Texture Pipeline User's Guide

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

About This Document 3
Purpose 3
Typographic Conventions 3
Related Documentation and Other Resources 3
1 Introduction 4
GXT Format 4
2 Using the GXT Conversion Tool 5
Command-line Usage 5
Options 5
Input Formats 6
Examples 7
3 Using the PVRT Compressor (psp2pvrt) 8
Input Formats 8
Command-line Usage 8
Compression Options 9
Examples 9
4 DLL Interfaces 10
Files 10
Sample Code 10
Custom Output Formats 12
Multiple Textures/Palettes 12
Reverting GXT Data 12
5 DCC Plugins 14
Autodesk Maya 14
Autodesk 3ds Max 14
Adobe Photoshop 15
Plugin Source Code 16

About This Document

Purpose

This document describes the tools provided as part of the SDK for producing files in the GXT texture format. The purpose of the texture pipeline is to provide a minimal offline pipeline to prepare textures in a runtime-ready format that matches hardware constraints, especially memory alignment constraints.

Typographic Conventions

The typographic conventions used in this guide are explained in this section.

Text

- Names of keyboard functions or keys are formatted in a bold serif font. For example, **Ctrl**, **Delete**, **F9**.
- File names, source code, and command-line text are formatted in a fixed-width font. For example:

```
host_tools\bin\psp2gxt.exe
```

Hyperlinks

Hyperlinks (underlined and in blue) are available to help you to navigate around the document. To return to where you clicked a hyperlink, select **View** > **Toolbars** > **More Tools** from the Adobe Reader main menu, and then enable the **Previous View** and **Next View** buttons.

Related Documentation and Other Resources

Errata

You can find any updates or amendments to this guide in the release notes that accompany the SDK release packages.

Sample Code

Source code for the tools described in this document is located in:

```
host_tools\graphics\src\sce_texture\
```

Related Documents

For further information about the GXT format, refer to the *libgxt Overview*.

For detailed descriptions of the GPU data layout for each texture format and type, refer to the *GPU User's Guide*.

1 Introduction

GXT Format

GXT (**GX**m **T**exture format) is a file format for storing texture data used by libgxm. The main purpose of GXT is to provide a format that can be used directly in the runtime without any additional formatting of the data.

Features of the GXT format:

- Texture data is in a memory ready format, properly aligned.
- Supports multiple textures in one file.
- Supports compressed, swizzled, and linear texture layouts.
- Supports mipmaps.
- Supports palettized textures.

Tools are provided to convert from existing image formats to GXT. Currently supported input formats are `.dds`, `.pvr`, and `.tga`. For a more detailed description, see Chapter 2, Using the GXT Conversion Tool.

GXT files have the extension "`.gxt`".

For further information about the GXT format, refer to the *libgxt Overview*.

Source code for the tools described in this document is located in:

```
host_tools\graphics\src\sce_texture\
```

2 Using the GXT Conversion Tool

A command-line utility is supplied for converting `.dds`, `.pvr` and `.tga` files to GXT format. The executable is located in:

```
host_tools\bin\psp2gxt.exe
```

Command-line Usage

The basic syntax to convert one or more textures to a single `.gxt` file is as follows:

```
psp2gxt -i <input file(s)> -o output.gxt [-p4 <P4 palette file(s)>]
[-p8 <P8 palette file(s)>] [options]
```

The optional arguments `-p4` and `-p8` allow the user to define the palettes included in the `.gxt` file, by specifying image files containing 16 and 256 color palettes, respectively. The image data in such palette files is ignored (unless the palette files are also passed as arguments to `-i`). The order in which these files are passed as argument parameters prescribes the order in which the corresponding palettes can be found in the `.gxt` data.

By default, textures using the same palette will refer to a single array entry (in other words, palette data is shared within a given `.gxt` file). If any of the input texture files uses a different palette, this palette will be included as an additional entry in the `.gxt` palette arrays, appearing after any palette entries specified by the `-p4` or `-p8` options.

Options

Table 1 describes the options that can be used with the conversion tool.

Table 1 GXT Conversion Tool Options

Option	Description
`-l / -linear`	Force linear conversion. For all textures that have power of 2 dimensions, swizzle conversion is performed by default.
`-v / -verbose`	Show texture information.
`-rp / -replicatePalettes`	Create a separate palette for each indexed input texture, even if they use the same palette as other inputs.

Note: Swizzle conversion is always performed for block compressed textures. For block compressed textures which have dimensions that are not powers of 2, additional padding will be added to the output so that the dimensions of the memory layout are powers of 2.

Input Formats

Currently, the tool supports `.dds`, `.pvr`, and `.tga` files as inputs.

Table 2 lists the supported input formats for each of these file types, and their corresponding input and output formats.

Table 2 Supported .dds, .pvr, and .tga Formats

Input File	Input Format	Output Format (in .gxt)
.dds	A8	SCE_GXM_TEXTURE_FORMAT_U8_R000
	L8	SCE_GXM_TEXTURE_FORMAT_U8_1RRR
	A1R5G5B5	SCE_GXM_TEXTURE_FORMAT_U1U5U5U5_ARGB
	X1R5G5B5	SCE_GXM_TEXTURE_FORMAT_X1U5U5U5_1RGB
	A4R4G4B4	SCE_GXM_TEXTURE_FORMAT_U4U4U4U4_ARGB
	X4R4G4B4	SCE_GXM_TEXTURE_FORMAT_X4U4U4U4_1RGB
	L16	SCE_GXM_TEXTURE_FORMAT_U16_1RRR
	R5G6B5	SCE_GXM_TEXTURE_FORMAT_U5U6U5_RGB
	A8R3G3B2	SCE_GXM_TEXTURE_FORMAT_U8U3U3U2_ARGB
	A8B8G8R8	SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ABGR
	A8R8G8B8	SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ARGB
	X8B8G8R8	SCE_GXM_TEXTURE_FORMAT_X8U8U8U8_1BGR
	X8R8G8B8	SCE_GXM_TEXTURE_FORMAT_X8U8U8U8_1RGB
	R8G8B8	SCE_GXM_TEXTURE_FORMAT_U8U8U8_RGB
	G16R16	SCE_GXM_TEXTURE_FORMAT_U16U16_00GR
	R16F	SCE_GXM_TEXTURE_FORMAT_F16_000R
	G16R16F	SCE_GXM_TEXTURE_FORMAT_F16F16_00GR
	R32F	SCE_GXM_TEXTURE_FORMAT_F32M_000R
	A8L8	SCE_GXM_TEXTURE_FORMAT_U8U8_GRRR
	A2R10G10B10	SCE_GXM_TEXTURE_FORMAT_U2U10U10U10_ARGB
	A2B10G10R10	SCE_GXM_TEXTURE_FORMAT_U2U10U10U10_ABGR
	AF16BF16GF16RF16	SCE_GXM_TEXTURE_FORMAT_F16F16F16F16_ABGR
	UBC1 (DXT1)	SCE_GXM_TEXTURE_FORMAT_UBC1_ABGR
	UBC2 (DXT3)	SCE_GXM_TEXTURE_FORMAT_UBC2_ABGR
	UBC3 (DXT5)	SCE_GXM_TEXTURE_FORMAT_UBC3_ABGR
	SBC4	SCE_GXM_TEXTURE_FORMAT_SBC4_000R
	UBC4 (ATI1)	SCE_GXM_TEXTURE_FORMAT_UBC4_000R
	SBC5	SCE_GXM_TEXTURE_FORMAT_SBC5_00GR
	UBC5 (ATI2)	SCE_GXM_TEXTURE_FORMAT_UBC5_00RG
	UBC5	SCE_GXM_TEXTURE_FORMAT_UBC5_00GR
.pvr	PVRT I 2bpp	SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_ABGR
	PVRT I 4bpp	SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_ABGR
	PVRT II 2bpp	SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR
	PVRT II 4bpp	SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR
.tga	A8B8G8R8	SCE_GXM_TEXTURE_FORMAT_U8U8U8U8_ABGR
	B8G8R8	SCE_GXM_TEXTURE_FORMAT_U8U8U8_BGR
	Indexed - 16 colors	SCE_GXM_TEXTURE_FORMAT_P4_ABGR
	Indexed - 256 colors	SCE_GXM_TEXTURE_FORMAT_P8_ABGR

Examples

The following sample converts `input.dds` to `output.gxt`:

```
psp2gxt -i input.dds -o output.gxt
```

The following sample converts `input0.dds`, `input1.dds`, and `input2.pvr` to `output.gxt`, forcing all textures to be encoded linearly (this is not optimal):

```
psp2gxt -i input0.dds input1.dds input2.pvr -o output.gxt -l
```

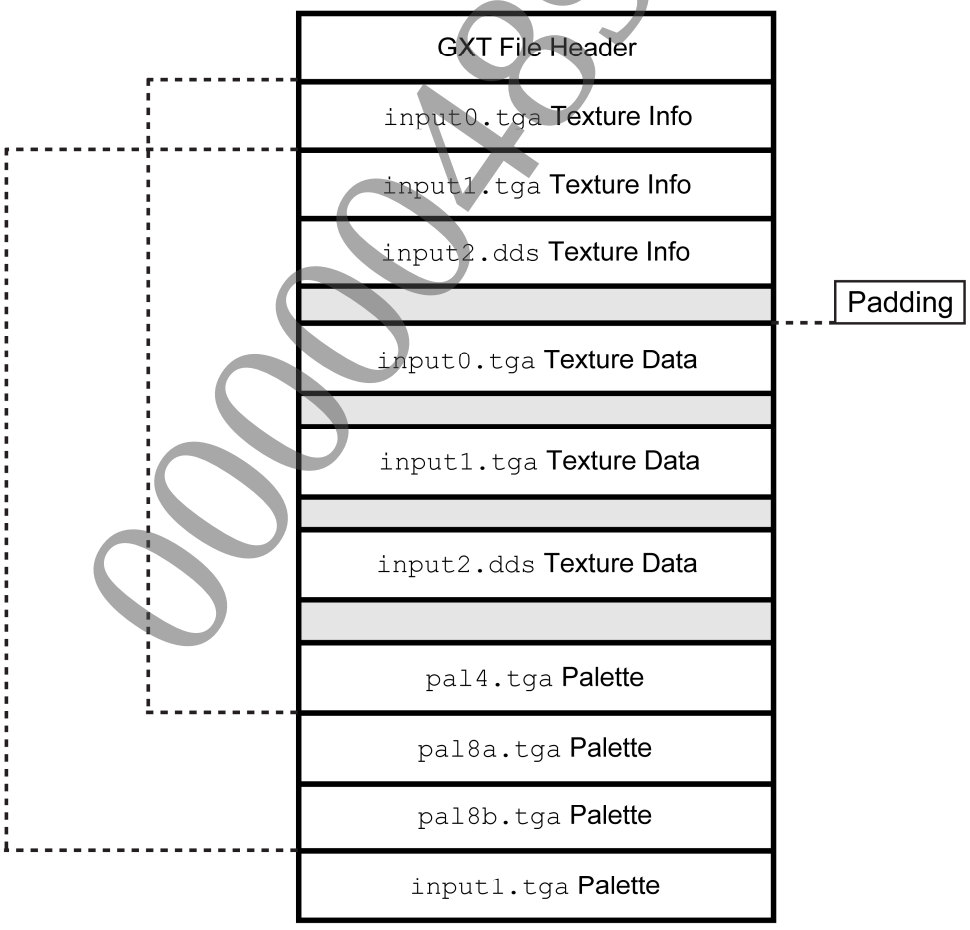
> **Note:** The "flip" flag in the `.pvr` header is ignored by this tool; therefore, textures converted from `.pvr` files that have this flag set may appear mirrored vertically. The PVRT compressor tool (described in the next chapter) does not set this flag.

The sample below creates a `.gxt` file containing one 16-color palette extracted from `pal4.tga`, and two 256-color palettes extracted from `pal8a.tga` and `pal8b.tga`; the `.gxt` file also includes texture image data converted from files `input0.tga`, `input1.tga` and `input2.dds`:

```
psp2gxt -p4 pal4.tga -p8 pal8a.tga pal8b.tga -i input0.tga input1.tga input2.pvr
-o output.gxt
```

If any of these input files uses a palette that is not included in the arguments to `-p4` or `-p8`, it will be automatically added to the file. Figure 1 shows the layout of the `.gxt` file resulting from this command if (a) `input0.tga` used the same palette as `pal8a.tga` and (b) `input1.tga` used a 256-entry palette that does not match any of the `-p8` parameters.

Figure 1 Example .gxt Layout



3 Using the PVRT Compressor (psp2pvrt)

This chapter describes how to use the PVRT compressor tool to output `.pvr` files in one of the four compression format variants shown in Table 3.

Table 3 PVRT Compression Formats

Format		Compression Ratio (vs. 24 bits RGB)	Compression Ratio (vs. 32 bits RGBA)
PVRT-I	2 bits per pixel	12:1	16:1
	4 bits per pixel	6:1	8:1
PVRT-II	2 bits per pixel	12:1	16:1
	4 bits per pixel	6:1	8:1

The executable is located in:

```
host_tools\bin\psp2pvrt.exe
```

There is an additional version of the tool for 64-bit host operating systems located in:

```
host_tools\bin\psp2pvrt64.exe
```

which may provide performance benefits in some cases.

The PVRT-II format introduces new encoding modes to avoid some of the most common compression artifacts of its predecessor; therefore if using PVRT compression, it is advisable always to use PVRT-II rather than PVRT-I. For more details, refer to Appendix B of the *GPU User's Guide*.

Note: `.pvr` files are not in a GPU-ready format and should be converted to `.gxt` using the psp2gxt tool.

Input Formats

Currently, the PVRT compressor tool takes as input `.dds` files, in any uncompressed linear format, and `.tga` files, in uncompressed, RLE-compressed or indexed formats.

`.dds` files can include mipmap chains of any length as well as cubemaps. However, all six faces must be provided for any cubemap inputs.

Internally, any input file will be converted to either `B8G8R8` or `A8B8G8R8` formats, depending on whether the input texture has any non-opaque alpha information. Components will be taken in the same order as they appear in the source format (for example, luminance in `L8` formats will be mapped to the red component). All other components for which there is no matching source information will be set to 0 (this produces better compression results). Floating-point components will be clamped to byte range.

Note: In general, compressing 1 or 2-component textures with PVRT may be counterproductive, because 3 is the fewest number of components supported by the format.

Command-line Usage

The basic syntax required to compress a texture is as follows:

```
psp2pvrt -i input.dds -o output.pvr [options]
```

Compression Options

Table 4 describes the options that can be used with the psp2pvrt tool.

Table 4 Compression Options

Option	Description
`-d / -decomp`	Specifies a `.dds` file in which to store an uncompressed copy of the compression results.
`-f / -format`	Defines the output compression format : • `SCE_GXM_TEXTURE_FORMAT_PVRT2BPP_ABGR: PVRT-I, 2BPP` • `SCE_GXM_TEXTURE_FORMAT_PVRT4BPP_ABGR: PVRT-I, 4BPP` • `SCE_GXM_TEXTURE_FORMAT_PVRTII2BPP_ABGR: PVRT-II, 2BPP` • `SCE_GXM_TEXTURE_FORMAT_PVRTII4BPP_ABGR: PVRT-II, 4BPP` By default, this option is set to `PVRT-II, 4BPP`.
`-t / -tech`	Specifies the compression technique to use: • `RangeFit`: Finds the bounding values by computing the image range along the axes of maximum variation, without taking into account the effects of discretization. • `ClusterFit`: Applies a cluster fitting algorithm to obtain the optimal bounding values given a quantization set. Cluster fitting is much more computationally expensive than range fitting and contributes very little error reduction; therefore, by default this option is set to `RangeFit`.
`-b2 / -bpp2`	Specifies the block modulation mode to use in 2BPP compression formats: • `1`: 1 bit per pixel modulation • `2`: 2 bits per pixel modulation • `mixed`: adaptive choice based on local error RMS By default, this option is set to 2 bits per pixel. For more details, see *Figure 34* and *Figure 35* in the *GPU User's Guide*.
`-m / -mips`	Specifies the mipmap levels generated from the input texture. If the input texture includes only the base level, setting this option to -1 will generate a full mipmap chain. If the input `.dds` includes a mipmap chain, whether partial or full, this option is ignored.
`-it / -iters`	Sets the number of iterations to be used by the global optimization algorithm refining the results of the compression. This increases the results' quality, and also the compression time. By default, this option is set to 4.
`-th / -threads`	Sets the number of threads to be used by the compressor. By default, this option is set to -1, which creates 1 thread per processor core. This is optimal when running as a single process. **Note:** The results of the compression may be numerically different when using different thread counts; even though the differences are negligible, this will result in `.pvr` files that are not binarily identical.

Examples

The following sample compresses `input.dds` and all its possible mipmap levels as `output.pvr`. It uses the PVRT-I 2BPP compression format with 1 bit per pixel modulation, and refines the result for two iterations.

```
psp2pvrt -i input.dds -o output.pvr -f SCE_GXM_BASE_TEXTURE_FORMAT_PVRT2BPP
-b2 mixed -m -1 -it 2
```

The following sample compresses `input.dds` as `output.pvr` using the PVRT-II 4BPP format and the cluster fitting compression technique. It runs the compression in parallel across four threads.

```
psp2pvrt -i input.dds -o output.pvr -f SCE_GXM_BASE_TEXTURE_FORMAT_PVRTII4BPP
-th 4 -t ClusterFit
```

4 DLL Interfaces

Dynamic Link Library (DLL) versions of the texture tools are provided.

Files

The DLL versions of the tools are provided as the following sets of files (Table 5).

Table 5 DLL Interface Files

File Name with Relative Path	Description

`sdk\host_tools\include\pvrt_compression.h`
`sdk\host_tools\lib\pvrt_compression.dll`
`sdk\host_tools\lib\pvrt_compression.lib`
	PVRT Compression header file, DLL and import library for 32-bit Windows.

`sdk\host_tools\include\pvrt_compression.h`
`sdk\host_tools\lib.x64\pvrt_compression.dll`
`sdk\host_tools\lib.x64\pvrt_compression.lib`
	PVRT Compression header file, DLL and import library for 64-bit Windows.

`sdk\host_tools\include\gxt_conversion.h`
`sdk\host_tools\lib\gxt_conversion.dll`
`sdk\host_tools\lib\gxt_conversion.lib`
	GXT Conversion header file, DLL and import library for 32-bit Windows.

`sdk\host_tools\include\gxt_conversion.h`
`sdk\host_tools\lib.x64\gxt_conversion.dll`
`sdk\host_tools\lib.x64\gxt_conversion.lib`
	GXT Conversion header file, DLL and import library for 64-bit Windows.

Detailed descriptions of the DLL interfaces can be found in the relevant header files. The DLL interfaces are essentially transcriptions of the GXT texture tool commands and command line options to C code.

The core functions `sceTexturePvrtCompression()` and `sceTextureGxtConversion()` expect as input a complete in-memory copy of the input file. This input file is a `.dds` file for `sceTexturePvrtCompression()` and either a `.dds`, `.tga`, or `.pvr` file for `sceTextureGxtConversion()`.

The restrictions on these input files are the same as those for the executable versions, described in Chapter 2, Using the GXT Conversion Tool and Chapter 3, Using the PVRT Compressor (psp2pvrt).

Utility functions are provided that initialize the input options to suitable default values. The user can then customize these options before passing them to the core functions.

The core functions output a buffer and the size of that buffer in bytes. The user should copy the result from this buffer (to either another memory location or disk) and then free the resources allocated in the compression or conversion function by calling `sceTexturePvrtCompressionFinalize()` or `sceTextureGxtConversionFinalize()`, respectively.

All of the compression, conversion, and finalization functions accept user-defined logging and memory allocation handlers. The user can also specify `NULL` for these handlers, in which case suitable defaults will be used.

Sample Code

```
void*   textureBuffer = NULL;
size_t  textureBufferSizeInBytes = 0;

/* Read a complete .dds file from disk. */
UserAllocAndReadFile(inputFilename, &textureBuffer,
&textureBufferSizeInBytes);
```

```
/* Initialize the compression options with the defaults for PVRT-II @ 4bpp. */
SceTexturePvrtCompressionOptions myPvrtCompressionOptions;

SCE_DBG_ASSERT(SCE_OK == sceTextureInitializeDefaultPvrtCompressionOptions(
                         &myPvrtCompressionOptions,
                         SCE_GXM_TEXTURE_BASE_FORMAT_PVRTII4BPP));

/* Set up an allocator struct to be used in all of the texture routines. */
SceMemoryAllocator userAllocator;
userAllocator.Allocate = UserAllocateFunction;
userAllocator.Deallocate = UserDeallocateFunction;

void*   pvrtData = NULL;
size_t  pvrtDataSizeInBytes = 0;


/* Perform the actual PVRT compression. */
       SCE_DBG_ASSERT(SCE_OK == sceTexturePvrtCompression(
                                                &pvrtData,
                                                &pvrtDataSizeInBytes,
                                                textureBuffer,
                                                textureBufferSizeInBytes,
                                                &myPvrtCompressionOptions,
                                                &UserLogger,
                                                &userAllocator));



SceTextureGxtConversionOptions myGxtConversionOptions;


/* Initialize GXT conversion options to the defaults. */

SCE_DBG_ASSERT(SCE_OK == sceTextureInitializeDefaultGxtConversionOptions(
                         &myGxtConversionOptions,
                         pvrtData,
                         pvrtDataSizeInBytes));

void* gxtData = NULL;
size_t gxtDataSizeInBytes = 0;

/* Perform the actual conversion to GXT.*/
SCE_DBG_ASSERT(SCE_OK == sceTextureGxtConversion(
                                                &gxtData,
                                                &gxtDataSizeInBytes,
                                                &myGxtConversionOptions,
                                                &UserLogger,
                                                &userAllocator));

/*  A user-supplied function to write the file to disk. */
UserWriteFile(outputFilename, gxtData, gxtDataSizeInBytes);

/*  Release the resources allocated during compression and conversion. */
sceTexturePvrtCompressionFinalize(&UserLogger, &userAllocator, pvrtData,
                                  pvrtDataSizeInBytes);
sceTextureGxtConversionFinalize(&UserLogger, &userAllocator, gxtData,
                                gxtDataSizeInBytes);
```

Custom Output Formats

By default, the output format in the GXT conversion options will be set to `SCE_TEXTURE_GXT_DEFAULT_OUTPUT_FORMAT`. This tells the conversion routine to choose a default output format based on the input format, as shown in Table 2.

To set a custom output format, make sure that the base format matches the base format of the default format. For example, if the input file is in `G16R16` format, a valid output format is `SCE_GXM_TEXTURE_FORMAT_U16U16_GRGR`, but `SCE_GXM_TEXTURE_FORMAT_U8U8_GRGR` is not a valid output format.

Multiple Textures/Palettes

The function `sceTextureGxtConversionEx()` is used to convert multiple textures/palettes to a single GXT file. An array of `SceTextureGxtConversionOptions` structures is passed along with values for the number of textures, P4 palettes, and P8 palettes respectively. The array should be ordered so that the texture options come first, followed by the P4 palette options, followed by the P8 palette options.

For example:

```
/* Perform the actual conversion to GXT.*/
SCE_DBG_ASSERT(SCE_OK == sceTextureGxtConversionEx(
    &gxtData,
    &gxtDataSizeInBytes,
    myGxtConversionOptionsArray,    // An array of 6 options
    2,                              // 2 texture inputs
    1,                              // 1 P4 palette input
    3,                              // 3 P8 palette inputs
    SCE_TEXTURE_SHARE_PALETTES,     // Share palettes where possible
    &UserLogger,
    &userAllocator));
```

Note: Palette inputs must come from `.tga` files.

Reverting GXT Data

The function `sceTextureGxtRevert()` is used to revert a single texture from a GXT file to either `.dds`, `.pvr`, or `.tga` format. The container type of the output buffer is determined by the format of the input texture and can be queried by calling `sceTextureQueryBufferType()`.

For example:

```
/* Revert a GXT texture.*/
SCE_DBG_ASSERT(SCE_OK == sceTextureGxtRevert(
    &outputBuffer,
    &outputBufferSizeInBytes,
    inputGxt,
    inputGxtSizeInBytes,
    0,                              // Revert texture 0
    &UserLogger,
    &userAllocator));

/* Query the container type of the output buffer.*/
SceTextureContainerType type;
SCE_DBG_ASSERT(SCE_OK == sceTextureQueryBufferType(
    &type,
    outputBuffer,
    outputBufferSizeInBytes));
```

```
if(type == SCE_TEXTURE_DDS_CONTAINER)
   /* The output buffer is in .dds format */

else if(type == SCE_TEXTURE_PVR_CONTAINER)
   /* The output buffer is in .pvr format */

else if(type == SCE_TEXTURE_TGA_CONTAINER)
   /* The output buffer is in .tga format */
```

5 DCC Plugins

Plugins are provided to load and save the GXT texture format with third-party Digital Content Creation (DCC) applications.

Autodesk Maya

The Maya GXT texture format plugins are located under:

- `sdk/host_tools/graphics/plugins/maya/` (32-bit versions)
- `sdk/host_tools/graphics/plugins.x64/maya/` (64-bit versions)

The relevant plugin must be loaded in Maya before you can load GXT textures. If you do not already have a procedure for distributing Maya plugins, refer to the section called "Distributing individual files" in the Autodesk Maya API Guide documentation.

After the plugin is loaded, you can load a GXT texture in the same way as any other texture file. GXT format files will not appear in the file browser by default. To display GXT format files, change the "Files of type" option to "Best Guess (*.*)".

The supported versions of Maya are Maya 2010, Maya 2011, and Maya 2012 in 32-bit and 64-bit Windows. A separate plugin is provided for each supported version.

Limitations

- Only the first face of cube maps is loaded.
- Only the first mipmap of a face is loaded.
- For palettized textures, only the first palette is used.
- HDR texture data is converted to 8 bits per channel.
- Example source code, which shows how to utilize the GXT texture library from within a Maya Hardware shader, is provided in addition to the official plugin. This allows full usage of cubemap faces and mipmap levels. The code for this example is located under:

  `sdk/host_tools/src/sce_texture/plugins/maya_gxt_hwshader/`

Autodesk 3ds Max

The 3ds Max GXT texture format plugins are located under:

- `sdk/host_tools/graphics/plugins/3dsmax/` (32-bit versions)
- `sdk/host_tools/graphics/plugins.x64/3dsmax/` (64-bit versions)

The relevant plugin must be loaded in Max before you can load GXT textures. If you do not already have a procedure for distributing Max plugins, refer to the section called "Plug-In Manager" in the Autodesk 3ds Max Help.

After the plugin is loaded, you can load a GXT texture in the same way as any other texture file.

The supported versions of Max are Max 2010, Max 2011, and Max 2012 in 32-bit and 64-bit Windows. A separate plugin is provided for each supported version.

Limitations

- Only the first face of cube maps is loaded.
- Only the first mipmap of a face is loaded.
- Only the first palette is used.
- HDR texture data is converted to 8 bits per channel.

Adobe Photoshop

The Adobe Photoshop GXT texture format plugins are located under:

- `sdk/host_tools/graphics/plugins/photoshop/` (32-bit versions)
- `sdk/host_tools/graphics/plugins.x64/photoshop/` (64-bit versions)

The relevant plugin must be placed in the correct location before you can load or save GXT textures. If you do not already have a procedure for distributing Photoshop plugins, refer to the section called "Plug-ins" in Photoshop Help.

After the plugin is installed, you can load or save a GXT texture in the same way as any other image file.

Load Options

If the GXT texture contains mipmaps, a dialog box will appear when you try to load it (it will not appear if the GXT does not contain mipmaps). If you select the "Load Mipmaps" option, mipmaps will be loaded side by side into a single image.

Save Options

When you save a texture to GXT format, a dialog box will appear. This dialog box allows you to choose the output format for your `.gxt` file. Only output formats that are suitable for your current image mode are shown. The current image mode is found in the **Image** > **Mode** menu.

Table 6 shows Adobe Photoshop output formats.

Table 6 Adobe Photoshop Output Formats

Mode	Bits	Output Formats
RGB Color	8, 16	`A8, L8, R5G6B5, A1R5G5B5, X1U5U5U5_1RGB, A8L8, L8A8, A4R4G4B4, X4U4U4U4_1RGB, U8U3U3U2_ARGB, L16, A8R8G8B8, A8B8G8R8, X8U8U8U8_1RGB, X8U8U8U8_1BGR, U8U8U8_RGB, U2U10U10U10_ABGR, U2U10U10U10_ARGB, U16U16_00GR, PVRT2BPP, PVRT4BPP, PVRTII2BPP, PVRTII4BPP, UBC1, UBC2, UBC3`
	32	`F16F16F16F16_ABGR, F16F16F16F16_ARGB, RF16, GF16RF16, RF32M`
Indexed Color	4	`P4_ABGR, P4_ARGB, P4_1BGR, P4_1RGB, P8_ABGR, P8_ARGB, P8_1BGR, P8_1RGB`
	8	`P8_ABGR, P8_ARGB, P8_1BGR, P8_1RGB`

A "Bits" value of 4 in Table 6 indicates that only the first 16 colors in the palette are used by the image. Photoshop does not directly support 4-bit indexed colors.

Photoshop does not support indexed alpha. The plugin loads the alpha channel of an indexed RGBA GXT texture into a non-indexed 8-bit alpha channel. The plugin will only allow saving to an indexed RGBA GXT output format if every pixel that has the same color index also has the same alpha value. For this reason, creating indexed RGBA images in Photoshop is difficult and not recommended.

The output formats prefixed with PVRT and UBC are compressed and inherently lossy. Every time you save an image to one of these formats, image quality will be lost. For this reason it is unadvisable to use these formats for image creation. The "Save Options" dialog box displays a warning when a lossy format is selected.

Many of the output formats have lower color depth than Photoshop's equivalent image mode. When a texture is first saved to such formats, some color depth will be lost. Unlike the PVRT and UBC formats, a second load and save of the image will not cause any further loss of quality. The Save Options dialog box displays a warning when precision loss will occur.

Limitations

- Mipmaps cannot be saved.
- Only the first palette is loaded.

- Multiple palettes cannot be saved.
- Alpha is not palletized in Photoshop.

Plugin Source Code

The source code for all the DCC plugins can be found at:

- `sdk/host_tools/graphics/src/sce_texture/plugins`

Third-Party SDKs

Due to license restrictions, SCE cannot distribute the DCC-application SDKs that are required to build the plugins. Before building, you will need to put such SDKs in the location expected by the plugin solution (`plugins.sln`). Alternatively, if you already have the SDK in another location on your hard drive, you could use Windows directory junctions to duplicate it in the expected location without using additional disk space.

The SDKs are expected to be contained in `sdk/host_tools/graphics/src/third_party` in subfolders described in Table 7.

Table 7 Third-Party SDKs

Expected Subfolder	Expected Contents
`max/3ds Max 2010 SDK`	Max 2010 SDK. A separate installer on the same disk as the Max application.
`max/3ds Max 2011 SDK`	Max 2011 SDK. A separate installer on the same disk as the Max application.
`max/3ds Max 2012 SDK`	Max 2012 SDK. A separate installer on the same disk as the Max application.
`maya/win32/2010`	Maya 2010 32-bit installation folder.
`maya/win32/2011`	Maya 2011 32-bit installation folder.
`maya/win32/2012`	Maya 2012 32-bit installation folder.
`maya/x64/2010`	Maya 2010 64-bit installation folder.
`maya/x64/2011`	Maya 2011 64-bit installation folder.
`maya/x64/2012`	Maya 2012 64-bit installation folder.
`adobe_photoshop_cs5_sdk_win`	Adobe Photoshop CS5 SDK. You can download it from http://www.adobe.com/devnet/photoshop/sdk.html Some versions of the SDK do not include all the required libraries.
`squish/squish-1.11`	Squish library source. Hosted on Google code: http://code.google.com/p/libsquish/