Table of Contents

1 Introduction

With normal and bump mapping becoming increasingly frequently used techniques in modern real-time graphics applications it becomes clear that some approach is required to provide effective compression for these texture maps. Developers have been commenting that managing the size of their normal maps is becoming a serious problem – these are usually stored in a 32-bit ARGB format, and at high resolutions to provide sufficient surface detail, and in some upcoming titles represent the majority of the texture maps.

Palletized texture maps have been used in the past to store normal information, but this format is becoming marginalized - it has been overtaken by DXT compression for general image compression as DXT usually gives higher quality and has friendlier characteristics for hardware decompression. Normal map compression is one of the few exceptions where palettisation can still give higher quality than DXT compression.

The challenge is therefore to investigate whether creative uses of existing compression methods can preserve equivalent or better overall quality than palettisation, while using techniques that can be supported across all hardware. Largely this limits us to either reducing bit depth or using one of the existing DXT block-based compression formats.

This document will explore these possibilities, and can hopefully be used as a reference to evaluate alternatives for ISVs.

2 Normal map representation

When examining possible formats for normal maps we should examine various possible formats for the normal representations.

2.1 Space

Normal maps are usually stored as a representation in one of two spaces – either in model space, or in the local tangent space of each triangle. In current applications the tangent space format is becoming more prevalent. Normal maps specified in model space must generally have three components since all directions must be representable. Normal maps in tangent space can be specified with only two components – since the tangent space of the triangle describes a hemispherical region the third component can be derived in a pixel shader.

2.2 Bit depth

Current normal maps are usually stored with a resolution of 8 bits per component. In the future we expect higher resolution formats to be used to avoid some quantisation errors and improve overall quality.

2.3 Normalized representation

Normal maps are usually stored in a normalized form, where the magnitude of the represented vector is always 1. This creates fewer problems with aspects such as MIP mapping, but decreases the number of possible directions available from a given type of texture (given any fixed resolution of components only a small percentage of the possible texel values are normalized). A denormalized representation with an implicit normalization step can give some additional directional accuracy at the cost of additional renormalizing work in the pixel shader.

However, some developers may use the side effects of mipmap filtering to deliberately denormalise the bump vectors in the distance. Although this behavior is actually incorrect it has some desirable side effects as it reduces aliasing on distant objects (if the normals remain normalized then as the lighting equation jumps between normal values pointing in effectively random directions on distant objects specular highlights will tend to pop in and out.) In producing alternative compression formats we also have to consider this problem and possible solutions.

2.3.1 888 Colorspace

This table shows the number of available model-space normals in an 888 colorspace with various accuracy tolerances

Percentage from normalized	Number of available Normals	Percentage of Colorspace
1.0	522329	3.11
2.0	1035699	6.17
3.0	1546033	9.22
4.0	2042853	12.18
5.0	2535215	15.11

2.3.2 565 Colorspace

This table shows the number of available model-space normals in a 565 colorspace with various accuracy tolerances

Percentage from normalized	Number of available Normals	Percentage of Colorspace
1.0	1015	1.55
2.0	2069	3.16
3.0	3138	4.79
4.0	3969	6.06
5.0	5032	7.68

3 Possible hardware normal map formats

On modern hardware we may have many available formats for storing normal maps at varying levels of accuracy. Looking at the specification for DX9 we can see that some of the possible formats are:

- D3DFMT_Q8W8V8U8 / D3DFMT_A8R8G8B8
- D3DFMT_A2R10G10B10
- D3DFMT_R5G6B5
- D3DFMT_G16R16 / D3DFMT_V16U16
- D3DFMT_A8L8 / D3DFMT_V8U8
- D3DFMT_CxV8U8
- D3DFMT_DXTn

I have listed both the signed and unsigned versions of these formats for completeness. In this document we will be concentrating on the 'compressed' representations (ie. using less than 32 bits), and using normalized forms for the moment.

3.1 5_6_5

In this format it can be seen that the number of representable normals is significantly lower than in an 888 colorspace. We have roughly $1/500^{th}$ of the number of normals at any given level of desired accuracy, so although the general representation of direction is good and consistent we would expect to see banding artifacts developing in the shading equation over any areas of shallow curvature. There is no signed version of this format available in D3D, although signed versions may be exposed through other APIs.

3.2 8_8

These formats are only useful for normal maps in tangent space. Their precision is equivalent to the 8_8_8_8 formats, but they use only half the storage. Since the tangent space calculation needs only a few instructions in the vertex shader this representation may be preferable as a general case. Denormalisation is only available through an additional texture map. These formats may be widely supported in DX9 class hardware, but still only offer marginal compression (2:1) – developers are generally wanting a higher compression ratio.

3.3 Cx_8_8

This format is similar to the above, but has the 3^{rd} component implicitly derived in the texture sampler by $C = sqrt(1-U^2 – V^2)$ instead of using the pixel shader. This format may be exposed by some hardware but has never become standard, and as pixel shaders become longer and processors more powerful it seems likely to be phased out in favor of using more generalized derivations in the shader. This may be useful on some older hardware that lacks sufficiently flexible pixel shaders to support the third component derivation either by instructions or a generalized texture lookup. Drawbacks are generally as for 8_8 in that the compression ratio is lower than we would ideally like.

3.4 DXTn

The standard DirectX compressed formats have several problems when used for normal map compression, since they were primarily designed for compression of colors, not generalized vectors. The distribution of values in normal maps are generally very different to RGB images – the 3-dimensional curves that values generally take in the normal map do not compress well to lines through the colorspace (on which DXT

compression depends). Additionally the linear interpolation through the space will cause the normalized values in the original to become highly denormalized, which can potentially cause large errors in the shading calculation. The availability of only 4 distinct normal directions per 4x4 block of texels will also tend to result in unacceptable blocking artifacts.
In terms of component-level accuracy DXTn has some advantages since the overall accuracy over any given component can be significantly higher than 565 with a good compressor.

We can see that all these formats have some drawbacks, but we will concentrate on the DXTn formats here since they offer the most compelling current alternative to palettisation (the storage costs will be the same or less than a palletized version)

4 DXTn compression of normal maps

Attempts to compress normal maps with DXTn have generally been fairly unsuccessful – there are several major problems that we need to address.

- Block based compression can cause blocking artifacts due to a lack of distinct normals within each 4x4 area.
- Available precision is lower than 8_8_8_8 (although generally higher than 5_6_5 with a good compressor)
- Linear interpolation of the colors causes denormalization of values within blocks
- Poorly constructed texture maps can cause problems around the edges of objects
- The approximations generated of the directions can alter the effective shapes of the objects for the purposes of lighting (and other equivalent calculations).

We will examine the use of DXTn compression as tested on current applications using a high quality DXT compressor, and see what enhancements we can make to the overall quality.

Our basic idea here is to make creative use of the DXT5 format, which provides an additional high-quality compressed channel. By separating one of the vector components to the alpha channel it can be compressed independently, and with high accuracy. This will improve quality, and also opens up the possibility of some additional compressor optimizations. There may be some minor extra overhead required in some cases to swizzle the resulting data into the correct channels in the shader.

The use of the separated channel increases the number of possible vector directions represented by each DXT block from 4 to 32, which is likely to largely eliminate the blocking seen with standard DXT1.

It should be noted that variations on this technique can provide higher quality compression for other data as well as normal vectors, but we will concentrate on vector compression here.

4.1 Vs. palettisation

Palletized normal maps cannot retain the smooth variations of the original image, whereas DXTn compression can generally represent these variations quite accurately. However palette entries can be generated in such a manner that they are always normalized which can be seen as a distinct advantage. An additional advantage of palettisation is that major directional errors should not generated by the palletizing process – it will generally represent the approximate direction of the vector accurately in all circumstances. A DXTn encoder may introduce significantly incorrect directions due to the block-based nature of the compression, and the requirement of the compressor to compromise between the different vectors in the block. Current DXTn encoders could be modified to improve this situation by using different error criteria to prioritize preservation of direction – it is currently more common to minimize errors in overall luminance since the eye is more sensitive to luminance errors. A compressor that favors directional accuracy will tend to result in more blocking artifacts, however, since in the extreme case blocks may be reduced to using only two explicitly encoded ‘representative’ directions.

4.2 Modifications to the basic compressor

In our experiments a basic DXTn compressor was somewhat rewritten – a standard DXT compressor deals with colours, and is built to favour the weightings of the colours with respect for the human eye’s responses.

For model-space normals this behaviour is undesirable so the weightings were equalised. After much experimentation a system was arrived at that dynamically alters the weightings from block to block to attempt to favour the major axis directions of each block – this modification was found to reduce the blocking artifacts significantly in the final output, and generally helps to preserve the major direction of the vectors.

In addition to this alteration the compressor was modified to ignore certain values. Knowing that normal maps are constructed of normalised values, but are usually poorly constructed so that objects tend to be bounded by black areas (or some other null value) the compressor was altered so that it would only consider normalised values when compressing a block. In this fashion we ignore false values that would otherwise skew the compression quality around the edges of objects – we treat all the false values as if they had the median value of the remaining vectors in the block. Ideally for the purposes of compression developers would construct textures in such a fashion that edge values are extruded to the appropriate block boundary, which would fix this problem without intervention from the compressor.

4.2.1 Future modifications (evaluation of these is in progress)

- It would be very interesting to produce a new version of the compressor that assumes that renormalisation of the vectors is performed in the pixel shader after reading the texture map. This would allow the compressor far more options since the magnitude of the resulting compressed vectors can be completely ignored - all that needs to be preserved in the compression is the overall direction. A compressor based around this assumption may give improved results.
- On the DXT5 compressor where one of the vectors is separated it might be useful to alter the directions of the vectors slightly to try to preserve normalization (in the case where the pixel shader does not renormalise). This could be done either by re-matching the Alpha component with the colour components to try to preserve normalization, or more appropriately by constructing a new alpha block after compressing the colour block – this alpha block will contain appropriate values for the final vector component to achieve normalization (care must be taken to preserve the direction of the original alpha component). By doing this we may be able to get away without a renormalisation step in the shader, but the correctness of the direction of the vectors will be sacrificed. Blocking artifacts also will tend to become more obvious with this technique as we will lose the extra directions provided by the independent nature of the axis encoded in the alpha block.
- When the compressor detects that it has generated serious directional errors it might be useful to have it fall back to a behaviour where it encodes two representative directions for the block accurately in the endpoints, and map all the texels in the block to these values as discussed above.

5 Image Quality Tests for Model Space Maps

By compressing and decompressing normal-map textures from an application and making use of the frame dumping capabilities we are able to carry out image quality comparisons between various source texture formats. We chose an application that uses model space normal maps first to evaluate compression quality in these cases.

For the DXT formats we tried two main situations – in one of them we assume that the pixel shader uses the raw, filtered values obtained from the texture map. In the other it is assumed that the pixel shader contains code that renormalizes the values after filtering before using them in the lighting equation.

When using DXT5 compression we extracted the X (or R) channel from the vector into the alpha block, so the color block contains only the Y and Z components (in the green and blue channels respectively). This was found to be the optimum case for this application. This choice could potentially be varied on a per-texture basis to improve the compression quality.

5.1 DXT1 without renormalisation

- Highly visible noise is introduced into the lighting equation.
- Visible directional lighting errors are introduced.
- Visible lighting magnitude changes on specular highlights.
- Specular highlights on some surfaces are lost almost completely.
- Significant blocking in some areas.

5.2 DXT5 without renormalisation

- Noise is reduced, but still very significant.
- Directional lighting errors are somewhat corrected
- Lighting magnitude changes are reduced
- Specular highlights are still frequently broken
- Blocking still a problem, but usually on specular regions only.

5.3 DXT1 with renormalisation

- Noise is greatly improved.
- Lighting directional errors are still visible.
- Lighting magnitudes are correct, but with significant detail loss.
- Specular highlights are the correct brightness.
- Specular blocking errors are massively reduced. Magnitude of highlights is now roughly correct

5.4 DXT5 with renormalisation

- Very slight noise remains, but almost invisible.
- All other areas look essentially identical to the original.
- Specular blocking has almost completely disappeared. Specular highlights are the correct brightness.

6 Two component DXT5 For Tangent Space Maps

In general it seems that tangent space normal mapping is being more widely adopted by developers than the model space equivalent - this opens up much more interesting possibilities for compression with DXT than those available for model-space since we now only need to store two components. We can expect to generate excellent results with a good encoder. In fact, hopefully, the results are good enough for this to be a generalized replacement for uncompressed normal maps in all cases on all modern hardware.

The idea is essentially to produce a compressed version of the 8_8 scheme as described briefly earlier. One component is represented by the green component of the RGB block, and the other by the alpha component from the alpha block. This gives a sweep of 3 or 4 positions across one axis (although this might be limited to 4 component sweeps only on some hardware due to differences in the decompression process) and 6 or 8 across the other, with high precision endpoints (6 bits and 8 bits respectively).

The third component is derived mathematically in the shader as it would be for an 8_8 format. Additional overhead for swizzling the alpha component to appropriate channels comes to at most one instruction, and in general shaders on all current hardware this overhead will almost certainly disappear entirely through internal optimization (either by the availability of generalized swizzles, or through parallel instruction execution). Generally it should be overwhelmingly likely that the performance of compressed normal maps will be equal to or higher than uncompressed ones in almost all cases.

In tests, achieved directional accuracy is very good, generally very close to the uncompressed 32-bit RGB source unless there is a huge variation of direction within a single block. The accuracy of the normal representation across shallow gradients is also generally close to 8-bit per component original data when compressed with a high-quality encoder.

By choosing on a per-texture basis which component to store in the high-quality (alpha) block, and which in the color block the quality of the compressed textures can potentially be maximized. To support both alternatives will require different shaders for the two cases (to swizzle appropriately) but developers can naturally decide whether the increased quality is worth the effort. Generally the compression quality is probably good enough in either case to be better than a palletizing approach in pretty much all circumstances.

One argument against 2-component formats in general is the denormalising side effect used by developers as the texture mip levels go from high detail to low detail. This can be solved for these formats by providing an additional luminance-only map that holds a denormalising value to multiply through - this does have disadvantages, however, since it doubles up the storage cost and also requires the second map to go through the same filtering as the first (which could be expensive if high-tap anisotropic is used).

With the DXT5 format we do have a second option. In one of the unused components of the RGB block we can store a denormalising value for that block. We choose this value such that it is constant over the whole block, and as such its presence will not affect the color compressor (possibly small variations could be permitted, but I expect that one value per block may prove to be sufficient anyway). At the highest mip-level this value will be one, and it can decrease as desired with each lower level (based on how denormalised the vectors in the block become – over relatively flat areas the blocks will naturally remain close to normalised). This allows specular highlights to fade with distance as before to alleviate aliasing artifacts in the lighting equation, requires no extra storage and only marginal extra shader time to extract and use the data appropriately. In addition the value naturally goes through the same filtering as the other components for free which could make it a better performance option than using a second texture.

7 Conclusions

On the evidence we have collected DXT5 format can be leveraged to provide compelling alternatives to original 32 bit format images for representation of normals. Directional errors are rare, and usually minor (although naturally it is possible to produce a normal map that breaks this, just as it's possible to produce a color map that doesn't compress well).

The main proviso here is that model-space shaders really need a normalization step after sampling the texture map, otherwise errors will definitely be visible (particularly on specular areas where the errors tend to be raised to some power). It should be noted that if bilinear filtering is being used then it is arguable that this normalization step should really be made anyway, regardless of the source texture format.

If texture maps are constructed in a friendly way for compression (extruding edges of objects to block boundaries) then many model-space normal maps can be compressed and the results will be of high quality, probably preserving more of the original information than moving to a palletized representation.

In tangent space the arguments for a 2-component DXT5 representation from a quality standpoint are far stronger than that for 3-component compressed maps. The results achievable with 2-component compression are generally very good, and comparable to the uncompressed representation.

In the future with some additional research we may be able to further improve the quality of the compression, and may eliminate some of the remaining problems. Certainly DXT textures can offer good alternatives as a stop-gap measure until more advanced normal map compression formats become available in the future. As with all compression there is always the risk of some bad cases, but if the artifacts of palettisation are acceptable then it seems that any loss of quality from using this approach should be more acceptable.

8 Example Code

The example uses a Rendermonkey workspace that uses two channel normal maps. There are several textures provided both in compressed and uncompressed forms. By loading different textures into the bump map and compressed bump map variables in the workspace, and then selecting which of these to display a comparison can be made of the overall image quality. I have included one very difficult compression example as well as the more standard tests to try to show a 'worst case'. The texture with concentric rings is extremely difficult to compress effectively as it is designed in a way that makes it very hard to compress with block-based techniques. The rings cut through blocks at all angles, which causes additional compression errors due to mismatches of intersections at block boundaries (as an 'edge' cuts through different blocks at different angles and positions it becomes highly likely that neighboring blocks hold very different information - the compressor is likely to make significantly different compression choices for the blocks – this large difference in block content usually causes significant additional low-frequency blocking artifacts). In addition to this the ring structure cause wildly varying directions to occur within single blocks, which is a very difficult case for the compressor to represent. Even in this extremely difficult case it can be seen that the character of the bumps is still generally well preserved – when combined with a texture map the quality would still probably be acceptable. In the other, more general cases provided it is often difficult to tell any significant difference between the compressed and uncompressed representations.

The bump maps have been compressed using ATI's Compressonator tool, which provides a very high-speed DXT compressor that retains fairly high quality. This gives a good feel for the levels of quality that can be achieved with real-time compression of texture maps at load time. Better results than the ones shown can be obtained with slower, higher quality compressors (such as S3's original reference compressor).

Examining the typical artifacts to caused by block-based compression in turn (Blocking, loss of fine detail and banding/quantisation noise.)

- It can be seen that pretty much all of the fine detail is preserved in the compressed representations.
- Blocking artifacts are kept to a minimum, and are generally not visible. They can occasionally be caused by areas that represent several highly independent normal directions (harsh corners of brickwork would be a good example)
- Banding/quantisation is slightly increased from the original in areas of shallow gradient but is significantly higher than 16 bits of overall base accuracy. Generally shallow gradients appear to be represented with a resolution of between 7 and 8 bits per component.