Texture Swizzling

Version 1.0 - 2003-06-20 - PS2Linux edition

Chapter 1 - The theory behind texture swizzling

Introduction

The GS can store textures in VRAM in many different formats as explained in the GS User Manual (section 8). If an array of data is sent to the GIF specifying PSMCT32 as the transfer format, it will be stored in VRAM differently than if the same data was sent specifying PSMT4 as the transfer format. This is completely transparent for the programmer, all that is required is to have the raw 4bit data in main memory and specify PSMT4, or the raw 32bit data and specify PSMCT32.
The concern is, as always, speed. An array of data sent in PSMT4 format will take longer to send than the same array in PSMCT32. The speed factor is about 200% to 300% in favour of PSMCT32, so it would be interesting to be able to use the full speed of the bus. The reason PSMCT32 is faster to transfer the same amount of data than the other format is purely due to internal GS bit conversion.

What is texture swizzling?

Texture swizzling is the "art" of swapping the pixels around in a texture so that when it is sent in PSMCT32 format, it will be stored in exactly the same way it would have been if it had been transferred using the original format. This allows for greater transfer speeds, while the texture sizes don't change and the geometry does not have to be modified in any way.

When is it done?

Texture swizzling is an offline process, so in the end it is completely free. Typically, the swizzling code is part of the export toolchain, when the models are exported from the modelling software and saved in a PS2 friendly format.
It also happens that the toolchain converts 32bit textures to 4 or 8bit using your favourite quantitiser; and after that conversion it is a good time to swizzle.

How is it done?

The principle is very easy, and doing it on the PS2 itself is trivial, even though it isn't ideal. An easy way to understand the process is to do it on the PS2, but for a better approach, see the next paragraph.
On the PS2, all you have to do is send the texture to the GS using the original transfer format and then grab it back to main memory using PSMCT32; however, the texture size has to be taken in consideration (see the "*Texture sizes*" section). Sending the resulting texture to the GIF using PSMCT32 now uses the full speed of the bus, and writes the data in the correct layout to be used by the geometry using the original format settings.

A much better way is to write a number of functions in C that you can integrate in your texture conversion tool easily to swizzle the pixels in the same way as the GS does. Luckily, this has been done and distributed by **Victor Suba**, so you don't have to do it yourself (see "*Related links*").
Victor implemented functions which read and write data to a simulated GS memory (i.e. a statically allocated 4MB array). All the formats are supported so that you can write to the "VRAM" in the original format and read it back in PSMCT32 to swizzle the texture; or you can write in 32bit and read in the original format to unswizzle it.

Texture sizes

One thing to take into account is that although the size (in bytes) of the data doesn't change, the size (width and height) of the texture depends on the format you are using. Take for example a 256x128 texture (*Figure 1*). If the original format is 8bit, then swizzling it to 32bit will change the size to 128x64 (*Figure 2*). If the original format is 4bit, then swizzling it to 32bit will change the size to 128x32 (*Figure 3*). The 3 textures have the same size in bytes.

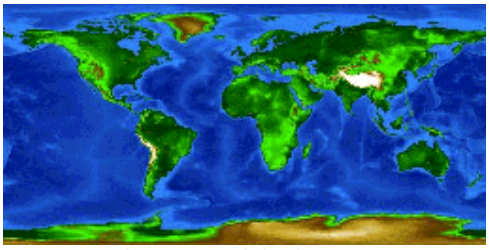


Figure 1: Original texture

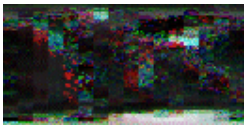


Figure 2: 8 to 32bit

Figure 3: 4 to 32bit

Since the texture size changes, you have to either store or calculate the original size when you want to fill the TEX0 register when you use the texture.

If you are prebuilding your DMA chains, then the exporter has to set TEX0 according to the original size of the texture, not the size of the swizzled version. If you are building your chains at run-time, it may be a little bit trickier; you may have to add user-data to the texture file to specify the original format and size of the texture (the TIM2 format allows you to do that, very handy).

	Width	Height
PSMT4	2	4
PSMT8	2	2
PSMCT16	1	2

Figure 4: Factors table when converting to PSMCT32

The conversion table for texture sizes in Figure 4 shows the width and height factors to use when converting to and from PSMCT32. For example, converting from PSMCT16 to PSMCT32, divide the height by two and keep the same width.

Speed tests

It is very easy to test the speed increase achieved by the swizzling technique; a simple program to send a texture repeatedly can be used to measure the raw gain of switching from PSMT4/8 to PSMCT32. The old sample TexTrans (see "*Related links*") does exactly that, and the table in "*Speed comparison table*" lists the respective gain for a number of texture sizes. N/A is specified when the size does not allow the conversion to work because of the layout of the blocks. It is a lot easier if the textures are sized to fit GS pages, as it helps the texture manager find where to upload the texture and the swizzle works a lot better.

As a general rule, it is usually better to try and pack a number of related textures into one when possible. Of course, this can cause problems with the CLUT (although you can have one CLUT for each part of the packed texture, but that means that file format you use has to support multiple CLUTs). It may also involve some changes in the CLAMP register, but nothing major.

It is also good to notice that the speed difference between PSMCT16 and PSMCT32 is very small, so converting to one format or the other is just as good. But generally conversions work best from 4bit to 16bit and from 8bit to 32bit, as shown in "*Speed comparison table*".

Speed comparison table

Size	Depth	Normal transfer	PSMCT16	PSMCT32	Max gain
16 x 32	4bit	1244	2380	N/A	191%
16 x 64	4bit	1223	2243	N/A	183%
16 x 128	4bit	1144	2039	N/A	178%
32 x 16	4bit	1224	2331	N/A	190%
32 x 32	4bit	1189	2223	2215	187%
32 x 64	4bit	1155	2039	N/A	177%
32 x 128	4bit	1020	1758	N/A	172%
64 x 16	4bit	1224	2262	N/A	185%
64 x 32	4bit	1164	2061	N/A	177%
64 x 64	4bit	1012	1755	1747	173%
64 x 128	4bit	782	1356	N/A	173%
128 x 16	4bit	1144	2013	N/A	176%
128 x 32	4bit	832	1736	N/A	209%
128 x 64	4bit	519	1242	N/A	239%
128 x 128	4bit	299	774	943	315%
256 x 16	4bit	839	1777	N/A	212%
256 x 32	4bit	520	1252	N/A	241%
256 x 64	4bit	297	759	N/A	256%
256 x 128	4bit	160	434	N/A	271%
256 x 256	4bit	83	333	332	401%
16 x 8	8bit	1137	N/A	2404	211%
16 x 16	8bit	1142	N/A	2331	204%
16 x 32	8bit	1113	N/A	2260	203%
16 x 64	8bit	1054	N/A	2077	197%
32 x 8	8bit	1129	N/A	2331	206%
32 x 16	8bit	1116	2244	2220	201%
32 x 32	8bit	1060	N/A	2071	195%
32 x 64	8bit	970	N/A	1754	181%
64 x 8	8bit	1122	N/A	2175	194%
64 x 16	8bit	1060	N/A	2073	196%
64 x 32	8bit	965	1754	1781	185%
64 x 64	8bit	842	N/A	1360	162%
128 x 8	8bit	1065	N/A	2040	192%
128 x 16	8bit	965	N/A	1767	183%
128 x 32	8bit	842	N/A	1365	162%
128 x 64	8bit	660	778	947	143%
128 x 128	8bit	461	N/A	584	127%
128 x 256	8bit	288	N/A	332	115%
256 x 8	8bit	974	N/A	1746	179%
256 x 16	8bit	834	N/A	1365	164%
256 x 32	8bit	648	N/A	941	145%
256 x 64	8bit	430	434	584	136%
256 x 128	8bit	256	N/A	332	130%
256 x 256	8bit	140	N/A	179	128%

The values correspond to the number of textures that can be sent in 200 scanslines; the more the better, obviously.

Conclusion

While swizzling is a simple and straightforward concept, the actual process is a bit fiddly. Not all texture sizes are "swizzleable", and different sizes give very different results.
It may be useful to add some user information in the texture file to say what the original size and format of the texture are (used for TEX0) to make it easy to enable swizzling per texture. That way, you upload the texture using the actual texture size and format, and set TEX0 to the original values, and you don't have to worry whether the texture is actually swizzled or which format it is swizzled to.
If a texture is causing problems when swizzled, simply don't swizzle it. The gain on a single texture is negligible, and it is not worth losing hair over a few textures. However, try rotating it and update the STs in the geometry, as this may fix the problem.
Once the exporters are tweaked and the renderer is set up, the process is completely transparent and done once and for all.

For swizzling code samples, check out ezSwizzle which features a Windows GUI and batch processing.

Related links

Victor Suba's code:
http://www.playstation2-linux.com/projects/ezSwizzle/

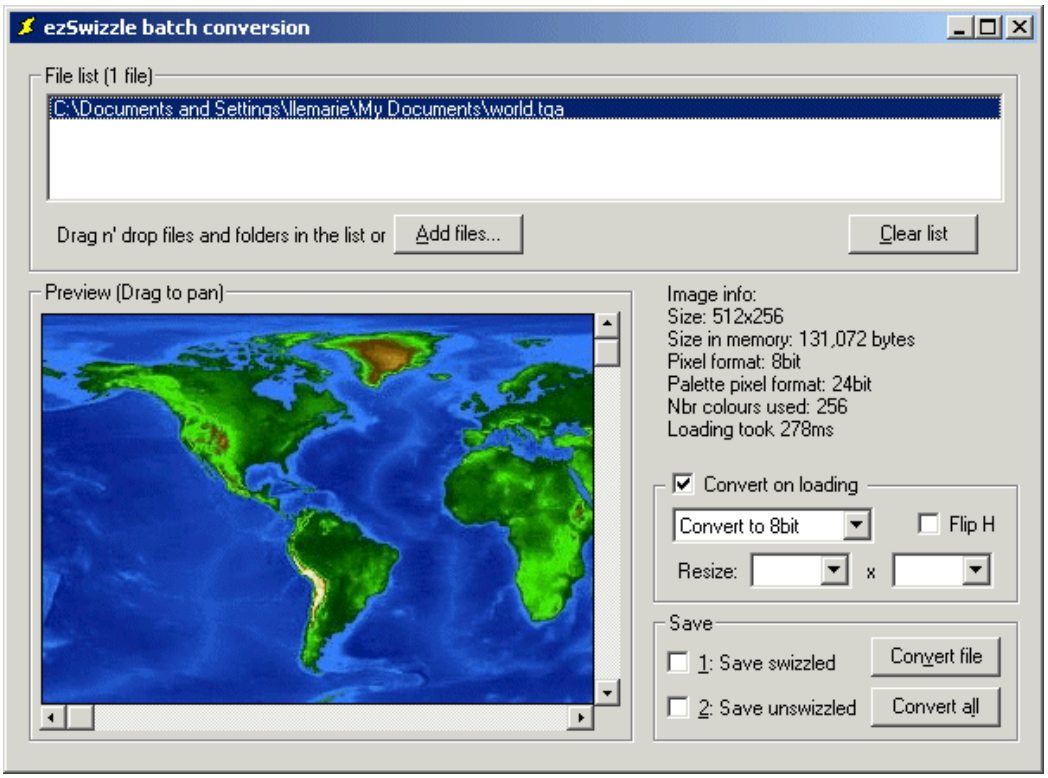
ezSwizzle:
http://www.playstation2-linux.com/projects/ezSwizzle/

Contact: Lionel Lemarié
llemarie@playstation2-linux.com

Chapter 2 - In practice: ezSwizzle

About the software

This Windows MFC application is a sample of tool used to convert textures from a variety of formats to a customised TIM2 format. Because the TIM2 format is flexible, the images can still be read by conventional editors; but ezSwizzle can also read the extra information to unswizzle the texture if necessary. Also, with the extra information, the code running on the PS2 can easily use the texture with very little extra overhead.
The batch feature allows you to select a number of textures, convert them to 8bit if so you desire, swizzle them, and save them with a suffix.



How to use it

You can drag n' drop a selection of files or folders into the file list, or click on "`Add files...`" to use the Open File dialog.
When you select a file in the list, the preview displays the picture and some information on the image is provided. You then have the choice to convert the image to 8bit or 32bit, flip it horizontally, and resize it to your liking.
The final part lets you decide whether you want to swizzle, unswizzle or leave the image as is. Then you can either save the current file, or batch convert the whole list.

Customising the file format

With the provided sources, it is easy to customise the output file format and use the one expected by your engine instead. All you have to do is create a `SaveMyFormat(CString strFilename)` function in the `CEzSwizzleDlg` class (in the .c and the .h) and call it instead of `SaveTIM2(strFilename)` in the `OnButtonConvertfile()` function.

All the image information is held in the following class members:

`iImageWidth`	Width of the converted image in pixels.
`iImageHeight`	Height of the converted image in pixels.
`iImageDepth`	Depth of the converted image in bits per pixel.
`iImageOriginalWidth`	Width of the original image in pixels.
`iImageOriginalHeight`	Height of the original image in pixels.
`iImageOriginalDepth`	Depth of the original image in bits per pixel.
`pImageData`	Converted image data buffer.
`iDataSize`	Size of the data buffer in bytes.
`pOriginalPalette`	Original palette buffer.
`iPalBytesPerPixel`	Depth of the palette in bytes per pixel.

When saving the file, keep the following points in mind:
- You can test if the image has been swizzled or not by comparing iImageWidth and iImageOriginalWidth; the former will be smaller than the latter if the image is swizzled.
- If the image is swizzled, don't forget to save the CLUT, even though the image is now in 16/32bit !
- Set the TEX0 register using iImageOriginalWidth, IImageOriginalHeight, and iImageOriginalDepth.
- Set up the texture transfer using iImageWidth, iImageHeight, iImageDepth.

Support and bug reports

If you have any problems, suggestions or comments, please email us at the usual address: llemarie@playstation2-linux.com.

If you want to complain/comment about the code, or if you have feature requests, please post them on the project on the web site:
http://playstation2-linux.com/bug/ezswizzle