Using Texture Compression in OpenGL

Sébastien Dominé
sebastien.domine@nvidia.com
NVIDIA Corporation

Overview

The OpenGL extension, ARB_texture_compression, used in conjunction with the EXT_texture_compression_s3tc extension offers a tremendous boost for the rendering pipeline at different levels:

- Faster rendering
- Lower texture memory requirements
- Faster texture downloads into texture memory
- Lower disk space requirements for storage and faster disk access.

Additionally, the use of compress textures allows the application to utilize higher resolution textures with the same memory footprint.

Note: Comments will be made throughout the document to reflect specific S3TC information and issues. A bold S3TC tag will be located in the margin to notify the reader.

Let's take a look at how a developer can best make user of texture compression in their titles. We can divide up the process into two parts: development and runtime usage.

Creating Compressed Textures During Development.

A. Production
a. Generic Method
    1. Compress uncompressed images using GL
    2. Save the compressed image onto disk
b. S3TC DDS file format
    1. Compress uncompressed images using ISV's S3TC compression tools
    2. Load DDS files in GL

Runtime Usage.

B. Runtime
    1. Load compressed images from disk
    2. Upload compressed images to GL
    3. Do not compress dynamic textures

Production

The optimal method for using compressed textures is to pre-compress all the textures during the production of the title. This reduces the disk storage requirements for the textures and accelerates runtime texture loading from disk. There are two different approaches to performing the compression of the texture bitmaps. The first method uses OpenGL to effectively compress the textures. The second describes an S3TC alternative file format supported by ISV's that can be used directly with OpenGL.

Generic Method

The ARB_texture_compression extension allows an uncompressed texture to be compressed on the fly through the `glTexImage2D` call by setting its `<internalFormat>` parameter accordingly. This can be done in one of two ways: use a generic compressed internal format from Table 1 or use an explicit internal format like one offered by the S3TC extension listed in Table 2. Basically, the "compressed internal format" works just like a texture with "base internal format" except that the data is compressed.

Table 1: Generic compressed internal formats and their corresponding uncompressed input format.

Generic Compressed Internal Format	Base Internal Format
GL_COMPRESSED_RGB_ARB	RGB
GL_COMPRESSED_RGBA_ARB	RGBA
GL_COMPRESSED_ALPHA_ARB	ALPHA
GL_COMPRESSED_LUMINANCE_ARB	LUMINANCE
GL_COMPRESSED_LUMINANCE_ALPHA_ARB	LUMINANCE_ALPHA
GL_COMPRESSED_INTENSITY_ARB	INTENSITY

S3TC

Table 2: S3TC compressed internal formats and their corresponding uncompressed input format.

S3TC Compressed Internal Format	Base Internal Format
GL_COMPRESSED_RGB_S3TC_DXT1_EXT	RGB
GL_COMPRESSED_RGBA_S3TC_DXT1_EXT	RGBA
GL_COMPRESSED_RGBA_S3TC_DXT3_EXT	RGBA
GL_COMPRESSED_RGBA_S3TC_DXT5_EXT	RGBA

Note: The S3TC compressor only accepts uncompressed RGB and RGBA input formats to be compressed.

Additionally, the use of proxy textures can indicate if whether a texture image is going to be compressed, before it actually gets compressed.

Next, make sure that the image has been properly compressed by checking the `glGetTexLevelParameteriv` with `<pname>` set to `GL_TEXTURE_COMPRESSED_ARB`. The returned `<params>` will be non-zero if the texture is effectively compressed. On the other hand, if the driver didn't compress the image (for whatever reason), it will treat the image as if `<internalFormat>` were the corresponding base internal format.

Then, if a generic compressed internal format was used, query OpenGL for the actual `<internalFormat>` that has been automatically selected by OpenGL. To do so, one calls

`glGetTexLevelParameteriv` again with `<pname>` set to `GL_TEXTURE_INTERNAL_FORMAT`.

In the next step, query OpenGL to obtain the size of the compressed texture image through another `glGetTexLevelParameteriv` call. This time, `<pname>` should be set to `GL_TEXTURE_COMPRESSED_IMAGE_SIZE_ARB`. Now create a buffer of the returned size.

Next, fetch the compressed texture data by calling `glGetCompressedTexImageARB` and passing the address of the buffer in `<img>`.

All that's left is to save the vital information describing the compressed texture to disk. Specifically, save the following attributes, in addition to the compressed texture itself, for later use at runtime:

- Size of the buffer
- Compressed internal format
- Width
- Height
- Border - If not S3TC (see S3TC note below)
- Depth - Only relevant for 3D textures and if not using S3TC (see S3TC note below)

S3TC

- `<border>` should be set to zero when the compressed internal format is one of the S3TC's. This means that `glCompressedTexImage2DARB` will produce an `INVALID_OPERATION` error if `<border>` is non-zero.
- `<depth>` is irrelevant for S3TC formats since they are only applicable to 2D textures, meaning, `glCompressedTexImage1DARB` and `glCompressedTexImage3DARB` will produce an `INVALID_ENUM` error if `<internalFormat>` is an S3TC format.

Cross platform issues can arise when the compressed internal format is not standard, and so ensure that the platform supports the specific format. One may enumerate the supported compressed texture formats by calling `glGetIntegerv` with `GL_NUM_COMPRESSED_TEXTURE_FORMATS_ARB` and `GL_COMPRESSED_TEXTURE_FORMATS_ARB`. A code example is given in the Code Listing 1.

```
GLint * compressed_format;
GLint   num_compressed_format;

glGetIntegerv(GL_NUM_COMPRESSED_TEXTURE_FORMATS_ARB, &num_compressed_format);

compressed_format = (GLint*)malloc(num_compressed_format * sizeof(GLint));

glGetIntegerv(GL_COMPRESSED_TEXTURE_FORMATS_ARB, compressed_format);
```

Code Listing 1: How to enumerate the supported Compressed Internal Format.

S3TC

Be aware that the enumerated format list is not necessarily the same as the list of supported formats. The main purpose of the format enumeration is to enumerate "normal-looking" formats so that an application can try them out without having any knowledge about specific extensions. This explains why the S3TC format enumeration doesn't refer to the `GL_COMPRESSED_RGBA_S3TC_DXT1_EXT` format, which is understood as not being a "normal" RGBA format.

The Code Listing 2 summarizes this process.

```
glBindTexture(GL_TEXTURE_2D, compressed_decal_map);
glTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGB_ARB, width, height,
                 0, GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);

glGetTexLevelParameteriv(GL_TEXTURE_2D, 0, GL_TEXTURE_COMPRESSED_ARB, &compressed);

/* if the compression has been successful */
if (compressed == GL_TRUE)
{
  glGetTexLevelParameteriv(GL_TEXTURE_2D, 0, GL_TEXTURE_INTERNAL_FORMAT,
&internalformat);

  glGetTexLevelParameteriv(GL_TEXTURE_2D, 0, GL_TEXTURE_COMPRESSED_IMAGE_SIZE_ARB,
&compressed_size);

  img = (unsigned char *)malloc(compressed_size * sizeof(unsigned char));

  glGetCompressedTexImageARB(GL_TEXTURE_2D, 0, img);

  SaveTexture(width, height, compressed_size, img, internalFormat, 0);
}
```

Code Listing 2: Compressing an uncompressed image and saving it out to disk.

There are several ways of evaluating the quality of a given compression extension. One can either query the estimated image quality parameters (`RED_BITS`, `GREEN_BITS`, …) or actually compress the texture map. For the later evaluation method, one can access the uncompressed image buffer by calling `glGetTexImage` appropriately.

S3TC DDS file format

S3TC

The DDS file format provides an alternative to handling compressed textures. ISV utilities like S3's Adobe PhotoShop plug-in and Microsoft DirectX's Dxtex to do the conversion from generic file formats to .dds files. DDS stands for Direct Draw Surface, which means that the file is basically a memory dump of a DirectX Direct Draw Surface onto disk. Thus, the DDS file reader given in Code Listing 3 has to include DirectX's `ddraw.h` include file in order to get the `DDSURFACEDESC2` definition. Another implication in dealing with DirectX structures is the fact that DirectDraw's screen coordinate origin is the top-left corner of the screen and OpenGL's origin is the bottom-left corner. This leads to the need of flipping the texture vertically before converting the image or taking the opposite `t` texture coordinate.

```
#include <ddraw.h>

gliGenericImage *
ReadDDSFile(const char *filename, int * bufsize, int * numMipmaps)
{
  gliGenericImage *genericImage;
  DDSURFACEDESC2 ddsd;
  char filecode[4];
  FILE *fp;

  /* try to open the file */
  fp = fopen(filename, "rb");
  if (fp == NULL)
    return NULL;

  /* verify the type of file */
  fread(filecode, 1, 4, fp);
  if (strncmp(filecode, "DDS ", 4) != 0) {
    fclose(fp);
    return NULL;
  }

  /* get the surface desc */
  fread(&ddsd, sizeof(ddsd), 1, fp);

  genericImage = (gliGenericImage*) malloc(sizeof(gliGenericImage));
  memset(genericImage,0,sizeof(gliGenericImage));
  /* how big is it going to be including all mipmaps? */
  *bufsize = ddsd.dwMipMapCount > 1 ? ddsd.dwLinearSize * 2 : ddsd.dwLinearSize;
  genericImage->pixels = (unsigned char*)malloc(*bufsize * sizeof(unsigned char));
  fread(genericImage->pixels, 1, *bufsize, fp);
  /* close the file pointer */
  fclose(fp);

  genericImage->width       = ddsd.dwWidth;
  genericImage->height      = ddsd.dwHeight;
  genericImage->components  = (ddsd.ddpfPixelFormat.dwFourCC == FOURCC_DXT1) ? 3 : 4;
  *numMipmaps = ddsd.dwMipMapCount;
  switch(ddsd.ddpfPixelFormat.dwFourCC)
  {
    case FOURCC_DXT1:
      genericImage->format = GL_COMPRESSED_RGBA_S3TC_DXT1_EXT;
      break;
    case FOURCC_DXT3:
      genericImage->format = GL_COMPRESSED_RGBA_S3TC_DXT3_EXT;
      break;
    case FOURCC_DXT5:
      genericImage->format = GL_COMPRESSED_RGBA_S3TC_DXT5_EXT;
      break;
    default:
      free(genericImage->pixels);
      free(genericImage);
      return NULL;
  }

  /* return data */
  return genericImage;
}
```

Code Listing 3: DDS file reader

The last warning that needs to be made about DDS files regards mipmaps. DDS files contain all the compressed mipmaps. This is problematic since GL takes one mipmap at a time. The buffer read for the DDS file cannot be passed in directly to GL. You need to compute the buffer offsets for each mipmap and pass in the correct data address and attributes to `glCompressedTexImage2DARB`. The Code Listing 4 shows how to

implement the process. For a full explanation on how to use the `glCompressedTexImage2DARB` function, please refer to the Runtime section, page 8.

```
/* load the .dds file */
  ddsimage = ReadDDSFile("flowers.dds",&ddsbufsize,&numMipmaps);
  height   = ddsimage->height;
  width    = ddsimage->width;
  offset   = 0;
  blockSize = (ddsimage->format == GL_COMPRESSED_RGBA_S3TC_DXT1_EXT) ? 8 : 16;

  glBindTexture(GL_TEXTURE_2D, dds_compressed_decal_map);
  /* load the mipmaps */
  for (i = 0; i < numMipmaps && (width || height); ++i)
  {
    if (width == 0)
      width = 1;
    if (height == 0)
      height = 1;

    size = ((width+3)/4)*((height+3)/4)*blockSize;
    glCompressedTexImage2DARB(GL_TEXTURE_2D, i, ddsimage->format, width, height,
      0, size, ddsimage->pixels + offset);

    GLErrorReport();
    offset += size;
    width  >>= 1;
    height >>= 1;
  }
```

Code Listing 4: How to download the DDS mipmaps into GL

Finally, we notice that Microsoft's DirectX SDK DXTex utility offers the capability to compress cube maps with mipmaps into a single DDS file. This feature is handy since it encloses all the data needed for one cubemap. The DDS file reader should be modified accordingly in order to read the different faces of the cube map. The ordering is matching the DirectX cube map documentation, which also matches GL.

S3TC information

S3TC compression ratios are fixed on a per-format basis. A list of compression ratios is given below.

Table 3: Compression ratios per Compressed S3TC format.

S3TC Compression Format	Compression (bits/texel)	Compression ratio (8 bits/channel)
GL_COMPRESSED_RGB_S3TC_DXT1_EXT	4	6:1 / 8:1
GL_COMPRESSED_RGBA_S3TC_DXT3_EXT	8	4:1
GL_COMPRESSED_RGBA_S3TC_DXT5_EXT	8	4:1

Note: Because 24-bit textures are actually stored as 32-bit textures, `GL_COMPRESSED_RGB_S3TC_DXT1_EXT` format can be seen as having an effective 8:1 compression ratio. Also, the S3TC compression formula is:

ImageSize = blockSize * ceil(width / 4) * ceil(height / 4)

Where bloackSize is 8 bytes for DXT1 and 16 bytes for DXT3/5.

The images below compare the quality between compressed and uncompressed images at various resolutions using different minification and magnification filters.

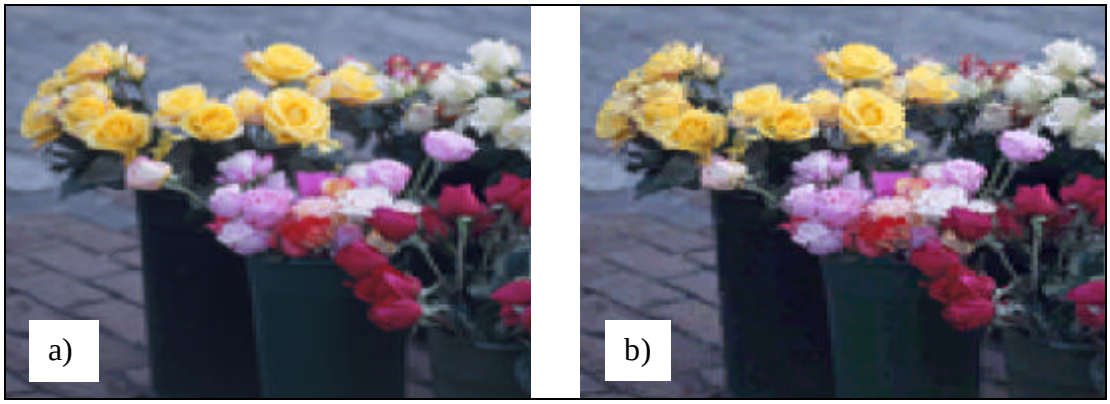

Figure 1: On the left, an uncompressed 128x128 texture; on the right, its compressed S3TC RGB DXT1 counterpart (Minification and magnification filter set to GL_NEAREST)

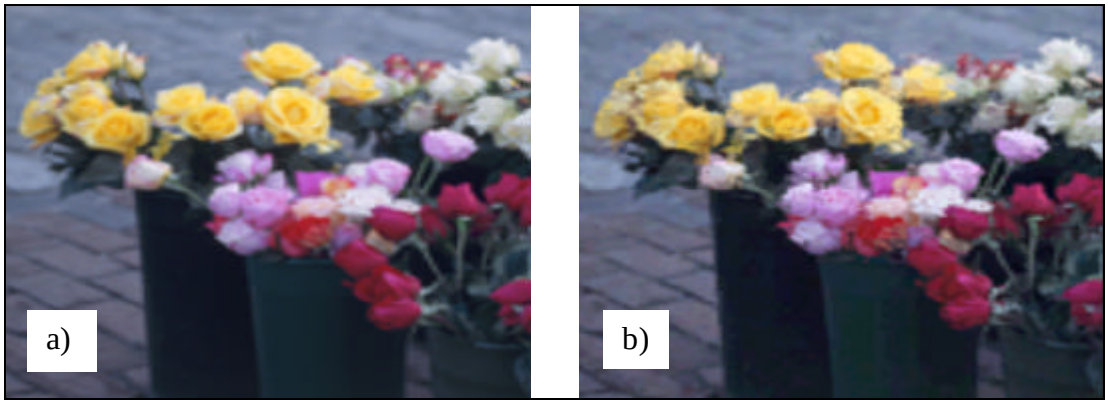

Figure 2: a) uncompressed 128x128 texture b) same texture compressed with S3TC RGB DXT1 (Minification and magnification filter set to GL_LINEAR)

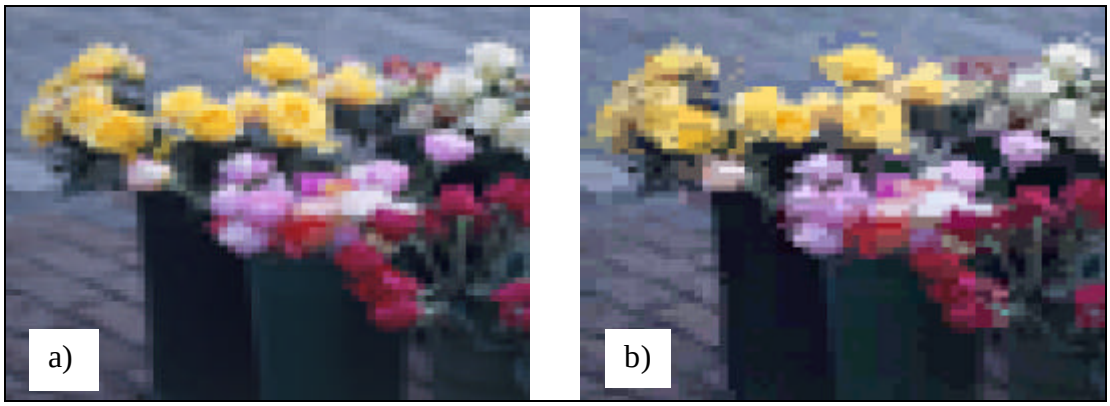

Figure 3: On the left, an uncompressed 64x64 texture; on the right, its compressed S3TC RGB DXT1 counterpart (Minification and magnification filter set to GL_NEAREST)

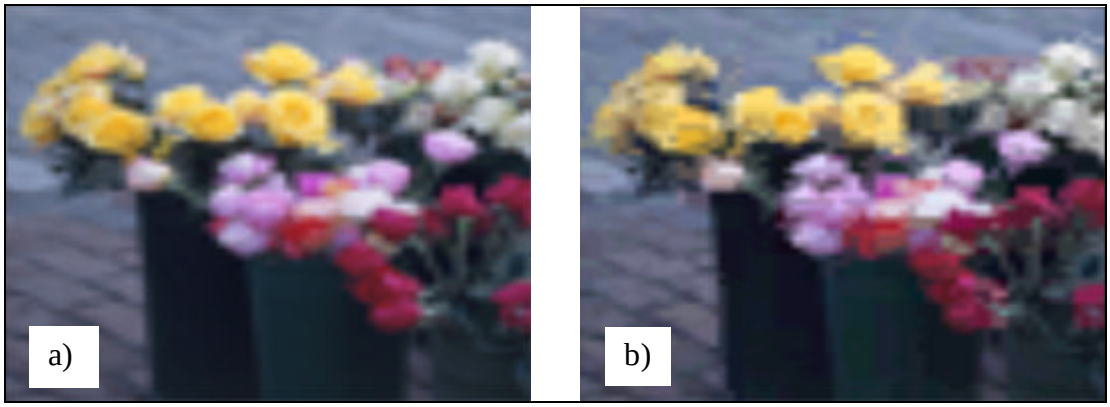

Figure 4: On the left, an uncompressed 64x64 texture; on the right, the same texture compressed with S3TC RGB DXT1 (Minification and magnification filter set to GL_LINEAR)

Another interesting comparison is to put one uncompressed 64x64 side by side with a compressed 128x128. In this particular case, the memory footprint is identical but the compressed image appears to be better looking (See Figure 5).

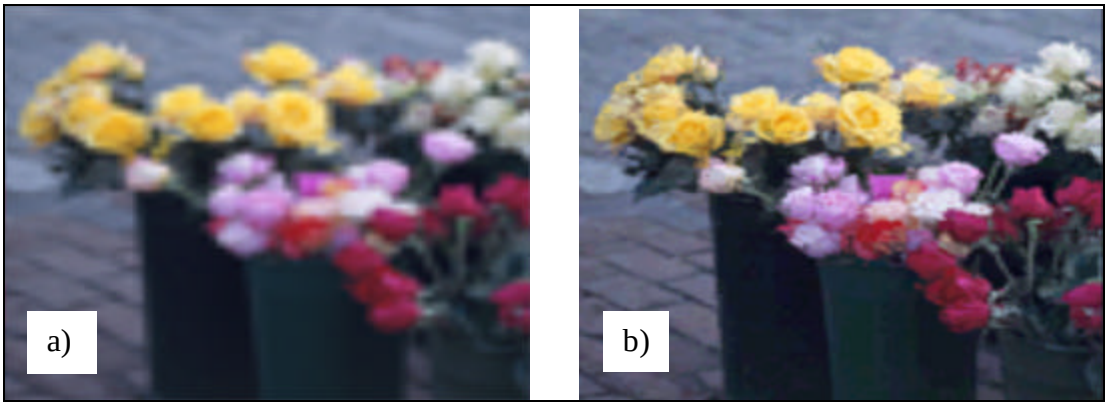

Figure 5: On the left, an uncompressed 64x64 texture; on the right a compressed RGB_DXT1 128x128 sibling (Minification and magnification filter set to GL_LINEAR). One should notice that those two textures take the same amount of memory, but the compressed version appears much sharper.

Runtime

Once the textures are compressed, they need to be decompressed during runtime. applications may now take full advantage of compressed textures by loading them directly from disk and directly downloading them to OpenGL. This method offers several advantages. First, it minimizes the size of the compressed texture on disk, offering a fast load from the media. Second, the memory transfer from the host memory to the texture memory is optimized because compressed textures are much smaller than their uncompressed counterparts. Most important, the individual textures within texture memory are smaller, allowing more textures to be resident in video memory

simultaneously, thus minimizing the potential for costly texture fetching from AGP or host memory. Finally, compressed textures reduce the memory bandwidth consumption while rendering. Each of these can contribute to faster rendering. As an example, running Quake3 in 32-bit color with texture detail set to maximum leads to a 20% speed increase.

To get started with the runtime procedure, first read the texture files from disk. Once the width, height, border, depth, internal format, the size of the buffer and the image itself are available, `glCompressedTexImage2D` is called with `<internalFormat>` set to the internal format value read from disk. By doing this, runtime compression doesn't occur because the compressed texture is being transferred directly to texture memory.

```
glBindTexture(GL_TEXTURE_2D, already_compressed_decal_map);
glCompressedTexImage2D(GL_TEXTURE_2D, 0, internalFormat, width, height,
                  border, image_size, data);
```

Code Listing 5: How to transfer a compressed texture directly into texture memory.

Finally, *never* compress dynamic textures at runtime, since the compression process is slow.

Conclusion

By following these straightforward steps, significant improvement in application's performance through use of texture compression in OpenGL can occur, though image fidelity may vary as a consequence. The extensions described above are available for all NVIDIA GPUs with Release 5.xx drivers.

Bibliography and further information.

NVIDIA Developer web site (OpenGL Code example – OpenGL S3TC):

- http://www.nvidia.com/OpenGL/S3TCExample

OpenGL Extension Registry:

- ARB_texture_compression specification
  http://oss.sgi.com/projects/ogl-sample/registry/ARB/texture_compression.txt
- EXT_texture_compression_s3tc
  http://oss.sgi.com/projects/ogl-sample/registry/EXT/texture_compression_s3tc.txt

DirectX 7.0 documentation (MSDN search for: Texture Compression)

- http://search.microsoft.com/us/dev/default.asp