ATI RADEON X800

3Dc™ White Paper

Introduction

3Dc™ is a new compression technology developed by ATI, and introduced in the new RADEON X800 series of Visual Processing Units. This technology is designed to allow game developers to pack more detail into real time 3D images than ever before, making 3Dc a key enabler of the HD Gaming vision.

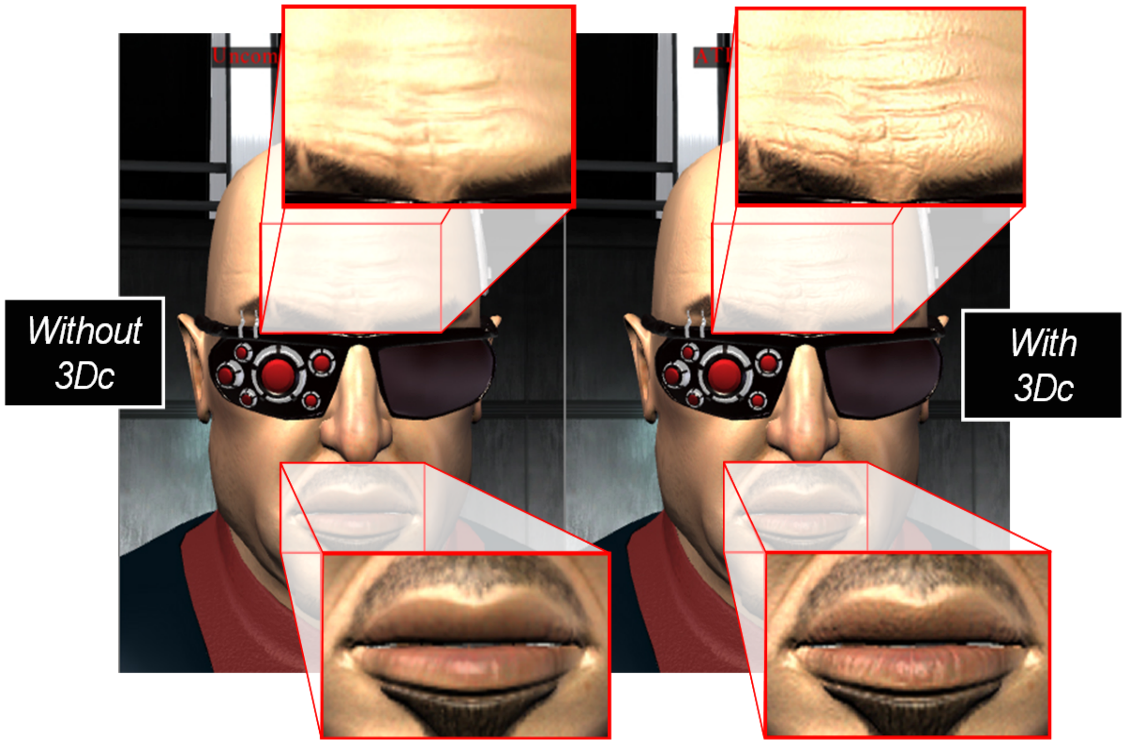

Figure 1: A close-up of the Optico character from ATI's Double Cross demo showing the increase in fine detail made possible by 3Dc compression.

Today's graphics processors rely heavily on data and bandwidth compression techniques to reach ever increasing levels of performance and image quality. Rendering a scene in a modern 3D application requires many different kinds of data, all of which must compete for bandwidth and space in the local memory of the graphics card. For games, texture data tends to be the largest consumer of these precious resources, making it one of the most obvious targets for compression.

A set of algorithms known as DXTC (DirectX Texture Compression) has been widely accepted as the industry standard for texture compression techniques. Introduced in 1999, along with S3TC (its counterpart for the OpenGL API), DXTC has been supported by all new graphics hardware for the past few years, and has seen widespread adoption by game developers. It has proven particularly effective for compressing two-dimensional arrays of color data stored in RGB and RGBA formats.

With the appearance of graphics hardware and APIs that support programmable pixel shaders in recent years, researchers and developers have come up with a variety of new uses for textures. A variety of material properties such as surface normals, shininess, roughness, and transparency can now be stored in textures along with the traditional color information. Unfortunately, DXTC is often not as effective at compressing textures carrying these different types of data as it is at compressing color data. This makes it challenging to fit all of the necessary data into graphics memory, and also impacts performance because of the increased memory bandwidth requirements.

3Dc is designed to resolve these issues by going beyond the capabilities of DXTC, and offering efficient and effective compression of new types of texture data. Whereas DXTC is optimized for compressing 3- and 4-channel data formats, 3Dc focuses on 2-channel data formats. It is particularly effective at two increasingly common tasks – compressing normal maps, and packing multiple pieces of data into a single compressed texture.

Normal Maps

Normal maps are special textures that are used to add detail to 3D surfaces. They are an extension of earlier “bump map” textures, which contained per-pixel height values and were used to create the appearance of bumpiness on otherwise smooth surfaces. Normal maps contain more detailed surface information, allowing them to represent much more complex shapes.

You can think of a normal as an arrow that points outward from a surface, and is perpendicular to that surface at its point of origin. The usefulness of normals stems from the fact that, given the location of a light source, they can be used to accurately determine the direction in which incoming light will reflect off of a surface at any given point.

Normals are typically stored at each vertex in a 3D model. Depending on the number of vertices and how far apart they are, there may be large gaps between vertices in which the direction of the normals cannot be precisely determined.   This prevents the normals from being used to create fine detail. To address this problem, a look-up into a normal map can be performed for each pixel, making it possible to more precisely define the direction of the normal at any location on a surface.

Normal maps provide a much more efficient way of capturing the fine details on a surface than using very large numbers of polygons. Their only limitation is that they cannot be used for large scale features, because they do not actually affect the shape of an object. The three dimensional appearance of the details on normal mapped surfaces are only an illusion, enabled by accurately modeling light and shadow. This can become apparent when looking at the silhouette of a normal mapped surface.

The most common way to use normal maps is to use just enough polygons to give the shape of an object and its silhouette a satisfactory appearance, and then use normal maps to add in any finer details. These details may be small in scale, but they can have a large impact on the realism and image quality of a rendered image.

Normal maps can be generated by creating two versions of each 3D model, one with a high polygon count, and one with a much lower polygon count. The low polygon version has just enough detail to capture the general shape of the object. The high polygon version uses as many polygons as are required to capture all of the finer details. A simple program can then be run that takes these two models as inputs, calculates the differences between them at many points on the surface, and stores the results in a normal map texture. For improved image quality, this normal map can be combined with a detail map, which includes details too small to be captured by even the high polygon version of the model. Often this can be a simple repeating pattern that adds realistic texture to a surface.

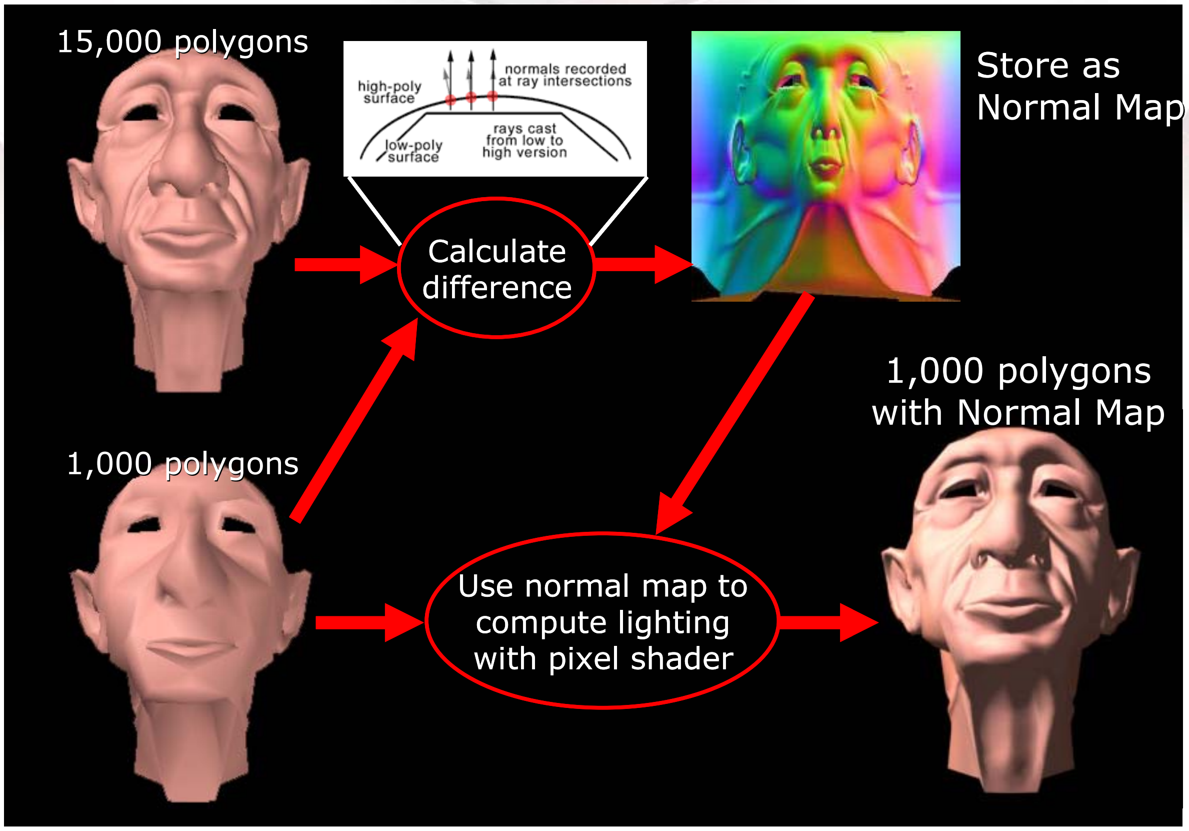
15,000 polygons
high-poly surface
normals recorded at ray intersections
low-poly surface
rays cast from low to high version
Calculate difference
Store as Normal Map
1,000 polygons with Normal Map
1,000 polygons
Use normal map to compute lighting with pixel shader

Figure 2: Generating normal maps

This normal mapping technique is one of the key things that makes the new generation of games like Half-Life 2, Doom 3, and Far Cry look so much better than previous titles. The only limitation to the effectiveness of this technique is the size of the textures required. In an ideal case where every surface had both a color texture map and a normal texture map, the texture memory and bandwidth requirements would double compared with just using color maps alone.

In fact, the problem is much worse because DXTC & S3TC are ineffective at compressing normal maps. They tend to have trouble capturing the small edges and subtle curvature that normal maps are designed to capture, and they also introduce unsightly block artifacts. Because normal maps are used to capture light reflections and realistic surface highlights, these problems are amplified relative to their impact on color textures. The results are sufficiently poor that game artists and developers would rather not use normal maps at all on most surfaces, and instead limit themselves to lower resolution maps on selected parts of the rendered scene.

3Dc provides an ideal solution to the normal map compression problem. It provides up to 4:1 compression of normal maps, with image quality that is virtually indistinguishable from the uncompressed version. The technique is hardware accelerated, so the performance impact is minimal. Thus, developers are freed to use higher resolution, more detailed normal maps, and/or use them on all of the objects in a scene rather than just a select few.

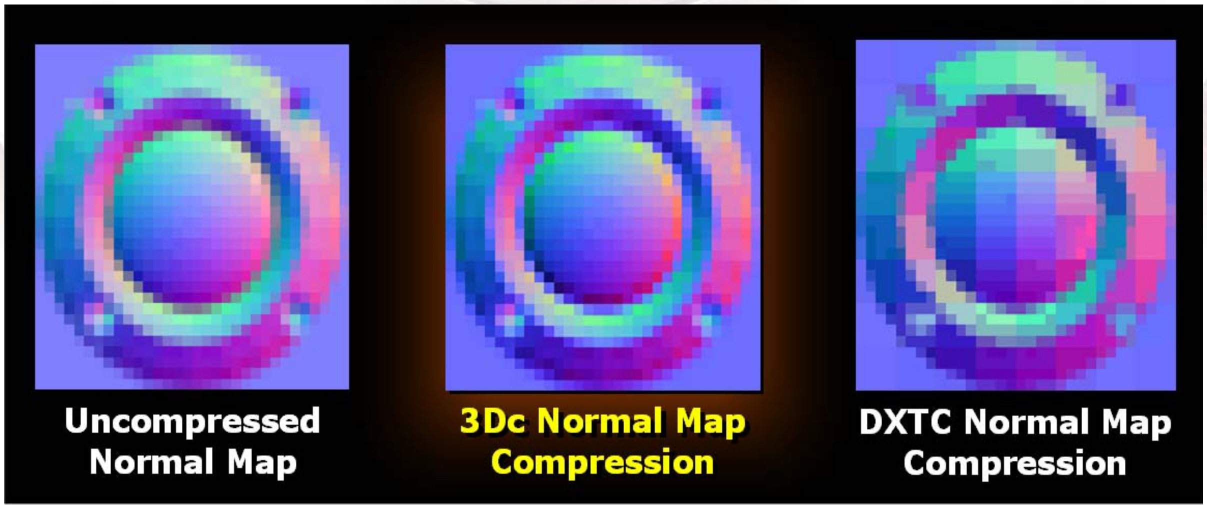

Figure 3: Comparison of normal maps compressed with 3Dc and DXTC. Note how the 3Dc compressed normal map looks virtually identical to the original, while the DXTC version exhibits blocky artefacts and loss of detail.

How 3Dc Works

3Dc is a block-based compression technique. It breaks a texture map up into 4x4 blocks containing 16 values each. These values must consist of two components. Each component is compressed separately. A maximum and minimum value is determined for each block, and these are stored as 8-bit values. A set of six intermediate values are then calculated, spaced equally between the minimum and the maximum. This gives a total of eight values that each component can take within a block. Each component is assigned a 3-bit index corresponding to whichever of these values is closest to its original value.

The resulting compressed blocks consist of four 8-bit values and thirty-two 3-bit values, for a total of 128 bits. Since the original blocks consisted of sixteen 32-bit values, for a total of 512 bits, this represents a compression ratio of 4:1. If the original values were 16-bit rather than 32-bit, then a compression ratio of 2:1 can still be achieved.

Using 3Dc to compress normal maps requires an additional step. This is because each value in a normal map is actually a 3D vector, consisting of 3 components (x, y & z). These values must be reduced to 2-component values in order to work with 3Dc. Fortunately, this can be handled in a simple way by assuming that all of the normal vectors have a length of 1. Given the values of two components of a vector, the value of the third component can be found using the following mathematical relationship: $z = \sqrt{(1-(x^2+y^2))}$. This formula can be implemented using just a couple of pixel shader instructions.

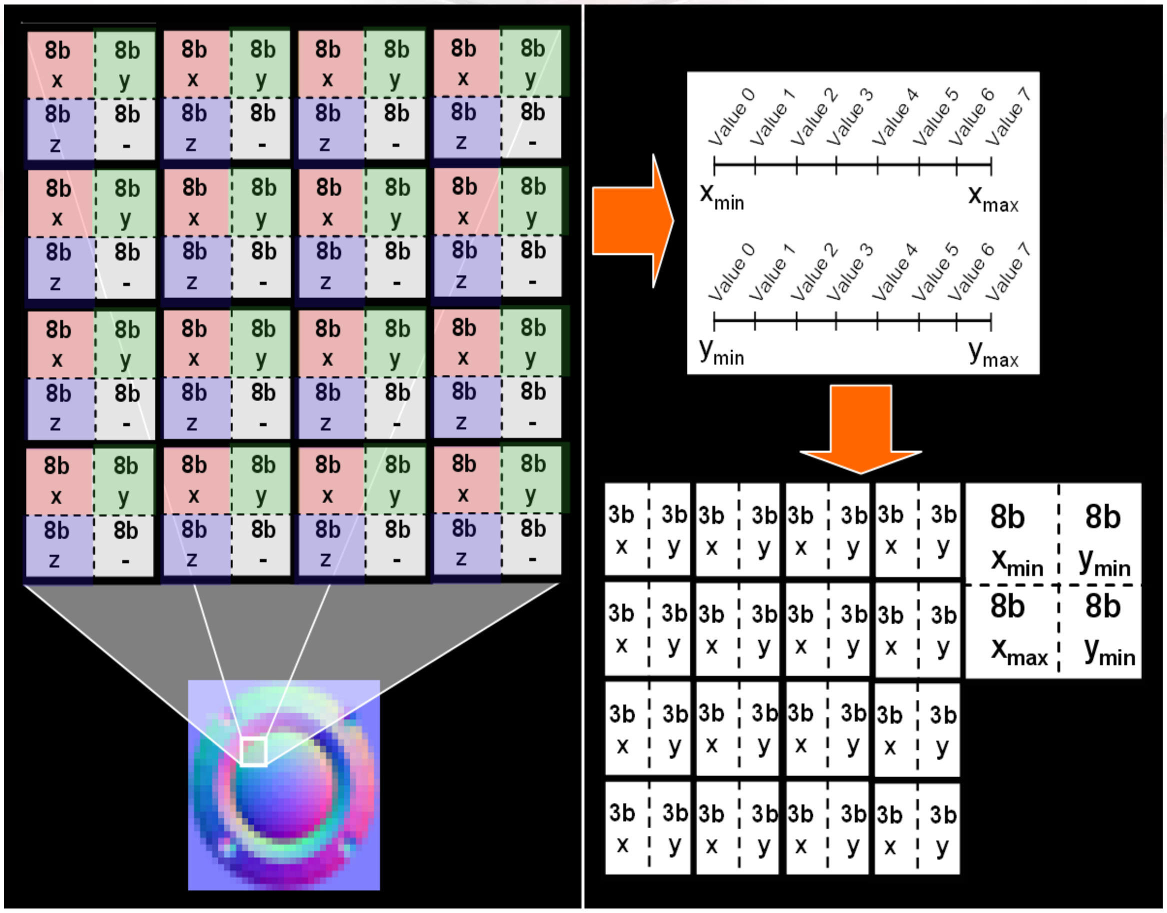

Figure 4: Block data formatting for a standard normal map (upper left) and 3Dc compressed normal map (lower right). The generation of intermediate values during the compression process is shown in the upper right.

Benefits of 3Dc

3Dc makes it possible to use much higher resolution and more detailed normal maps than would otherwise be possible due to memory restrictions. Since the decompression of 3Dc compressed textures is hardware accelerated, the performance cost of achieving this higher level of detail is minimal.

Alternatively, 3Dc can be used to reduce the memory footprint of existing normal maps and allow a wider variety of them to be stored, meaning they can be used on larger number of different surfaces. It can also be used to optimize performance in applications that are strongly limited by memory bandwidth, since the normal maps are read from memory in compressed format.

Another important benefit of 3Dc is its ease of implementation. In most cases it can be used with existing art assets. The same compression tools used for DXTC/S3TC can be used for 3Dc with only minor modifications. Pixel shader changes consist of only two extra instructions in the case of compressed normal maps. The simplicity of 3Dc allows it to work easily with all other image quality enhancement techniques including anti-aliasing, texture filtering, shadows, HDR, and post-processing effects.

The new 3Dc texture format is supported in DirectX 9.0 using a FourCC code, and in OpenGL using a vendor-specific extension. 3Dc is expected to be incorporated in some form into future versions of these programming interfaces. To assist in this process, ATI is making all the details of this format openly available.

Summary

3Dc is an exciting new compression technology designed to bring out fine details in games while minimizing memory usage. It is the first compression technique optimized to work with normal maps, which allow fine per-pixel control over how light reflects from a textured surface. With up to 4:1 compression possible, this means game designers can now include up to 4x the detail without changing the amount of graphics memory required and without impacting performance.