Revolution SDK

Compressed Texture Format

Version: 1.01

The content of this document is highly confidential and should be handled accordingly.

RVL-06-0180-001-D
Released: March 27, 2009

Table of Contents

Code

Tables

Figures

Revision History

Version	Revision Date	Description
1.0.1	2008/08/28	Chapter 4: Fixed corrupted characters in Figures 4-1 and 4-2.
1.0.0	2008/08/01	Initial version.

1 Introduction

The Wii console supports compressed texture format using S3TC (S3 Texture Compression), which is developed by S3, Inc.

2 S3TC and the Wii Console

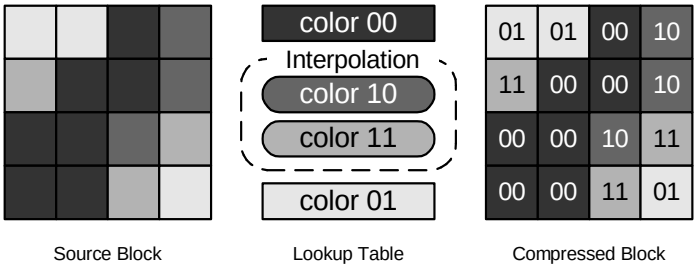
S3TC compressed texture format is one of the texture formats supported by the Wii console. This format makes it possible to greatly reduce data size compared with other non-compression formats, while supporting a high level of reproducibility. The development of compressed texture is carried out in real time using hardware, so there are fewer disadvantages in terms of speed when compared with other texture formats.

2.1 Compressed Texture Format

S3TC divides texture maps into 4x4 texel blocks. Two RGB565-formatted colors (representative colors), expressed in 16 bits, are linked to each texel block. For nontransparent texel blocks, a four-color lookup table is created from these two colors (00 and 01) and from two colors that are linearly interpolated (10 and 11) using these two colors. A two-bit index to the color lookup table is assigned to each texel in the texel block.

Figure 2-1 shows, from the left, the source texel block, the color lookup table, and the compressed texel block with the two-bit index assignments. Each of the texels in the compressed texel block is assigned the color that is closest to the source text from the four colors in the lookup table.

Figure 2-1 The Principle Behind S3TC

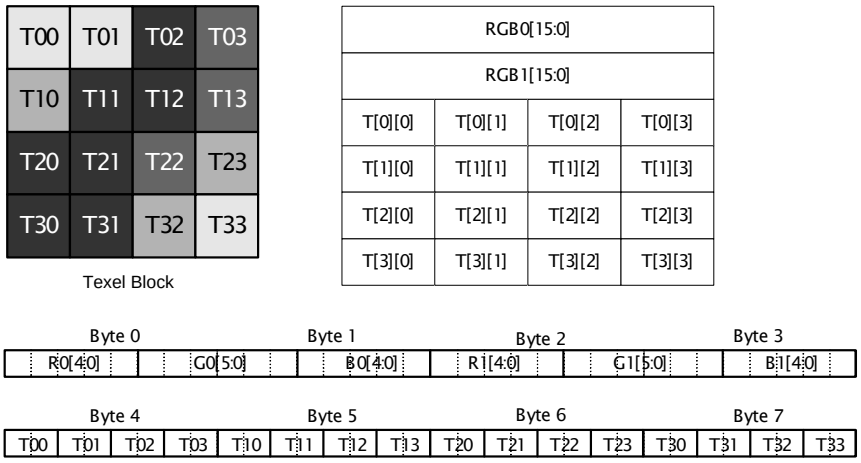


Compressed textures for the Wii console can have two stages of transparency: non-transparent or completely transparent. The numerical size of the two representative colors in the color lookup table determine if a texel block is completely non-transparent.

Figure 2-2 is the compressed texture texel block format for the Wii console. RGB0 and RGB1 are compared numerically, and if RGB0 > RGB1 the texture is judged to be a non-transparent texture. Otherwise, it is judged to be a texture containing transparent texels. If it is determined that the block contains transparent texels, the texels specified with (11) in the index will be the transparent texels. While those texels specified with (10) will be the average color of the two representative colors.

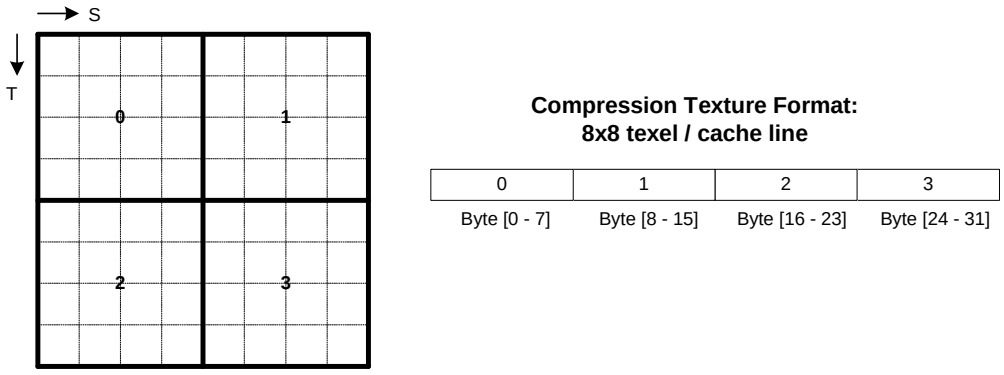
For the data size per texel block, there will be 2 16-bit colors and 16 2-bit indices for a total of 64 bits. The average texel will be four bits.

Figure 2-2 Compressed Texture Texel Block Format



Texture cache hardware actually retrieves the texture images in units of 32-byte tiles. With compressed texture, one tile is made up of an 8x8 grid of texels.

Figure 2-3 Texture Tile Format Using Compressed Texture



2.2 Advantages of Compressed Textures

- Reduced Data Size

  Compared with other types of texture format, S3TC compressed texture format allows a substantial reduction in data size (see Table 1). It is possible to improve image quality by allocating memory that has been freed up by compression to texture size, number, or mipmaps. Additionally, a reduction in data size means that less bandwidth is required to read in texture data, which leads to improvements in overall system performance.

Table 2-1 Comparison of Saved Data Size by Texture Format
(in bytes: GX_TF_C8 and GX_TF_C4 include texture look-up table)

Texture Size	GX_TF_RGB565	GX_TF_C8 (GX_TL_RGB565)	GX_TF_C4 (GX_TL_RGB565)	GX_TF_CMPR
8×8	128	576	64	32
64×64	8,192	4,608	2,080	2,048
256×256	131,072	66,048	32,800	32,768
1024×1024	2,097,152	1,049,088	524,320	524,288

- Expansion Speed

  Depending upon the hardware, compressed texture format can allow expansion without any speed penalties.

- Memory Conservation

  Compressed textures are stored, compressed, in the texture cache and are expanded immediately prior to texture filtering. For this reason, the main memory and texture cache can be conserved and the memory bandwidth from the main memory to the texture cache can be reduced. Additionally, hit rate of the texture cache is improved.

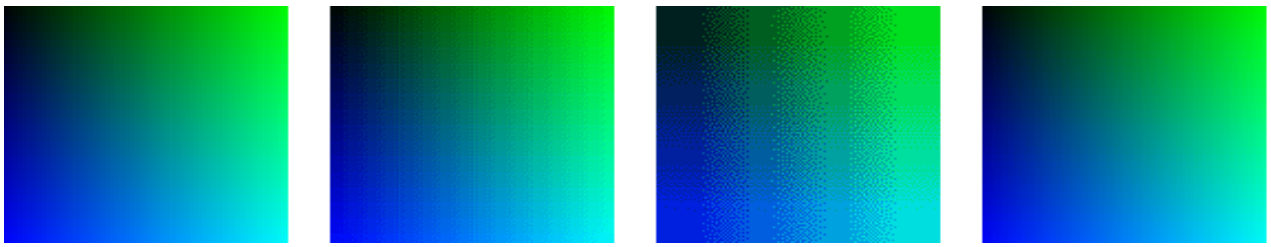
- Image Quality

  The color interpolation of the lookup table is performed with 24-bit precision. The result is that the compressed texture has a color quality rivaling 16- to 24-bit uncompressed textures.

- Images Suited to Compressed Texture

  Natural images with smooth color changes and gradation are reproduced with a high degree of accuracy (see Figure 2-4).

Figure 2-4 Comparison 1 of Images by Texture Format (128 x 128)



2.3 Disadvantages of Compressed Textures

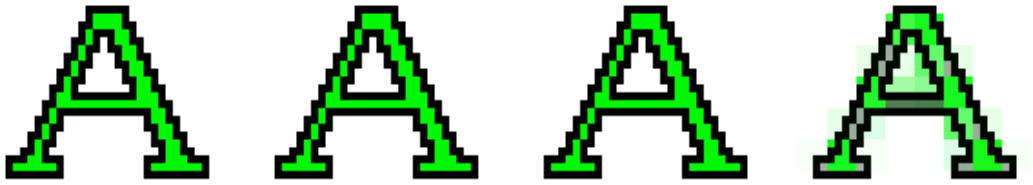
- Limited Transparency

  Compressed texture can only have non-transparent texels or completely transparent texels. Multi-stage transparency requires special handling, such as layering intensity textures and multi-textures (see Chapter 5 Using Compressed Textures).

- Images Not Suited to Compressed Texture

  Degradation is apparent in images with sharp color changes, such as pixel art (see Figure 2-5).

Figure 2-5 Comparison 2 of Images by Texture Format (32 x 32)



GX_TF_RGB565 GX_TF_C8 GX_TF_C4 GX_TF_CMPR

2.4 Comparison with Color Index Textures

- Data Size

  Table 2-2 shows a comparison of color index texture format and compressed texture format in terms of data size. One characteristic of color index texture format is that the larger the texture size, the more advantageous it is in terms of data size. Even so, compressed texture has more advantages for a variety of texture sizes.

Table 2-2 Comparison of Index Format and Compressed Format in Terms of Data Size (in bytes; GX_TF_C8 and GX_TF_C4 include texture look-up table)

Texture Size	GX_TF_C8 (GX_TL_RGB565)	GX_TF_C4 (GX_TL_RGB565)	GX_TF_CMPR
8×8	576	64	32
64×64	4,608	2,080	2,048
256×256	66,048	32,800	32,768
1024×1024	1,049,088	524,320	524,288

- Image Quality

  The number of colors that can be used in an entire picture is limited for each format with color index textures, so image quality declines as the number of colors increases. At the same time,

there is no limit to the number of colors with compressed textures, but there is a limit of four colors (three colors if there is a transparent texel) per texel block. It is impossible to say which image quality is better in all cases.

- Mipmaps

  With color index texture, you cannot specify GX_NEAR_MIP_LIN or GX_LIN_MIP_LIN in filter mode, when the texture is in a reduced area. This means that you cannot have smooth LOD interpolation when using mipmaps. At the same time, compressed textures allow the use of mipmaps the same way that other direct-reference textures do.

- Delays

  When multi-texturing with a mix of index textures and RGBA textures, performance can decline due to the weight of switching settings.

- Texture Look-Up Table (TLUT) Animation

  The ability to use TLUT animation is a significant advantage to index texture format.

3 Creating Compressed Textures

3.1 Tools That Can Create Compressed Textures

The following are some tools that can be used to create compressed textures.

- NintendoWare for Revolution Photoshop Plug-in

  This plug-in is used in Adobe Photoshop to generate texture files for NintendoWare for Revolution.

- `TexConv.exe`

  `TexConv.exe` is an application used to generate TPL files. A default plug-in file reader is provided for TGA files. For information on TPL files, refer to the *Texture Palette Library* (`TPL.pdf`).

- OPTPiX iMageStudio for Wii

  This is a product of Web Technology. It supports the TPL format.

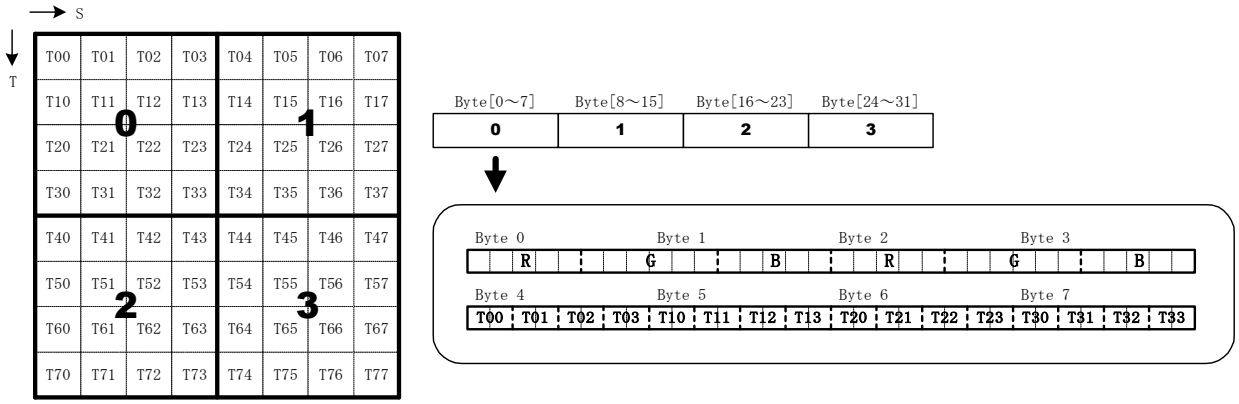
3.2 Creating Converters

3.2.1 Conversion to Hardware Format

The following points must be taken into consideration when converting image data into the hardware format.

- Each texel block contains 2 16-bit colors and 16 2-bit indices, for a total of 8 bytes. The texture cache hardware requests these converted texel blocks in 32-byte units, so compressed textures must be made up of individual 2x2 texel blocks (8x8 texels) (see Figure 3-1).
- Bytes 0 – 3 in each texel block are made up of the two 2-byte colors. These colors must be in big-endian format to be used on the Wii console hardware.
- Bytes 4 – 7 in each texel block are made up of the 16 2-bit indices. Each byte stores four 2-bit indices corresponding to a single (horizontal) row of the texel block. Refer to Figure 3-1 for the order of these indices. The bits in each 2-bit index are handled in little-endian format.
- A compressed texture's dimensions may be any multiple of four texels. The Wii console hardware requests texture images in 32-byte units, which for compressed textures is equivalent to 8x8 texels. If the dimensions of a source texture are not a multiple of eight texels, the region to the right or bottom must be padded using some value, such as zero. However, because the padding data will be wasted, it is probably best to create textures with a size based on a multiple of eight texels.

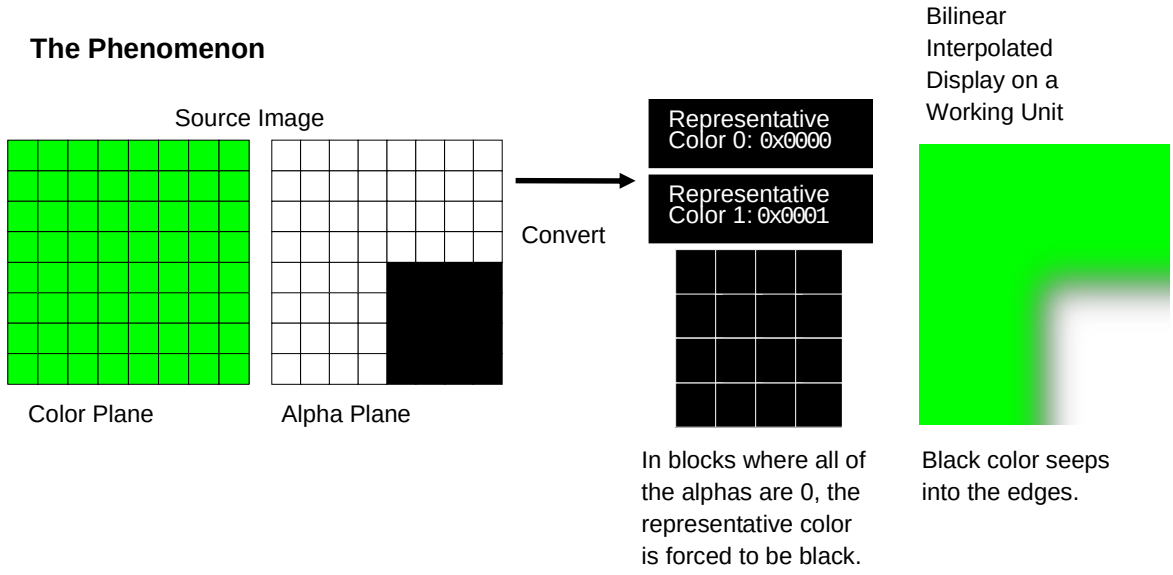
Figure 3-1 Structure of the 32-Byte Tile Requested by the Hardware



4 The Problem of Completely Transparent Texel Blocks

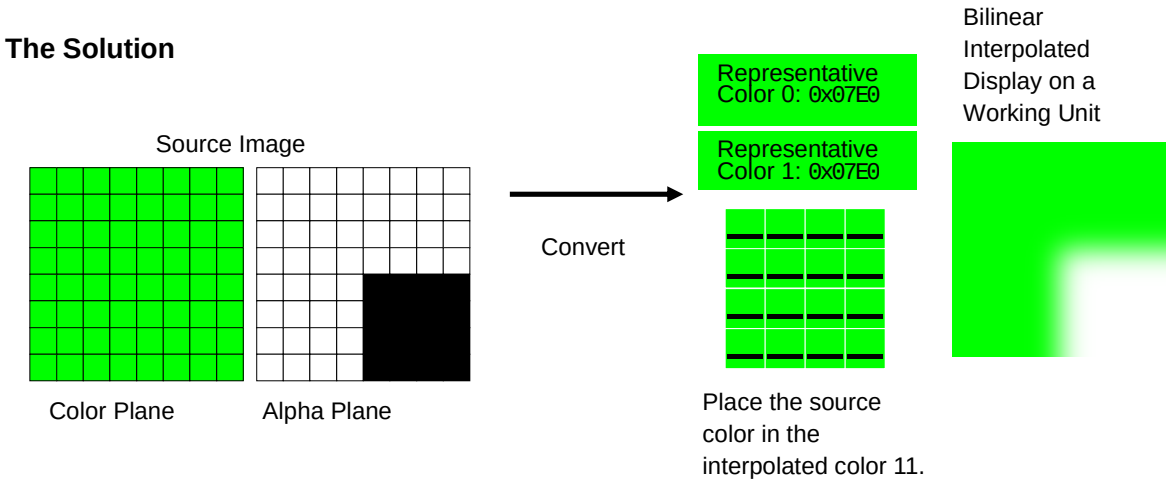
If GX_LINEAR is the setting for the texture filter mode, the texel value will be calculated by means of bilinear interpolation, using the decimal part of the ST value from the 4 texels near the ST coordinates of the pixel. Here, the color of the transparent texel block (a texel block for which all of the 4x4 texels are transparent) is also referenced as a reference texel color. If an inappropriate color is set as the representative color of a transparent texel block, displaying using bilinear interpolation will result in improper color components (see Figure 4-1).

Figure 4-1 The Problem of the Completely Transparent Texel Block



To avoid this problem, set an appropriate representative color in the texel block colors. When creating a texture, it is probably best to set the transparent color to be the same color as the surrounding area, and then have the converter adopt that color as the representative color, as shown in Figure 4-2.

Figure 4-2 Solution



5 Using Compressed Textures

Compressed textures can be used with the same types of procedures as other textures.

In the following example, to resolve the compressed texture problem of having only two stages of transparency, the I8 texture is layered with a multi-texture to effect multi-stage transparency.

Code 5-1 Example of Realization of Multi-Stage Transparency Using Compressed Texture and I8 Texture

```
#include <demo.h>
#define TEX_ID       0

/*---------------------------------------------------------------------------*
                          Model Data
 *---------------------------------------------------------------------------*/

// Vertex Coordinate Array
static s8 Vert_s8[] ATTRIBUTE_ALIGN(32) =
{
   -100,  100, 0,         // 0
    100,  100, 0,         // 1
   -100, -100, 0          // 2
};

//  Texture Coordinate Array
static u8 TexCoords_u8[] ATTRIBUTE_ALIGN(32) =
{
//   s    t
   0x00, 0x00, // 0
   0x01, 0x00, // 1
   0x00, 0x01  // 2
};

/*---------------------------------------------------------------------------*
                           Function Declaration
 *---------------------------------------------------------------------------*/

static void CameraInit( Mtx v );

/*---------------------------------------------------------------------------*
                          Main Loop
 *---------------------------------------------------------------------------*/
```

```
void main ( void )
{
    PADStatus    pad[4];
    GXTexObj     CmprTexObj;
    GXTexObj     I8TexObj;
    Mtx          v;
    u8           i;
    TEXPalettePtr CmprTpl = 0;
    TEXPalettePtr I8Tpl   = 0;

    pad[0].button = 0;

    DEMOInit(NULL);
    CameraInit(v);
    GXLoadPosMtxImm(v, GX_PNMTX0);

    GXClearVtxDesc();
    GXSetVtxDesc(GX_VA_POS,  GX_INDEX8);
    GXSetVtxDesc(GX_VA_TEX0, GX_INDEX8);

    GXSetArray(GX_VA_POS,  Vert_s8, 3*sizeof(s8));
    GXSetArray(GX_VA_TEX0, TexCoords_u8, 2*sizeof(u8));

    GXSetVtxAttrFmt(GX_VTXFMT0, GX_VA_POS,  GX_POS_XYZ,  GX_S8,    0);
    GXSetVtxAttrFmt(GX_VTXFMT0, GX_VA_TEX0, GX_TEX_ST,   GX_U8,    0);

    TEXGetPalette(&CmprTpl, "CmprTexture.tpl");
    TEXGetPalette(&I8Tpl, "I8Texture.tpl");

    TEXGetGXTexObjFromPalette(CmprTpl, &CmprTexObj, TEX_ID);
    TEXGetGXTexObjFromPalette(I8Tpl, &I8TexObj, TEX_ID);

    GXLoadTexObj(&CmprTexObj, GX_TEXMAP0);
    GXLoadTexObj(&I8TexObj, GX_TEXMAP1);

    GXSetTexCoordGen(GX_TEXCOORD0, GX_TG_MTX2x4, GX_TG_TEX0, GX_IDENTITY);
    GXSetTexCoordGen(GX_TEXCOORD1, GX_TG_MTX2x4, GX_TG_TEX0, GX_IDENTITY);

    GXSetNumTexGens(2);

    GXSetNumChans(0);
```

```
GXSetNumTevStages(2);

// Stage 0 hands over the CMPR Texture Colors to the subsequent stage without
// modification.
GXSetTevOrder(
   GX_TEVSTAGE0,
   GX_TEXCOORD0,
   GX_TEXMAP0,    // CMPR Texture
   GX_COLOR_NULL);
GXSetTevOp(GX_TEVSTAGE0, GX_REPLACE);

// Stage 1 outputs the colors it receives from the previous stage unchanged, and
// substitutes the alphas with the input from the I8 texture.
GXSetTevOrder(
   GX_TEVSTAGE1,
   GX_TEXCOORD1,
   GX_TEXMAP1,        // I8 Texture
   GX_COLOR_NULL);
GXSetTevColorIn(
   GX_TEVSTAGE1,
   GX_CC_ZERO,
   GX_CC_ZERO,
   GX_CC_ZERO,
   GX_CC_CPREV );     // Input = Previous Stage Output Color
GXSetTevColorOp(
   GX_TEVSTAGE1,
   GX_TEV_ADD,
   GX_TB_ZERO,
   GX_CS_SCALE_1,
   GX_ENABLE,
   GX_TEVPREV );      // Output Color = Previous Stage Output Color
GXSetTevAlphaIn(
   GX_TEVSTAGE1,
   GX_CA_ZERO,
   GX_CA_ZERO,
   GX_CA_ZERO,
   GX_CA_TEXA );      // Input = I8 Texture Alpha
GXSetTevAlphaOp(
   GX_TEVSTAGE1,
   GX_TEV_ADD,
   GX_TB_ZERO,
```

```
        GX_CS_SCALE_1,
        GX_ENABLE,
        GX_TEVPREV );      // Output Alpha = I8 Texture Alpha

    GXSetBlendMode(GX_BM_BLEND, GX_BL_SRCALPHA, GX_BL_INVSRCALPHA, GX_LO_CLEAR);


    while(!(pad[0].button & PAD_BUTTON_MENU))
    {
        DEMOBeforeRender();
        GXBegin(GX_TRIANGLES, GX_VTXFMT0, 3);
        for (i = 0; i < 3; i++) {
            GXPosition1x8(i);
            GXTexCoord1x8(i);
        }
        GXEnd();
        DEMODoneRender();
        PADRead(pad);
    }

    OSHalt("End of demo");
}

static void CameraInit ( Mtx v )
{
    Mtx44 p;
    Vec   camPt = {0.0F, 0.0F, 800.0F};
    Vec   at    = {0.0F, 0.0F, -100.0F};
    Vec   up    = {0.0F, 1.0F, 0.0F};

    MTXFrustum(p, 240.0F,-240.0F,-320.0F, 320.0F, 500, 2000);
    GXSetProjection(p, GX_PERSPECTIVE);
    MTXLookAt(v, &camPt, &up, &at);
}
```

6 Summary

On earlier gaming devices, it was a general practice to use color index textures to reduce the space that textures occupy. However, with the Wii console, you could say that compressed textures are the most advantageous in terms of efficiency. Additionally, there is the limitation that mipmaps cannot be used in color index textures, but there are no such limitations when using compressed textures. They can be used in the same way as normal direct-reference textures.

Although image quality may suffer in some cases when there are sharp changes in color (such as for pixel art), good results should be possible most of the time.

We recommend you consider compressed textures if you are unsure of which texture format to use.

S3TC (S3 Texture Compression) is a texture compression technology developed by S3, Inc.

Adobe and Photoshop are the trademarks or registered trademarks of Adobe Systems Incorporated.

OPTPiX, web technology, and iMageStudio are the trademarks or registered trademarks of Web Technology, Inc.

All other company and product names are the trademarks or registered trademarks of their respective companies.