DMPGL 2.0 Data Manipulation Specifications

Version 1.6

PROVISIONAL TRANSLATION

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

Table of Contents

Code

Revision History

Version	Revision Date	Description
1.6	2010/07/30	Deleted section 2.4 Rendering to a Texture about raising an error depending on the transfer method.
1.5	2010/04/23	Fixed typos.
1.4	2010/04/02	Revised section 2.3 Default Settings. Changed error conditions in section 2.1.3 No FCRAM Copies and VRAM(A/B) Access From PICA.
1.3	2010/03/12	Added error processing for partial updates of vertex buffer regions.
1.2	2010/02/15	Changed specifications related to partial updates of vertex buffer regions.
1.1	2009/12/25	Added information about partial transfers.
1.0	2009/09/10	Initial version (changed the default location for placing texture data).
0.9	2009/07/22	Rough draft.

1 Overview

This document explains how to use the DMPGL 2.0 development hardware driver to manipulate texture and vertex data, and specify the location of the regions allocated for the render and display buffers.

2 Texture and Vertex Buffers

This chapter explains how to manipulate data in the texture and vertex buffers.

2.1 Uploading Data to the Texture and Vertex Buffers

The following two types of macros specify how to upload data for the texture and vertex buffers.

- Macros that specify the region accessed by PICA (`NN_GX_MEM_FCRAM`, `NN_GX_MEM_VRAMA`, `NN_GX_MEM_VRAMB`)
- Macros that specify whether to copy data into FCRAM (`GL_COPY_FCRAM_DMP`, `GL_NO_COPY_FCRAM_DMP`)

These macros are specified using a bitwise OR as the ***target*** argument to **glTexImage2D**, **glCompressedTexImage2D**, and **glBufferData**.

Combinations of the aforementioned macros result in the following four configurations.

- No FCRAM copies and FCRAM access from PICA
- FCRAM copies and FCRAM access from PICA
- No FCRAM copies and VRAM(A/B) access from PICA
- FCRAM copies and VRAM(A/B) access from PICA

Each configuration changes behaviors such as region allocation and DMA transfers. Further details are given in the following sections.

2.1.1 No FCRAM Copies and FCRAM Access from PICA

Code 2-1 Sample Specification 1

```
glTexImage2D(GL_TEXTURE_2D | NN_GX_MEM_FCRAM | GL_NO_COPY_FCRAM_DMP,
          level, internalformat, width, height, border, format, type, data);
glBufferData(GL_ARRAY_BUFFER | NN_GX_MEM_FCRAM | GL_NO_COPY_FCRAM_DMP,
          size, data, usage);
```

PICA accesses the FCRAM address as it is specified by the ***data*** argument of **glTexImage2D**, **glCompressedTexImage2D**, and **glBufferData**. The API does not allocate memory. The application must preserve the specified FCRAM data while it is used for rendering. A `GL_INVALID_OPERATION` error occurs if `NULL` is specified as the data address or if the texture does not use the native PICA format.

2.1.2 FCRAM Copies and FCRAM Access From PICA

Specify the bitwise OR of `NN_GX_MEM_FCRAM` and `GL_COPY_FCRAM_DMP` as the ***target*** argument.

Code 2-2 Sample Specification 2

```
glTexImage2D(GL_TEXTURE_2D | NN_GX_MEM_FCRAM | GL_COPY_FCRAM_DMP,
          level, internalformat, width, height, border, format, type, data);
glBufferData(GL_ARRAY_BUFFER | NN_GX_MEM_FCRAM | GL_COPY_FCRAM_DMP,
          size, data, usage);
```

The API allocates a region in FCRAM into which the CPU copies the FCRAM data specified by the ***data*** argument of **glTexImage2D**, **glCompressedTexImage2D**, and **glBufferData**. PICA accesses the copied region. The application can destroy the specified FCRAM data immediately after the function call finishes. If NULL is specified as the data address, a region is allocated but data is not copied.

2.1.3 No FCRAM Copies and VRAM(A/B) Access From PICA

Specify the bitwise OR of NN_GX_MEM_VRAMA(B) and GL_NO_COPY_FCRAM_DMP as the ***target*** argument.

Code 2-3 Sample Specification 3

```
glTexImage2D(GL_TEXTURE_2D | NN_GX_MEM_VRAMA | GL_NO_COPY_FCRAM_DMP,
          level, internalformat, width, height, border, format, type, data);
glBufferData(GL_ARRAY_BUFFER | NN_GX_MEM_VRAMA | GL_NO_COPY_FCRAM_DMP,
          size, data, usage);
```

The API allocates a region in VRAM(A/B) into which a DMA transfer copies the FCRAM data specified by the ***data*** argument of **glTexImage2D**, **glCompressedTexImage2D**, and **glBufferData**. PICA accesses the target region of the DMA transfer. The application must preserve the specified FCRAM data until the DMA transfer is complete. If NULL is specified as the data address, a region is allocated but no DMA transfer is run. A GL_INVALID_OPERATION error occurs if the texture is not in the native PICA format and the data address is NULL.

2.1.4 FCRAM Copies and VRAM(A/B) Access From PICA

Specify the bitwise OR of NN_GX_MEM_VRAMA(B) and GL_COPY_FCRAM_DMP as the ***target*** argument.

Code 2-4 Sample Specification 4

```
glTexImage2D(GL_TEXTURE_2D | NN_GX_MEM_VRAMA | GL_COPY_FCRAM_DMP,
          level, internalformat, width, height, border, format, type, data);
glBufferData(GL_ARRAY_BUFFER | NN_GX_MEM_VRAMA | GL_COPY_FCRAM_DMP,
          size, data, usage);
```

The API allocates a region in both VRAM(A/B) and FCRAM. Next, the CPU copies the FCRAM data specified by the ***data*** argument of **glTexImage2D**, **glCompressedTexImage2D**, and **glBufferData** into the FCRAM region. Finally, DMA is used to transfer the copied data into the VRAM region. PICA accesses the VRAM region into which DMA transferred data. The application can destroy the specified FCRAM data immediately after the function call finishes. A GL_INVALID_OPERATION error occurs if NULL is specified as the data address.

2.2 Copying a Color Buffer to a Texture

When the content of a color buffer is copied into a texture, the following macros are used to specify the destination.

- Macros that specify the region accessed by PICA (NN_GX_MEM_FCRAM, NN_GX_MEM_VRAMA, NN_GX_MEM_VRAMB)

These macros are specified using a bitwise OR as the ***target*** argument to **glCopyTexImage2D**.

Code 2-5 Example

```
glCopyTexImage2D(GL_TEXTURE_2D | NN_GX_MEM_FCRAM,
          level, internalformat, x, y, width, height, border);
glCopyTexImage2D(GL_TEXTURE_2D | NN_GX_MEM_VRAMA,
          level, internalformat, x, y, width, height, border);
```

Memory is allocated in the specified region and color buffer content is transferred there via DMA. PICA accesses the transferred data.

2.3 Default Settings

The default settings are applied when macros are not specified. The default settings are applied if macros are not specified for specifying the PICA access memory, or for specifying whether to copy FCRAM.

The default settings for **glBufferData**, **glTexImage2D**, and **glCompressedTexImage2D** are NN_GX_MEM_FCRAM and GL_COPY_FCRAM_DMP. The default setting for **glCopyTexImage2D** is NN_GX_MEM_FCRAM. However, the default settings for **glTexImage2D** are NN_GX_MEM_VRAMB and GL_NO_COPY_FCRAM_DMP when NULL is specified as the ***data*** argument.

2.4 Updating Partial Regions of a Vertex Buffer

When you use glBufferSubData to partially update a vertex buffer, you cannot specify how the data is uploaded. Instead, the original settings made by glBufferData are used.

If glBufferData has disabled FCRAM copies and configured PICA to access FCRAM (NN_GX_MEM_FCRAM | GL_NO_COPY_FCRAM_DMP), the vertex buffer region is in application memory and glBufferSubData therefore only flushes the cache for a partial region without updating partial region data. Partial regions should be updated by the application. In this case, a GL_INVALID_VALUE error is generated if ***data*** in **glBufferSubData** is not specified as the sum of ***offset*** and the original buffer address set by **glBufferData**.

If glBufferData has disabled FCRAM copies and configured PICA to access VRAM (NN_GX_MEM_VRAMA(B) | GL_NO_COPY_FCRAM_DMP), you must guarantee the content of the region specified to glBufferSubData through data until the DMA finishes.

3 Render Buffer

Use the following macros to specify the location at which to allocate the render buffer region.

- NN_GX_MEM_VRAMA
- NN_GX_MEM_VRAMB

These macros are specified using a bitwise OR as the ***target*** argument to **glRenderbufferStorage**. The NN_GX_MEM_VRAMA macro is considered to be specified if no others are.

4 Display Buffer

Use the following macros to specify the location at which to allocate the display buffer region.

- `NN_GX_MEM_FCRAM`
- `NN_GX_MEM_VRAMA`
- `NN_GX_MEM_VRAMB`

These macros are specified using a bitwise OR as the ***target*** argument to **`nngxDisplaybufferStorage`**.