_UINT	0x51
RGBA 32323232	signed int		DX10_R32G32B32A32_SINT	DX10_R32G32B32A32_SINT	0x52
RGB 323232	float	High precision formats with no alpha support	DX10_R32G32B32_FLOAT	DX10_R32G32B32_FLOAT	0x53
RGB 323232	unsigned int		DX10_R32G32B32_UINT	DX10_R32G32B32_UINT	0x54
RGB 323232	signed int		DX10_R32G32B32_SINT	DX10_R32G32B32_SINT	0x55
RGBA 16161616	float	16-bit precision formats with alpha support	DX10_R16G16B16A16_FLOAT	DX10_R16G16B16A16_FLOAT	0x56
RGBA 16161616	unsigned normalised int		DX10_R16G16B16A16_UNORM	DX10_R16G16B16A16_UNORM	0x57
RGBA 16161616	unsigned int		DX10_R16G16B16A16_UINT	DX10_R16G16B16A16_UINT	0x58
RGBA 16161616	signed normalised int		DX10_R16G16B16A16_SNORM	DX10_R16G16B16A16_SNORM	0x59
RGBA 16161616	signed int		DX10_R16G16B16A16_SINT	DX10_R16G16B16A16_SINT	0x5A
RG 3232	float	High precision two channel formats	DX10_R32G32_FLOAT	DX10_R32G32_FLOAT	0x5B
RG 3232	unsigned int		DX10_R32G32_UINT	DX10_R32G32_UINT	0x5C
RG 3232	signed int		DX10_R32G32_SINT	DX10_R32G32_SINT	0x5D
RGBA 1010102	unsigned normalised int	10-bit precision format with basic 2 bit support for alpha.	DX10_R10G10B10A2_UNORM	DX10_R10G10B10A2_UNORM	0x5E
RGBA 1010102	unsigned int		DX10_R10G10B10A2_UINT	DX10_R10G10B10A2_UINT	0x5F
RGBA 8888	unsigned normalised int	32-bit formats with alpha support	DX10_R8G8B8A8_UNORM	DX10_R8G8B8A8_UNORM	0x61
RGBA 8888	unsigned normalised int sRGB colour		DX10_R8G8B8A8_UNORM_SRGB	DX10_R8G8B8A8_UNORM_SRGB	0x62

	space				
RGBA 8888	unsigned int		DX10_R8G8B8A8_UINT	DX10_R8G8B8A8_UINT	0x63
RGBA 8888	signed normalised int		DX10_R8G8B8A8_SNORM	DX10_R8G8B8A8_SNORM	0x64
RGBA 8888	signed int		DX10_R8G8B8A8_SINT	DX10_R8G8B8A8_SINT	0x65
RG 1616	float	16-bit precision two channel formats	DX10_R16G16_FLOAT	DX10_R16G16_FLOAT	0x66
RG 1616	unsigned normalised int		DX10_R16G16_UNORM	DX10_R16G16_UNORM	0x67
RG 1616	unsigned int		DX10_R16G16_UINT	DX10_R16G16_UINT	0x68
RG 1616	signed normalised int		DX10_R16G16_SNORM	DX10_R16G16_SNORM	0x69
RG 1616	signed int		DX10_R16G16_SINT	DX10_R16G16_SINT	0x6A
R 32	float	32-bit single channel formats	DX10_R32_FLOAT	DX10_R32_FLOAT	0x6B
R 32	unsigned int		DX10_R32_UINT	DX10_R32_UINT	0x6C
R 32	signed int		DX10_R32_SINT	DX10_R32_SINT	0x6D
RG 88	unsigned normalised int	8-bit precision two channel formats	DX10_R8G8_UNORM	DX10_R8G8_UNORM	0x6E
RG 88	unsigned int		DX10_R8G8_UINT	DX10_R8G8_UINT	0x6F
RG 88	signed normalised int		DX10_R8G8_SNORM	DX10_R8G8_SNORM	0x70
RG 88	signed int		DX10_R8G8_SINT	DX10_R8G8_SINT	0x71
R 16	float	16-bit single channel formats	DX10_R16_FLOAT	DX10_R16_FLOAT	0x72
R 16	unsigned normalised int		DX10_R16_UNORM	DX10_R16_UNORM	0x73
R 16	unsigned int		DX10_R16_UINT	DX10_R16_UINT	0x74
R 16	signed normalised int		DX10_R16_SNORM	DX10_R16_SNORM	0x75

R 16	signed int		DX10_R16_SINT	DX10_R16_SINT	0x76
R 8	unsigned normalised int	8-bit single channel formats	DX10_R8_UNORM	DX10_R8_UNORM	0x77
R 8	unsigned int		DX10_R8_UINT	DX10_R8_UINT	0x78
R 8	signed normalised int		DX10_R8_SNORM	DX10_R8_SNORM	0x79
R 8	signed int		DX10_R8_SINT	DX10_R8_SINT	0x7A
A 8	unsigned normalised int	8-bit single channel alpha format	DX10_A8_UNORM	DX10_A8_UNORM	0x7B
R 1	unsigned normalised int	1-bit per pixel texture format	DX10_R1_UNORM	DX10_R1_UNORM	0x7C
BC 1	unsigned normalised int	Microsoft S3TC format, 4 bits per pixel with no alpha information.	DX10_BC1_UNORM	DX10_BC1_UNORM	0x80
BC 1	unsigned normalised int sRGB colour space	Microsoft S3TC format, 4 bits per pixel with no alpha information.	DX10_BC1_UNORM_SRGB	DX10_BC1_UNORM_SRGB	0x81
BC 2	unsigned normalised int	Microsoft S3TC format, 8 bits per pixel. Good for sharp alpha transitions.	DX10_BC2_UNORM	DX10_BC2_UNORM	0x82
BC 2	unsigned normalised int sRGB colour space	Microsoft S3TC format, 8 bits per pixel. Good for sharp alpha transitions.	DX10_BC2_UNORM_SRGB	DX10_BC2_UNORM_SRGB	0x83
BC 3	unsigned normalised int	Microsoft S3TC format, 8 bits per pixel. Good for smooth alpha	DX10_BC3_UNORM	DX10_BC3_UNORM	0x84

		transitions.			
BC 3	unsigned normalised int sRGB colour space	Microsoft S3TC format, 8 bits per pixel. Good for smooth alpha transitions.	DX10_BC3_UNORM_SRGB	DX10_BC3_UNORM_SRGB	0x85

6.2. OpenVG

Format	Description	Command Line Identifier	Identifer Enum	Enum Value
RGBX 8888 sRGB	32 bits per pixel, no alpha support, sRGB colour space	OVG_RGBX_8888_SRGB	ePT_VG_sRGBX_8888	0x90
RGBA 8888 sRGB	32 bits per pixel, alpha support, sRGB colour space	OVG_RGBA_8888_SRGB	ePT_VG_sRGBA_8888	0x91
RGBA 8888 sRGB PRE	32 bits per pixel, pre-multiplied alpha support, sRGB colour space	OVG_RGBA_8888_SRGB_PRE	ePT_VG_sRGBA_8888_PRE	0x92
RGB 565 sRGB	16 bits per pixel, no alpha support, sRGB colour space	OVG_RGB_565_SRGB	ePT_VG_sRGB_565	0x93
RGBA 5551 sRGB	16 bits per pixel, punch-through alpha support, sRGB colour space	OVG_RGBA_5551_SRGB	ePT_VG_sRGBA_5551	0x94
RGBA 4444 sRGB	16 bits per pixel, alpha support, sRGB colour space	OVG_RGBA_4444_SRGB	ePT_VG_sRGBA_4444	0x95
L 8 sRGB	Single channel 8 bits per pixel format, sRGB colour space	OVG_L_8_SRGB	ePT_VG_sL_8	0x96
RGBX 8888 lRGB	32 bits per pixel, no alpha support, lRGB colour space	OVG_RGBX_8888_LRGB	ePT_VG_lRGBX_8888	0x97
RGBA 8888 lRGB	32 bits per pixel, no alpha support, lRGB colour space	OVG_RGBA_8888_LRGB	ePT_VG_lRGBA_8888	0x98

RGBA 8888 lRGB PRE	32 bits per pixel, pre-multiplied alpha support, sRGB colour space	OVG_RGBA_8888_LRGB_PRE	ePT_VG_lRGBA_8888_PRE	0x99
L 8 lRGB	Single channel 8 bits per pixel format, lRGB colour space	OVG_L_8_LRGB	ePT_VG_lL_8	0x9A
A 8	Alpha texture 8 bits per channel	OVG_A_8	ePT_VG_A_8	0x9B
1 BPP	Single bit per pixel B&W texture	OVG_1_BPP	ePT_VG_BW_1	0x9C
XRGB 8888 sRGB	32 bits per pixel, no alpha support, sRGB colour space	OVG_XRGB_8888_SRGB	ePT_VG_sXRGB_8888	0x9D
ARGB 8888 sRGB	32 bits per pixel, alpha support, sRGB colour space	OVG_ARGB_8888_SRGB	ePT_VG_sARGB_8888	0x9E
ARGB 8888 sRGB PRE	32 bits per pixel, pre-multiplied alpha support, sRGB colour space	OVG_ARGB_8888_SRGB_PRE	ePT_VG_sARGB_8888_PRE	0x9F
ARGB 1555 sRGB	16 bits per pixel, punch-through alpha support, sRGB colour space	OVG_ARGB_1555_SRGB	ePT_VG_sARGB_1555	0x100
ARGB 4444 sRGB	16 bits per pixel, alpha support, sRGB colour space	OVG_ARGB_4444_SRGB	ePT_VG_sARGB_4444	0x101
XRGB 8888 lRGB	32 bits per pixel, no alpha support, lRGB colour space	OVG_XRGB_8888_LRGB	ePT_VG_lXRGB_8888	0x102
ARGB 8888 lRGB	32 bits per pixel, alpha support, lRGB colour space	OVG_ARGB_8888_LRGB	ePT_VG_lARGB_8888	0x103
ARGB 8888 lRGB PRE	32 bits per pixel, pre-multiplied alpha support, lRGB colour space	OVG_ARGB_8888_LRGB_PRE	ePT_VG_lARGB_8888_PRE	0x104
BGRX 8888 sRGB	32 bits per pixel, no alpha support, sRGB colour space	OVG_BGRX_8888_SRGB	ePT_VG_sBGRX_8888	0x105

BGRA 8888 sRGB	32 bits per pixel, alpha support, sRGB colour space	OVG_BGRA_8888_SRGB	ePT_VG_sBGRA_8888	0x106
BGRA 8888 sRGB PRE	32 bits per pixel, premultiplied alpha support, sRGB colour space	OVG_BGRA_8888_SRGB_PRE	ePT_VG_sBGRA_8888_PRE	0x107
BGR 565 sRGB	16 bits per pixel, no alpha support, sRGB colour space	OVG_BGR_565_SRGB	ePT_VG_sBGR_565	0x108
BGR 5551 sRGB	16 bits per pixel, punch-through alpha support, sRGB colour space	OVG_BGR_5551_SRGB	ePT_VG_sBGRA_5551	0x109
BGRA 4444 sRGB	16 bits per pixel, alpha support, sRGB colour space	OVG_BGRA_4444_SRGB	ePT_VG_sBGRA_4444	0x10A
BGRX 8888 lRGB	32 bits per pixel, no alpha support, lRGB colour space	OVG_BGRX_8888_LRGB	ePT_VG_lBGRX_8888	0x10B
BGRA 8888 lRGB	32 bits per pixel, alpha support, lRGB colour space	OVG_BGRA_8888_LRGB	ePT_VG_lBGRA_8888	0x10C
BGRA 8888 lRGB PRE	32 bits per pixel, pre-multiplied alpha support, lRGB colour space	OVG_BGRA_8888_LRGB_PRE	ePT_VG_lBGRA_8888_PRE	0x10D
XBGR 8888 sRGB	32 bits per pixel, no alpha support, sRGB colour space	OVG_XBGR_8888_SRGB	ePT_VG_sXBGR_8888	0x10E
ABGR 8888 sRGB	32 bits per pixel, alpha support, sRGB colour space	OVG_ABGR_8888_SRGB	ePT_VG_sABGR_8888	0x10F
ABGR 8888 sRGB PRE	32 bits per pixel, pre-multiplied alpha support, sRGB colour space	OVG_ABGR_8888_SRGB_PRE	ePT_VG_sABGR_8888_PRE	0x110
ABGR 1555 sRGB	16 bits per pixel, no alpha support, sRGB colour space	OVG_ABGR_1555_SRGB	ePT_VG_sABGR_1555	0x111
ABGR	16 bits per pixel,	OVG_ABGR_4444_SRGB	ePT_VG_sABGR_4444	0x112

4444 lRGB	alpha support, sRGB colour space			
XBGR 8888 lRGB	32 bits per pixel, no alpha support, lRGB colour space	OVG_XBGR_8888_LRGB	ePT_VG_lXBGR_8888	0x113
ABGR 8888 lRGB	32 bits per pixel, alpha support, lRGB colour space	OVG_ABGR_8888_LRGB	ePT_VG_lABGR_8888	0x114
ABGR 8888 lRGB PRE	32 bits per pixel, pre-multiplied alpha support, lRGB colour space	OVG_ABGR_8888_LRGB_PRE	ePT_VG_lABGR_8888_PRE	0x115