

# PVRTC Texture Compression

## Usage Guide

Copyright © 2008, Imagination Technologies Ltd. All Rights Reserved.

This publication contains proprietary information which is protected by copyright. The information contained in this publication is subject to change without notice and is supplied 'as is' without warranty of any kind. Imagination Technologies and the Imagination Technologies logo are trademarks or registered trademarks of Imagination Technologies Limited. All other logos, products, trademarks and registered trademarks are the property of their respective owners.

Filename : PVRTC Texture Compression.Usage Guide.mht  
Version : 1.1f (PowerVR SDK 2.03.23.1153)  
Issue Date : 10 Jun 2008  
Author : PowerVR

## Contents

1.	<b>Texture Compression</b> .....	4
1.1.	Why Texture Compression? .....	4
1.2.	Reduced Memory footprint .....	4
1.3.	Higher performance .....	4
1.4.	Reduced power consumption .....	4
2.	<b>PVRTC</b> .....	5
2.1.	What is PVRTC? .....	5
2.2.	How does PVRTC work? .....	5
2.2.1.	PVRTC Block description .....	5
2.2.2.	Decompressing PVRTC .....	6
2.2.3.	Modulation data .....	7
2.3.	Benefits of PVRTC .....	8
3.	<b>PVRTC Examples and comparisons</b> .....	9
4.	<b>Getting the best from PVRTC</b> .....	12
4.1.	Non-tiling textures .....	12
4.1.1.	Example .....	12
4.2.	Sky boxes .....	14
4.2.1.	Example .....	16
4.3.	Texture pages .....	18
4.3.1.	Example .....	18

## List of Figures

Figure 1	PVRTC block description .....	5
Figure 2	A and B colour data arrangement within a PVRTC block .....	6
Figure 3	PVRTC Decompression .....	7
Figure 4	Modulation weights for PVRTC4 .....	8
Figure 5	PVRTC4 vs S3TC (DXT1) comparison with skybox texture .....	9
Figure 6	PVRTC vs S3TC (DXT1) comparison with a photograph .....	9
Figure 7	PVRTC vs S3TC (DXT1) Comparison with a Normal Map .....	10
Figure 8	Usage of compressed Normal Maps from Figure 7 showing DOT3 Bump Mapping .....	11
Figure 9	Original non-tiling Texture .....	12
Figure 10	PVRTC Compression .....	12
Figure 11	Top and Bottom Rows of Original Image .....	13
Figure 12	Top and Bottom Rows of PVRTC Image .....	13
Figure 13	Resized image with Border Area .....	13
Figure 14	Border Area with Mirrored Data .....	13
Figure 15	PVRTC compressed image with Border containing Mirror Data .....	13
Figure 16	Final Result .....	13
Figure 17	Valid Top and Bottom Rows of PVRTC Image with Border Area fix .....	14
Figure 18	Skybox Overview. Zneg is implied to be at the very front of the cube .....	14
Figure 19	Resized cube map faces with border area to be filled in .....	15
Figure 20	Single Cube Face Texture with its neighbours .....	15
Figure 21	Single Cube Face with border area filled in with correct data .....	16
Figure 22	Example skybox geometry .....	17
Figure 23	Skybox using PVRTC without Border .....	17
Figure 24	Skybox using PVRTC with Border .....	17

---

Figure 25 Normal Map Texture for a human character .....	19
Figure 26 Border Expansion – only border shown for clarity.....	20
Figure 27 Final Image with full Border Expansion.....	21

## List of Equations

Equation 1: Using modulation data to interpolate between upscaled version of A and B.....8

## 1. Texture Compression

### 1.1. Why Texture Compression?

Modern graphics applications like games often need to use a large amount of textures in order to represent a scene with satisfying quality. Compressing textures offer significant advantages over uncompressed ones; the remainder of this section will examine those in further details.

### 1.2. Reduced Memory footprint

Texture compression reduces memory footprint. This allows game to fit all required textures in a constrained amount of texture memory, or to use larger (or more) textures for the same memory budget for extra quality.

Those savings in memory requirements are very useful for embedded systems where memory is scarce.

### 1.3. Higher performance

Because compressed textures occupy less memory, there is less data to transfer from memory to the chipset, allowing significant bandwidth savings to be made. Those bandwidth savings can significantly increase performance in situations where memory transfers are the bottleneck.

Most texture compression schemes are block-based, which enable the 3D hardware to optimize the fetching of texel data, considerably improving memory cache effectiveness in the process.

### 1.4. Reduced power consumption

Memory accesses are the main cause of increased power consumption, which is a key feature of any embedded systems featuring real-time graphics processing capabilities. The bandwidth savings and better cache usage resulting from texture compression both contribute to less memory accesses being made, therefore reducing power consumption requirements.

## 2. PVRTC

### 2.1. What is PVRTC?

PVRTC is PowerVR's texture compression scheme. PVRTC boasts very high quality for competitive compression ratios: 4 bits per pixel (PVRTC4) and 2 bits per pixel (PVRTC2). Those represent savings in memory footprint of 8:1 (PVRTC4) and 16:1 (PVRTC2) compared to 32 bits textures. Like most texture compression schemes PVRTC is block-based, however it also uses data from adjacent blocks in order to reconstruct the original texture with more fidelity. This represents a considerable visual quality advantage compared to compression techniques using a single block to reconstruct the totality of the texture data within that block.

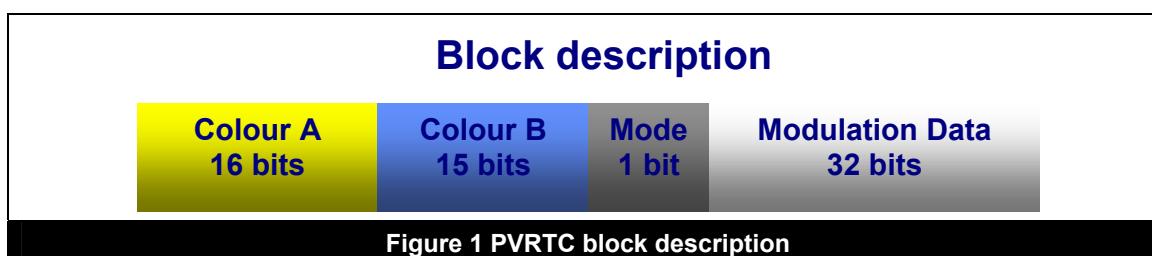
PVRTC support both opaque (RGB) and translucent (RGBA) textures (unlike other format e.g. S3TC that require a dedicated format to support full alpha variants). All data is encoded on a per-block basis, not for the entire texture, which has the benefit that fully opaque blocks will not be impacted with the encoding of unnecessary alpha information.

### 2.2. How does PVRTC work?

PVRTC uses a sophisticated amplitude modulation scheme to compress textures. It encodes the data as two low-resolution images along with a full resolution, low bit precision modulation signal.

#### 2.2.1. PVRTC Block description

PVRTC is block-based; each block is 64 bit in size and corresponds to 4x4 pixels in the case of PVRTC4, and 8x4 pixels in the case of PVRTC2. A block contains two colours A and B, a boolean specifying modulation mode, and modulation data for the entire block. Figure 1 illustrates the contents of a PVRTC block.



Colour information within a block is specified differently depending on whether the block contains translucent data information or not. The first bit of each colour A and B indicates what arrangement to use. In the case of colour B only 15 bits are available therefore the blue channel is encoded with less bits (blue is chosen because the human eye is less sensible to blue variations than other colours). Figure 2 illustrates the two possible colour data arrangements.

## Opaque Colour Format



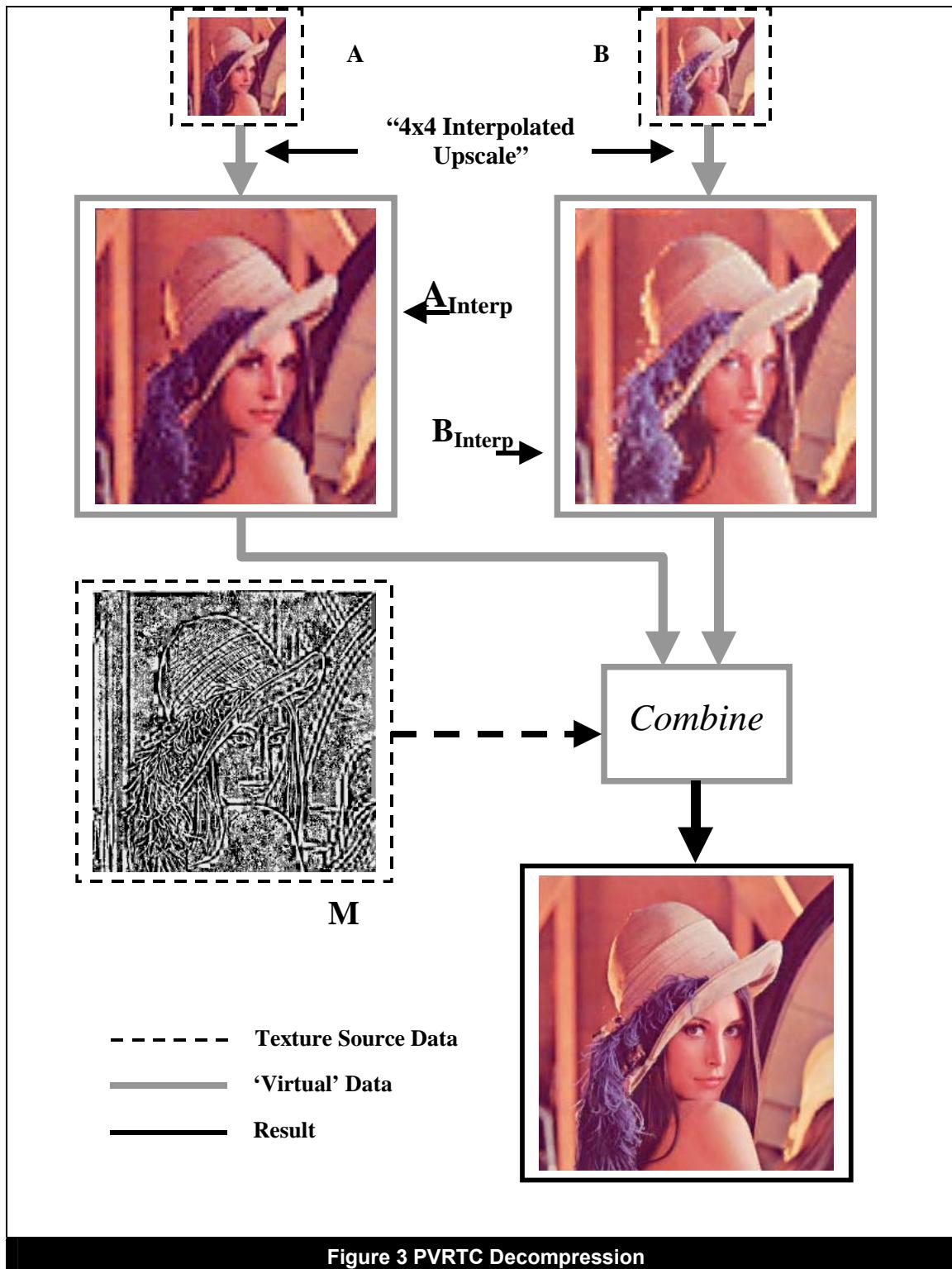
## Translucent Colour Format



Figure 2 A and B colour data arrangement within a PVRTC block

### 2.2.2. Decompressing PVRTC

When decompressing PVRTC, four adjacent blocks are fetched. The colours of each block are bilinearly interpolated in order to produce 4x4 up-scaled versions of A and B. The modulation data is then used to linearly interpolate between the up-scaled versions of A and B. Figure 3 illustrates the steps involved in decompressing PVRTC.



**Figure 3 PVRTC Decompression**

### 2.2.3. Modulation data

Modulation data is encoded differently depending on the mode set in each block. In the case of PVRTC4 each pixel is encoded with 2 bits ( $4 \times 4 \times 2 = 32$  bits), and the modulation mode bit selects the

weights corresponding to the four possible cases of modulation data. Those weights are used in the linear interpolation of A and B to produce the final pixel colour according to Equation 1.

$$\text{Colour} = A_p + M_p(B_p - A_p)$$

**Equation 1: Using modulation data to interpolate between upscaled version of A and B**

In modulation mode 1 one of the modulation combination sets alpha to 0; this can be used for punch-through textures. Figure 4 shows the weights for the two PVRTC modulation cases.

<b>MODULATION MODE 0</b>		<b>MODULATION MODE 1</b>	
<b>Code</b>	<b>Weight</b>	<b>Code</b>	<b>Weight</b>
<b>00</b>	<b>0</b>	<b>00</b>	<b>0</b>
<b>01</b>	<b>3/8</b>	<b>01</b>	<b>4/8</b>
<b>10</b>	<b>5/8</b>	<b>10</b>	<b>4/8, <math>\alpha=0</math></b>
<b>11</b>	<b>1</b>	<b>11</b>	<b>1</b>

**Figure 4 Modulation weights for PVRTC4**

In the case of PVRTC2 modulation data corresponds to an area of 8x4 pixels. As with PVRTC4 the interpretation of modulation data depends on the modulation mode set in the PVRTC block.

In modulation mode 0, each modulation pixel is one bit and corresponds to the weights to use during the linear interpolation of A and B. As only a single bit is present either A or B will be chosen, with no intermediate values available.

In modulation mode 1, each other modulation pixel is encoded with 2 bits and other values are produced by interpolation, allowing intermediate values to be used.

For more information about PVRTC readers are invited to read the following white paper that describes and explains PVRTC in greater details:

<http://www.powervr.com/Downloads/Factsheets/PVRTextureCompression.pdf>

### 2.3. Benefits of PVRTC

PVRTC boasts attractive compression ratios of 8:1 (PVRTC4) and 16:1 (PVRTC2) that considerably reduce texture memory requirements of graphic applications. Both modes support an alpha channel, which enable developers to effectively compress a greater variety of textures with the same formats. PVRTC compares favourably to other compression formats like S3TC; the inclusion of four adjacent blocks in the decompression process allow PVRTC to represent smooth gradients, resulting in higher quality images. The next section focuses on examples of PVRTC results and how they compare to S3TC.

### 3. PVRTC Examples and comparisons

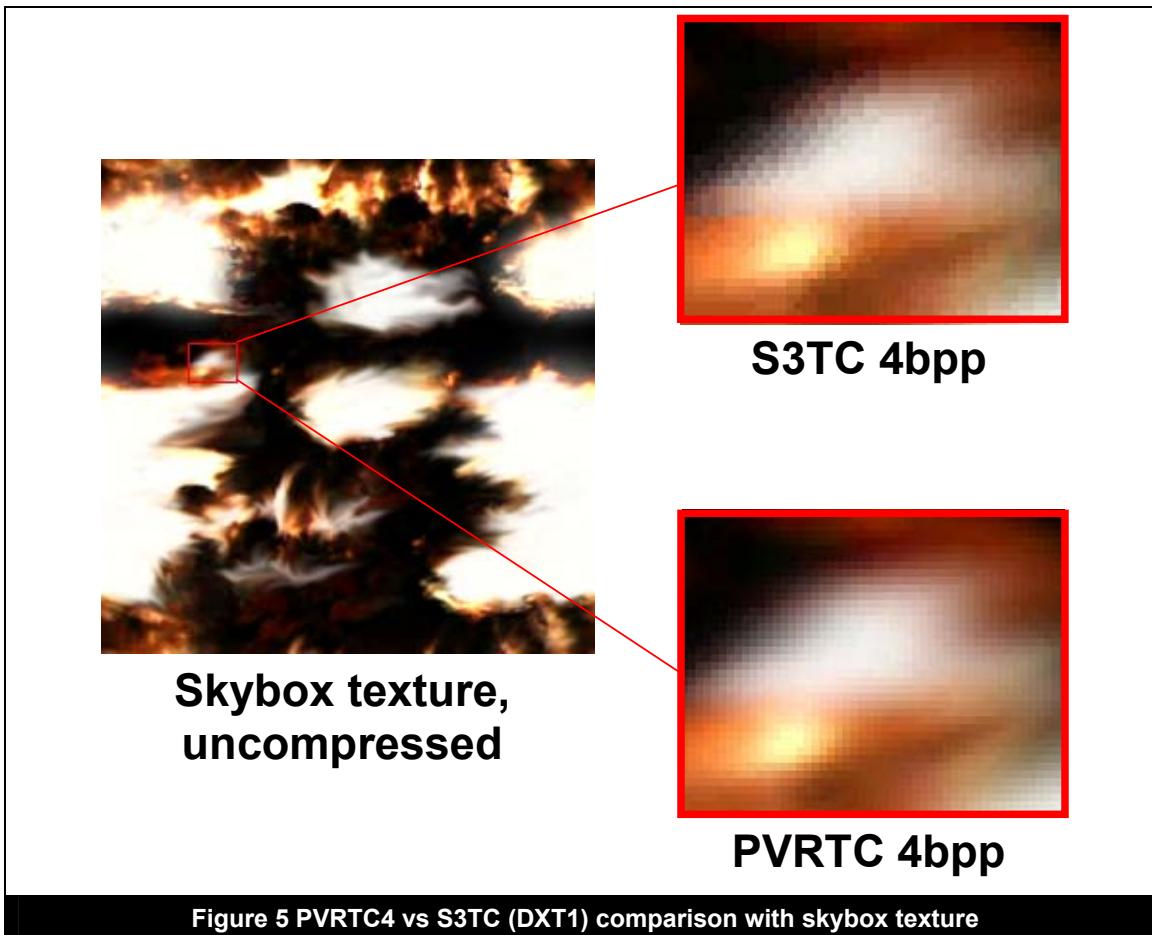


Figure 5 PVRTC4 vs S3TC (DXT1) comparison with skybox texture

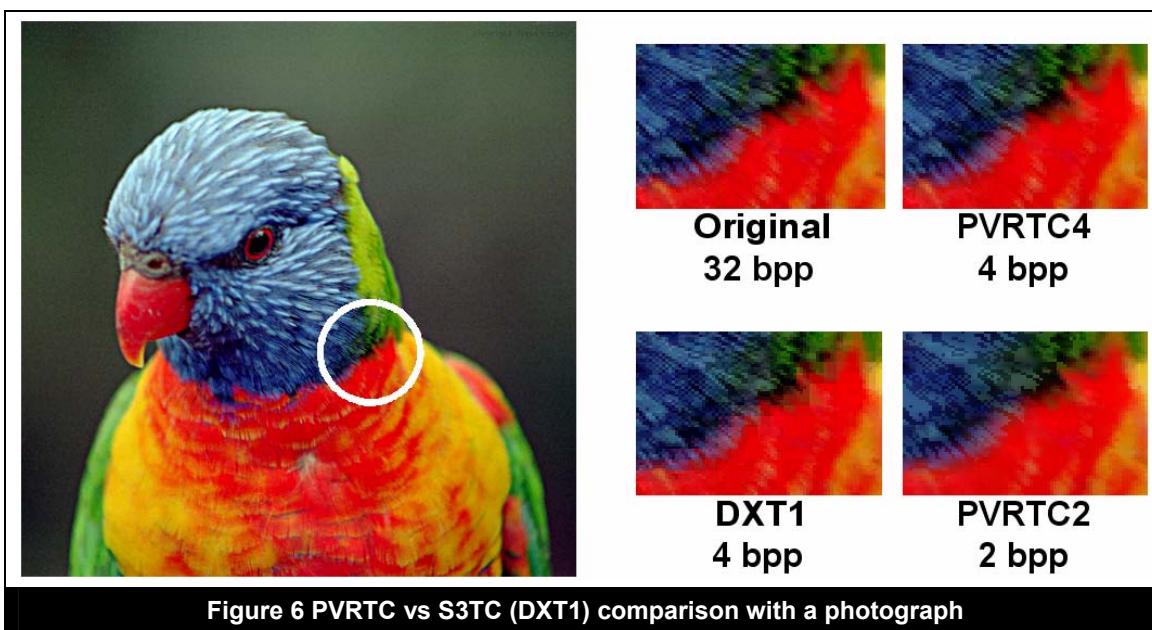


Figure 6 PVRTC vs S3TC (DXT1) comparison with a photograph

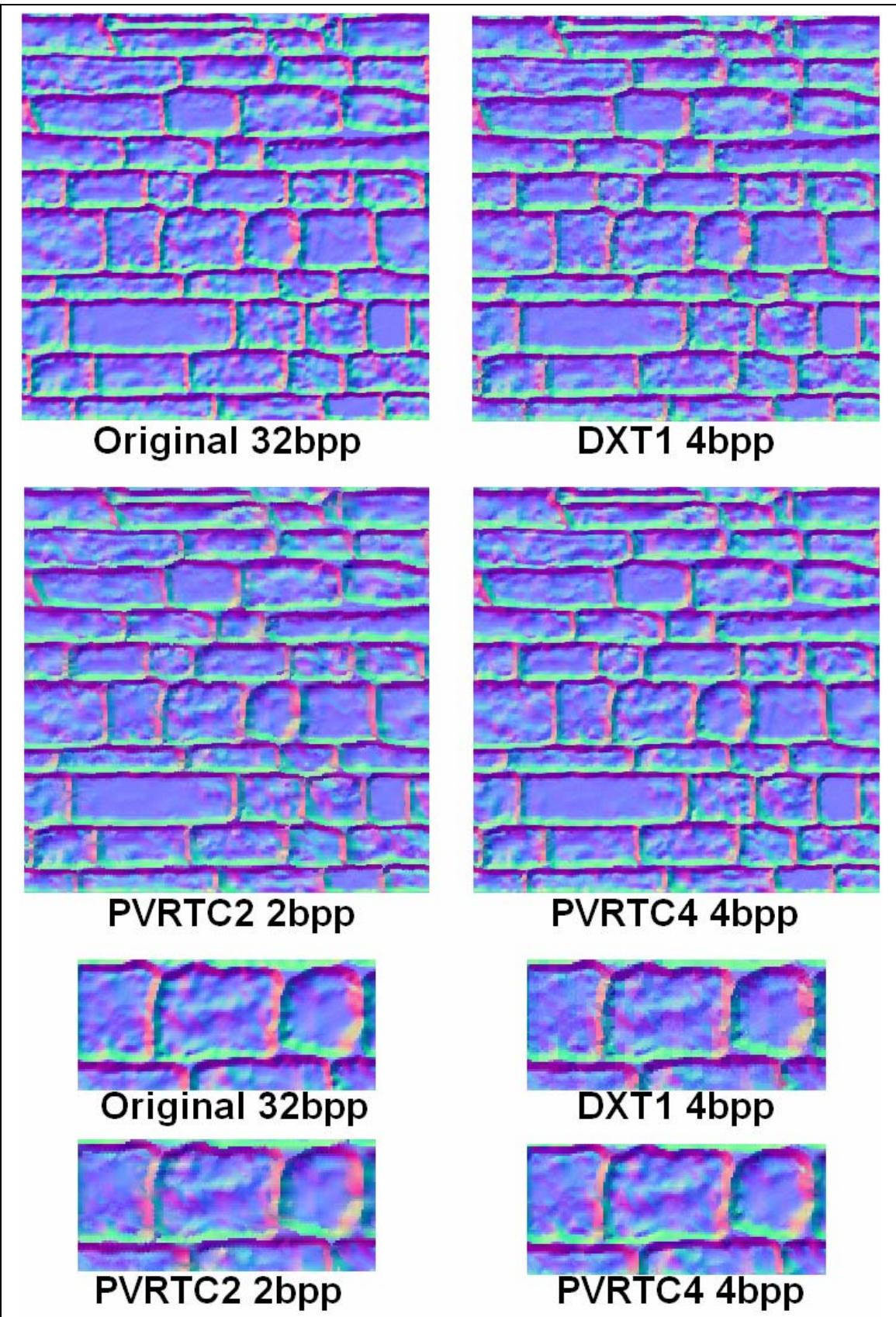
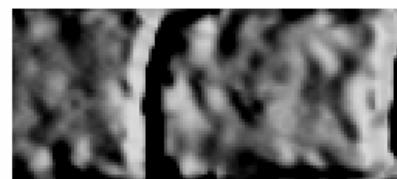
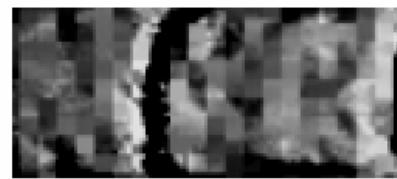


Figure 7 PVRTC vs S3TC (DXT1) Comparison with a Normal Map



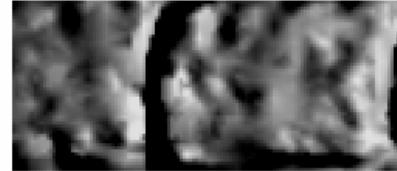
Original 32bpp



DXT1 4bpp



PVRTC2 2bpp



PVRTC4 4bpp

Figure 8 Usage of compressed Normal Maps from Figure 7 showing DOT3 Bump Mapping

## 4. Getting the best from PVRTC

### 4.1. Non-tiling textures

For PVRTC texture data is assumed to be continuous across texture edges, which is a common case in graphic applications where various material textures like rock, brick, grass etc. are tiled together to represent high texture detail for a larger area. This can occasionally result in minor compression artefacts along the edges of texture data that does not tile.

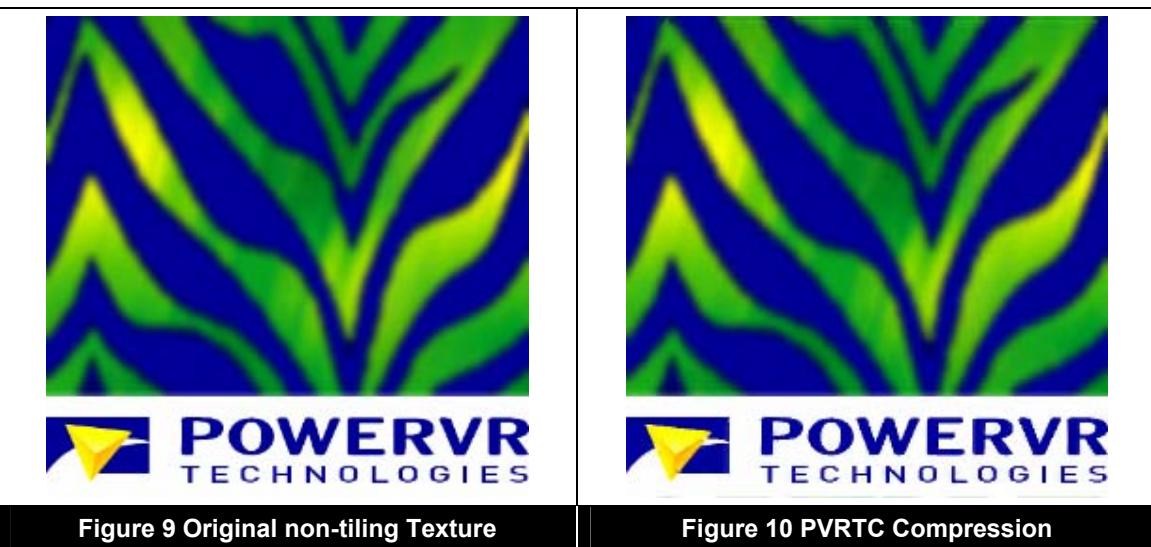
Whenever results are deemed unsatisfying, this issue can easily be resolved by adding a border around the original texture data. This border will then absorb any possible artefacts related to tiling. This technique is illustrated step-by-step below:

1. Create space for adding the border
  - a. Resize the original texture to a size equal to (SizeX-(2xBorderPixels)) x (SizeY-(2xBorderPixels)). For a 512x512 texture with a 4 pixel border this would be (512-2x4)x(512-2x4) = 504x504.
  - b. Centre the resized texture data
2. Fill in the border pixels
  - a. Mirror the resized texture data from the image into the top and bottom area of the border.
  - b. Mirror the result of the previous step into the left and right area of the border
3. Adapt texture coordinates to only use the central non-border area of the generated texture.

The above operation influences the actual texture resolution and the resize operation can result in some minor blurring. This can be avoided by creating artwork directly at the required resolution.

#### 4.1.1. Example

The following images illustrate PVRTC compression of a non-tiling texture. Figure 9 shows the original uncompressed texture data. Figure 10 shows the result of PVRTC compression done on this texture.



The minor artefacts introduced can be seen when comparing the top and bottom four pixels of the images as illustrated in Figure 11 and Figure 12.

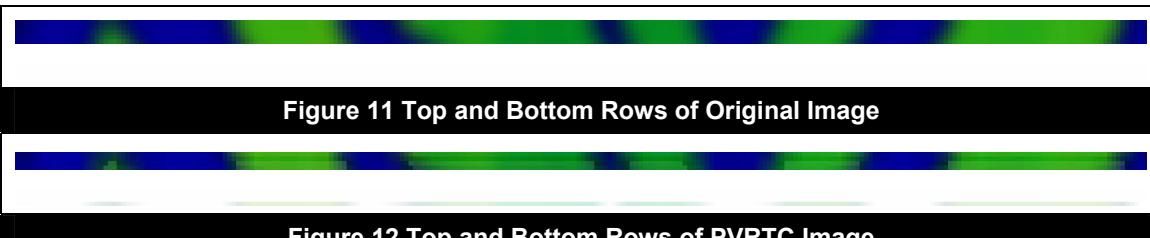


Figure 13 shows the resized texture with a black 4-pixel border area. Figure 14 shows the border filled with mirrored data from the central texture data.

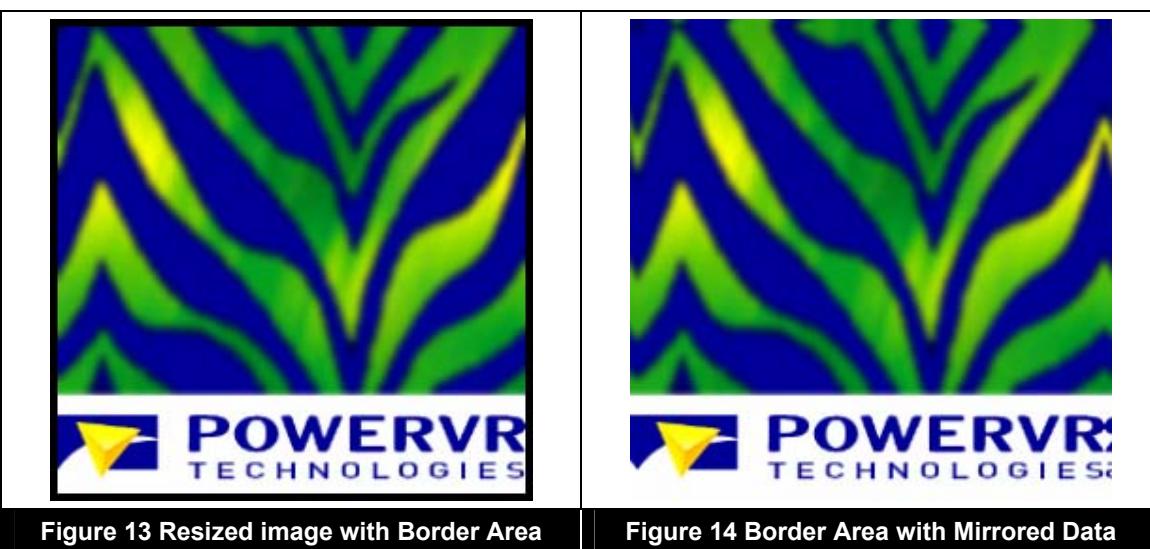


Figure 15 shows the PVRTC result of the texture with a border area filled with mirrored texture data. Note how the artefacts are now contained within the border area. Figure 16 shows the final result by applying the texture to a polygon with texture coordinates that only map to the central (non-border) region of the texture.

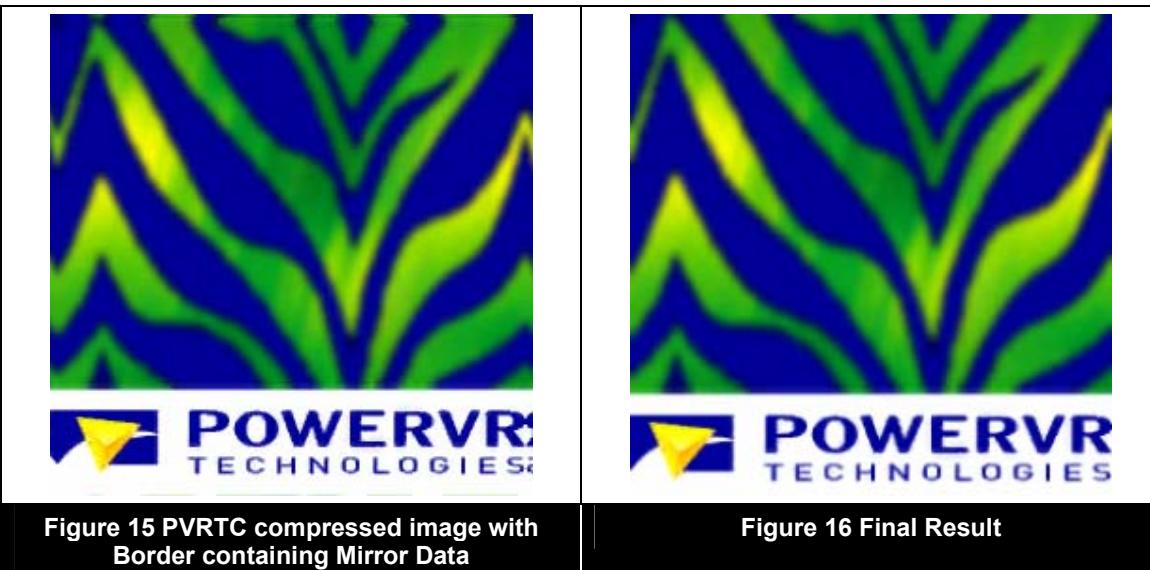


Figure 17 shows the Top and Bottom rows of the valid area of the PVRTC Image with border area fix. Note the lack of artefacts when comparing with Figure 11.

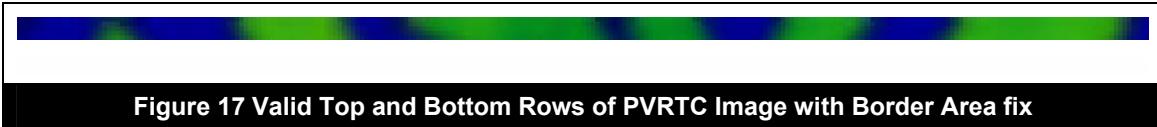


Figure 17 Valid Top and Bottom Rows of PVRTC Image with Border Area fix

## 4.2. Sky boxes

Sky Box textures are a collection of six textures which are mapped onto a cube to represent the sky or background of a 3D scene. These textures are a special type of non-tiling textures with a very specific tiling behaviour from one texture to another. When filling in the border pixels this special behaviour needs to be taken into account to obtain the best possible end-result in combination with PVRTC. The specific tiling behaviour is illustrated in Figure 18 for one of the faces of the skybox. Sky boxes have six textures, specifically two textures for each major axis: Xpos and Xneg, Ypos and Yneg, Zpos and Zneg.

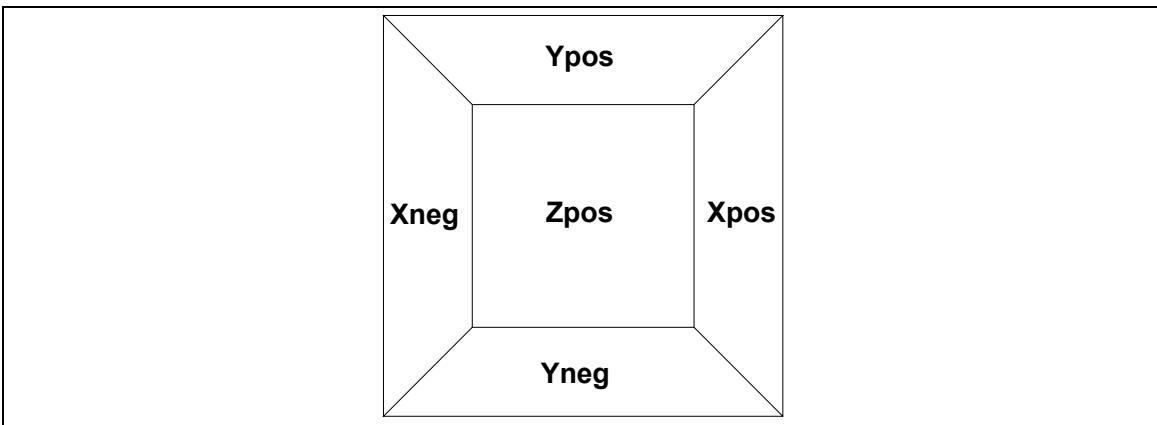


Figure 18 Skybox Overview. Zneg is implied to be at the very front of the cube.

To remove potential tiling or texture filtering artefacts the border regions of each texture face need to be filled with the correct data from the neighbouring textures. For instance the borders of the Zpos texture get the top data from the Ypos texture, the bottom border gets data from the Yneg texture, the left border gets data from the Xneg texture and finally the right border gets pixels from the Xpos texture. An overview of this procedure is shown in Figure 19, Figure 20 and Figure 21

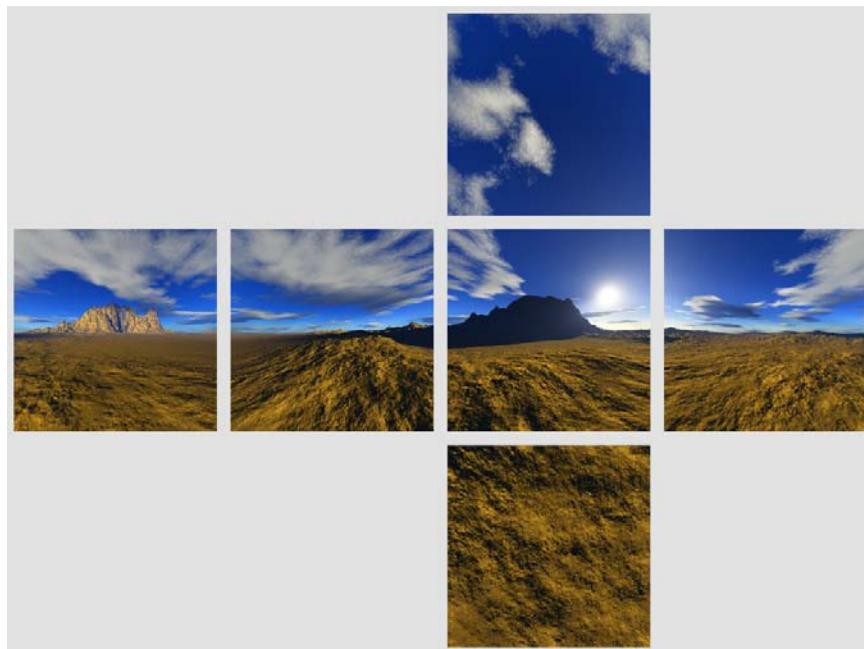


Figure 19 Resized cube map faces with border area to be filled in.

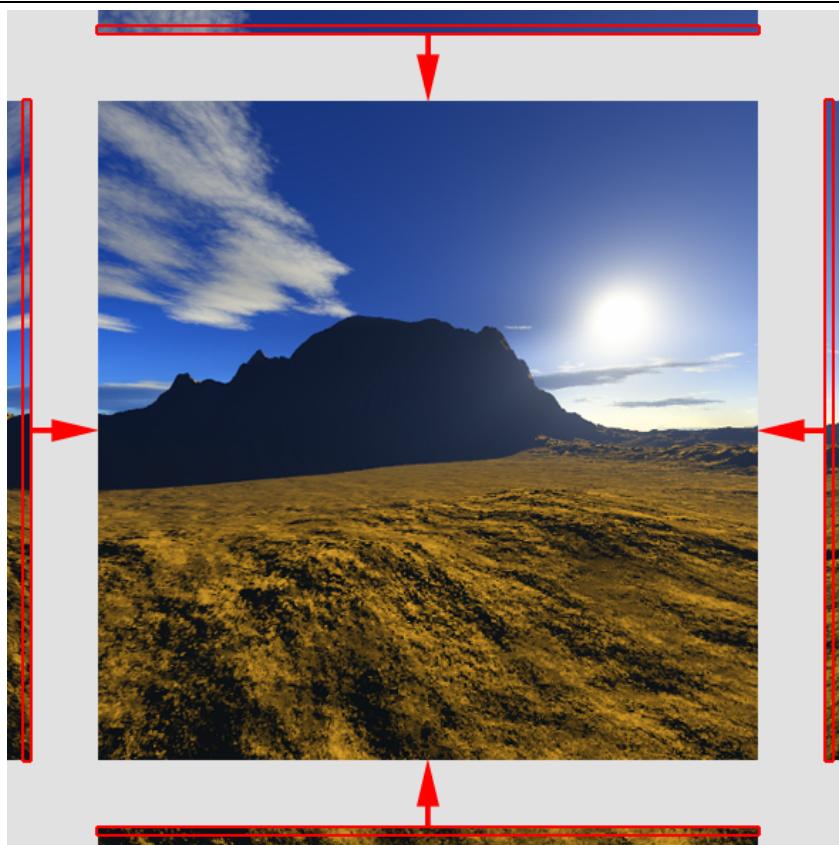


Figure 20 Single Cube Face Texture with its neighbours.

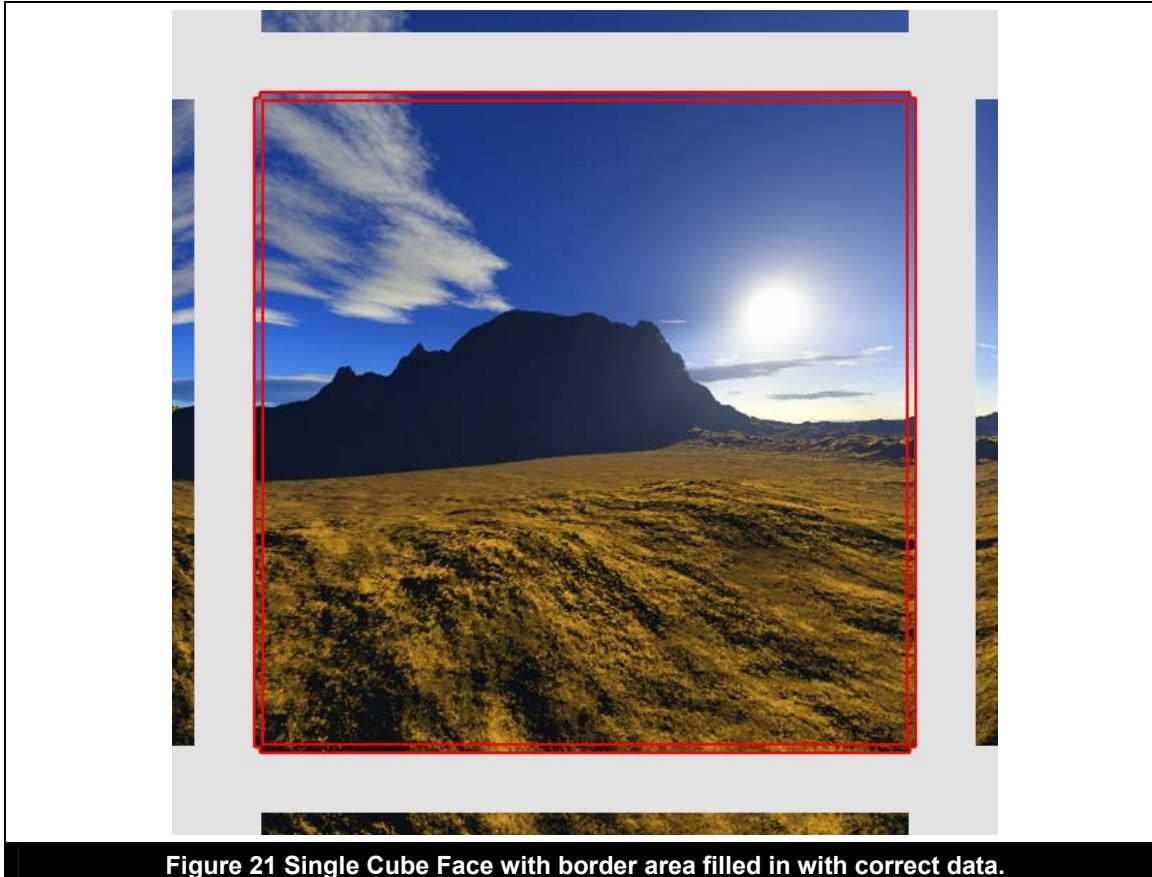


Figure 21 Single Cube Face with border area filled in with correct data.

When mapping these textures with border onto the final skybox cube geometry the texture coordinates need to be scaled and shifted such that only the central non-border area is mapped. As well as matching edge pixels, this technique has the extra advantage of achieving correct texture filtering access the skybox faces.

This same technique can be utilised for other types of textures with a specific well-known tiling behaviour where the border area can be filled in with the correct data from neighbouring textures.

#### 4.2.1. Example

Figure 22 shows an overview of the example skybox. Figure 23 shows the environment using PVRTC compressed textures with non-corrected borders. Figure 24 shows the environment using PVRTC compressed textures with border padding; note the perfect continuity between the cube surfaces.

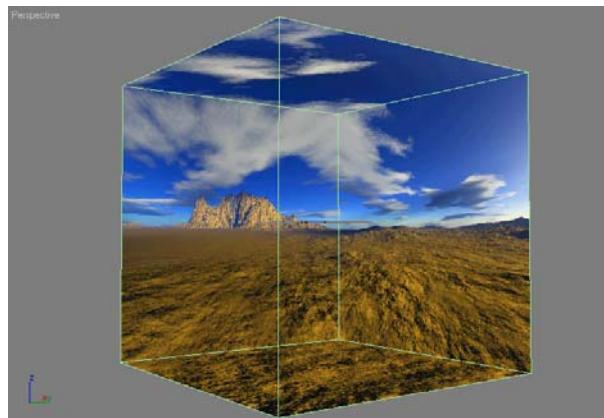


Figure 22 Example skybox geometry



Figure 23 Skybox using PVRTC without Border



Figure 24 Skybox using PVRTC with Border

### 4.3. Texture pages

To reduce the number of draw calls and render state changes applications often group multiple textures into a single larger texture - this concept is known as a texture page (sometimes called a texture atlas). A similar result can be seen from semi-automated unwrapping tools which unwrap/flatten the geometry of an object into a texture. For example the upper body is mapped into the top half of a texture while the bottom half of a characters body is mapped into the lower half of a texture.

Both these techniques often use a background colour that surrounds the various texture elements contained within the texture – in most cases this colour is black and bears no resemblance to the true texture contents. This lack of relation between background and actual contents is an unnecessary complication for a texture compressor since better results are achieved when a texture has colour continuity (no sharp transitions). Because the background pixels are not actually used as part of the texture mapping (excluded through the texture coordinates) it is thus possible and advisable to change the background area pixels into a colour that results in optimal texture compression results.

For PVRTC the best way to fill in these background pixels is to use a colour that is continuous with the actual colours used within the textured areas. This can be achieved by expanding the true texture contents out into the background area of the texture.

#### 4.3.1. Example

Figure 25 shows a normal map for a human character. Note how the whole character is mapped using a single texture but how different body parts are spread out over the whole texture area using a black background colour.



Figure 25 Normal Map Texture for a human character

Figure 26 shows the border grown from the Normal Map seen in Figure 25; this can easily be done using any image processing package and/or specific tools. Adding this border results in smooth continuous texture contents for the compressor to work on, there is no odd background colour to add to the compression complexity and hence better results can be achieved. Figure 27 shows the texture with borders added.





Figure 27 Final Image with full Border Expansion