Revolution SDK

Texture Palette Library (TPL)

Version 1.1

Contents

Code Examples

Figures

Revision History

Version	Date Revised	Item	Description
1.1	2008/07/24	-	Integrated content with the *TPL Files and TexConv* document.
		3.1	Added "3.1 How to Use TexConv" on page 7.
		4	Removed “Making CLUT into Instances in the Texture Palette”. Added "4 TPL File Access" on page 15, and added descriptions of structures and functions.
1.00	2006/03/01	-	First release by Nintendo of America Inc.

1 Introduction

The TPL (texture palette) image format was a part of the Character Pipeline SDK provided in the Nintendo GameCube development environment. It was very versatile and is used in some of the GX library's sample demos. The Revolution SDK provides a smaller version of the TPL library to access this TPL format.

Using the TPL library provides easy access to TPL format files and allows users to render graphics that use these files.

This document describes the following.

- Features and characteristics of the TPL format and of TexConv, an application for texture conversion
- How to convert into the TPL format using TexConv
- How to access TPL files using the TPL library

2 Features and Characteristics of the TPL Format and TexConv

The TPL format and TexConv have the following features and characteristics.

- Disc loads are faster because multiple textures are stored in a single file.
- Efficiency is improved by storing multiple textures that share temporal locality with respect to an application in a single file.
- Many different texture maps and color lookup tables of different formats can be stored in a single file.
- Except for Z textures, all of the texture formats (`RGBA`, `IA`, `I`, `CI`, and `CMPR`) supported by the Wii console can be stored in a TPL file (TexConv does not support some formats).
- A TPL file uses 32-byte tiles for all of its texture data, so it can be used directly by the GX library.
- Input palettes can be converted into the `RGB565` and `RGB5A3` output formats.
- Mipmaps can be generated.
- S3TC textures can be compressed.
- Colors (for example, `RGBA` to `IA`, `CI` to `RGB`, or `RGB` to `RGBA`) can be automatically converted.

2.1 Optimized Texture Creation

When creating textures, a designer typically wants to add many detailed revisions to the original image file, until he or she is satisfied with its final appearance. To make image generation more efficient, the TexConv texture converter updates only those parts of a TPL file that have changed since TexConv was last executed. This speeds conversion by removing the overhead associated with recreating an entire TPL file every time a small change is added to a texture.

When TexConv is run, it compares each of the new images and palettes with the contents of the most recently created TPL file. If a data block matches, the existing block will be copied directly to a new TPL file, skipping the conversion process.

2.2 Support for Color Index Textures

Compression and animation are two large reasons to use a color index texture (`CI` texture). However, many developers use the `CMPR` (S3TC) texture compression format instead of the `CI` format because the former can provide high-quality compressed textures from true color images. Consequently, TexConv was designed without support for creating CLUTs (color lookup tables) from true color images. Do not confuse this with the TexConv feature that automatically converts from a `CI` input format to an `RGB` output format.

TexConv does not support encoding methods for `CI` texture animations. This is because it is difficult to define a CLUT encoding method for `CI` texture animations. For the most part, these encoding methods are unique to the animation being requested, and the relevant animation techniques themselves are unique to each game. Ultimately, TexConv only has very simple support for `CI` textures and is not the ideal way to create `CI` textures specific to a game. Instead, it is used to show developers how to use CLUTs and `CI` textures on the Wii console.

TexConv only supports `CI8`, but it stores 16-bit color index textures internally. The output indices are copied directly from the input indices. No bit masks or bit shifts are used.

Consequently, it is the developer's responsibility to ensure that the input color index is placed precisely within the output bit range. If the output format is `CI8`, the input indices must be between 0 and 255.

3 TPL File Conversion

The texture conversion application TexConv converts other files to TPL files according to script file specifications.

3.1 How to Use TexConv

By using TexConv, you can convert image files in the Truevision Graphics Adapter (TGA) format into TPL files. Run TexConv with the following command.

Code 3–1 TexConv Command

```
TexConv (TCS file) (TPL file)
```

A TGA file is not directly specified as an argument to TexConv. Instead, specify a texture conversion script (TCS) file, which contains the path of the TGA file to convert as well as other settings.

3.2 Texture Conversion Script File

The following is a sample texture conversion script (TCS) file.

Code 3–2 Texture Conversion Script

```
                        ; Comments start with a semicolon and continue to the end of the line

path      = c:/level1/mario ; Sets the path of the directory in which files exist
                        ; If “path” is NULL, the full path is required in the filename
                        ; “path” will be attached to all of the filenames that come after it
                        ; “path” can be set on any line
                        ; “path” can be an absolute or relative path

file 0    = marioHead_rgba8.tga
                        ; The complete filename is
                        ; c:/level1/mario/marioHead_rgba8.tga
image 0   = 0, 0, RGBA8     ; With image0[RGB]=file0 and image0[A]=file0,
                        ; convert into RGBA8.
texture 0 = 0, x            ; The image index for texture 0
                        ; in the TPL file is image0.
                        ; This is not a color index texture,
                        ; so the CLUT index is “x”.

path      = d:/temp/mario/  ; change the path
file 1    = marioArm_rgb565.tga
                        ; The complete filename is
                        ; d:/temp/mario/marioArm_rgb565.tga
image 1   = 1, x, RGB565, 0, 3, 0
                        ; Generate mipmaps 0 to 3 (four levels of detail) and remap to 0
texture 1 = 1, x

file 2    = marioFoot_ci8.tga
image 2   = 2, x, CI8       ; Convert to CI format
palette 0 = 2, RGB565       ; Create palette 0 from image 2
                        ; Convert palette entries into RGB565 format
texture 2 = 2, 0            ; texture 2[RGB]  = image 2,
                        ; texture 2[CLUT] = palette 0

path      = NULL            ; No path setting
file 3    = c:/marioBody_i8.tga
                        ; In this case the full path is required here
image 3   = 3, x, I8, GX_REPEAT, GX_REPEAT
                        ; TexConv automatically converts
                        ; RGB input into intensity output.
                        ; The S and T wrap modes can be configured
                        ; in order to repeat the image.
                        ; The wrap mode can be set in the same
                        ; way as the mipmap arguments.
texture 3 = 3, x
```

3.2.1 Basic Formatting Rules

Script files have the following basic rules.

- Spaces are ignored. Both "`image1=1`" and "`image  1  =     1`" are considered to be identical.
- Text is parsed line by line. A line ends with a newline character and cannot contain more than 255 characters.
- Images, textures, and palettes can be listed in any order, as can their indices. TexConv sorts each list in ascending order before converting them.
- Comments start with a semicolon and continue to the end of the line.
- Unspecified elements are written as "`x`". For example, "`image 1 = 1, x, RGB565`" describes an image without an alpha plane, while "`texture 1 = 1, x`" describes a texture without a palette.
- The directory path name is specified as either an absolute or a relative path from the current directory.

3.2.2 Commands

This section describes the script file commands recognized by TexConv.

3.2.2.1 Command to Change a Reference Directory Path

```
path = directory_path
```

Input files exist in the *directory_path* directory. The *directory_path* will be attached to all of the filenames that come after it.

The contents of *path* will be maintained until they are changed. Set *path* equal to 0 or `NULL` to disable it.

3.2.2.2 Command to Specify a Reference File

```
file file_id = file_name
```

The *file_id* number refers to the *file_name* file used by the `image` and `texture` commands. Any integer value of zero or greater is a valid number.

When *path* is disabled, *file_name* must include the full path. The path is either an absolute path or a relative path from the current directory.

When *path* has been configured, *directory_path* is attached to *file_name* in advance.

3.2.2.3 Command to Specify a Reference Image

```
image image_id = rgb_image_file_id, alpha_image_file_id,
                 format [,start_lod, end_lod, remap_lod] [,wrapS, wrapT]
```

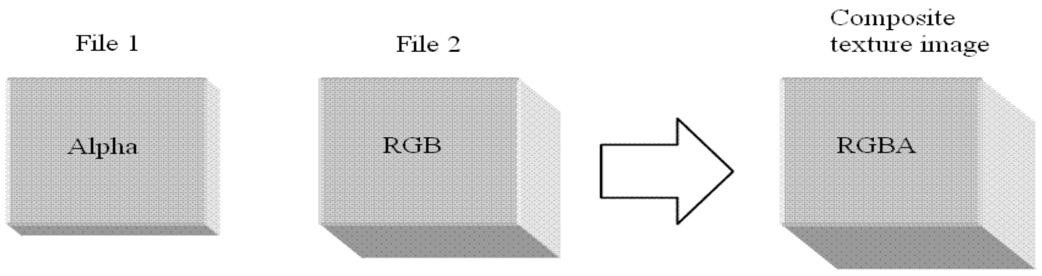
The *image_id* number refers to the image used by the `texture` command. Any integer value of zero or greater is a valid number.

The RGB portion of this image is contained in the *rgb_image_file_id* file.

The alpha portion of this image is contained in the *alpha_image_file_id* file.

As described above, the `image` command is extremely powerful. An output `RGBA` image can be constructed from two different files using *rgb_image_file_id* and *alpha_image_file_id*. The TGA format completely supports alpha planes, so this feature might not be used.

Figure 3–1 Creating an RGBA Image from Two Separate Files



- The output texture format is specified in *format*. Table 3–1 lists the formats that can be specified.

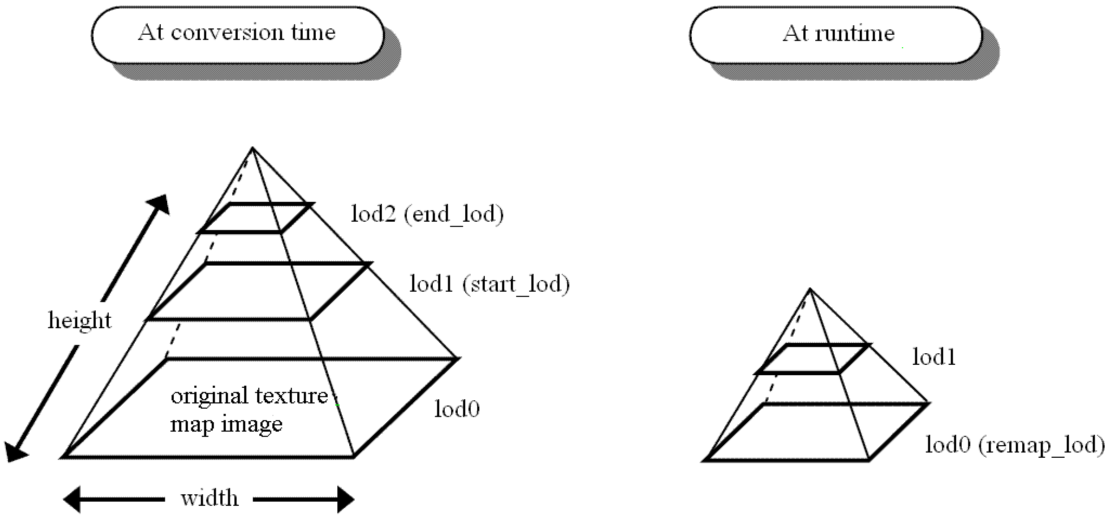
Table 3–1 List of Specifiable Texture Formats

format	**Details**
`I4`	4-bit intensity
`I8`	8-bit intensity
`IA4`	8-bit intensity + alpha (4 + 4)
`IA8`	16-bit intensity + alpha (8 + 8)
`CI8`	8-bit color index
`RGB565`	16-bit RGB
`RGB5A3`	16-bit RGB + alpha (RGB555 or RGBA4443)
`RGBA8`	32-bit RGBA
`CMPR`	Compressed format with 4 bits per texel

- The following are optional arguments and only required when creating mipmaps: *[, start_lod, end_lod, remap_lod]*. The mipmap levels that must be generated by TexConv are specified by *start_lod* and *end_lod*. The method to use for runtime mapping to a LOD is specified by *remap_lod*. Generally, *remap_lod* is 0.
- The optional arguments *[, wraps, wrapT]* are added to overwrite the wrap mode in the *s* and *t* directions. The possible values are `GX_REPEAT`, `GX_CLAMP`, or `GX_MIRROR`. By default, these are `GX_REPEAT` when the width and height dimensions are a power of 2, and `GX_CLAMP` otherwise.

The following example shows how to generate 2 LOD levels and remap to lod0 at runtime.

Figure 3–2 Generating and Remapping LOD Levels



3.2.2.4 Command to Specify Generated Palette Entries

```
palette CLUT_palette_id = CLUT_file_id, CLUT_format
```

The *CLUT_palette_id* is the number of the palette entry used by the `texture` command. Any integer value of zero or greater is a valid number.

The *CLUT_file_id* is the number of the image to use when creating this palette entry.

The two palette entry formats supported by *CLUT_format* are RGB565 and RGB5A3.

3.2.2.5 Command to Specify Generated Textures

```
texture texture_id = image_id, CLUT_palette_id
```

The `texture` command defines a texture in the TPL file. An integer of zero or greater must be set for the value of *texture_id*.

The *image_id* is the number of the image depicted by this texture.

The *CLUT_palette_id* is the number of the palette entry used by this texture. This parameter is “`x`” for textures that do not use the color index format.

3.3 List Order

When script file parsing begins, all image, palette, and texture lists are also started. Each line of text is recognized by its first word (`image`, `palette`, `texture`, and so on), and a new element is added to the appropriate list.

Once script file parsing has completed, TexConv will sort each list in ascending order. As a result, images, textures, palettes, and their corresponding indices can be listed in any order within the script file. In other words, though the images, textures, and palettes are listed interchangeably in "Code 3–2 Texture Conversion Script" on page 8, all the images could be listed as a group after all of the palettes.

Palettes and texture images are each stored in the TPL file in the order that they occur in their respective sorted lists.

Normally, the list index matches the position in the sorted list, so that an index of 0 is the start of the list, and an index of (n-1) is the end of the list. Although this is convenient for commenting out several script file elements in an attempt to set an optimal TPL file structure, note that the script file may specify an index that does not match the position in the sorted list. Once the final TPL file structure has been settled, the script file indices should be redistributed so that they are continuous from 0 to (n-1). Even though this is not obligatory, it is strongly recommended to avoid confusion when accessing textures at runtime.

3.4 Image Files that Can Be Used as Input

Image files in the Truevision Graphics Adapter (TGA) format are used as input to TexConv.

The TGA format was chosen as the format to support because it has a long history; it can be imported and exported by many graphics tools; and it can store many texture formats, including true color, intensity, intensity alpha and color-indexed images, and 256-color palettes.

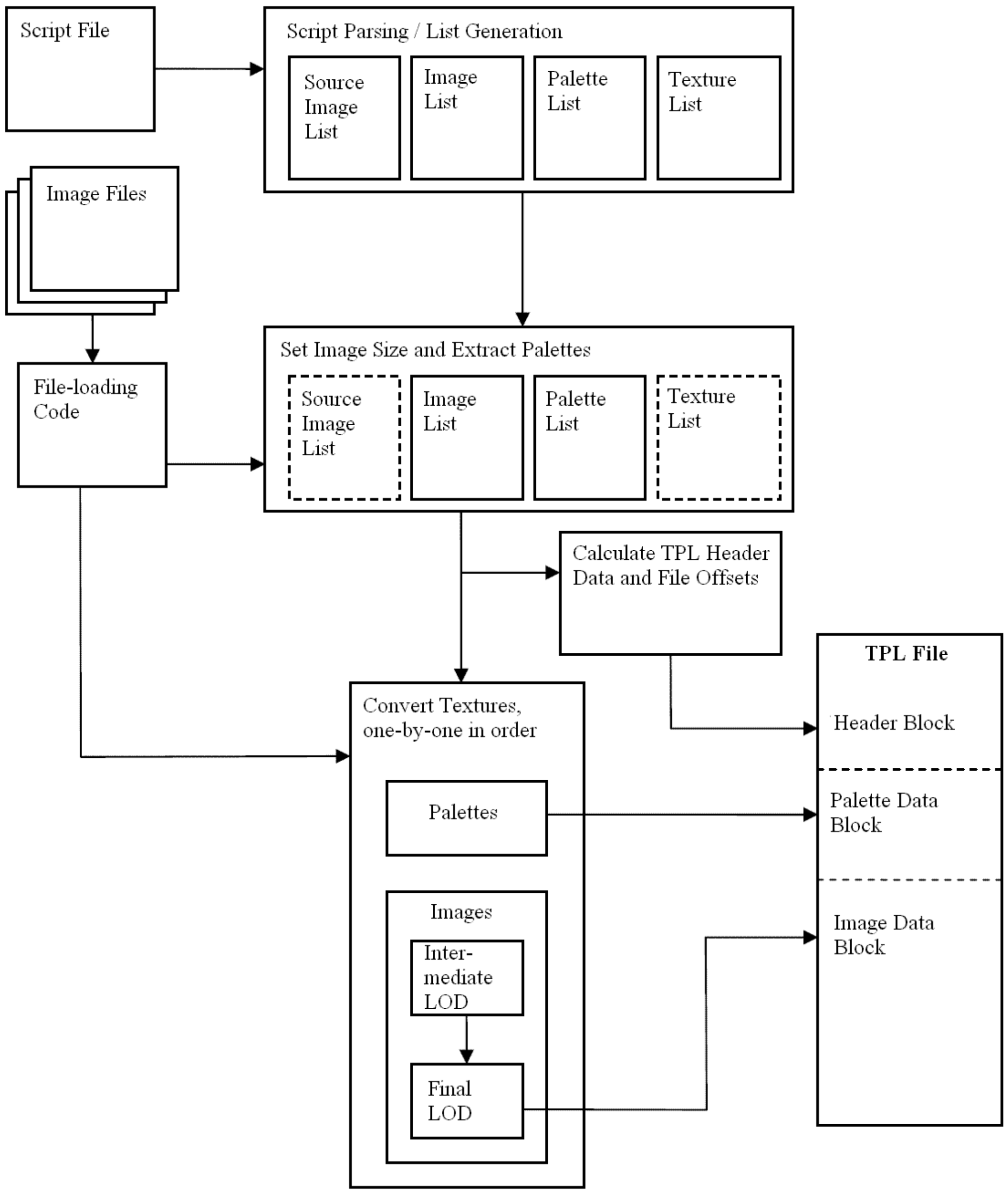
Refer to the *Encyclopedia of Graphics File Formats, 2nd Ed.* (ISBN 1-56592-161-5), pp. 860-879, for details on TGA files.

3.5 Conversion Process Flow

The TexConv source code is provided. To better understand the code, we provide the following overview of the major processes performed by TexConv to convert the original TGA file into a TPL file.

Figure 3–3 shows the logical flow of TPL file generation with TexConv.

Figure 3–3 Conversion Process Flow for TPL Files



TPL file conversion can roughly be divided into three stages.

3.5.1 Input

Script files are loaded and parsed. An error check confirms that all of the data is consistent and all of the requested input files exist. If there is a problem with any of this, the program will output an error message and exit.

3.5.2 Setup

TexConv gathers the input data and structures it for conversion. At this stage, TexConv creates and manages four information lists. These lists are as follows.

- Source Image List

  Lists all of the original input files, along with their size and format information.
- Image List

  Contains information for the final images, such as how different files will be combined. For example, a single image can take RGB information from one source and alpha information from another source.
- Palette List

  Holds palette information from the source data. Palettes are extracted from the source files and stored here.
- Texture List

  Contains information for combining files to create images with palettes in the output TPL file.

3.5.3 Convert and Export

TexConv reads the texture list from beginning to end and converts the original data into a format that can be handled by the Wii console. If data that has already been converted exists, it will be used and conversion skipped. TexConv generates mipmap levels as necessary during conversion. Each time a texture has finished converting and is generated, it will be exported to the new TPL file. TexConv will close the TPL file after it is done processing all textures in the texture list. This completes the runtime library preparations.

3.6 Cache File

In the process of creating a texture, designers will repeatedly edit, convert, and then preview the texture. Each time, almost all of the textures within the TPL file will probably be unchanged since the previous conversion results. TexConv saves conversion time by making block copies of unchanged textures from the previous TPL file and converting only the modified textures.

To determine if an image or palette needs to be converted, TexConv manages a cache file (CSF) that describes the content of the previous TPL file. TexConv consults the CSF file, the previous TPL file, and the source image files to check for changes; if nothing has changed, it copies the data that exists in the cache file.

The CSF file is saved in `C:\Temp\tplCache.csf`.

The CSF file itself does not need to be read or changed. However, if this is necessary, the `.csf` file format is entirely explained in comments and structures at the beginning of the file below.

```
$(REVOLUTION_SDK_ROOT)/build/libraries/tc/src/TCTPLToolBox.cpp
```

The following situations will result in complete reconversion.

- The CSF file does not exist.
- The previous TPL file does not exist.
- The previous TPL file has been changed since it was created.
- The version number of the previous TPL file does not match the current “code” version number.
- The previous TPL file and the new TPL file do not have any common images or palettes.

When determining whether to convert or copy data blocks, TexConv checks whether the previous TPL file has image or palette blocks with the same filename and attributes as the images or palettes to convert. This means that even if the script file index or TPL filename is changed between conversions, the data that has already been converted can be used.

The following situations will result in partial reconversion.

Table 3–2 Causes of a Partial Reconversion

State	TexConv Behavior
The Last Modified date on a source image file has changed	Reconvert the image and palette for this file
The script file includes changes	Reconvert the corresponding images and palettes
Content has been added to the script file	Convert the new images and palettes

Note: To determine whether data can be reused, TexConv checks script file tokens and the timestamp at which changes were added to a file. The check does not extend to the actual bits in the source image. This would cause errors whenever the conversion code has been rewritten between consecutive invocations. In such cases, the previous TPL file can no longer be synchronized with the current conversion code. To force a complete reconversion, delete either the CSF file or the previous TPL file.

4 TPL File Access

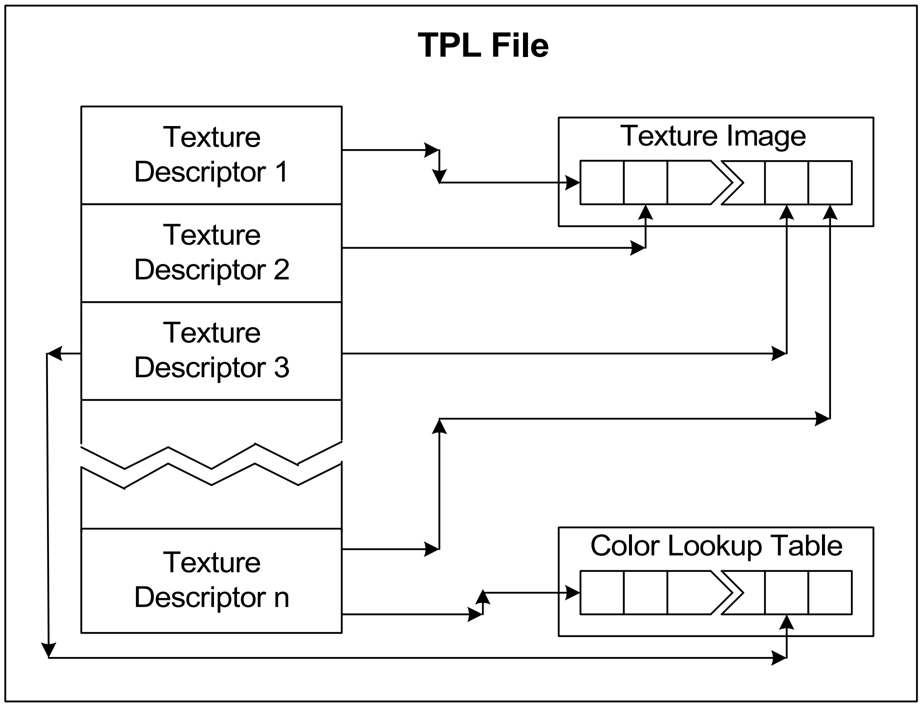
The TPL library provides access to textures. It offers basic functionality for obtaining information describing individual textures in a texture palette that has been loaded from disc and decompressed into memory, and for loading textures into hardware.

Refer to the "TPL library" section of the *Revolution Function Reference Manual* for details on its various functions and structures.

4.1 Texture Palettes in Memory

When a TPL file is loaded into memory and decompressed, it is simply expressed as a list of texture descriptors. The texture descriptors maintain pointers to the texture images and, if required, pointers to color look-up tables (CLUTs).

Figure 3–4 A Texture Palette in Memory



4.2 Initialization

When an application uses a TPL file, it will load the file into memory, decompress the internal pointers immediately afterwards, and then store the texture palette information in a texture palette structure, `TPLPalette`.

Code 4–3 Texture Palette Structure

```
typedef struct
{
   u32              versionNumber;
   u32              numDescriptors;
   TPLDescriptorPtr descriptorArray;

} TPLPalette, *TPLPalettePtr;
```

The TPL library provides an initialization function to perform the aforementioned operations.

Code 4–4 Initialization Function

```
void TPLBind ( TPLPalettePtr pal )
```

4.3 Obtaining Texture Information

The TPL library provides a function to get texture information from a texture palette structure.

Code 4–5 Function for Getting Texture Information

```
TPLDescriptorPtr TPLGet ( TPLPalettePtr pal, u32 id )
```

By using the `TPLGet` function, you can extract a texture descriptor structure containing texture information from a texture palette structure.

Code 4–6 Texture Descriptor Structure

```
typedef struct
{
   TPLHeaderPtr      textureHeader;
   TPLClutHeaderPtr  CLUTHeader;

} TPLDescriptor, *TPLDescriptorPtr
```

The texture descriptor structure stores pointers to a texture header structure and a CLUT structure.

Code 4–7 Texture Header Structure

```
typedef struct
{
   u16 height;
   u16 width;
   u32 format;
   Ptr data;

   GXTexWrapMode wrapS;
   GXTexWrapMode wrapT;
   GXTexFilter   minFilter;
   GXTexFilter   magFilter;

   float LODBias;
   u8    edgeLODEnable;
   u8    minLOD;
   u8    maxLOD;
   u8    unpacked;

} TPLHeader, *TPLHeaderPtr
```

The texture header structure stores texture information such as the size and format of texture images.

Code 4–8 Color Lookup Table (CLUT) Structure

```
typedef struct
{
   u16       numEntries;
   u8        unpacked;
   u8        pad8;
   GXTlutFmt format;
   Ptr       data;

} TPLClutHeader, *TPLClutHeaderPtr
```

The CLUT structure stores the number of palette entries in the color lookup table, the data format, and other information.

In addition to obtaining texture data via the texture descriptors, the TPL library provides two functions to directly initialize `GXTexObj` and `GXTlutObj`. These functions simply look at the texture descriptors and initialize the specified GX object as appropriate.

Code 4–9 Functions that Obtain Texture Data Using `GXTexObj` and `GXTlutObj`

```
void TPLGetGXTexObjFromPalette ( TPLPalettePtr pal, GXTexObj *to, u32 id )

void TPLGetGXTexObjFromPaletteCI ( TPLPalettePtr pal, GXTexObj *to,
                                   GXTlutObj *tlo, GXTlut tluts, u32 id )
```

4.4 Simple Utility Functions

The Character Pipeline SDK provided with the Nintendo GameCube development environment was designed to automatically allocate memory from within the library and load from the disc during TPL file decompression. To provide the same convenience, the following utility function is registered as an inline function in `demo.h` in the DEMO library.

Code 6–10 Simple TPL File Decompression and Deallocation

```
void TPLGetPalette ( TPLPalettePtr *pal, char *name )

void TPLReleasePalette ( TPLPalettePtr *pal )
```

These internally call functions such as `OSAlloc` and `DVDRead` to implement the same automatic file decompression as the Character Pipeline SDK.