VRAM Managers

Dynamically Allocating and Deallocating Memory to and from VRAM

2009/04/17

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

TWL-06-0071-001-B
Released: September 11, 2009

Table of Contents

Figures

Tables

Revision History

Revision Date	Description
2009/04/17	Revised the description of the procedure for allocating memory for standard textures.
2008/05/30	Revisions resulting from NITRO-System name change (updating to TWL-System).
2007/04/27	Corrected typographical errors. Changed Revision History dates to international format.
2005/01/05	Changed an instance of “NITRO” to “Nintendo DS.”
2004/10/03	Added a chapter on the Linked-List VRAM Manager.
2004/07/16	Initial version.

1 Introduction

When using the TWL or Nintendo DS 3D graphics engine to render textured polygons, textures need to be stored in VRAM. TWL-System includes VRAM managers so the programmer does not need to precisely manage texture locations in VRAM. The VRAM managers dynamically allocate and deallocate memory to and from VRAM.

2 Overview of the VRAM Managers

2.1 Common Functions of the VRAM Managers

Methods for efficiently handling VRAM content may differ widely by application. For this reason, a unique VRAM manager may be provided depending on the application. A mechanism is provided to allow replacing the VRAM manager used by the TWL-System graphics library.

Four functions are defined, as shown in the table below, to allow memory allocation and deallocation to be carried out using the same functions regardless of the type of VRAM manager being used with TWL-System.

Table 2-1 Common Functions for VRAM Managers

Function Name	Function Processing
NNS_GfdAllocTexVram	Allocates texture memory from VRAM.
NNS_GfdFreeTexVram	Deallocates texture memory allocated from VRAM.
NNS_GfdAllocPlttVram	Allocates palette memory from the palette RAM.
NNS_GfdFreePlttVram	Deallocates palette memory allocated from palette RAM.

These functions internally call functions registered to function pointers to do the actual processing. By default, a function that does not perform any processes and returns an error is registered. Function pointers referenced internally by these functions are defined as the following global variables.

Table 2-2 Function Pointers Referenced in the Common Functions

Functions Referencing a Pointer	Function Pointer Type	Function Pointer Variable
NNS_GfdAllocTexVram	NNSGfdFuncAllocTexVram	NNS_GfdDefaultFuncAllocTexVram
NNS_GfdFreeTexVram	NNSGfdFuncFreeTexVram	NNS_GfdDefaultFuncFreeTexVram
NNS_GfdAllocPlttVram	NNSGfdFuncAllocPlttVram	NNS_GfdDefaultFuncAllocPlttVram
NNS_GfdFreePlttVram	NNSGfdFuncFreePlttVram	NNS_GfdDefaultFuncFreePlttVram

The TWL-System graphics library that uses VRAM managers allocates and deallocates memory to and from VRAM by using these four functions. To change the VRAM manager used by the library, register to the function pointer the function for allocating and deallocating memory included in the desired VRAM manager.

2.2 Texture VRAM Manager and Palette VRAM Manager

There are two main VRAM managers. One is the texture VRAM manager for allocating and deallocating texture memory, and the other is the palette VRAM manager for allocating and deallocating palette memory.

2.2.1 Allocating and Deallocating Texture Memory with the Texture VRAM Manager

Use the functions below to allocate or deallocate texture memory with the texture VRAM manager.

```
// Allocating memory from the texture VRAM
NNSGfdTexKey NNS_GfdAllocTexVram(u32 szByte, BOOL is4x4comp, u32 opt);

// Deallocating memory from the texture VRAM
int NNS_GfdFreeTexVram(NNSGfdTexKey key);
```

The `NNS_GfdAllocTexVram` function allocates the amount of texture memory specified by `szSize` from the VRAM, and returns the `NNSGfdTexKey` type value indicating the allocated texture memory. If the allocation of the texture memory failed, the constant `NNS_GFD_ALLOC_ERROR_TEXKEY` is returned.

The `NNS_GfdFreeTexVram` function deallocates the texture memory specified by the `NNSGfdTexKey` type value. If the deallocation succeeds, 0 is returned.

2.2.1.1 Allocating Memory for 4x4 Texel Compressed Texture

The texture VRAM manager has a feature that allows allocation of a memory region for 4x4 texel compressed texture. If the second argument `is4x4comp` of the `NNS_GfdAllocTexVram` function, which is a function for allocating texture memory, is `TRUE`, a region for texture palette index data is allocated at the same time as allocation of a region for texture image data. If either of these regions cannot be allocated, the `NNS_GfdAllocTexVram` function call fails.

2.2.2 NNSGfdTexKey

`NNSGfdTexKey` is a 32-bit integer value that functions as a key for identifying texture memory allocated by the texture VRAM manager. `NNSGfdTexKey` is generated from the texture memory address and size, as well as a flag indicating the 4x4 texel compressed texture memory.

2.2.2.1 Operation for NNSGfdTexKey

To transfer texture data to allocated texture memory, you must get the actual VRAM address. The texture VRAM manager provides a function to find the VRAM address and size from `NNSGfdTexKey`.

```
// Getting the address from NNSGfdTexKey
u32 NNS_GfdGetTexKeyAddr( NNSGfdTexKey memKey );

// Getting the size from NNSGfdTexKey
u32 NNS_GfdGetTexKeySize( NNSGfdTexKey memKey );

// Finding out whether NNSGfdTexKey is for compressed texture
BOOL NNS_GfdGetTexKey4x4Flag( NNSGfdTexKey memKey );
```

2.2.3 Allocating and Deallocating Palette Memory with the Palette VRAM Manager

Use the following functions to allocate and deallocate palette memory with the palette VRAM manager.

```
// Allocating memory from the palette RAM
NNSGfdPlttKey NNS_GfdAllocPlttVram(u32 szByte, BOOL is4pltt, u32 opt);
```

```
// Deallocating memory from the palette RAM
int NNS_GfdFreePlttVram(NNSGfdPlttKey key);
```

NNS_GfdAllocPlttVram allocates the amount of palette memory specified by szSize from the palette RAM, and returns the NNSGfdPlttKey type value indicating the allocated palette memory. If the allocation of the palette memory failed, the constant NNS_GFD_ALLOC_ERROR_PLTTKEY is returned.

NNS_GfdFreePlttVRam then deallocates the palette memory specified by NNSGfdplttKey. If the deallocation succeeds, 0 is returned.

2.2.3.1 Allocating Memory for Four-Color Palette

For allocating four-color color palette memory using the palette VRAM manager, specify TRUE for the second argument is4pltt of NNS_GfdAllocPlttVram function, which is a function for allocating palette memory. When this is set, a determination is made whether memory was allocated at a position that allows for a four-color palette. (Four-color palettes cannot be placed at 0x10000 or higher.)

2.2.4 NNSGfdPlttKey

NNSGfdPlttKey is a 32-bit integer value that functions as a key for identifying palette memory allocated from the palette VRAM manager. NNSGfdPlttKey is generated from the address and size of the allocated palette memory.

2.2.4.1 Operation for NNSGfdPlttKey

To transfer palette data to allocated palette memory, you must get the actual palette RAM address. The palette VRAM manager provides a function to find the palette RAM address and size from NNSGfdPlttKey.

```
// Getting the address from NNSGfdPlttKey.
u32 NNS_GfdGetPlttKeyAddr( NNSGfdPlttKey memKey )

// Getting the size from NNSGfdPlttKey
u32 NNS_GfdGetPlttKeySize( NNSGfdPlttKey memKey )
```

2.3 VRAM Managers Provided in TWL-System

As explained in section 2.1 Common Functions of the VRAM Managers, the functions for allocating and deallocating memory provided by the TWL-System VRAM managers in their initial state do not do anything. In the future, the VRAM managers planned for TWL-System will have additional memory management functions for memory allocation and deallocation. Currently, however, only the frame VRAM manager (the frame texture VRAM manager and the frame palette VRAM manager) and the linked list VRAM manager (linked list texture VRAM manager and linked list palette VRAM manager) are provided.

3 Frame VRAM Managers

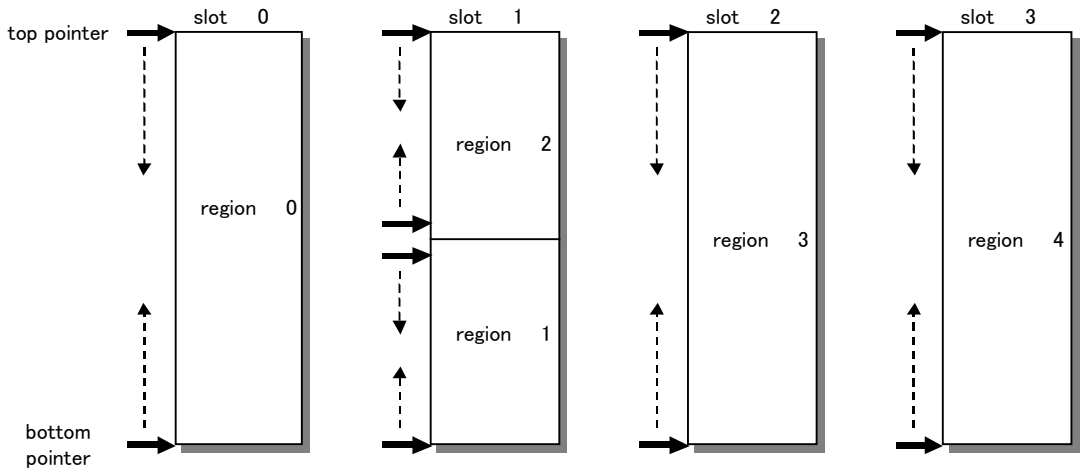
Like the Foundation library frame heap managers, the frame VRAM managers can only allocate memory blocks with a specified size and deallocate all allocated memory blocks simultaneously. The trade-off is that because the memory blocks do not have any management information, high memory efficiency is achieved. The frame VRAM managers have the following characteristics.

- No memory management region is required.
- Memory blocks are allocated from the lowest address and highest address of each VRAM slot without any gaps.
- Allocated memory blocks cannot be deallocated individually.
- All allocated memory blocks can be deallocated simultaneously.
- Memory block allocation states can be saved and restored.

3.1 Frame Texture VRAM Manager

Error! Reference source not found. shows the frame texture VRAM manager. The frame texture VRAM manager splits VRAM into five regions for management. Each region has two pointers, at the top and bottom of that region. These two pointers indicate the boundaries between the used and unused regions. The area between the two pointers is not used. Texture memory is allocated from the highest address and lowest address of the regions, and every time memory is allocated the two pointers move toward the center of the region.

Figure 3-1 Frame Texture VRAM Manager



3.1.1 Initializing the Frame Texture VRAM Manager

Before using the frame texture VRAM manager, it must be initialized. Use the following function to initialize.

```
void NNS_GfdInitFrmTexVramManager(u16 numSlot, BOOL useAsDefault);
```

The first argument `numSlot` specifies the number of slots for the frame texture VRAM manager to manage. The frame texture VRAM manager is initialized to manage the number of VRAM slots specified by `numSlot` beginning at VRAM slot 0. The maximum value that may be specified for `numSlot` is 4.

If TRUE is specified for the second argument, `useAsDefault`, a function pointer is initialized when either `NNS_GfdAllcTexVram` or `NNS_GfdFreeTexVram` (common functions for the texture VRAM manager) is called, so the memory allocation and deallocation functions of the frame texture VRAM manager can be used. This argument should be set to `TRUE` except in special situations such as when replacing the texture VRAM manager processing later on.

3.1.2 Allocating Memory for Texture

To allocate memory for texture, the texture VRAM manager common function `NNS_GfdAllocTexVram` is normally used.

```
NNSGfdTexKey NNS_GfdAllocTexVram(u32 szByte, BOOL is4x4comp, u32 opt);
```

The third argument `opt` of the `NNS_GfdAllocTexVram` function is not used with the frame texture VRAM manager.

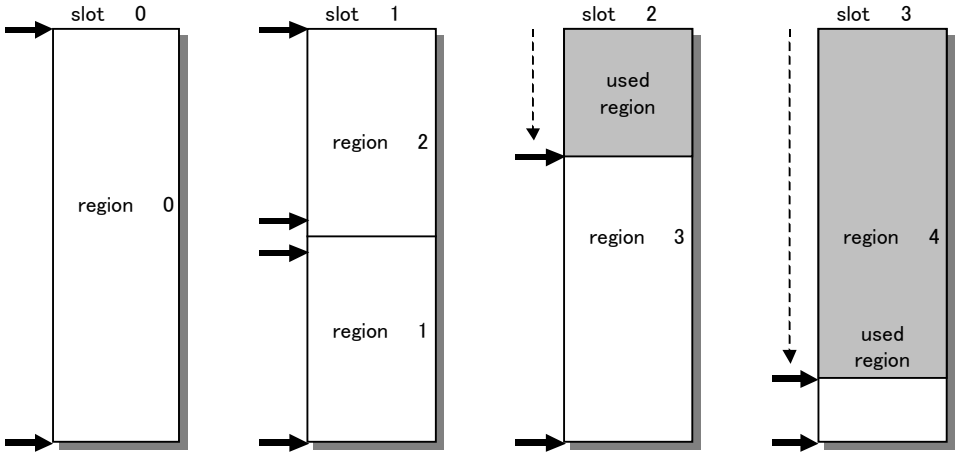
The specific operations of the managers are described below. Refer to the example in the description where 4 is passed as the parameter `numSlot` (the number of slots the manager manages) during initialization. (The manager will slightly change the search order of the empty region depending on the value of `numSlot`.)

3.1.2.1 Allocating Memory for Standard Texture

To allocate memory for standard texture (other than 4x4 texel compressed texture), specify `FALSE` for the second argument, `is4x4comp`.

To allocate memory for standard texture, allocate the memory using the top pointer for each region. The frame texture VRAM manager first tries to allocate memory from region 4. If there is not enough free space for allocation, the manager then attempts to allocate from regions 3, 2, 0 and 1, in that order.

Figure 3-2 Allocating Standard Texture Memory

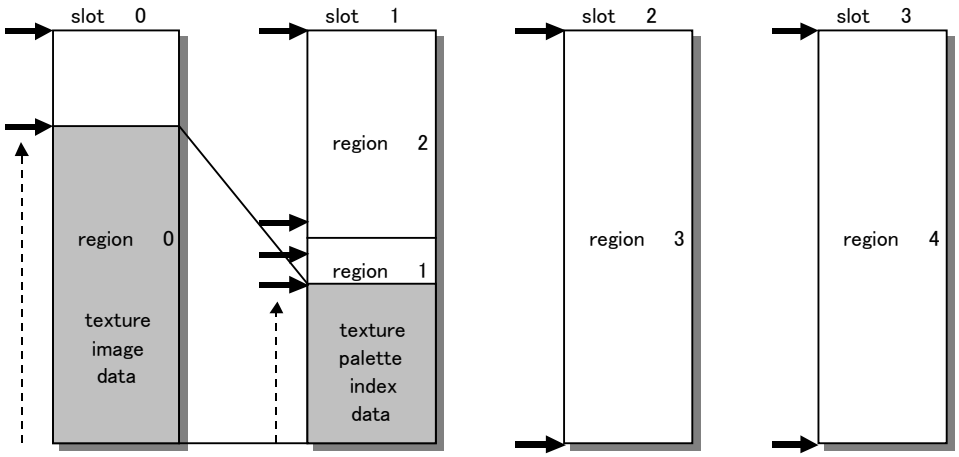


3.1.2.2 Operation When Requesting Memory for 4x4 Texel Compressed Texture

To allocate 4x4 texel compressed texture memory, specify `TRUE` for the second argument, `is4x4comp`.

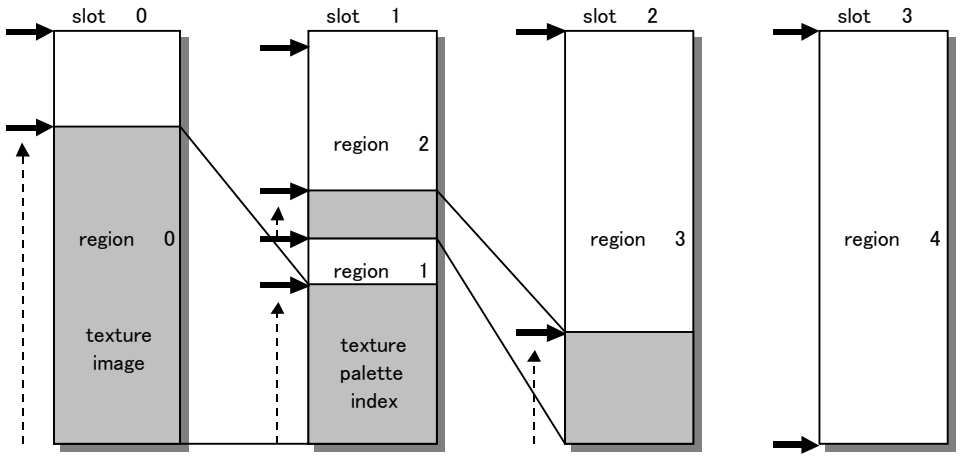
To allocate memory for 4 x 4 texel compressed texture, first an attempt is made to allocate that memory using the bottom pointer of region 0. To store 4x4 texel compressed texture in the VRAM, the texture palette index data also needs to be stored at the same time at the position corresponding to the address where the texture image data is placed. Therefore, a region for texture palette index data is allocated at the same time using the bottom pointer for region 1.

Figure 3-3 Allocating 4x4 Texel Compressed Texture Memory



If 4x4 texel compressed texture memory of the requested size cannot be allocated from region 0, an attempt is made to allocate memory using the bottom pointer for region 3. At the same time, a region for the texture palette index data is allocated using the bottom pointer for region 2.

Figure 3-4 Allocating 4x4 Texel Compressed Texture Memory from Region 3



3.1.3 Deallocating Texture Memory

Due to the characteristics of the memory management algorithm, the frame texture VRAM manager does not allow allocated texture memory to be deallocated individually.

The frame texture VRAM manager provides a `NNS_GfdFreeFrmTexVram` function to deallocate memory, though this function returns without processing anything. This function is provided to register the function pointer to `NNS_GfdFreeTexVram`, which is a common function for the texture VRAM manager.

3.1.4 Saving and Restoring Frame Texture VRAM Manager States

The frame texture VRAM manager has a feature to save the use state of texture memory and to restore a saved texture memory use state.

3.1.4.1 Saving the Use State of Texture Memory

The following function is used to save the use state of texture memory.

```
NNS_GfdGetFrmTexVramState(NNSGfdFrmTexVramState* pState);
```

When the `NNS_GfdGetFrmTexVramState` function is called, the current use state of the texture memory is written to the `NNSGfdFrmTexVramState` structure specified in the argument.

3.1.4.2 Restoring a Use State of Texture Memory

The following function is used to return the frame texture VRAM manager to a texture memory allocation state saved using the `NNS_GfdGetFrmTexVramState` function.

```
void NNS_GfdSetFrmTexVramState(const NNSGfdFrmTexVramState* pState);
```

This operation deallocates texture memory allocated after a texture memory allocation state specified by `pState` has been saved.

3.1.4.3 Returning to the Initial Use State of Texture Memory

The following function is used to return the frame texture VRAM manager to its initial state.

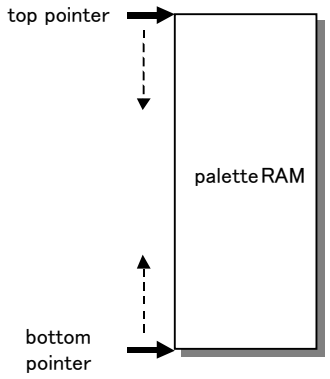
```
void NNS_GfdResetFrmTexVramState(void);
```

This operation deallocates all texture memory allocated from the frame texture VRAM manager.

3.2 Frame Palette VRAM Manager

Figure 3-5 Frame Palette VRAM Manager shows the frame palette VRAM manager. The frame palette VRAM manager can allocate palette memory from either the beginning or the end of palette RAM. Top and bottom pointers are provided for that purpose. These two pointers indicate the boundaries between the used and unused regions. The area between two pointers is unused.

Figure 3-5 Frame Palette VRAM Manager



3.2.1 Initializing the Frame Palette VRAM Manager

Before using the frame palette VRAM manager, you must initialize it. Use the following function for initialization.

```
void NNS_GfdInitFrmPlttVramManager(u32 szByte, BOOL useAsDefault);
```

The first argument `szSize` specifies the size of the palette RAM managed by the frame palette VRAM manager. The frame palette VRAM manager is initialized so as to manage the amount of palette RAM specified by `szSize` in bytes starting at the beginning of the palette RAM.

If `TRUE` is specified for the second argument `useAsDefault`, a function pointer is initialized when either `NNS_GfdAllocPlttVram` or `NNS_GfdFreePlttVram` (common functions for the palette VRAM managers) is called, so the memory allocation and deallocation functions of the frame palette VRAM manager are used. This should be set to `TRUE` except in special situations such as when replacing the palette VRAM manager processing later on.

3.2.2 Allocating Memory for Palette

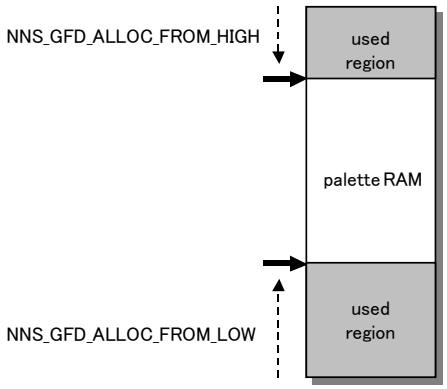
Normally, the palette VRAM manager common function `NNS_GfdAllocPlttVram` is used to allocate palette memory.

```
NNSGfdPlttKey NNS_GfdAllocPlttVram(u32 szByte, BOOL is4Pltt, u32 opt);
```

3.2.2.1 Direction of Allocation for Palette Memory

With the frame palette VRAM manager, the third argument `opt` can be used to specify the direction of the palette memory allocation. If `NNS_GFD_ALLOC_FROM_LOW` is specified in `opt`, allocation will be made from the lowest address of the palette RAM. If `NNS_GFD_ALLOC_FROM_HIGH` is specified in `opt`, allocation will be made from the highest address of the palette RAM.

Figure 3-6 Allocating Palette Memory



3.2.3 Deallocating Palette Memory

Due to the characteristics of the memory management algorithm, the frame palette VRAM manager does not allow allocated palette memory to be deallocated individually.

The frame palette VRAM manager provides a `NNS_GfdFreeFrmPlttVram` function for deallocating memory, though this function returns without processing anything. This is provided to register the function pointer of `NNS_GfdFreeFrmPlttVram`, which is a common frame palette VRAM manager function.

3.2.4 Saving and Restoring Frame Palette VRAM Manager States

The frame texture VRAM manager has the feature to save the use state of palette memory and to restore saved palette memory use states.

3.2.4.1 Saving the Use State of Palette Memory

The following function is used to save a use state of palette memory.

```
NNS_GfdGetFrmPlttVramState(NNSGfdFrmPlttVramState* pState);
```

When the `NNS_GfdGetFrmPlttVramState` function is called, the current use state of palette memory is written to the `NNSGfdFrmPlttVramState` structure specified in the argument.

3.2.4.2 Restoring a Use State of Palette Memory

To return the frame palette VRAM manager to the palette memory allocation state saved by using the `NNS_GfdGetFrmPlttVramState` function, the following function is used.

```
void NNS_GfdSetFrmPlttVramState(const NNSGfdFrmPlttVramState* pState);
```

This operation deallocates palette memory allocated after a palette memory allocation state specified by `pState` has been saved.

3.2.4.3 Returning to the Initial Use State of Palette Memory

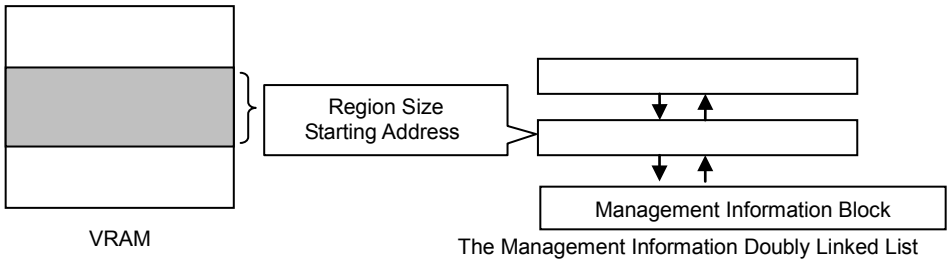
The following function is used to return the frame palette VRAM manager to its initial state.

```
void NNS_GfdResetFrmPlttVramState(void);
```

This operation deallocates all palette memory allocated from the frame palette VRAM manager.

4 Linked-List VRAM Manager

Figure 4-1 Linked-List VRAM Manager



The Linked-List VRAM Manager manages the VRAM region using the VRAM management information in the main memory. The management information consists of the starting address and the byte size of the VRAM region that will be managed.

The management information contains the reference information that references the previous and the next management information and is set up as a doubly linked list structure. The VRAM manager uses this doubly linked list of management information to manage the empty space scattered throughout VRAM.

The characteristics of the linked-list VRAM manager are as follows.

- It can perform the sectional deallocation of VRAM regions.
- Its management information must be allocated in main memory.
- It has a processing load for searching the list of empty regions when allocating and deallocating.

4.1 Linked-List Texture VRAM Manager

The texture VRAM manager individually manages the management region for the normal textures and the management region for 4x4 compressed textures. If there is a request to allocate a region, this manager searches for an empty region that fulfills that request from the empty region information list.

4.1.1 Initializing the Manager

```
void NNS_GfdInitLnkTexVramManager
(
u32     szByte,
u32     szByteFor4x4,
void*   pManagementWork,
u32     szByteManagementWork,
BOOL    useAsDefault
)
```

When initializing, individually designate the size of the texture management region and the size of the region within the texture management region to be used for 4x4 compressed texture. The texture management region size must be larger than the region size used for the 4x4 compressed texture. The manager initializes the free blocks with the designated size. The memory region used as the management information is passed as an argument to `pManagementWork`. To compute the size of the management information region, use:

```
u32 NNS_GfdGetLnkTexVramManagerWorkSize( u32 numMemBlk )
```

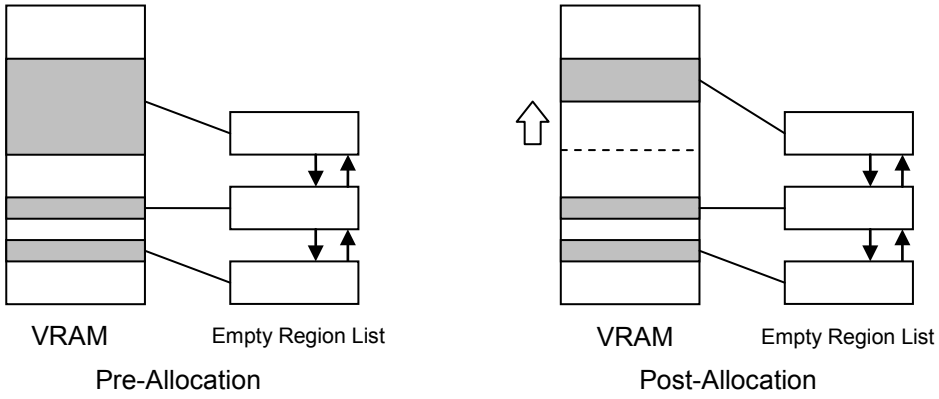
`numMemBlk` specified here is the maximum number of blocks into which the empty memory region can be broken.

The texture VRAM manager does not manage the region for the palette index of the 4x4 compressed texture. Therefore, the region for the palette index is always not yet in use, and the user can assume that it is always usable. Also, regions other than the regions for the palette index are initialized as empty regions for normal textures.

4.1.2 Allocating Memory for Textures

If there is a new allocation request, the manager will search the list of empty regions to find a region according to the type of region to be allocated (Normal or 4x4 Compressed). If an empty region that meets the requirements is found, the empty region information is updated with the information that subtracted the used region, and the allocated region is returned as a texture key.

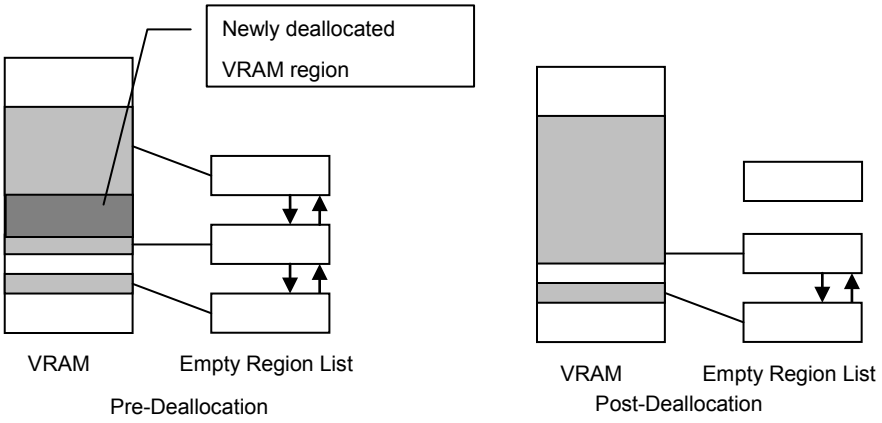
Figure 4-2 Allocating Memory for Textures



4.1.3 Deallocating Memory for Textures

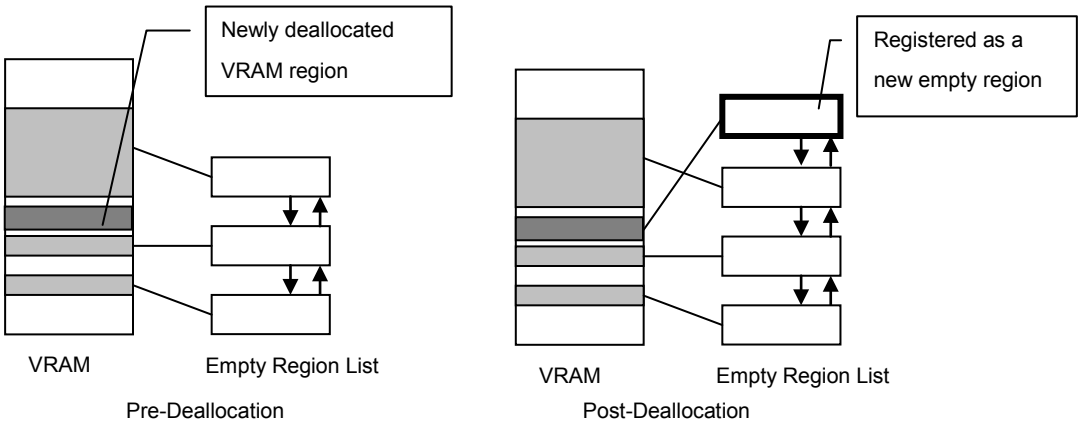
This section deals with the calculation of the memory region from the texture key to be deallocated. A linear search is performed on the empty region list to determine if there are contiguous empty regions at the highest address or the lowest address of the calculated region. If a contiguous region block is found, those neighboring blocks are joined to form one region block. This process makes it harder for memory fragmentation to occur.

Figure 4-3 Deallocating Memory for Textures



If the joining of the region blocks fails, a new empty region block is generated, and that block is added to and registered in the empty region list. In such an event, if there is insufficient management information for the manager and a new empty region block cannot be obtained, the deallocation of the memory region will fail.

Figure 4-4 An Example of When the Empty List Joining Process Fails



4.2 Linked-List Palette VRAM Manager

In the same way as the texture VRAM manager, the palette VRAM manager manages the empty space scattered throughout VRAM via a doubly linked list of management information.

4.2.1 Initialization

```
void NNS_GfdInitLnkPlttVramManager
(
u32     szByte,
```

```
void*   pManagementWork,
u32     szByteManagementWork,
BOOL    useAsDefault
);
```

Initialize the palette VRAM manager in the same way as the texture VRAM manager, by passing the memory region used in the management information and its size as an argument.

4.2.2 Allocating and Deallocating Palette Memory

Memory allocation and deallocation are performed in essentially the same manner as with the texture VRAM manager.

Since an 8-byte alignment is needed when allocating a 4-color palette, allocate an aligned region. The empty blocks generated when performing the alignment are registered as free blocks. In order to avoid unnecessary fragmentation of the management region, we recommend that you allocate the regions for 4-color palettes and palettes in other formats in good-sized groups.