FXT1 Texture Compression Technology White Paper

By
3dfx Interactive

Introduction:

The development of advanced 3D content continues to push the envelope of on-screen visual realism, necessitating the use of more textures at higher resolutions to render more detailed and realistic images. Current titles are using two or more textures per object at resolutions up to 2048x2048 per texture and these requirements will continue to increase as 3D content developers strive for increased realism. Using more textures, however, also requires the support of an advanced and flexible technological foundation for handling them. 3dfx's FXT1™ texture compression technology provides that foundation.

One of the issues when using large numbers of textures is simply that it takes much more memory to store those textures. Consider, for example, that a 2048x2048 32-bit per texel texture requires 16 Mbytes of texture storage space (and this is without even storing texture mipmaps!). Furthermore, this is only for a single high-resolution texture! It wasn't long ago that 3D accelerators didn't even have 16 Mbytes of *total* memory onboard. Texture compression is a powerful way to increase the amount of textures used without dramatically increasing the texture storage requirements.

Increasing the size and number of textures used in a scene also increases the amount of memory bandwidth required for texture lookup. Ultimately, as the amount of memory bandwidth required for texture accesses grows large, the overall fill-rate performance of the 3D accelerator is reduced and the resulting frame rate declines. One of the benefits of utilizing compressed textures is that the amount of memory bandwidth required for texturing is significantly reduced. As a result, games and applications taking advantage of texture compression can achieve higher sustained fill-rates and thus higher overall frame rates.

As a free, open-source, cross-platform compression algorithm, 3dfx's FXT1™ texture compression technology allows content developers to create textures at higher resolutions and use more textures in a given scene while simultaneously reducing the memory bandwidth required for texturing. By reducing the amount of storage required for a given texture, more textures can be stored in a given amount of memory. As a result, more textures can be used in a rendered scene which can substantially improve overall visual quality. By reducing the memory bandwidth needed for texture transfers and processing, the overall fill-rate of a 3D accelerator is increased, which increases frame-rates and also generates a much more realistic, immersive 3D experience.

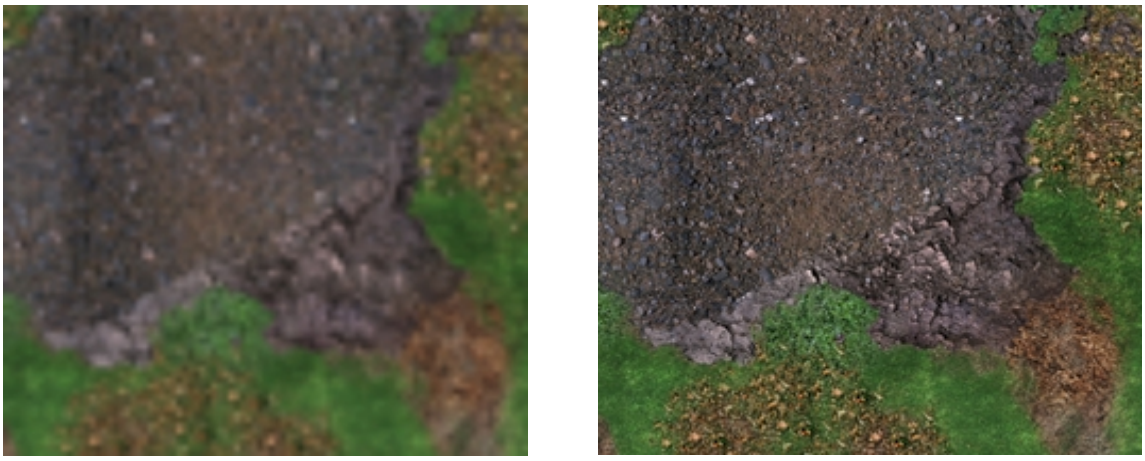
Advantages of FXT1™ Texture Compression:

-Increased total number of textures available for rendering
Texture compression technology reduces the amount of memory required to store a given texel, thus in turn reduces the amount of memory required to store an entire texture map. As a result of using texture compression, more textures can be stored in a given amount of texture memory. For example, a typical 3D accelerator storing a 256x256 texture in a 32 bit-per-texel format requires 256 Kbytes of texture memory storage. However, storing that same 256x256 texture in a compressed 4 bit-per-texel format requires only 32 Kbytes of texture memory storage, an effective savings of 224 Kbytes. Or, from a different perspective, the same amount of texture storage required to hold a single 256x256, 32 bit-per-texel texture can hold 8 unique 256x256, 4 bit-per-texel textures. These 8 unique textures can be used to dramatically improve the overall visual quality of a rendered scene.

Texture Size	**8-bit**	**16-bit**	**24-bit**	**32-bit**	**4-bit FXT1™**
64x64	6 KB	9 KB	13 KB	16 KB	2KB
128x128	18 KB	33 KB	49 KB	64 KB	8KB
256x256	66 KB	129 KB	193 KB	256 KB	32KB
512x512	258 KB	513 KB	769 KB	1024 KB	128KB
1024x1024	1,026 KB	2,049 KB	3,073 KB	4096 KB	512KB
2048x2048	4,098 KB	8,193 KB	12,289 KB	16384 KB	2048KB

This table shows the memory requirements for various sizes and color depths of textures without compression. With FXT1™ texture compression, 32-bit images are reduced by a ratio of 8:1 without a perceivable loss in image quality.

-Higher resolution textures for better image quality
With limited on-board memory, first and second generation consumer 3D accelerators were limited to texture maps with a maximum resolution of 256x256 texels. These low-resolution textures look fine when viewed at a distance, but when viewed at a closer range the textures become blurred and lack fine detail. However, utilizing higher resolution textures to improve visual quality can cause texture memory requirements to be too large to be economical. By utilizing texture compression, higher resolution textures can now be utilized to offer significantly improved visual realism in economical amounts of texture memory. Consider our example 256x256, 32 bit-per-texel texture map. The same amount of memory required to store this texture map, 256 Kbytes, can hold a single 4 bit-per-texel texture map at a resolution of 1024x512. Utilizing larger texture maps can significantly enhance the overall visual quality of a rendered scene.

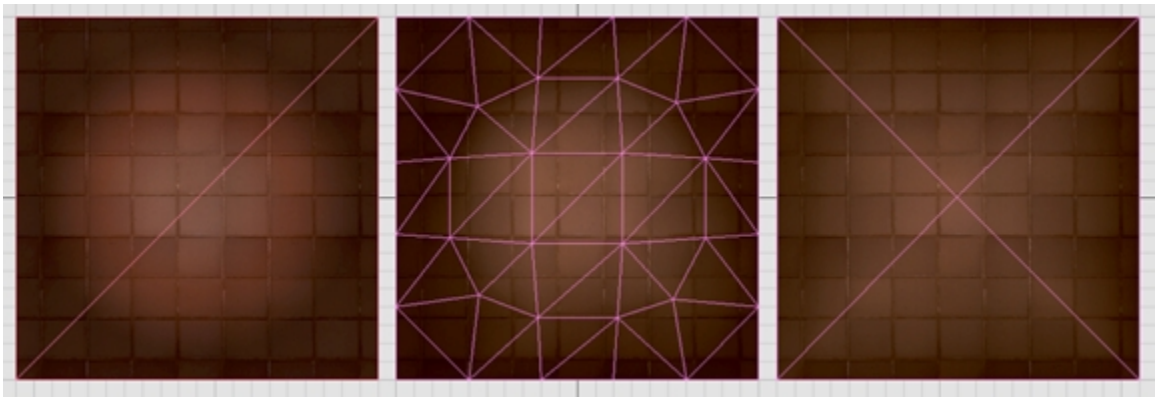


The 256x256 image on the left shows very little detail and looks blurry. The same image on the right at 2048x2048 shows very fine detail including individual rocks and blades of grass, without any of the associate blurriness.

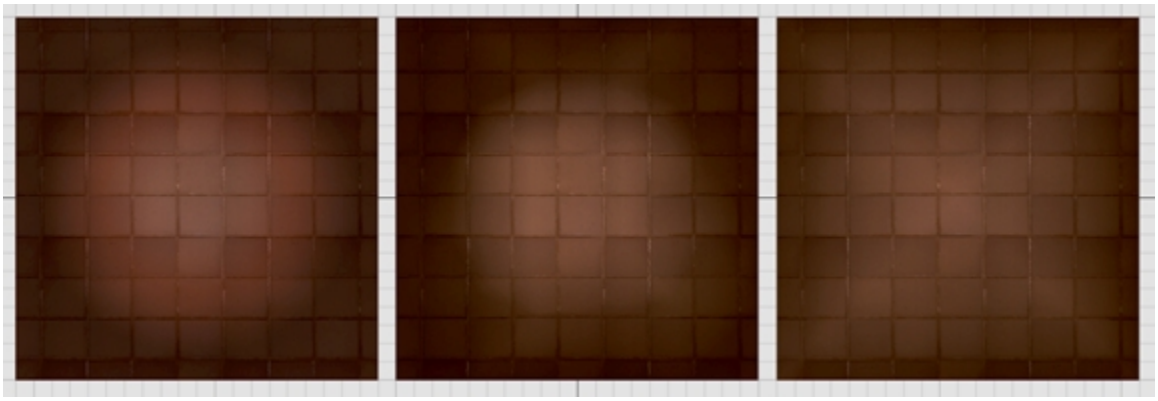
-More textures per polygon for advanced effects
With support for multiple textures per surface in all of the major 3D APIs, content developers are now using two or more textures per surface to increase image quality and create special effects. By using more than one texture per surface, effects like advanced real-time lighting, specular highlights, and bump mapping are possible, without increasing the total triangle count for the scene or overwhelming the available memory bandwidth.

With multiple textures, advanced lighting effects like spotlights and shadows can be created very easily and without using geometrically complex models. Lighting effects using multi-texturing hardware are typically performed by blending a light map (a black and white image that emulates the differences in light intensity) with the base texture of an object. When these two textures are blended, the user sees all the lighting detail without the performance burden caused by alternative techniques which utilize significantly higher geometric models and complex lighting equations.

Texture compression benefits these multi-texturing effects by decreasing the total bandwidth needed to transfer the extra texture data used for the second texture . Additionally, the resultant memory savings which comes from using compressed textures allows the application to use multi-texturing techniques on many more objects in the scene. Consider that a typical 512x512 texture at 32 bits-per-texel would require 1MB of texture storage space. If the 3D scene contained just eight objects with two textures each, current generation cards (with only 16MB of memory) would use up all available bandwidth and storage space with no room left over for mipmaps. With 4-bit-per-texel texture compression, a total of 40 or more objects could be multi-textured, including mip maps, before memory storage is depleted. This allows multi-texturing for entire rooms and all of the objects or characters in a given scene.



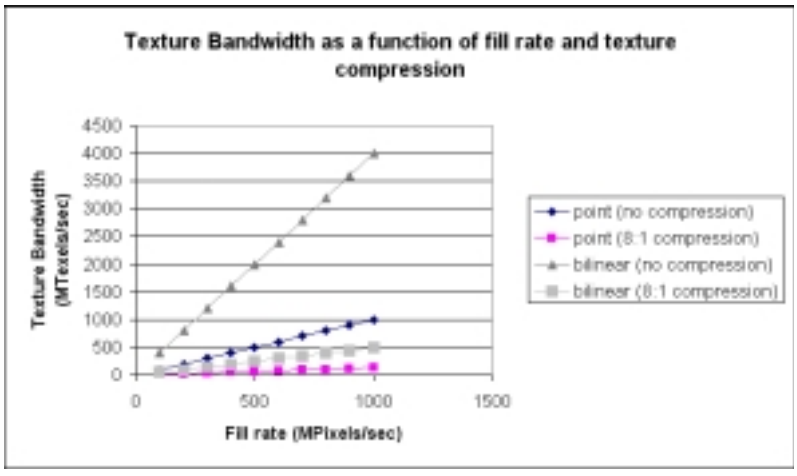
On the left, a light map is blended with a texture map to create the effect of a spotlight, without using extra triangles. In the middle, the same image is created by tessellating and adding many triangles to achieve accurate lighting. On the right, an image using only four triangles does a poor job of representing the spotlight effect.



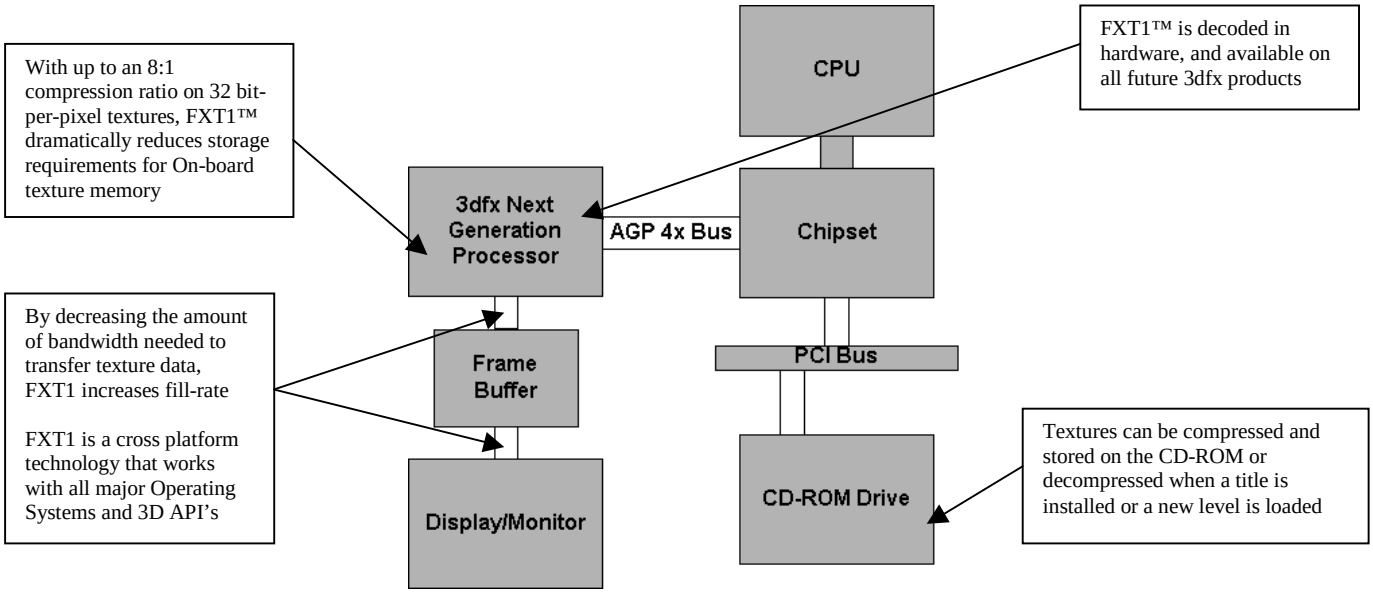
The same images as above, with the mesh removed from the picture.

-Lower bandwidth requirements for better fill-rate performance

Fill-rate is the number of pixels or texels that can be drawn in a given period of time. The higher the fill-rate of a 3D accelerator, the higher frame rates it will produce for a given piece of content. Of course higher frame rates are desirable as they create smooth 3D rendering without the jerkiness associated with lower frame rates. Overall, fill-rate performance is directly affected by the amount of texture memory bandwidth available. In the case of a 32-bit texture, 8 times the amount of texture memory bandwidth is required to render a given texel when compared to a 4-bit texture. This dramatic reduction of required texture memory bandwidth, as a result of utilizing texture compression, results in much higher fill-rates and frame rates, thus creating a much more enjoyable, immersive 3D experience.



This chart shows the relationship between texture bandwidth and fill-rate. Notice the dramatic increase in texture bandwidth needed to achieve a desired fill-rate (4x as much bandwidth is needed when using bilinear filtering!). Without enough available bandwidth, fill rate performance suffers.



Textures have a long way to travel before they are rendered on your screen. FXT1™ reduces their size considerably, thereby reducing bandwidth requirements making the transfer more efficient

How FXT1™ texture compression works:

-*Compression (also known as Encoding):*
FXT1™ texture compression works by dividing an image into multiple 4x4 and/or 4x8 texel blocks. Each texture (including opaque and transparent textures) is compressed to an average of four bits-per-texel. In the FXT1™ texture compression scheme, four different compression algorithms are used, with the best one chosen per block to generate the highest quality result. The four algorithms used are:

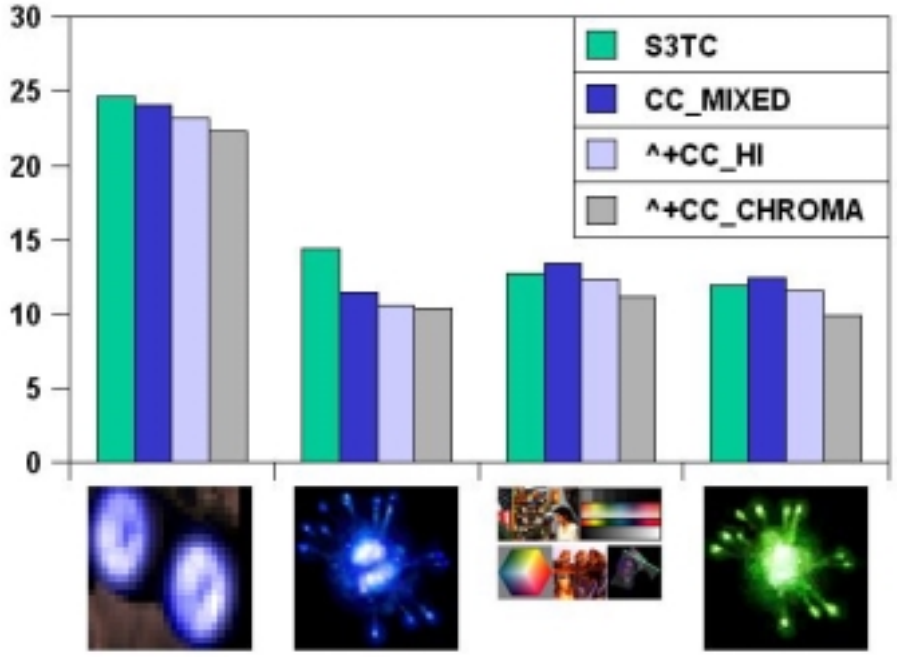
1. *CC_MIXED* (similar to other texture compression schemes): A 4x4 texel block is represented by two bits-per-texel for opaque textures. Additionally, each block has two 16-bit colors stored in an RGB 565 format. The two RGB 565 colors and two additional colors (created by interpolating between the two RGB 565 colors) form the primary colors for this texel block and its associated four color lookup table. A 2-bit index is used to determine which color from the lookup table will be used for each texel in the 4x4 block. Transparent textures are created by making one of the four colors transparent.

2. *CC_HI* (best for spatial resolution): A 4x8 texel block is represented by three bits-per-texel for opaque and transparent textures. Each block stores two 15-bit colors in an RGB 555 format. The two RGB 555 colors and five additional colors (created by interpolating between the two RGB 555 colors) form the primary colors for this texel block. Additionally, an eighth color is defined to be the transparent color. A 3-bit index is used to determine which color from the 8-entry lookup table will be used for each texel in the 4x8 block.

3. *CC_CHROMA* (good at complex color areas): A 4x8 texel block is represented by two bits-per-texel for opaque textures. Each block stores four 15-bit colors in an RGB 555 format. All four colors are used directly with no interpolation to form a 4-entry lookup table. The 2-bit index assigned to each texel in the block is used to determine which of the four colors is assigned to each individual texel. Note that *Colors4* only applies to opaque textures, as it does not support transparency.

4. *CC_ALPHA* (gives the best control over complex alpha transparencies at four bits-per-texel): A 4x8 texel block is represented by two bits-per-texel for opaque and transparent textures. Each block stores three 20-bit colors stored in a 5555 format. The first and second 20-bit colors are used for the primary colors of the left 4x4 block, while the second and third colors are used for the primary colors of the right 4x4 block. Two additional colors are created in each block by interpolating between the two primary

colors for that block. A 2-bit index is assigned to each texel in the block and a lookup table is used to determine which color is applied to each texel.



With up to 4 different techniques used to compress each image, FXT1™ provides the most accurate image reproduction when measuring the Root Mean Square error of each encoding algorithm.

-*Decompression (also known as Decoding):*

5. FXT1™ texture decompression happens during the texture mapping process and is implemented in the 3D hardware accelerator. A 2-bit field is stored in each block and is used to determine which of the 4 compression schemes was utilized by the compression algorithm for best visual quality. Depending on which algorithm is used for a given block, the proper decompression logic is applied to generate decoded 32-bit texels which can then be used by the texture mapping hardware.

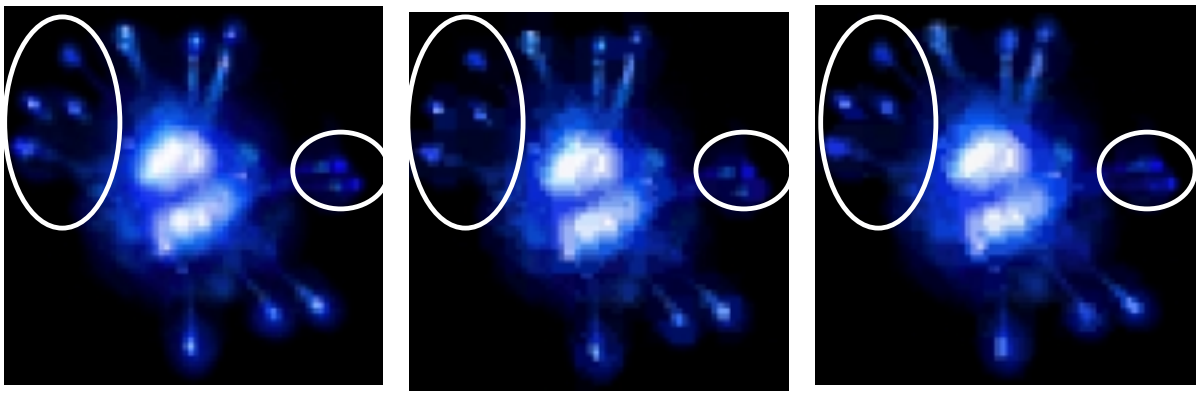
Advantages of FXT1™ texture compression as compared to S3TC™

-*Free, open source tool set:*

3dfx wants to encourage software content developers and 3D hardware accelerator companies (also known as "IHVs") to innovate with new encoding and decoding schemes that offer even greater compression ratios, better image quality, or ideally both. To that end, the compression and decompression tools, and source code are freely available. This will allow developers to use the FXT1™ compression technology across multiple platforms. Furthermore, the free license will provide IHV's with the opportunity to implement FXT1™ texture compression in future hardware designs, completely free of any license or royalty fees.

-*Multiple encoding schemes per image for better quality*

FXT1™ creates images with higher quality than other texture compression schemes by using four compression techniques for each texture (compared to only a single technique in other compression schemes). This allows the encoder to be more accurate in reproducing specific portions of an image and/or different types of images as the best possible technique is applied to each texel block.



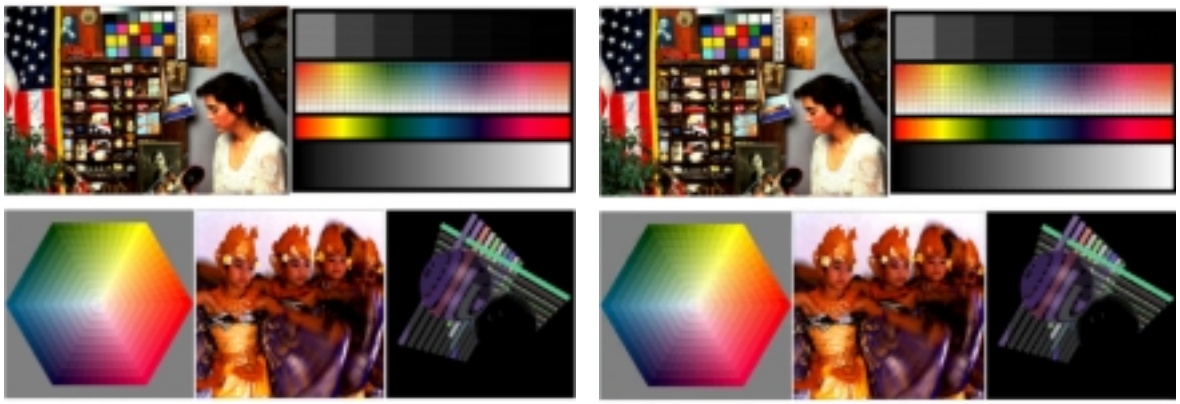
Uncompressed image (left) compared to the same image compressed using S3TC (center) compared to FXT1 (right). Note how much more accurate FXT1 is at resolving the finer details of the images.

-Better Compression Ratio for textures with multi-bit Alpha
Unlike S3TC™ which uses an 8-bit compression format when compressing textures with multi-bit alpha components (alpha is used for transparency information), FXT1™ uses a 4-bit format for the greatest possible compression ratio. As a result, the compression ratio of the FXT1™ compression algorithm is twice that of the S3TC algorithm when compressing 16 or 32-bit textures which include alpha information. This substantially increases the number of textures which can be stored in a given amount of memory, and also reduces the amount of bandwidth required for texturing.

-Cross Platform API support
All major 3D APIs support FXT1™ texture compression. Textures can be pre-compressed and stored on CD or they can be compressed in real-time by using the provided tools and source code for FXT1™ encoding. In Direct3D™, when an application creates a textured surface in memory, the application tells the hardware if there are any special codes used to designate a compressed image surface, called FOURCC codes, associated with that surface. If so, the hardware reads this code and knows that the surface must be decoded prior to being moved to the frame buffer. In OpenGL (on all platforms, including Windows, Linux and Macintosh), texture surfaces are marked with an FXT1™code in the file header. When a surface is created in memory, the surface has a flag which designates it as a compressed texture. The OpenGL driver recognizes this flag when the proper extension is implemented in the OpenGL driver. The texture is then decoded by the hardware, prior to being used internally. Glide provides native support for FXT1™ based textures; automatically recognizing FXT1™ encoded textures and decoding them accordingly in hardware.

Using FXT1™ Texture Compression

3dfx provides free tools and associated source code for encoding and decoding textures in the FXT1™ format. One tool is a command line utility which allows developers to convert back and forth between the FXT1™ format and other standard image formats, performing both FXT1™ compression and decompression. A second tool is an Adobe PhotoShop™ plug-in, which allows artists to work on a texture image within PhotoShop and then both export and import images in the FXT1™ image format. For developers wishing to do real-time compression in the FXT1™ format, free source code is available for writing an encoder as part of an application.



High-resolution image – Uncompressed (Left) vs. same high-resolution image – Compressed (Right). While the compressed image on the right suffers no loss in visual quality, it is represented using 6 times less data than the one on the right.

Conclusion

FXT1™ texture compression is provided free to software developers and 3D hardware accelerator companies who want to create and accelerate content utilizing many more texture images and higher resolution textures, all at the highest possible image quality. FXT1™ texture compression provides equal to or better compression ratios than any available hardware compression scheme. With up to four compression schemes used in the encoding of each texture, FXT1™ provides the most accurate texture compression available, providing little or no loss in image quality. Decoding FXT1™ textures can be done transparently in the 3D hardware at run-time, or decoded in software and converted to another hardware-supported texture format. In addition, cross platform 3D API (Glide, OpenGL, and Direct3D) support for FXT1™ allows developers to use it on Windows, Macintosh and Linux platforms without any additional coding by the developer. Finally, 3dfx provides free tools, and the associated source code, to encode and decode compressed textures to allow the user community to create newer and better encoding algorithms. These tools are currently available to developers registered with the 3dfx Developer Program. For additional information, please contact devprogram@3dfx.com.