PVRTexLib

User Manual

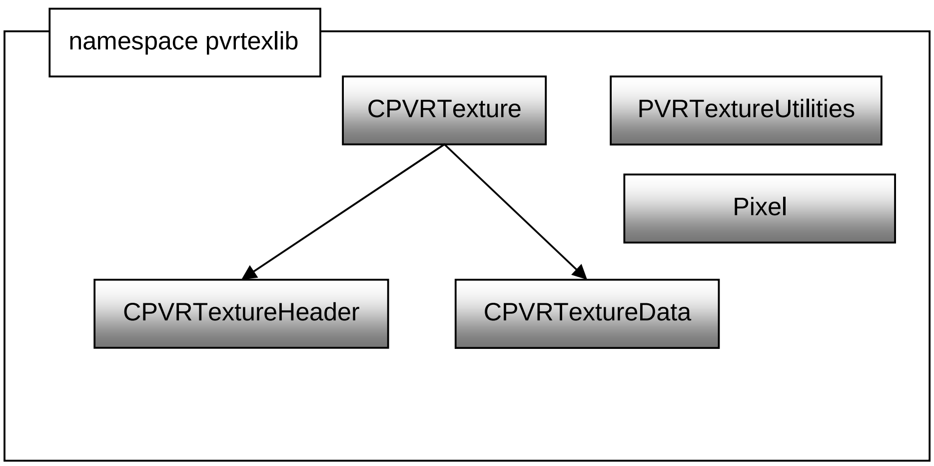
Filename	:	PVRTexLib.User Manual.mht
Version	:	1.3f (PowerVR SDK 2.03.23.1159)
Issue Date	:	28 Aug 2008
Author	:	PowerVR

Contents

1. Overview

PVRTexLib is a library for the management of PVR textures. It occupies the pvrtexlib namespace and provides the facility to:

- Load and save *.PVR texture files with data encoded in many texture formats.
- Decompress from any of the supported pixel formats to a standard set of pixel types.
- Process a texture in various ways when in one of these standard pixel types.
- Compress to any of the supported pixel formats from the same set of standard pixel types.
- Gather information about a texture loaded by the library.



1. PVRTexLib Overview

PVRTexLib provides container classes for PVR textures. For common functionality, it is recommended that developers avoid directly using the CPVRTextureHeader and CPVRTextureData classes and, instead use the CPVRTexture parent class, where possible.

An extended version of PVRTexLib is the backbone for the PVRTexTool applications and plug-ins in the Imagination Technologies, PowerVR SDK.

1.1. Accessing the Library

PVRTexLib is provided as the file PVRTexLib.dll on Windows and as libPVRTexLib.a on Linux. Also present should be a folder with the header files that are described later in this document. To access the library you will have to include these files and link to the library file. On Windows, the PVRTexLib.dll will need to be present for your application to run successfully. Also on Windows you will need to define _WINDLL_IMPORT somewhere in your project in order to access the functions and classes in the dll successfully.

1.2. Standard Pixel Types

PVRTexLib doesn't directly process all the pixel formats it supports but, instead, uses a set of standard pixel types for most operations. The raw unencoded formats for pixel textures split the formats available in the PVRTexLib into four precisions. These precisions are defined in the PVRTexLibGlobals.h file:

```
        enum PVR_PRECMODE
        {       // precision modes - correspond to standard pixel types
                ePREC_INT8=0,
                ePREC_INT16,
                ePREC_INT32,
                ePREC_FLOAT,
        } ;

        const PixelType       eInt8StandardPixelType        = DX10_R8G8B8A8_UNORM,
                              eInt16StandardPixelType       = D3D_ABGR_16161616,
                              eInt32StandardPixelType       = DX10_R32G32B32A32_UINT,
                              eFloatStandardPixelType       = D3D_ABGR_32323232F;
```

This means that there are four standard pixel types defined for working with each of these precisions.

- eInt8StandardPixelType
  Uncompressed 32-bit per pixel ABGR data. Produced by decompressing any pixel format of PrecMode: ePREC_INT8, for example. The uint8 typedef can represent each separate channel of data for a pixel and a whole pixel may be addressed using the Pixel<uint8> struct. *Note: the red and blue channel positions for this type have changed from previous versions of PVRTexLib.*
- eInt16StandardPixelType

  Uncompressed 64-bit per pixel ABGR data. Produced by decompressing any pixel format of PrecMode: ePREC_INT16, for example. The uint16 typedef can represent each separate channel of data for a pixel and a whole pixel may be addressed using the Pixel<uint16> struct.
- eInt32StandardPixelType

  Uncompressed 128-bit per pixel ABGR data. Produced by decompressing any pixel format of PrecMode: ePREC_INT32, for example. The uint32 typedef can represent each separate channel of data for a pixel and a whole pixel may be addressed using the Pixel<uint32> struct.
- eFloatStandardPixelType

  Uncompressed 128-bit per pixel ABGR data. Will be the result of decompressing any pixel format of PrecMode: ePREC_FLOAT. The float32 typedef can represent each separate channel of data for a pixel and a whole pixel may be addressed using the Pixel<float32> struct.

1.3. Exception Handling

Functions of PVRTexLib throw PVRException instances on failure using the PVRTHROW macro.

```
        class PVRException : public std::exception
        {
        public:
                PVRException(char* what)throw(): m_what(what){}
                char *  what()                                    {return m_what;}
                ~PVRException() throw(){}
        private:
                 char*  m_what;
        };

        #define PVRTHROW(A) {PVRException myException(A); throw(myException);}
        #define PVRCATCH(A) catch(PVRException& A)
```

To take advantage of this functionality, place the code you want to catch exceptions from inside a normal try block and catch using the PVRCATCH macro provided or catch(PVRException& A). The what() function should give some explanation towards the failure.

1.3.1. Example of Exception Handling

```
	CPVRTexture sDecompressedTexture, sEncodedTexture;
	try
	{
		PVRTextureUtilities::ptr()-
>DecompressPVR(sEncodedTexture,sDecompressedTexture);
	}
	PVRCATCH(myException)
	{
		fprintf(stderr,"Could not decompress texture:\n%s\n",myException.what());
	}
```

If something is failing when using PVRTexLib it may be informative to examine the exception that is thrown, as in this example.

2. PVRTexLib Reference

Further documentation than may be present here is available in the header files.

2.1. PVRTexLib.h - PVRTextureUtilities

This is a singleton class that has functions to work with the other classes in PVRTexLib. The functions provided in this class allow manipulation of textures and come in two forms: a form that works with CPVRTexture instances and ones that require separate instances of `CPVRTextureHeader` and `CPVRTextureData`. There is no difference in these functions other than this interface.

2.1.1. CompressPVR()

```
void CompressPVR(CPVRTexture& sDecompressedTexture,
               CPVRTexture& sCompressedTexture, const int nMode);
void CompressPVR(CPVRTextureHeader    &sSourceHeader,
               CPVRTextureData        &sSourceData,
               CPVRTextureHeader      &sCompHeader,
               CPVRTextureData        &sCompData,
               const int              nMode=0);
```

Use these functions to compress a texture in a standard format to one of the other formats.

`sDecompressedTexture` should be the texture you wish to compress, `sCompressedTexture` should be a texture with an identical header, but with the pixel type set to the desired compressed pixel type. To achieve this use the `setPixelType(PixelType)` methods in `CPVRTexture` or `CPVRTextureHeader`..

The nMode variable is only relevant for ETC compression. Suitable values are:

- 0 – Fast
- 1 – Medium
- 2 – Slow
- 3 – Fast Perceptual
- 4 – Medium Perceptual
- 5 – Slow Perceptual

2.1.2. DecompressPVR()

```
void DecompressPVR(CPVRTexture& sCompressedTexture,
               CPVRTexture& sDecompressedTexture);
void DecompressPVR(    CPVRTextureHeader      & sCompressedHeader,
               const CPVRTextureData          & sCompressedData,
               CPVRTextureHeader              & sDecompressedHeader,
               CPVRTextureData                & sDecompressedData);
```

Use these functions to decompress textures into a standard format for further processing.

`sCompressedTexture` should be the texture to be decompressed, `sDecompressedTexture` is an instance that will be the destination for the decompressed data.

2.1.3. ProcessRawPVR()

```
bool ProcessRawPVR(    CPVRTexture&             sInputTexture,
                       CPVRTexture&             sOutputTexture,
                       const bool               bDoBleeding,
                       const float              fBleedRed=0.0f,
                       const float              fBleedGreen=0.0f,
                       const float              fBleedBlue=0.0f,
                       PVR_RESIZE               eResizeMode = eRESIZE_BICUBIC);
bool ProcessRawPVR(CPVRTextureHeader&           sInputHeader,
                       CPVRTextureData&         sInputData,
                       CPVRTextureHeader&       sOutputHeader,
                       const bool               bDoBleeding,
                       const float              fBleedRed=0.0f,
                       const float              fBleedGreen=0.0f,
                       const float              fBleedBlue=0.0f,
                       PVR_RESIZE               eResizeMode = eRESIZE_BICUBIC);
```

This function allows some processing of a texture. Textures must be in a standard format in order to be processed. `sInputTexture` should be such a texture and `sOutputTexture` is a texture where the header has been set to the desired result of the processing. Please refer to the example later in this document for an illustration of how to use this function.

Operations possible with `ProcessRawPVR` include:

- Resize a texture by specifying a different width and/or height in the `sOutputTexture` (or `sOutputHeader`). The default resizing algorithm is bicubic – to choose others set `eResizeMode` to one of: `eRESIZE_NEAREST`, `eRESIZE_BILINEAR`.
- Encode a border on to the texture, similar to the border functions available in PVRTexTool. Use the `setBorder(true)` function of the `sOutputTexture` (or `sOutputHeader`) before calling `ProcessRawPVR()` to add the border.
- Generate a normal map from the red channel of the passed texture. To do this use `setNormalMap(value)` with a non-zero value, in your output texture (or header) before processing.
- Generate MIP-maps. Set the required number of MIP-map levels in the output texture/header using `setMipMapCount(number)`. Setting the number of MIP-levels to 0 will cause no MIP-maps to be generated: i.e. only the top image will be present in the processed texture. Positive numbers will cause MIP-levels in addition to this to be generated.
- 'Bleed' a chosen colour in the texture in the same way as the operation in PVRTexTool to help the appearance of the texture with blending. Pass the red, green and blue values for the colour to be bled into using the `fBleedRed`, `fBleedGreen` and `fBleedBlue` parameters and pass true for `bDoBleeding`. The colour channel values should be in the range 0.0f to 1.0f for `ePREC_FLOAT` textures, 0.0f to 255.0f for `ePREC_INT8`, 0.0f to 65535.0f for `ePREC_INT16`. Bleeding is currently unavailable for `ePREC_INT32`.
- Generate coloured MIP-map levels for debugging purposes. Process for MIP-map levels as normal (or use a texture that contains these), but also use `setFalseMips(true)` in the output texture/header.

2.2. CPVRTexture.h - CPVRTexture

A `CPVRTexture` class represents an entire texture in memory, including all descriptive info and holding all texture surface data. Further documentation is in the header file. For most operations, this class may be used as the atomic element of PVRTexLib without keeping any separate instances of `CPVRTextureHeader` or `CPVRTextureData`.

There are various accessor functions for gaining information from a texture and setting values for processing a texture.

2.3. CPVRTextureHeader.h - CPVRTextureHeader

A class holding the description of a PVR texture – roughly analogous to the header associated with a .PVR file.

2.4. CPVRTextureData - CPVRTextureData

A container class for the actual pixel data in a PVR texture.

2.5. PVRTexLibGlobals.h

Holds the macros, enums and constants used by PVRTexLib. Of particular interest should be the standard types, precision modes, PixelType enum and exception macros.

2.6. Pixel.h

Defines some structs useful for manipulating pixels held in various pixel types.

2.7. singleton.h

This is the singleton class template that the PVRTextureUtilities class uses.

3. Code Examples

Exception handling has been removed from these examples to aid in clarity and brevity.

3.1. Read and Decompress .PVR file

In this example, an existing .PVR file is read and decompressed, possibly for later processing, access to the image data or re-encoding.

The variable `strFilePath` is a string containing a resolved file path.

```
#include "PVRTexLib.h"
using namespace pvrtexlib;

...
               // get the utilities instance
               PVRTextureUtilities *PVRU = PVRTextureUtilities::ptr();

               // open and read a pvr texture from the file location specified by strFilePath
               CPVRTexture sOriginalTexture(strFilePath);

               // make an empty texture to decompress into
               CPVRTexture sDecompressedTexture;

               // decompress the texture
               PVRU->DecompressPVR(sOriginalTexture,sDecompressedTexture);

               // convert to uint8 precision
               sDecompressedTexture.convertToPrecMode(ePREC_INT8);
```

All being correct, the `CPVRTexture` instance `sDecompressedTexture` now contains a 32 bits per pixel decompressed version of the texture originally accessed from the .PVR file.

This code may also be used to load .dds and .ngt files.

3.2. Given Raw Pixel Data, Pre-process, Encode and Save

In this example, A section of raw pixel data in 32 bit per pixel, uncompressed format of an image 256 pixels by 256 pixels is converted into a normal map of the same dimensions with full MIP-map chain, encoded into PVRTC 4 bits per pixel and then saved to a file.

The variable strFilePath is a PVRT::string containing a resolved file path. pPixelData points to the raw data.

```
#include "PVRTexLib.h"
using namespace pvrtexlib;

...

        // get the utilities instance
        PVRTextureUtilities PVRU = PVRTextureUtilities::ptr();

        // make a CPVRTexture instance with data passed
        CPVRTexture sOriginalTexture( 256,                  // u32Width,
                                      256,                  // u32Height,
                                      0,                    // u32MipMapCount,
                                      1,                    // u32NumSurfaces,
                                      false,                // bBorder,
                                      false,                // bTwiddled,
                                      false,                // bCubeMap,
                                      false,                // bVolume,
                                      false,                // bFalseMips,
                                      true,                 // bHasAlpha,
                                      MGLPT_ARGB_8888,      // ePixelType,
                                      0.0f,                 // fNormalMap,
                                      pPixelData            // pPixelData
                                      );

        // make an empty texture for the destination of the preprocessing
        // copying the header settings
        CPVRTexture sProcessTexture(sOriginalTexture.getHeader());

        // specify desired mip map levels
        sProcessTexture.setMipMapCount(8);

        // specify desired normal map height factor
        sProcessTexture.setNormalMap(5.0f);

        // preprocess the texture; creates MIP-levels and calculates normal map
        PVRU->ProcessRawPVR(sOriginalTexture,sProcessTexture,0,false);

        // create texture to encode to
        CPVRTexture sCompressedTexture(sProcessTexture.getHeader());

        // set required encoded pixel type
        sCompressedTexture.setPixelType(OGL_PVRTC4);

        // encode texture
        PVRU->CompressPVR(sOriginalTexture, sCompressedTexture, 0);

        // write to file specified
        sCompressedTexture.writeToFile(strFilePath);
```

After this code a .PVR file containing a normal map and 8 MIP-levels should be at the location specified by strFilePath.

4. Pixel Format Reference

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

Please note that greyed out formats, whilst present in the PixelType enum, are not supported by PVRTexLib 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.	DXT4	`D3D_DXT4`	0x23

	Alpha is considered premultiplied.			
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
	Ericsson Texture Compression, 4 bits per pixel with explicit		`ETC_RGBA_EXPLICIT`	0x37

	alpha like DXT3.			
	Ericsson Texture Compression, 4 bits per pixel with interpolated alpha like DXT5.		`ETC_RGBA_INTERPOLATED`	0x38
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

4.1. 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
	float			DX10_R11G11B10_FLOAT	0x60
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 space		DX10_R8G8B8A8_UNORM_SRGB	DX10_R8G8B8A8_UNORM_SRGB	0x62
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
				DX10_R9G9B9E5_SHAREDEXP	0x7D
	unsigned normalised int			DX10_R8G8_B8G8_UNORM	0x7E
	unsigned normalised int			DX10_G8R8_G8B8_UNORM	0x7F

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 transitions.	DX10_BC3_UNORM	DX10_BC3_UNORM	0x84
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
BC 4	unsigned normalised int			DX10_BC4_UNORM	0x86
BC 4	signed normalised int			DX10_BC4_SNORM	0x87
BC 5	unsigned normalised int			DX10_BC5_UNORM	0x88

BC 5	signed normalised int			DX10_BC5_SNORM	0x89

4.2. 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 4444 lRGB	16 bits per pixel, alpha support, sRGB colour space	`OVG_ABGR_4444_SRGB`	`ePT_VG_sABGR_4444`	0x112
XBGR	32 bits per pixel,	`OVG_XBGR_8888_LRGB`	`ePT_VG_lXBGR_8888`	0x113

8888 lRGB	no alpha support, lRGB colour space			
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