
# PVRTC & Texture Compression

# User Guide



| | | |
|---|---|---|
| Filename | : | PVRTC & Texture Compression.User Guide |
| Version | : | PowerVR SDK REL_3.3@2810346a External Issue |
| Issue Date | : | 05 Mar 2014 |
| Author | : | Imagination Technologies Limited |

# Contents



# List of Figures



# List of Tables

# 1. Introduction

Texture compression is an important tool in the arsenal of developers that, due to its many advantages over uncompressed formats, should be used whenever it is feasible to do so. This document contains a brief explanation of why texture compression should be used as well as an introduction to PowerVR's texture compression, PVRTC.

PVRTC is available in two versions: PVRTC1 and PVRTC2. Each version offers two bit depths per version, 2bpp and 4bpp. Both versions are described in this document.

# 2. Texture Compression

## 2.1. Why Use Texture Compression?

Modern applications have become graphically intensive; certain types of software, such as games or navigation aids, often need large amounts of textures in order to represent a scene with satisfying quality. Texture compression can save or allow better utilization of bandwidth, power, and memory without noticeably losing graphical quality.

### 2.1.1. Performance Improvement

A smaller texture data size means smaller transfers from memory to the GPU. Memory bandwidth is precious, particularly in mobile platforms where shared memory architectures are prevalent. In situations where memory bandwidth is the limiting factor in an application's performance, texture compression can provide a significant improvement.

### 2.1.2. Storage Footprint vs. Memory Footprint

Texture compression reduces the memory footprint of a given texture; this allows applications to fit all their required textures in a constrained amount of texture memory, or to use larger (or more) textures for the same memory budget resulting in potentially, extra quality. In addition, any savings in memory requirements are very useful for mobile and tablet devices where, as mentioned, memory is shared across an entire SoC (System on Chip).

### 2.1.3. Power Consumption

Memory accesses are expensive in terms of power consumption on mobile devices where battery life is of the upmost importance. The bandwidth savings and better cache usage resulting from run-time texture compression both contribute to decreasing the quantity and magnitude of memory accesses; which in turn reduce the power consumption of an application.

## 2.2. Image File Storage Compression versus Texture Compression

Developers are familiar with compressed image file formats such as jpg or png; however, it is important to remember that there is a distinction between these forms of ‘storage’ compression and the texture compression discussed in this document.

The primary requirement of ‘storage’ compression schemes is that files compressed using them should occupy as small an amount of storage in a file system as possible, there is no requirement that the data stay compressed while in use. The result is that ‘storage-based’ image file formats tend to produce very small file sizes, often for very high (or lossless) image quality, but at the cost of immediate decompression for when the image is loaded into an application and used. This immediate decompression, usually to 24/32bpp, means that the image, while small on disk, consumes large amounts of bandwidth and memory when used as a texture.

Texture compression schemes, such as PVRTC, are designed to be directly usable by the GPU. The texture data exists in storage, in memory, and when transferred to the graphics hardware itself, in the compressed format. The only step in which full-precision colour values are extracted from a compressed state is when the texture sampling hardware inside the graphics accelerator passes texel values to the shader processing units – a process entirely separate from the main memory bus. A graphical representation of this can be seen in Figure 2-1 Image file compression vs. texture compression.

This allows all the advantages mentioned in Section 2.1, but puts some limits on the form the compression technique may take. In order to allow for direct use by the graphics accelerator, a texture format should be optimised for random access, with a minimal size of data from which to retrieve each texel’s values. Consequently, texture compression schemes are usually fixed bitrate with very high data locality; image file formats are not constrained by these requirements and thus can often achieve higher compression ratios and image quality for a given data size.

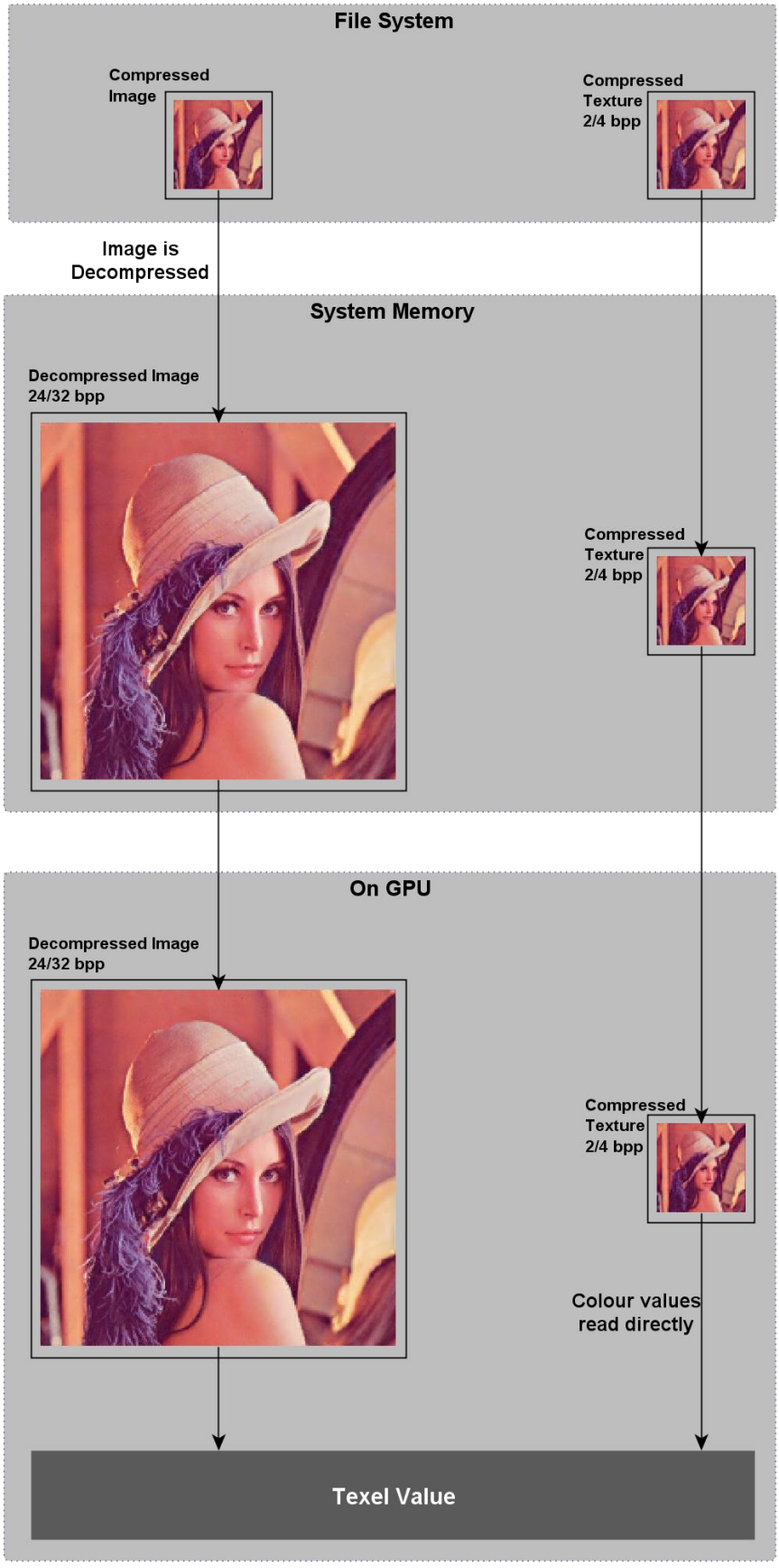


Figure 2-1 Image file compression vs. texture compression

# 3. PVRTC

## 3.1. Overview

PVRTC is PowerVR's proprietary texture compression scheme; it uses a sophisticated amplitude modulation scheme to compress textures: texture data is encoded as two low-resolution images along with a full resolution, low bit-precision modulation signal. More information can be found in the whitepaper:

Fenney, S. (2003) 'Texture Compression Using Low-Frequency Signal Modulation' *SIGGRAPH Conference*.

PVRTC uses 64-bit data-words containing the most significant weights for either a 4x4 or 8x4 sets of pixels; the increased quality of PVRTC comes from the use of adjacent data-words to reconstruct the original image. This represents a considerable visual enhancement compared to block-based compression techniques that use the contents of a single block alone to reconstruct the texel values within that individual block.

PVRTC supports both opaque (RGB) and translucent (RGBA) textures (unlike other formats such as S3TC that require a dedicated, larger form to support full alpha channels); and boasts a very high image quality for competitive compression ratios: 4 bits per pixel (PVRTC 4bpp) and 2 bits per pixel (PVRTC 2bpp). These represent savings in memory footprint of 8:1 (PVRTC 4bpp) and 16:1 (PVRTC 2bpp) compared to uncompressed, 32 bit per pixel textures. Finally, channel data is encoded on a per-data-word basis so that fully opaque sections of a texture do not suffer the precision cost of encoding unnecessary alpha information.

## 3.2. Why Use PVRTC?

In any given situation, the best texture format to use is the one that gives the required image quality at the highest rate of compression. The smaller the size of the texture data, the less bandwidth is required for texture fetches; this reduces power consumption, can increase performance, and allows for more textures to be used for the same budget. The smallest RGB and RGBA format currently available is PVRTC 2bpp; as such, it should be considered for every texture in an application. A larger format (such as PVRTC 4bpp) should only be used if a texture encoded using PVRTC 2bpp does not have sufficient quality.

- It is important to consider textures on a case by case basis rather than applying a blanket format to all textures that may result in unnecessary bandwidth wastage.
- Once a texture is loaded into the graphics API that is being used (e.g. OpenGL ES) then use of this texture is the same irrespective of format so that only the upload code need worry about variable formats. For an example of texture loading code that can deal with many formats please examine the PVRTTexture functions found in the PVRTools folder of the PowerVR Insider SDK.
- It is suggested that texture encoding be carried out using a script and the command line version of PVRTexTool (also available in the SDK) in order to ease this selective choice of format per texture.
- PVRTC support can be checked for at runtime through the graphics API – for example: check for "GL_IMG_texture_compression_pvrtc" in the OpenGL ES extension string.

# 4. PVRTC2

PVRTC2 is the second version of PVRTC. It supports the same features as PVRTC1 along with several improvements. If your target platform supports it then PVRTC2 should be preferred for your applications.

Support of PVRTC2 can be checked for in a similar way to PVRTC1: through the graphics API – in the case of OpenGL ES, look for GL_IMG_texture_compression_pvrtc2

## 4.1. PVRTC2 versus PVRTC1

As with PVRTC1, PVRTC2 supports RGB and RGBA textures at 4bpp and 2bpp, and also has the following advantages:

### 4.1.1. Image Quality

In general, PVRTC2 will give a better quality image than PVRTC1: in images with large areas of colour discontinuity, such as those in Figure 4-1 Texture atlas and normal map the quality will be significantly higher without the need for texture manipulation techniques such as border expansion. A border around images in a skybox is also no longer needed, and non-tiling textures will appear much improved. This is due to the addition of dedicated modes for dealing with areas of high contrast between sections of a texture.

### 4.1.2. Non-Power-Of-Two Dimensions

PVRTC2 offers the developer the use of arbitrary-sized, NPOT textures – textures that don't have dimensions that are limited to powers of two.

### 4.1.3. Sub-Texturing

Unlike PVRTC1, sub-texturing is supported in PVRTC2 at data-word boundaries (4x4 or 8x4 for PVRTC2 4bpp or PVRTC2 2bpp respectively). This further enables techniques such as texture atlasing for applications. It should be noted that this requires "hard-edge" mode to be enabled around the sections of a texture to be replaced – for more information on this PVRTC data mode see 4.3.2 PVRTC2 Data.

Figure 4-1 Texture atlas and normal map

## 4.2. Comparison Images

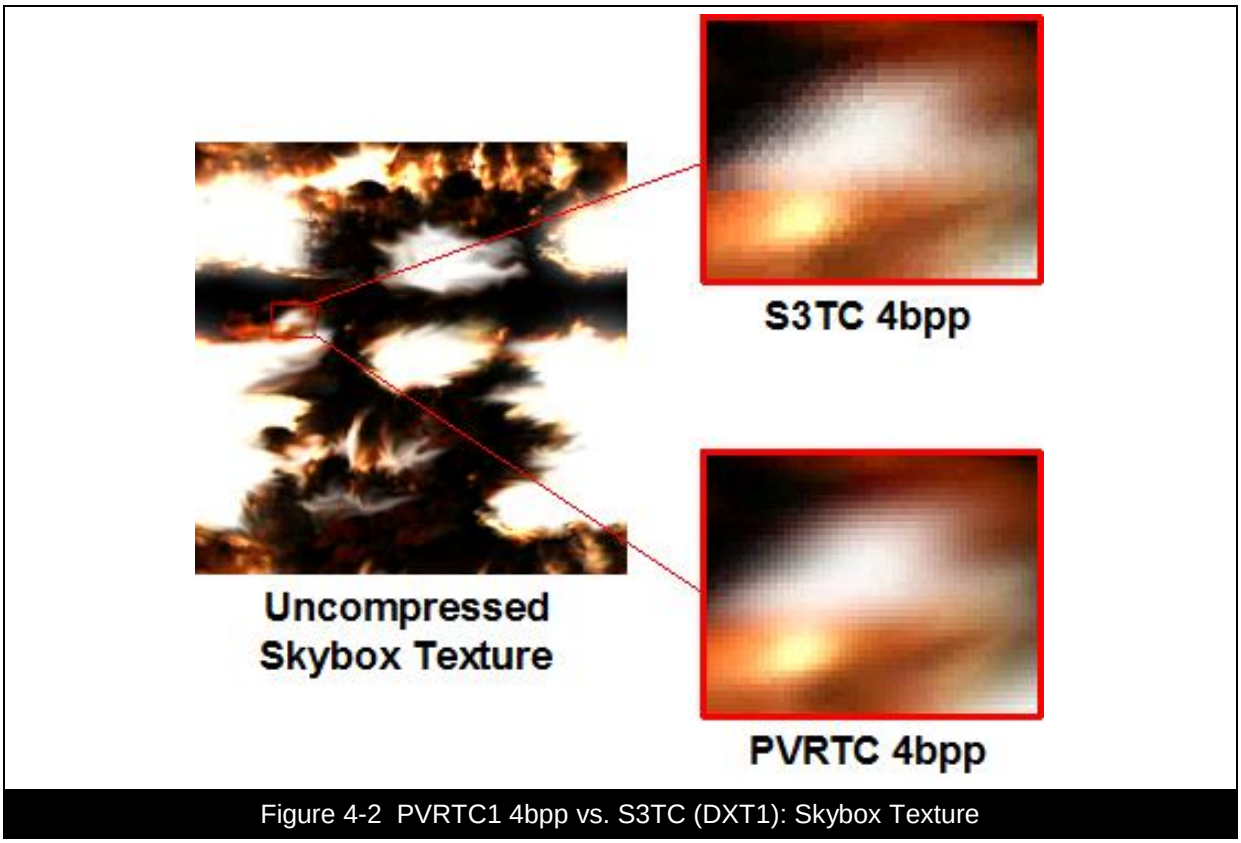


Figure 4-2 PVRTC1 4bpp vs. S3TC (DXT1): Skybox Texture

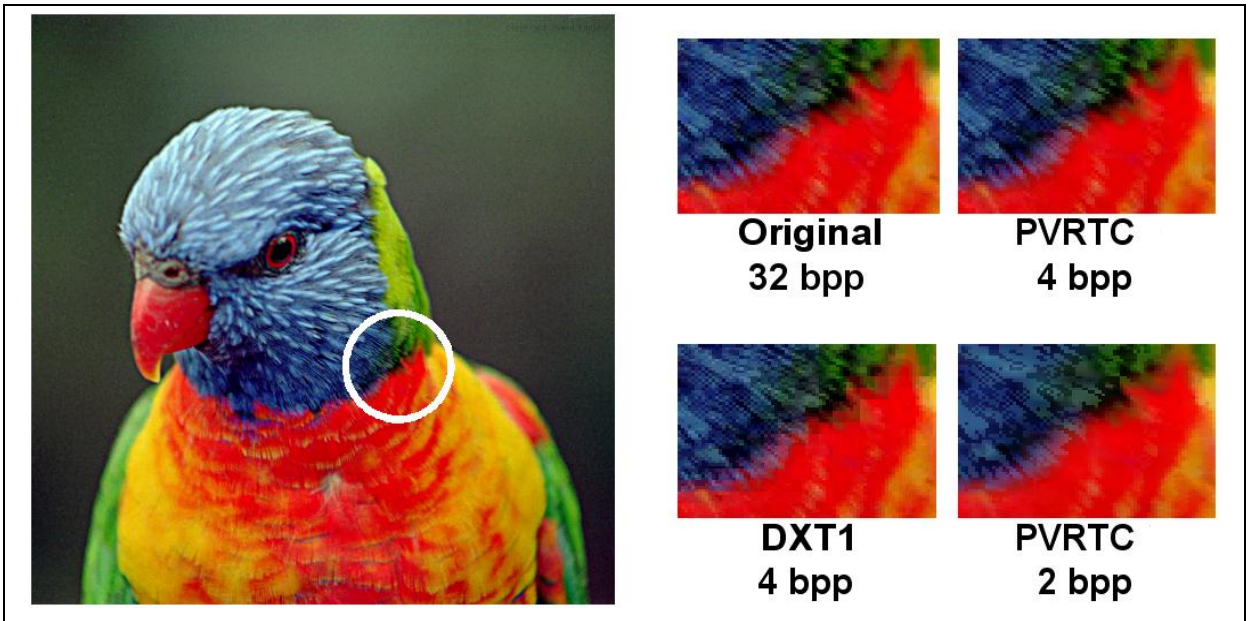


Figure 4-3 PVRTC1 vs. S3TC (DXT1): Photograph

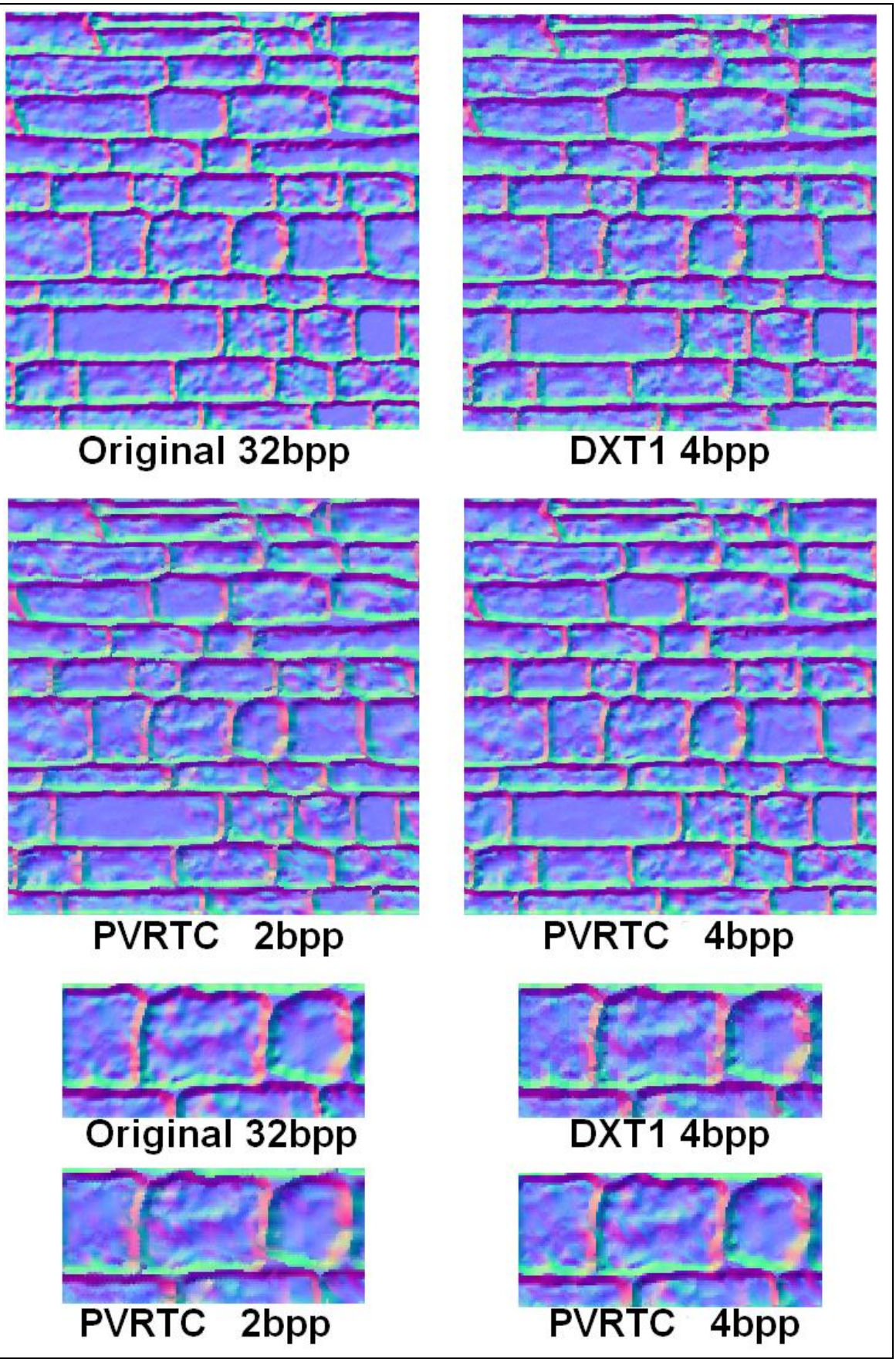


Figure 4-4 PVRTC1 vs. S3TC (DXT1): normal map

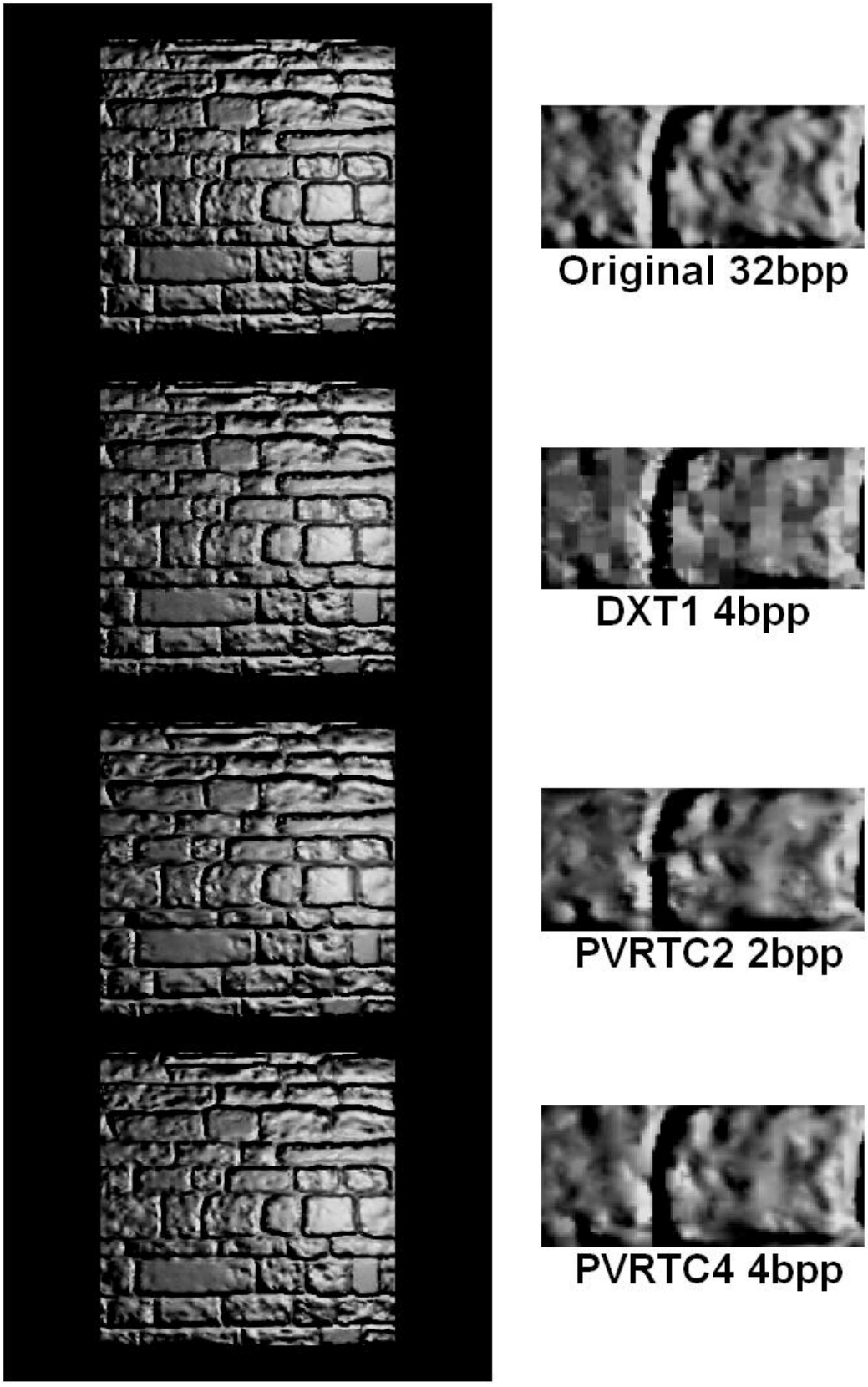


Figure 4-5 Usage of normal map from Figure 4-4

## 4.3. Decompression

Decompressing PVRTC can be generalised into three main steps:

1. Four adjacent data-words are fetched.
2. Bilinear interpolation is used to produce a 4x4 upscale of the two low resolution images (8x4 for PVRTC 2bpp).
3. The modulation data from the second half of each data-word is then used to linearly interpolate between the up-scaled versions of the low resolution images. This is illustrated in Figure 4-7 PVRTC decompression description.

### 4.3.1. PVRTC1 Data

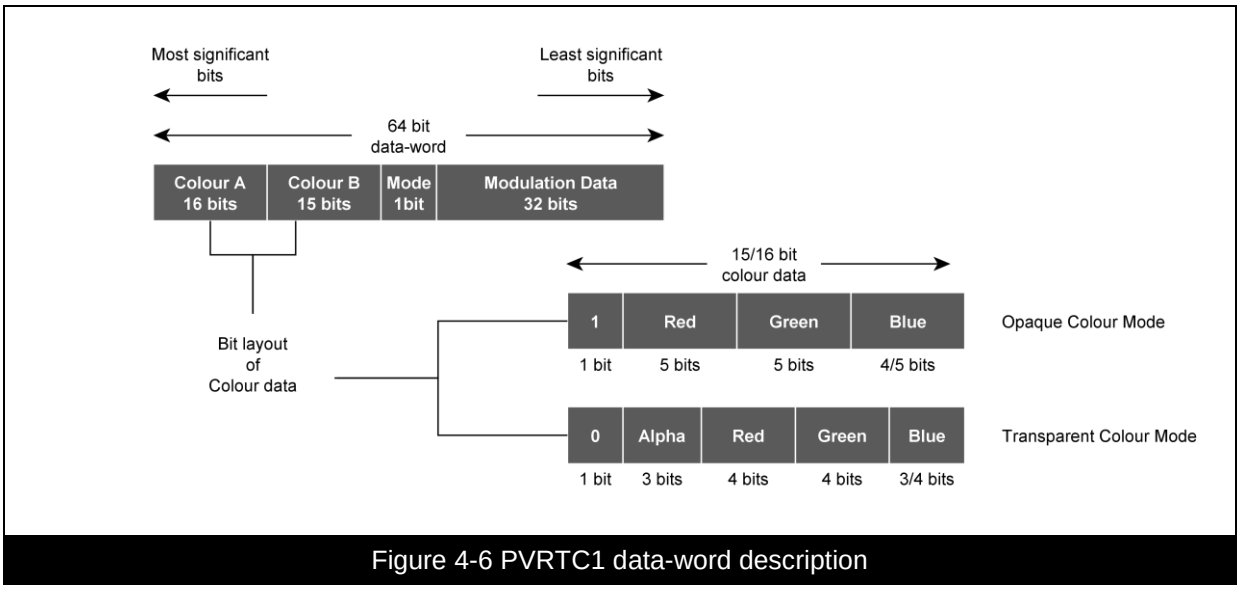


Figure 4-6 PVRTC1 data-word description

Colour information within a data-word is specified differently depending on whether the block contains alpha channel data or not. The first bit of each colour segment indicates whether that segment contains an alpha channel or not, with the remaining fourteen/fifteen bits containing three or four channels of data. In cases where alpha is at, or very close to 0% or 100% the mode bit will be set to '1', the colour segments will be laid out in Opaque Colour Mode and 'punch-through' alpha can be used.

**Modulation Data**

In the case of PVRTC 4bpp each pixel is encoded with 2 bits of modulation data; the weights associated with these bits are shown below:

**Table 1. Mode Bit = 1**

| Modulation Bits | Weight |
|---|---|
| 00 | 0 |
| 01 | 3/8 |
| 10 | 5/8 |
| 11 | 1 |

**Table 2. Mode Bit = 0**

| Modulation Bits | Weight |
|---|---|
| 00 | 0 |
| 01 | 4/8 |
| 10 | 4/8 – 'Punch-Through' |
| 11 | 1 |

As previously described, these weights are used to linearly interpolate the upscaled images using the following equation:

$$Output_{(t)} = A_{(t)} + Mod_{(t)}(B_{(t)} - A_{(t)})$$

In the case of PVRTC 2bpp modulation data corresponds to an area of 8x4 pixels. As with PVRTC 4bpp the interpretation of modulation data depends on the value of the Mode Bit.

With the Mode Bit set to '0' each pixel has one bit of modulation data; as only a single bit is present, either colour A or B will be chosen, with no intermediate values available. With the Mode Bit set to '1' every alternate pixel is encoded with 2 bits of modulation data. The modulation values for the remaining pixels are produced by interpolation; this allows intermediate values to be used as per PVRTC 4bpp.

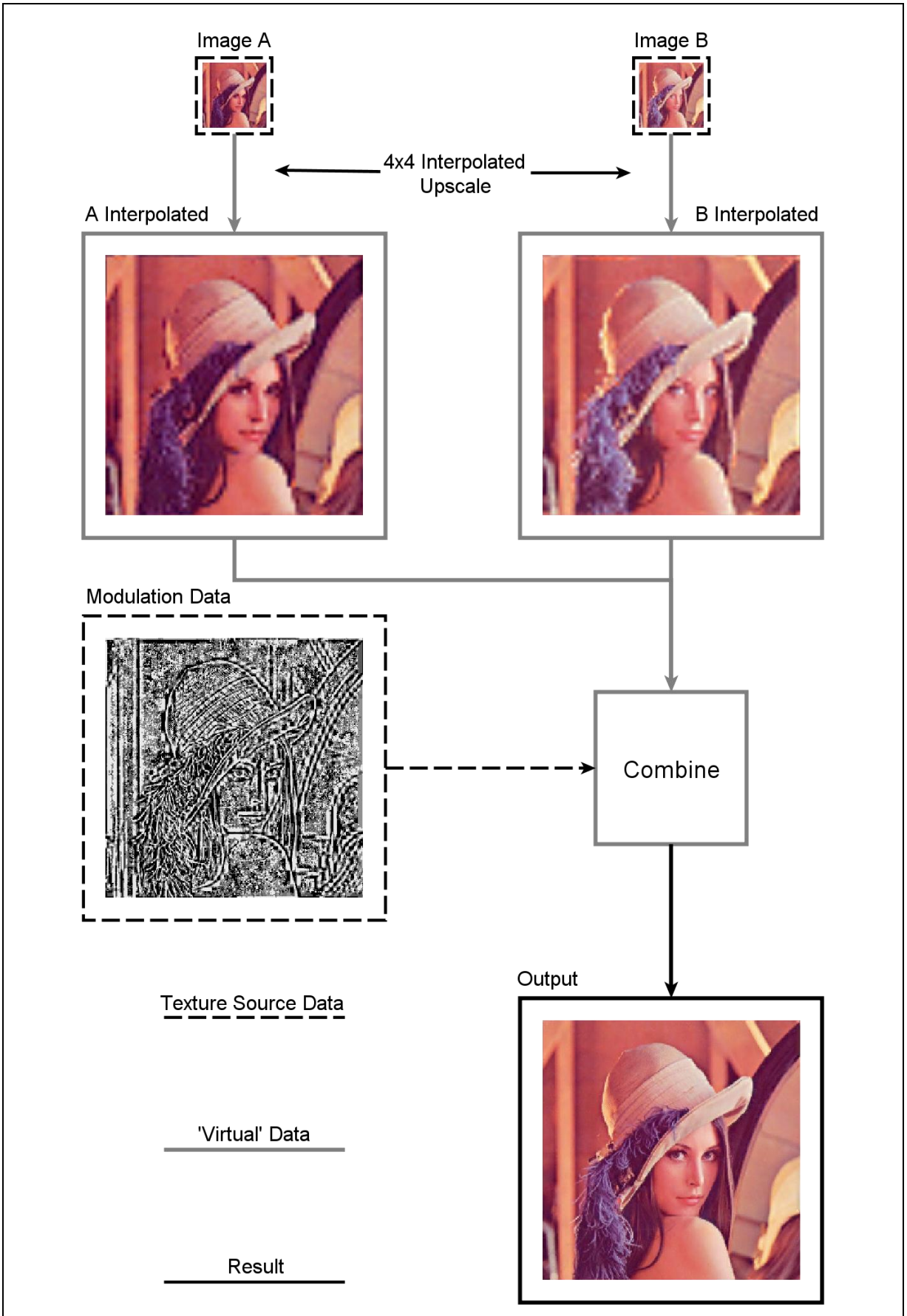


Figure 4-7 PVRTC decompression description

### 4.3.2. PVRTC2 Data

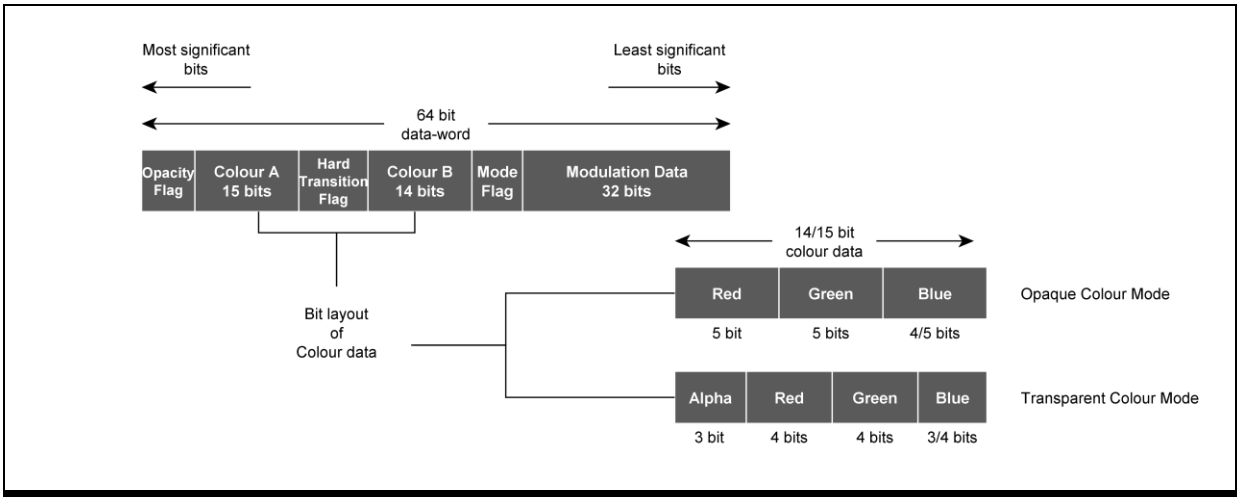


Figure 4-8 PVRTC2 data-word description

The data-word layout of PVRTC2 is very similar to that of PVRTC1, although two main differences exist:

- Only a single ‘Opacity Flag’ is used for the entire data-word as opposed to an ‘Opacity Flag’ per colour segment.
- A ‘Hard Transition Flag’ now exists.

When the single ‘Opacity Flag’ is set to ‘1’ both colour segments are opaque; when it is ‘0’ both segments must be interpreted in the Transparent Mode shown in Figure 4-8 PVRTC2 data-word description.

The ‘Hard Transition Flag’ is used, in conjunction with the ‘Mode Flag’, to provide additional options for interpreting the modulation data. This is in addition to the two available in PVRTC1.

The four options are listed in Table 3.

**Table 3. Interpreting modulation data**

| Modulation Flag | Hard Transition Flag | Modulation Mode |
|---|---|---|
| 0 | 0 | Standard Bilinear |
| 1 | 0 | Punch-through Alpha |
| 0 | 1 | Non-interpolated |
| 1 | 1 | Local Palette Mode |

The purpose of these two additional options, ‘non-interpolated’ and ‘local palette mode’, is to improve the quality of PVRTC even further when dealing with textures that contain colour discontinuities. In cases where textures do not tile ‘seamlessly’ (i.e. discontinuities exist on along the image boundary), ‘non-interpolated mode’ is used. In cases where a texture contains local areas which use a variety of very distinct colours ‘local palette mode’ is used.

# 5. Related Materials

**Training Courses**

- IntroducingPVRTools onwards

**Software**

- PVRTexTool
- PVRTexLib
- PVRTools

**Documentation**

- PVRTexTool User Manual
- PVRTexLib User Manual
- PVRTools Documentation
- PVR Texture Compression Whitepaper

# 6. Contact Details

For further support, visit our forum:
http://forum.imgtec.com

Or file a ticket in our support system:
https://pvrsupport.imgtec.com

To learn more about our PowerVR Graphics SDK and Insider programme, please visit:
http://www.powervrinsider.com

For general enquiries, please visit our website:
http://imgtec.com/corporate/contactus.asp