Compressed Texture Format

(Provisional Format)

Version: September 29, 2006

Compressed Texture Format (Provisional Format)

Version: September 29. 2006

Contents

Code Examples

Figures

Tables

Revision History

Version	Date Revised	Items (Chapter)	Description
9/29/2006	9/29/2006	-	Initial Version

1 Introduction

Wii supports compressed texture format using S3TC (S3 Texture Compression). Compressed textures using the S3TC format is developed by S3, Inc.

2 S3TC and Wii

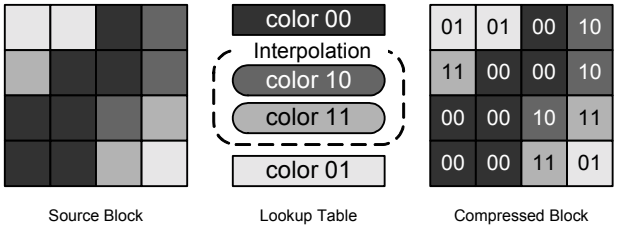
The S3TC compressed texture format is one of the texture formats supported by the Wii console. This format makes it possible to greatly reduce data size compared with other non-compressed formats, while supporting a high level of reproducibility. The decompression of a compressed texture is carried out in real time using hardware, so there is no disadvantage 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 format 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 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.

In the figure below, working from the left, the source texel block, the color lookup table and the compressed texel block with the two-bit index assignments are shown. 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 1 - The Principle behind S3TC

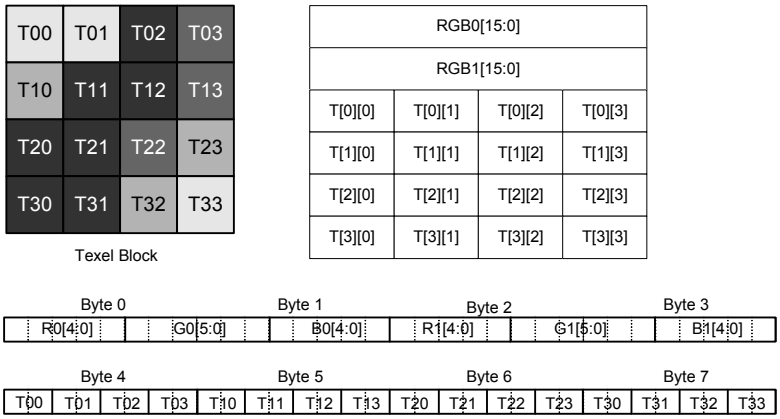


Wii compressed textures can have two stages of transparency, non-transparent or completely transparent. The numerical size of the two representative colors in the color lookup table determine if a texel block is completely non-transparent.

The following figure illustrates the compressed texture texel block format in Wii. 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.

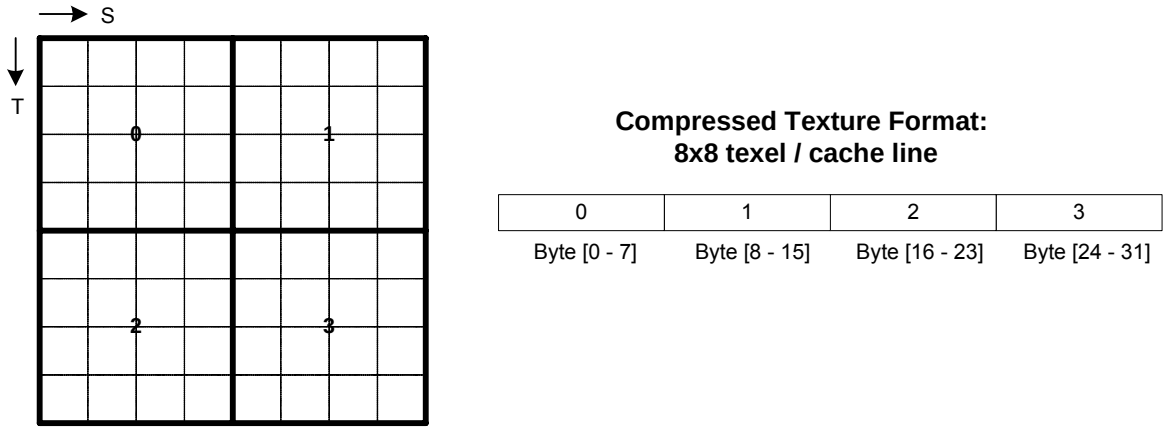
For the data size per texel block, there will be two 16-bit colors and 16 2-bit indices for a total of 64 bits. The average texel will be four bits.

Figure 2 - Compressed Texture Texel Block Format



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

Figure 3 - Texture Tile Format Using Compressed Textures



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 the table below). It is possible to improve image quality by allocating memory that has been freed up by compressed to the size or number of textures, 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 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 x 8	128	576	64	32
64 x 64	8,192	4,608	2,080	2,048
256 x 256	131,072	66,048	32,800	32,768
1024 x 1024	2,097,152	1,049,088	524,320	524,288

- Decompressed Speed

  Due to hardware, compressed texture format can allow decompressed without any speed penalties.

- Memory Conservation

  Compressed textures are stored, compressed, in the texture cache and are decompressed 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.

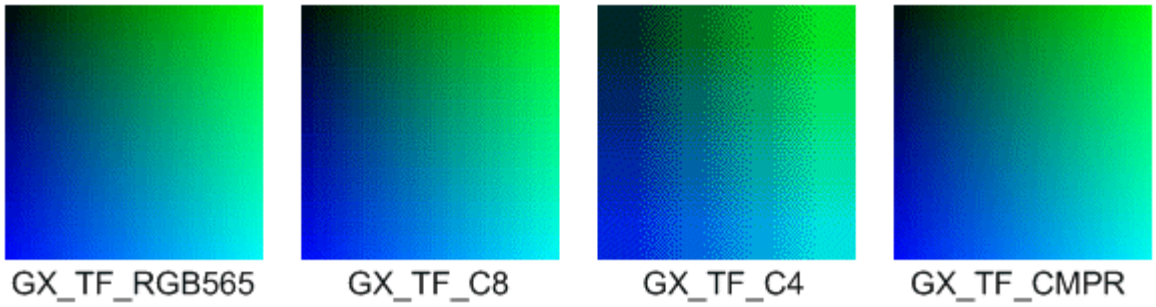
- Image Quality

  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 Textures

  Natural images with smooth color changes and gradation are reproduced with a high degree of accuracy (see the following figure).

Figure 4 - Comparison of Graphics by Texture Format #1 (128 x 128)



2.3 Disadvantages of Compressed Textures

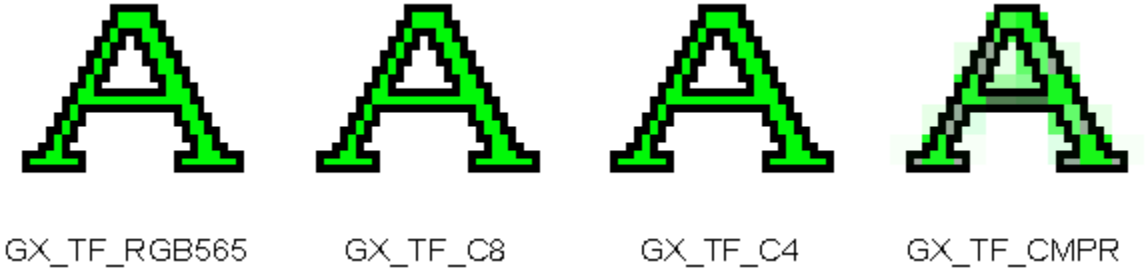
- Transparency Limitations

  Compressed textures can only have non-transparent texels or completely transparent texels. Multi-stage transparency requires special handling, such as layering intensity textures and multitextures (see "5 Using Compressed Textures" on page 11).

- Images Not Suited to Compressed Textures

  Degradation is apparent in images with stark color changes, such as dot images (see the following figure).

Figure 5 - Comparison of Graphics Using Texture Format 2 (32 x 32)



2.4 Comparison with Color Indexed Textures

- Data Size

  The table below shows a comparison of color indexed texture format and compressed texture format in terms of data size. One characteristic of color indexed texture format is: the larger the texture size, the more advantageous it is in terms of data size. Even so, compressed texture has advantages for all texture sizes.

  Table 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 x 8	576	64	32
64 x 64	4,608	2,080	2,048
256 x 256	66,048	32,800	32,768
1024 x 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 indexed 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 indexed textures, 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 indexed 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 indexed 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 for creating texture files for NintendoWare for Revolution.
- TexConv.exe

  TexConv.exe is an application that generates TPL files. There is a default plug-in file reader for TGA files. For information on TPL files, see “Texture Palette Library” (TPL.pdf).
- OPTPiX iMageStudio for Wii

  This is a product of Web Technology. It is compatible with TPL format.

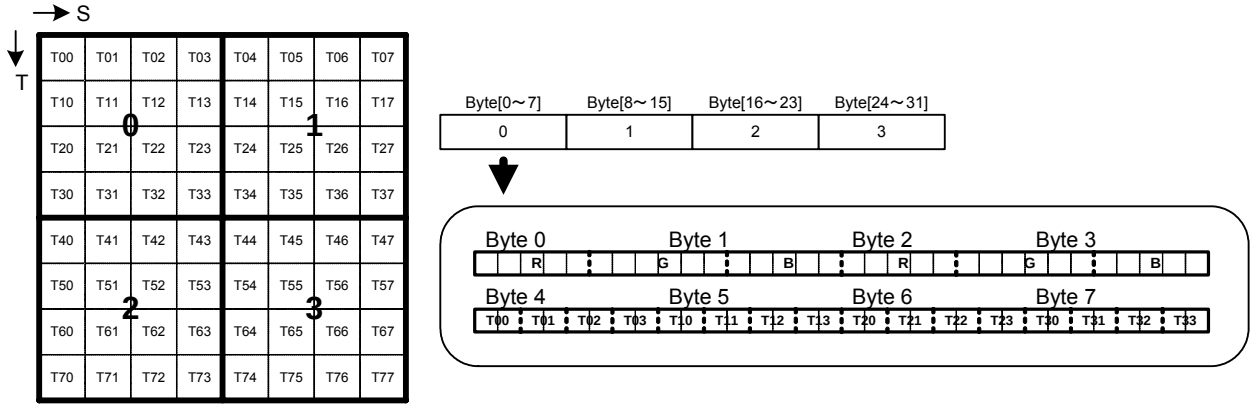
3.2 Converter Creation

3.2.1 Conversion to Hardware Format

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

- Each texel block is made up of a total of 8 bytes: two 16-bit colors and 16 2-bit indeces. The texture cache hardware takes these converted texel blocks in 32-byte units, so the compressed texture has to be prepared in 2x2 units of texel blocks, in other words, in 8x8 texel units (see "Figure 6 - Structure of the 32-byte Tile Required by the Hardware" on page 7).
- Bytes 0 – 3 in each texel block are made up of two 2-byte colors. These colors must be in big-endian format in order to use them on the Wii hardware.
- Bytes 4 – 7 in each texel block are made up of 16 2-bit indeces. For each byte there are four 2-bit indices, each corresponding to one horizontal row in the texel block. See Chapter 6 for information on ordering these indeces. Bits are treated as little-endian inside each of the 2-bit indeces.
- The compressed texture dimensions may be any multiple of the 4 texels. However, the Wii hardware will request a 32-byte aligned texture image, so it will be 8x8 texels when using compressed textures. If the source texture dimensions are not multiples of 8 texels, it will be necessary to pad the area on the right side or underneath with zeroes or another character. Note that the padding data will be wasted, so it would probably be a good idea to consider multiples of 8 texels as being the basic element when creating textures.

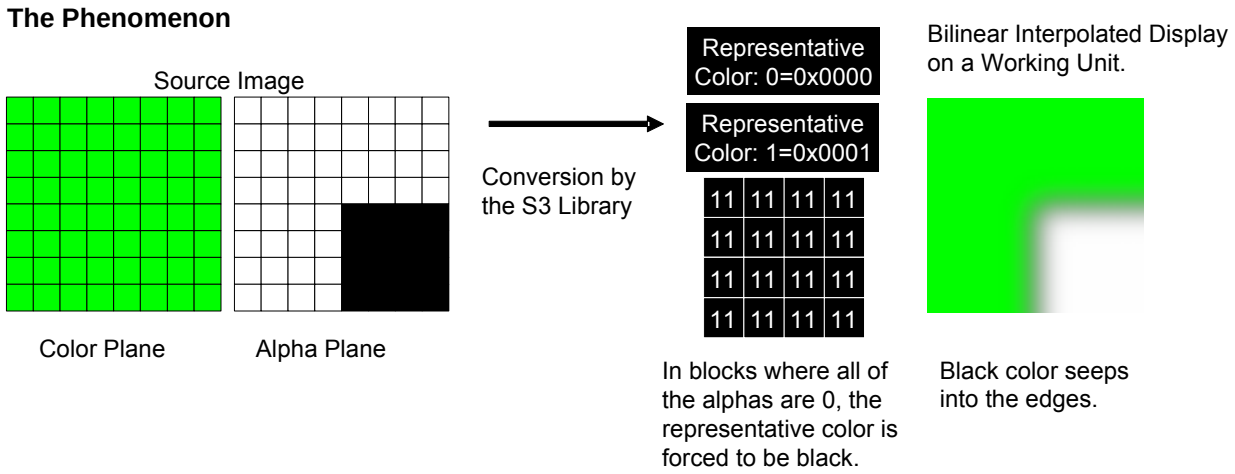
Figure 6 - Structure of the 32-byte Tile Required by the Hardware



4 Problems with Completely Transparent Texel Blocks

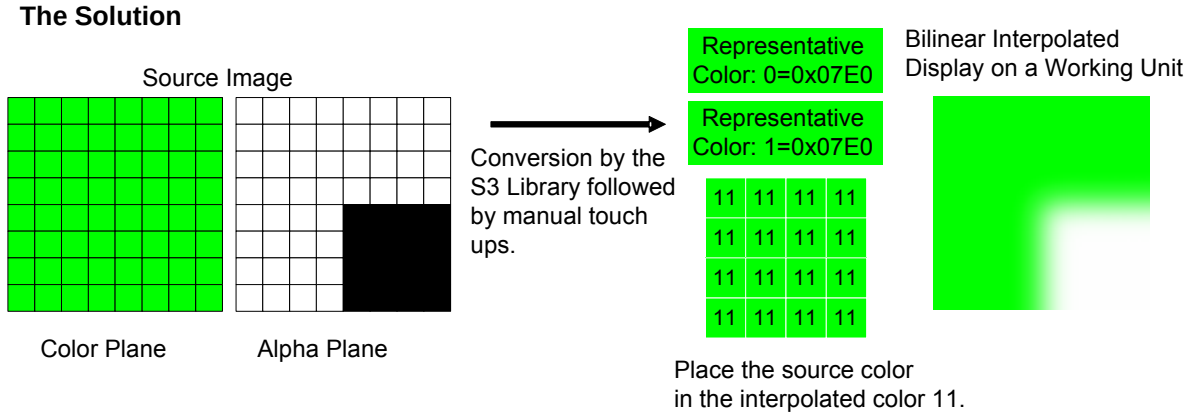
If GX_LINEAR is the setting for the texture filter mode, the texel value will be calculated by means of bilinear interpolation, using the decimal portion of the ST value from the 4 texels near the ST coordinates of the pixel. Here, the color of a transparent texel block (texel block in which all of the 4x4 texels are transparent) is also referenced as a reference texel color. As Figure 7 shows, if an inappropriate color is set as the representative color for transparent texel blocks, displaying bilinear interpolation will result in improper color components.

Figure 7 - An Issue with Completely Transparent Texel Blocks



To avoid this problem, you must set an appropriate representative color in the texel block colors. It is a good idea to also enter the color of the periphery as a transparent color, and have the converter use this color as a representative color (Figure 8).

Figure 8 - Solution



5 Using Compressed Textures

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

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

Code 1 - Example of Realization of Multi-Stage Transparency Using Compressed Textures 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();
        // Drawing Triangles
        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 a conventional gaming device, it has been a general practice to use color indexed textures to reduce the the space that textures occupy. However, with 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 indexed textures, but there are no such limitations when using compressed textures. They can be used in the same way as normal direct-reference textures.

Where there is the inconvenience of certain textures (such as pixel art) having stark color changes when compressed, good results should be possible in most cases with compressed textures.

When trying to decide which texture format to use for your game, we recommend considering compressed textures.