

Chapter 1
Introduction to the Nintendo 64

This chapter describes an overview of the major components, processes, and terminology related to the Nintendo 64.

1-1 N64 Game Machine Architecture

The following figure illustrates the whole N64 game configuration.

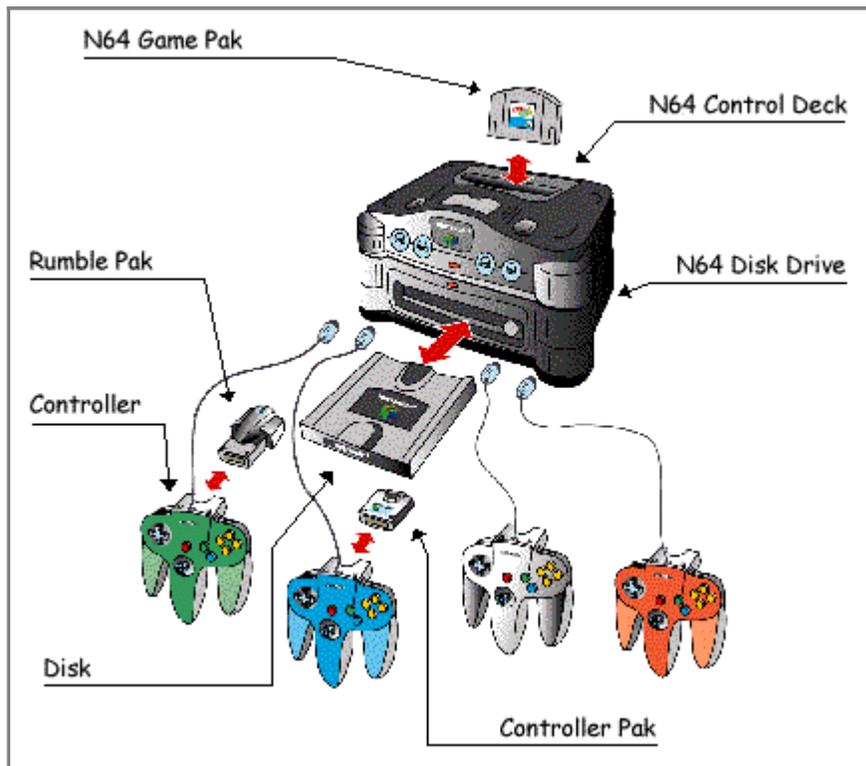


Figure1-1-1 N64 Game Configuration

The game software is supplied by an N64 Game Pak and/or an N64 Disk Drive Disk through the parallel interface.

The N64 Disk Drive is connected to the N64 Control Deck via the expansion slot located on the bottom of the Control Deck.

Up to 4 Controllers can be connected to the N64 Control Deck at any one time through the serial interface.

You can add a Controller Pak or a Rumble Pak to each Controller. The Controller Pak provides more data storage capacity as well as portability. The Rumble Pak causes the Controller to vibrate, adding realism to the game.

1-2 N64 Control Deck Architecture

The N64 System is composed of the total of four chips, as the core, which are the RCP, CPU and two RDRAM chips. Program execution is provided by the CPU and RCP. Both of these can execute the program at the same time. The processing units executed by them is called a [thread](#) and a task in the N64 system.

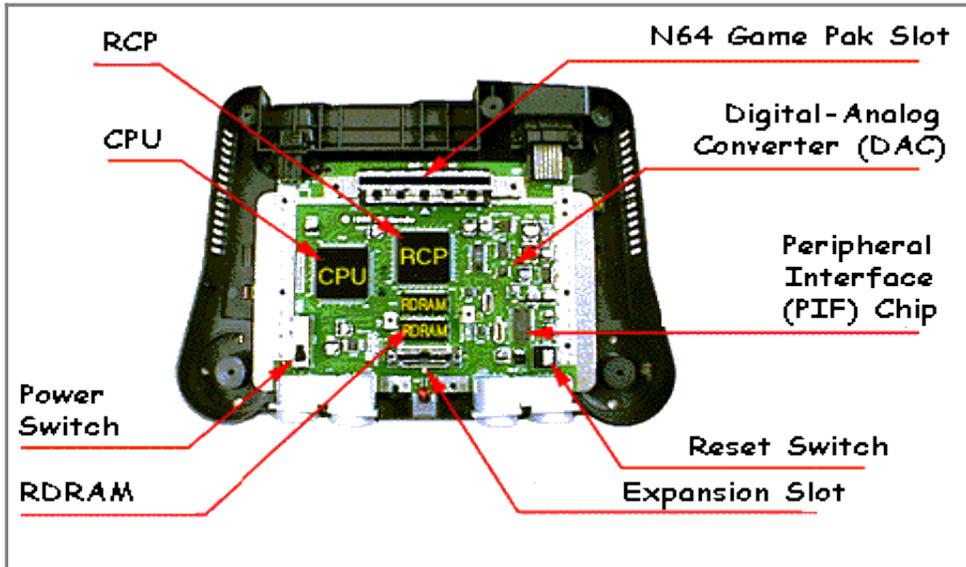


Figure1-2-1 N64 Control Deck Architecture

1-3 N64 Hardware and System Features

1-3-1 Hardware Features

The CPU is fast (about 100 MIPS) and includes the following on-chip cache memory:

The Instruction Cache is 16K bytes

The Data Cache is 8K bytes

Figure1-3-1 CPU Cache Memory

N64 doesn't require a special dedicated chip to process graphics and audio as was the case with Super NES. N64 uses software to execute almost all processes.

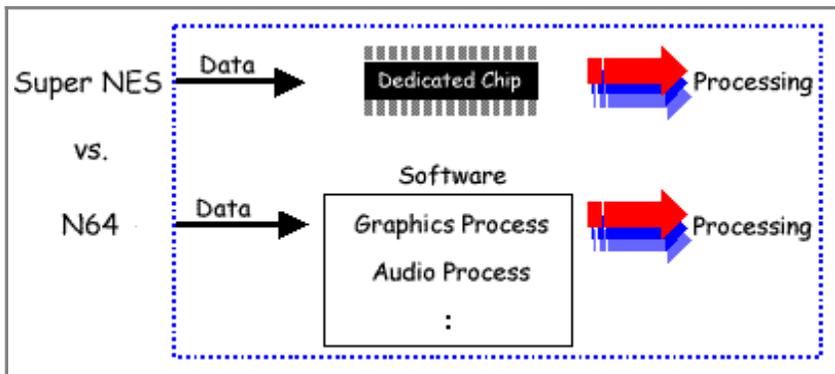


Figure1-3-2 Process are executed by software

The main memory (RDRAM) can be used by the entire system. Programmers are free to divide up the main memory into buffers (frame buffer, audio buffer, Z-buffer, [heap](#), texture buffer, and so on) as appropriate for each game program. It is a unified memory system.

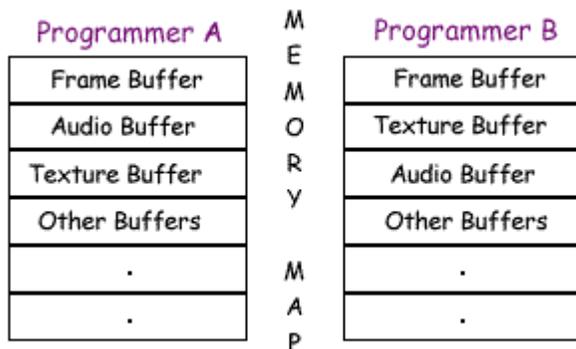


Figure1-3-3 Unified Memory System

Almost all processes are executed by the CPU and RCP working together.

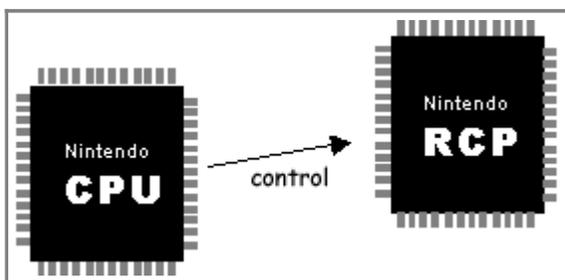


Figure1-3-4 CPU and RCP execute a process in common.

The RCP consists of two internal processors the RSP (Reality Signal Processor) and the RDP (Reality Display Processor), I/O Interface, and control logic.

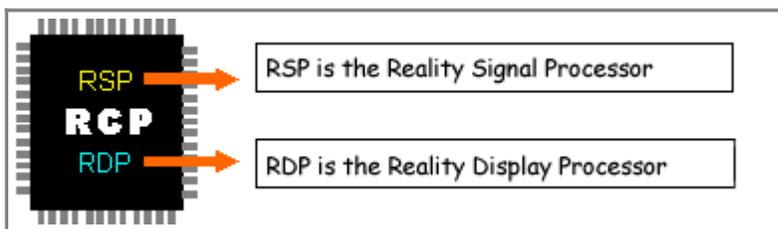


Figure1-3-5 The configuration of RCP

The RSP and RDP work together to execute GBI ([Graphics Binary Interface](#)) commands and render graphics into the frame buffer. The GBI commands are strung together into a specialized command list call the [display list](#).

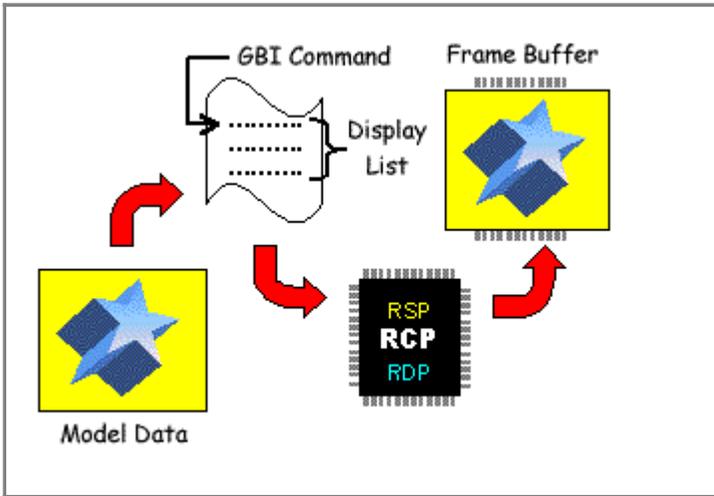


Figure1-3-6 RSP and RDP on the graphics process

The RSP also interprets ABI (Audio Binary Interface) commands and creates linear PCM data using the sampling rate specified by the audio buffer.

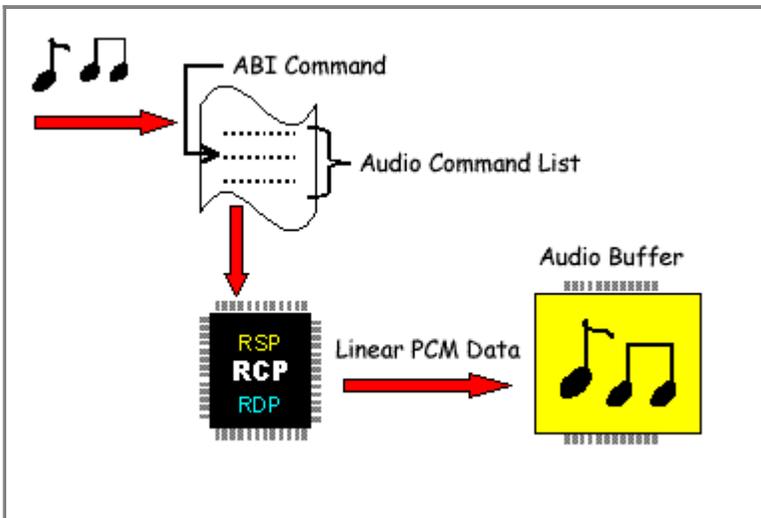


Figure1-3-7 RSP on the audio process

The RSP interprets GBI and ABI commands using software called [microcode](#).

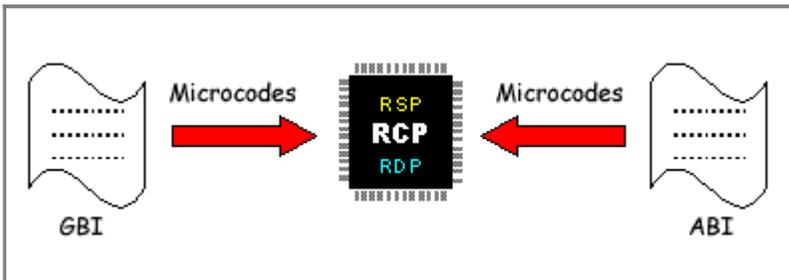


Figure1-3-8 RSP interprets commands by microcode

The image drawn in the frame buffer is transferred to the video DAC (Digital-to-Analog Converter) through DMA (direct memory access). The image becomes the TV image there in the video DAC.

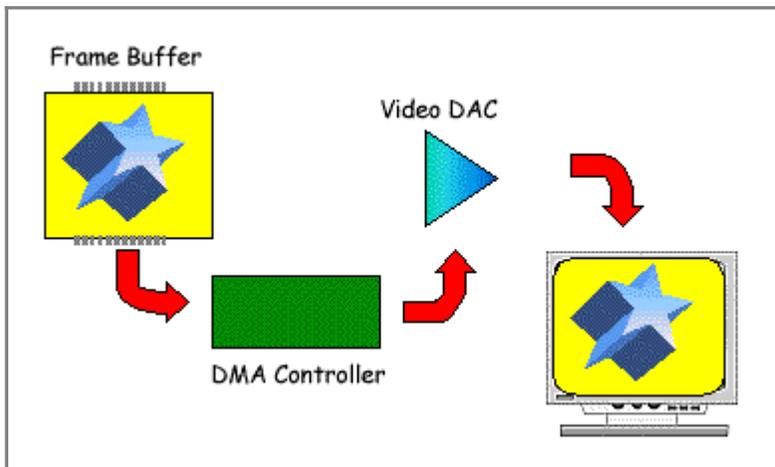


Figure1-3-9 Image Transferred to DAC through DMA

The PCM data placed in the audio buffer is transferred to audio DAC through DMA. The audio becomes sound there in the audio DAC.

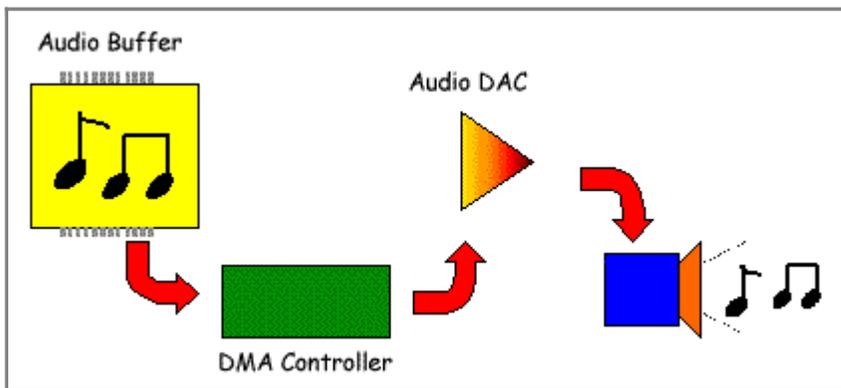


Figure1-3-10 The PCM data transferred to the audio DAC by DMA

For faster processing, game programs are not executed on the N64 Game Pak directly. Instead they are loaded into RDRAM first.

[Previous](#)

[N64](#)

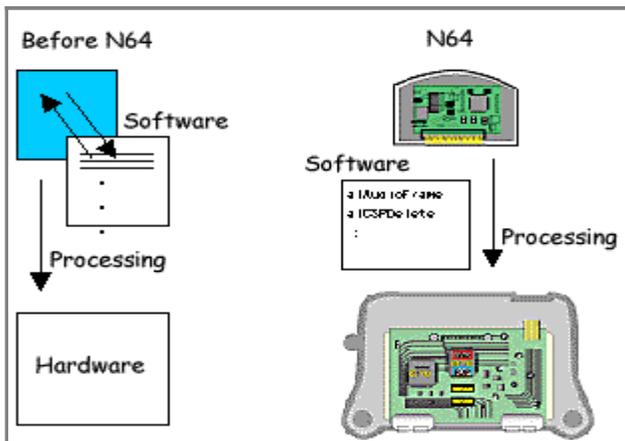


Figure1-3-11 The programs of the ROM cartridge is surely loaded on RDRAM

Data coming from the Controller is read by way of the PIF chip as serial data. Similarly, data in the Controller Pak is also read and written by way of the PIF chip through the Controller as serial data.

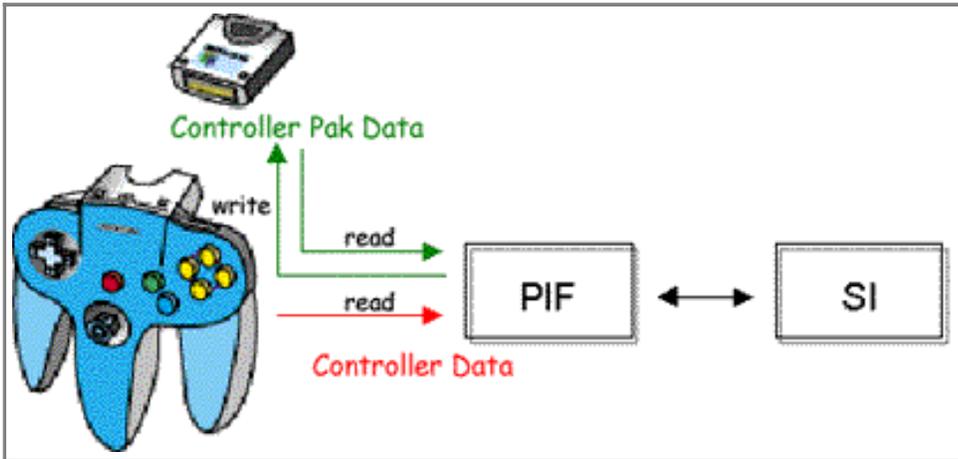


Figure1-3-12 The Data Communication of the Controller

The RDP creates the screen using the Z-buffer (depth [buffer](#)) as shown in the following illustration.

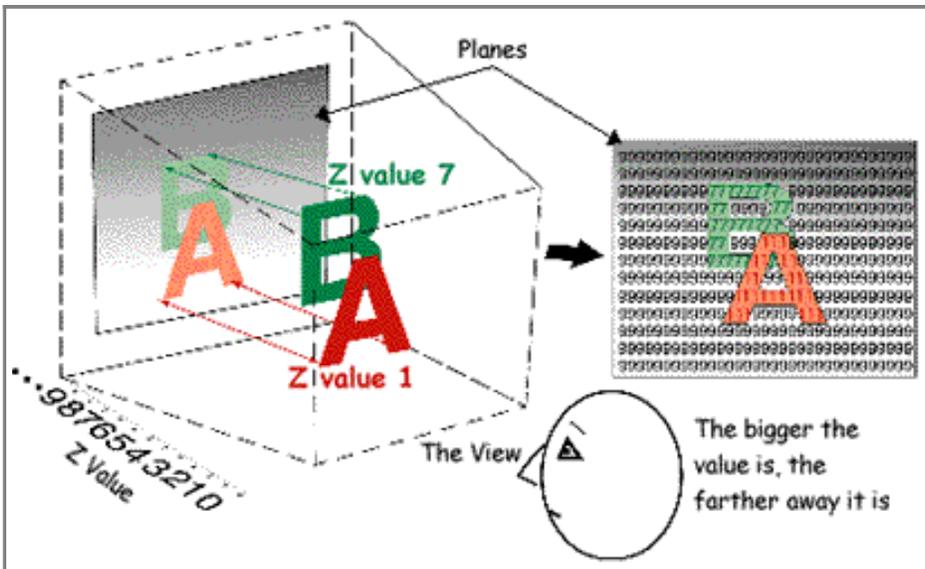


Figure1-3-13 Creating the Screen with the Z-buffer Using RDP

(Note that this is a conceptual illustration only. In practice, the Z-buffer values range from -1.0 to 1.0.)

1-3-2System (Software) Features

Game applications are executed in the multi-[thread](#) N64 Operating System.

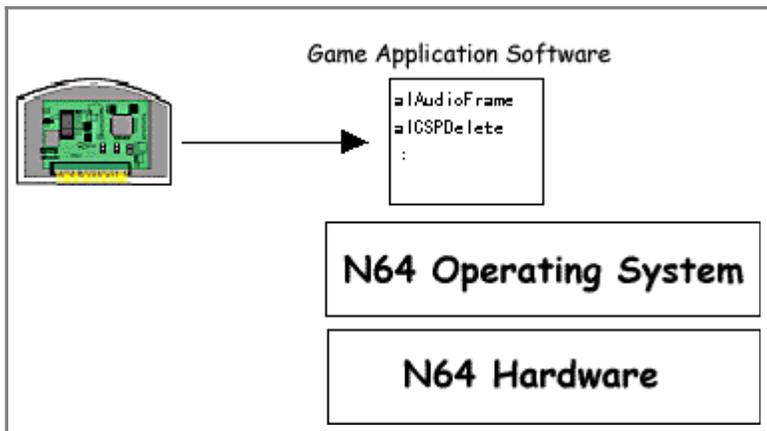


Figure1-3-14 The application software is executed on OS

Multi-thread programming provides the following advantages:

- You can run independent processes on a single piece of hardware at the same time
- Threads provide a natural way to structure your program
- Thread synchronization is assured because the threads communicate with each other by sending and receiving [messages](#)

As a result, there are fewer bugs and debugging time is reduced.

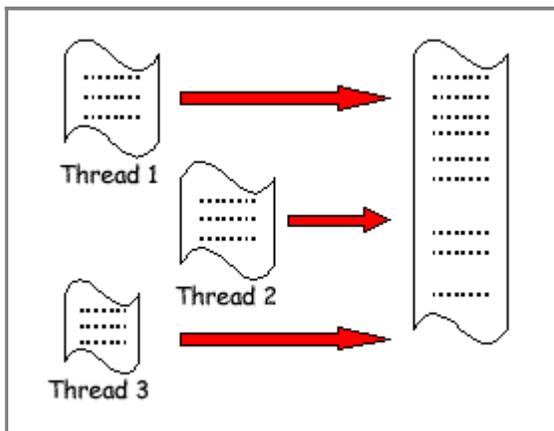


Figure1-3-15 Multi-thread programming

Messages provide information throughput between threads.

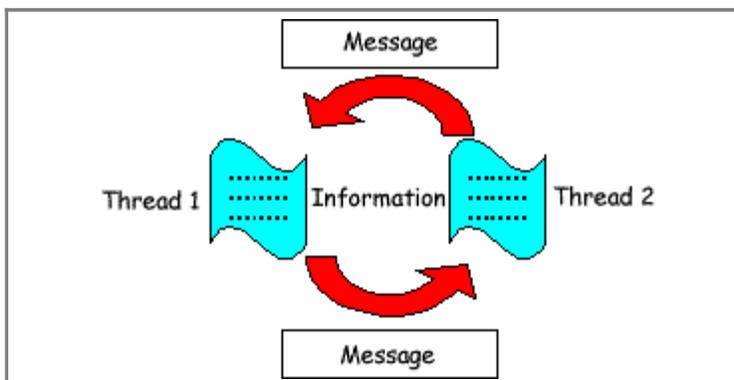


Figure1-3-16 "Message" Provide Information Throughput Between Threads

Threads are executed according to their priority, not their actual position in the program code.

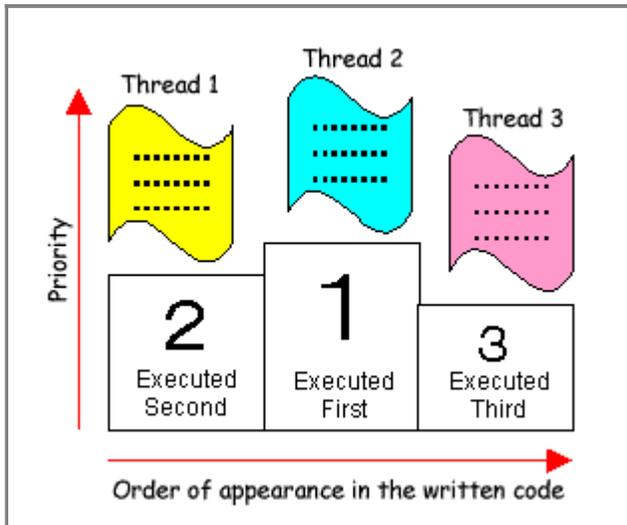


Figure1-3-17 Software execution is provided by priority of threads

Threads that receive interrupt information messages execute interrupt processing.

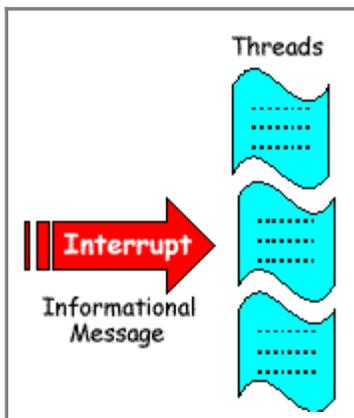


Figure1-3-18 Threads provide the interrupt processing

A high-priority thread called the Scheduler manages all the other threads. The Scheduler thread manages messages for the threads by using tasks or the VI retrace (the vertical synchronizing interrupt).

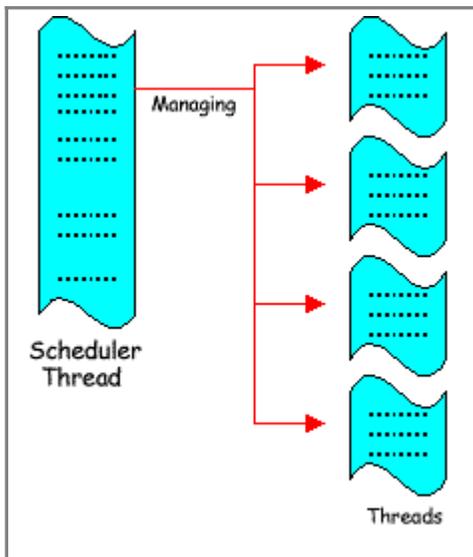


Figure1-3-19 The scheduler provide management of threads

The CPU executes threads comprised of several processing units (function calls, macros, and so on). The RSP in the RCP, on the other hand, executes processing units called tasks.

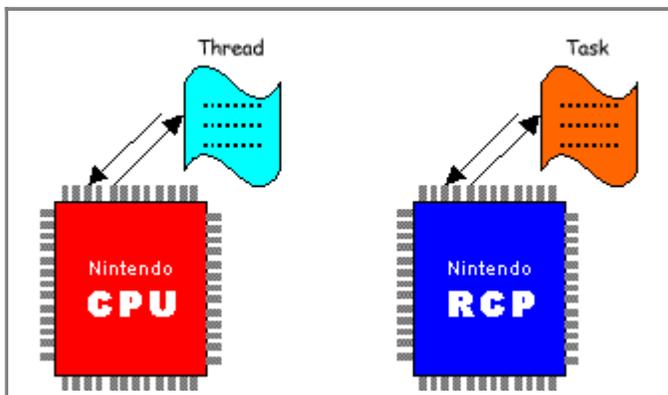


Figure1-3-20 Thread and Task

1-4 Memory

1-4-1 RDRAM Main Memory

Main memory is provided by two very high-speed memory chips called RDRAM (Rambus DRAM). Each of the two RDRAM chips in the N64 provide a memory map of 2 megabytes by 9 bits for a total memory size of 4 megabytes by 9 bits.

N64 has been designed so that all three processors (CPU, RSP, and RDP) can share this memory.

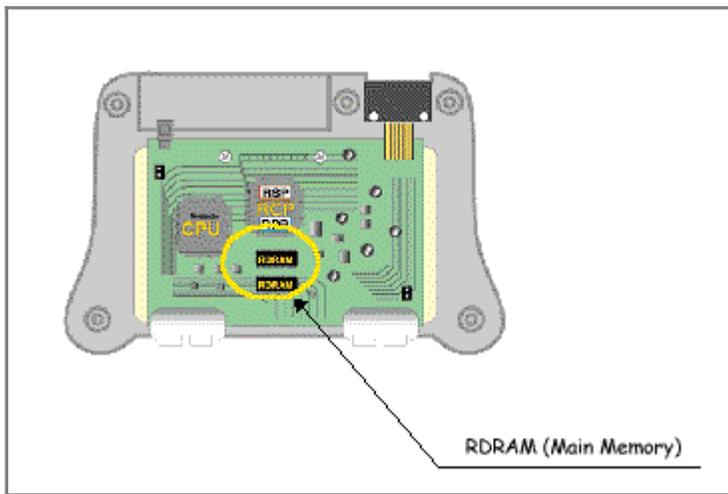


Figure 1-4-1 RDRAM

Also, N64 provides a tremendous advantage in that any place in RDRAM can be used not only to store the program or data but also to be the frame buffer, the [Z buffer](#), or the audio buffer. In addition, connecting the Memory Pak to the N64 Control Deck expands RDRAM by another 4 megabytes by 9 bits.

1-4-2 Memory Management

You are free to manage memory as appropriate for each game you develop. The N64 Operating System does not force you to use any specific method. The memory region library that dynamically allocates fixed-length memory blocks now fully supports all features of the *malloc* and *free* standard C functions.

1-4-3 CPU Addressing

Be careful when working with CPU addresses. They are [virtual addresses](#), not physical addresses. The CPU is operated in 32-bit [kernel](#) mode which means that each address space is 32 bits. (Note that this in no way excludes 64-bit integer arithmetic.) In 32-bit kernel mode, memory is divided into the following five [segments](#):

- | <Start> | <End> | <ID> | <Use> |
|------------|--------------|-------|-----------------------|
| 0x00000000 | - 0x7fffffff | KUSEG | TLB map |
| 0x80000000 | - 0x9fffffff | KSEG0 | Direct map, cache |
| 0xa0000000 | - 0xbfffffff | KSEG1 | Direct map, non-cache |
| 0xc0000000 | - 0xdfffffff | KSSEG | TLB mapping |
| 0xe0000000 | - 0xffffffff | KSEG3 | TLB mapping |

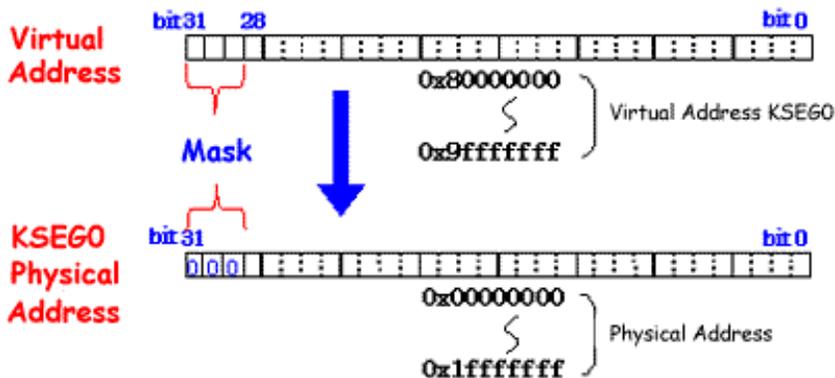


Figure 1-4-2 CPU Addressing

Usually, you'll want to use KSEG0 where mapping between physical and virtual addresses is simple. In this segment, the address that masks the upper 3 bits of the virtual address becomes the physical address. Of course, you can use the other segments that use the TLB (translation lookaside buffer).

1-4-4 RDP Addressing

The RSP uses a [segment address](#) system to identify the RDRAM address where the display list, matrix data, vertex data, or texture data is located. This information is used by the RSP-driven graphics [microcode](#), which can manage up to 16 segments.

In the 32-bit address given as the segment address, the four bits from bit 31 to 28 are ignored. The four bits from bit 27 to 24 form a segment ID to identify the base address of one of the 16 segments. The 24 bits from bit 23 to 0 hold the [segment offset](#). To find the physical address, simply add the segment offset to the segment base address specified by the segment ID.

The segment ID is reserved only to sue the physical address, and it is used after making the segment base address 0. Besides, it has a function for the clear indication to [GBI](#).

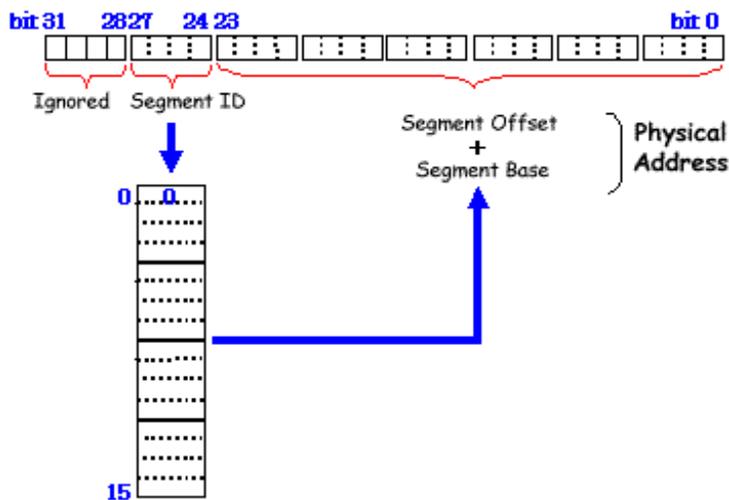


Figure 1-4-3 RDP Addressing

1-4-5 RDP Addressing

The address specification system directly used in RDP is the physical address. Because the display list is passed to RDP via RSP once, the segment address is supposed to be converted to the physical address at this time. Therefore, the display list can use the segment address in all case regardless that it is processed by RSP or RDP.

1-5 Graphics

In the N64 graphics process, the CPU transfers the data accepted from the N64 Game Pak ROM to RDRAM. CThe CPU creates the display list (GBI command list) in RDRAM.

Next, the RSP provides the [geometry](#) conversion by calculating the coordinate transformation by using the appropriate [microcode](#). After that, the RDP processes, [rasterizes](#) (Creates the pixels for the drawing), and transfers the data to the frame buffer in RDRAM. The CPU then transfers the data to the VI (Video Interface).

VI transfers the accepted digital data to the video DAC and it is output as the TV screen from there.

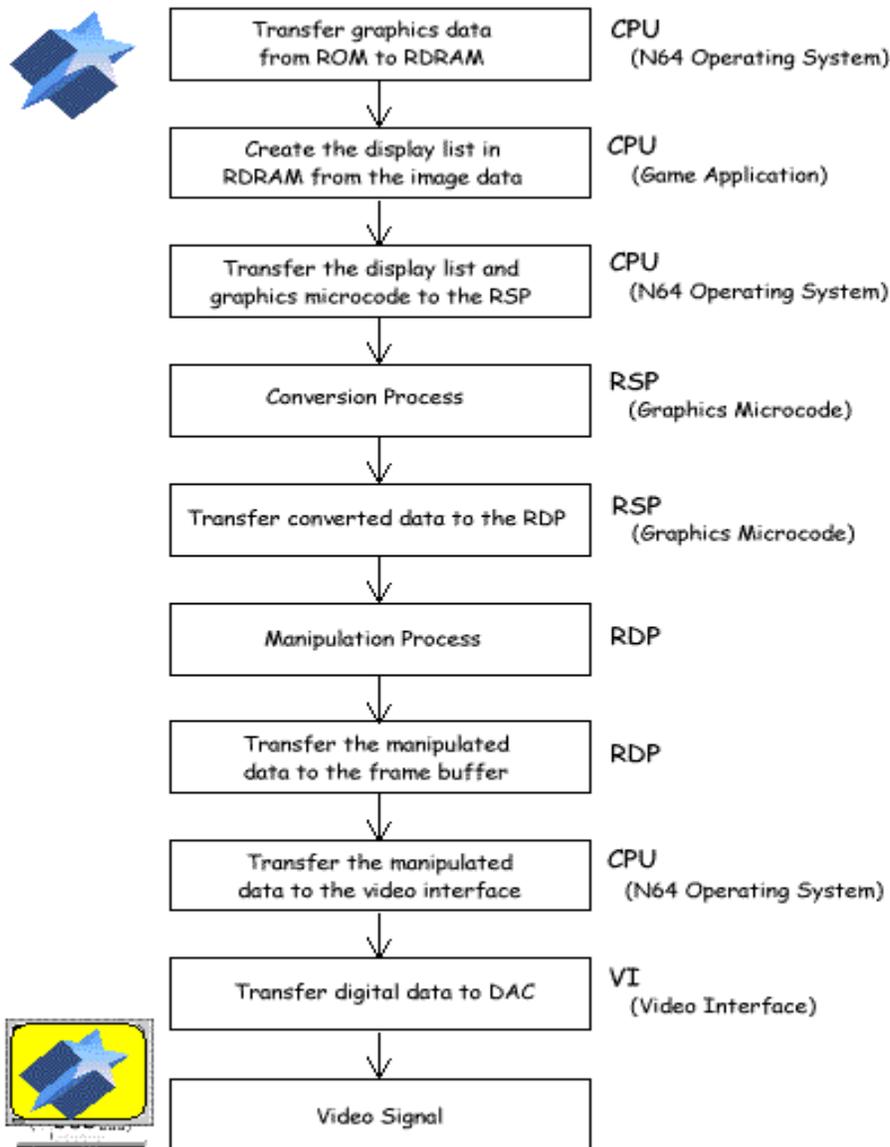


Figure 1-5-1 N64 Graphics Process

1-6 Audio

In the N64 audio process, the CPU transfers the data from the N64 Game Pak ROM to RDRAM. Then the CPU creates the audio command list ([ABI command](#) list) in RDRAM by using the synthesizer driver.

Next, the RSP performs the waveform synthesis process to transform the data into the 16-bit data of the L/R stereo by using the audio [microcode](#). Then the RSP transfers the data from the audio buffer to the AI (the Audio Interface) in RDRAM.

The AI transfers the accepted digital data to the audio DAC and it is output as sound.

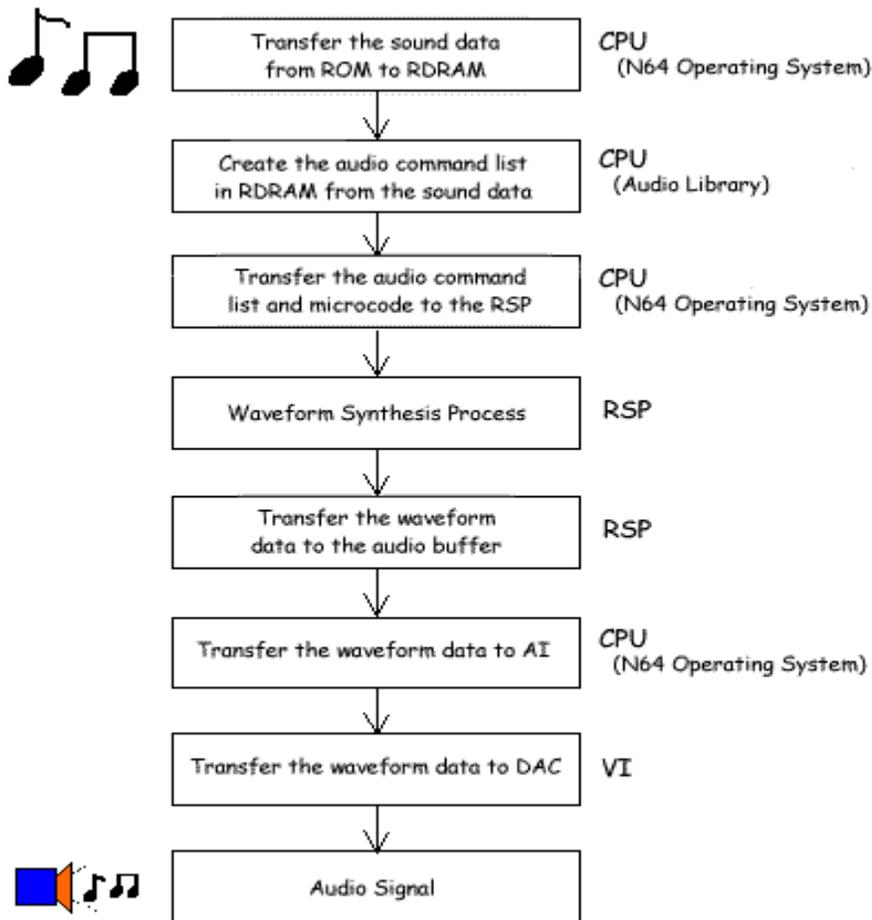


Figure 1-6-1 N64 Audio Process

1-7 Basic Terminology(thread, message, etc)

In N64 game development, various technical terms appear. Here, we will describe some of the basic terms. In addition, the glossary is attached as a separate volume.

Thread

A thread is a single processing unit operating in the CPU. A [thread](#) at any one time can have any one of the following four states:

((state of thread))

- *execution state
- *ready state
- *halt state
- *queued state

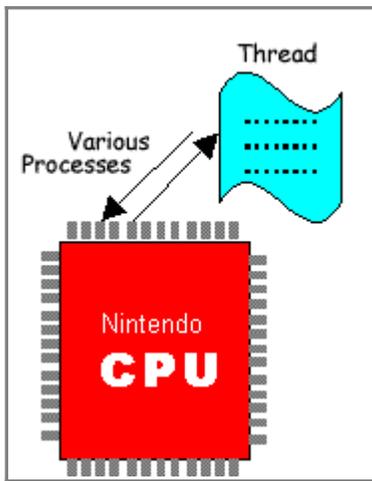


Figure 1-7-1 The thread is a process unit on CPU.

All threads execute in RDRAM under the management of the CPU.

In a standard game system, in addition to the threads created by game programmers, the following threads are reserved:

- Threads used by the N64 Operating System (PI manager, SI manager, and so on)
- The [idle thread](#)

Message

[Messages](#) are sent from and received by threads for the purpose of synchronizing information throughput between threads. Messages are managed by the message queue.

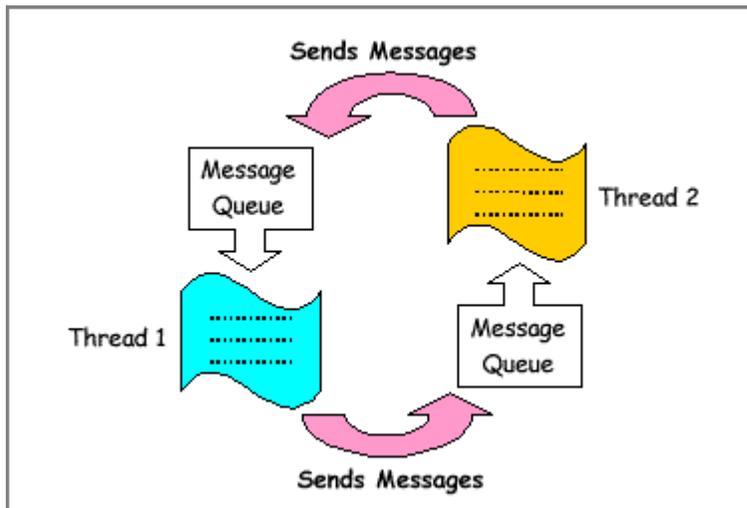


Figure 1-7-2 Messages provide synchroniztion and information throughput between threads and is managed by the queue.

Task

A task is a single processing unit operating in the RSP. The RSP in the RCP executes graphics and

audio processes. Each process is called a task.

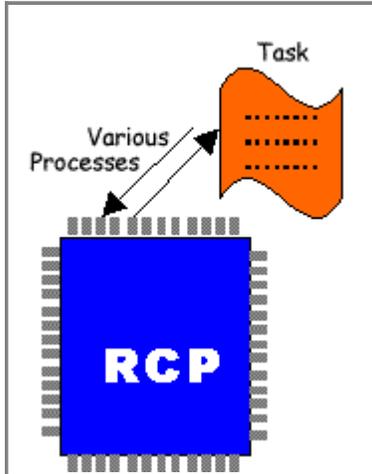


Figure 1-7-3 The task is the process provided by RSP

Scheduler

The Scheduler is a high-priority thread that manages all the other threads. It sends messages to registered threads to coordinate their efforts in the CPU. It can also pass control to another thread that manages or decides how to allocate tasks executed on the RCP.

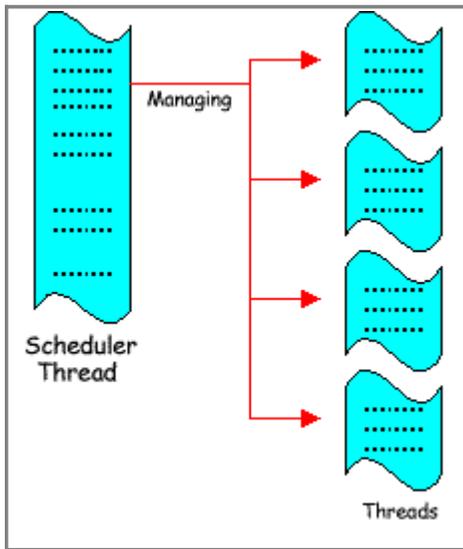


Figure 1-7-4 Scheduler manages threads

Event

A hardware interrupt is called an [event](#). The application recognizes the event by sending messages to the threads to tell them which event has occurred. The following are the main events:

- RSP

- RDP
- VI(Video Interface)
- AI(Audio Interface)
- SI(Serial Interface)
- PI(Parallel Interface)

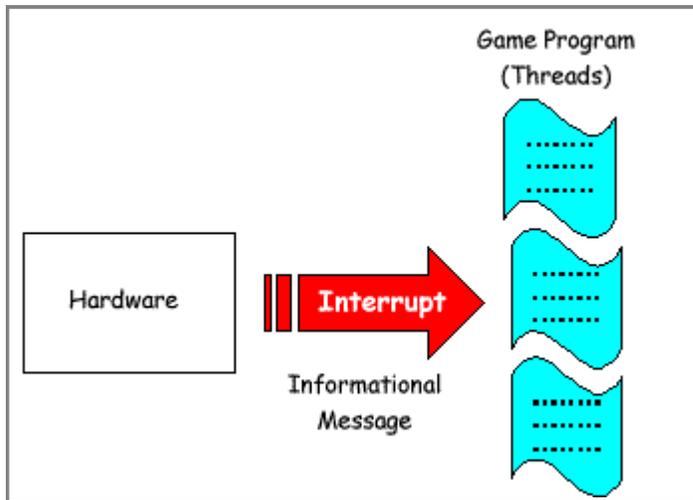


Figure 1-7-5 The events manages the interruption or exceptions

Microcode

Microcode is executed by RSP concurrently (simultaneously) with CPU processing. There are two basic types of microcode, graphics microcode and audio microcode.

*Graphics Microcode

Graphics microcode executes the graphics. There is graphics microcode for 2D and for 3D graphics. Graphics microcode analyzes graphics instructions, GBI(Graphics Binary Interface) commands, in 64-bit format and executes them. Two graphics microcodes are provided for 3D. These are for polygon and for line types of graphics. They should be used as appropriate.

It also depends upon how precise the drawing is.

Fast3D	Microcode
F3DEX	Microcode
S2DEX	Microcode

Figure1-7-6 Example of Graphics Microcode

*Audio microcode

Audio microcode executes the waveform synthesis. It analyzes audio instructions (ABI commands) in 64-bit format, and executes them.

```
aspMain  Microcode
n_aspMain Microcode
```

Figure1-7-7 Example of Audio Microcode

GBI Command (Graphic Binary Interface Command)

A GBI command is the pseudo-instruction code for drawing graphics.

Display List

The display list is just another name for the graphics command list. It is a string of GBI commands.

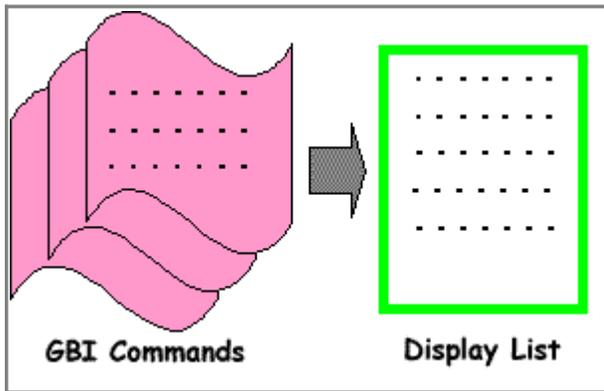


Figure1-7-8 GBI Command and [Display List](#)

ABI Command

An ABI command is the pseudo-instruction code for synthesizing waveforms.

Audio Command List

The audio command list is a string of ABI commands used to synthesize waveforms. (It is also called the ABI command list.)

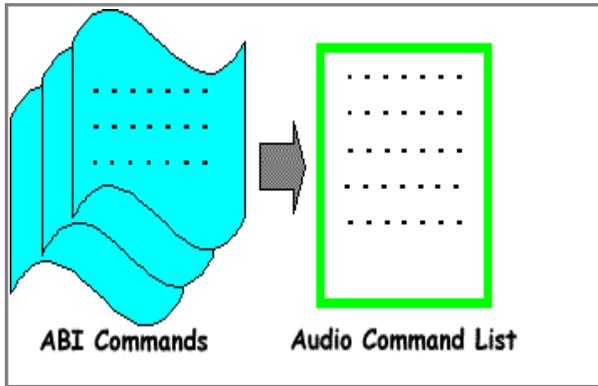


Figure1-7-9 ABI Command and Audio Command List

Z-Buffer

The Z-buffer stores the depth (z coordinate in an x,y,z coordinate system) information. By using the information in the Z-buffer, N64 can draw just those items that are in view; that is, all blocked views are not drawn. In N64, pixels are updated when the Z value is small. In the following illustration, because the Z value of the background is 9, the Z value of the letter B is 7, and the Z value of the letter A is 1, N64 knows that A is in front of (partially blocking) B.

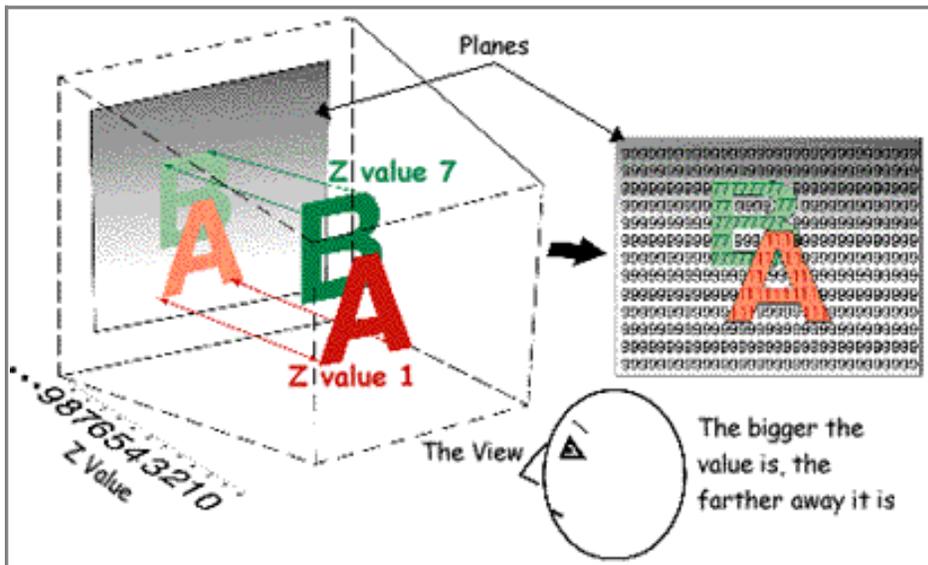


Figure1-7-10 Image of Z-buffer

(Note that this is just a conceptual illustration. In actual practice, the Z-buffer takes a value between -1.0 and 1.0.)

Pixel

A pixel is a dot on the screen. It is the minimum unit of drawing. A set of these dots becomes image data (the visual image).

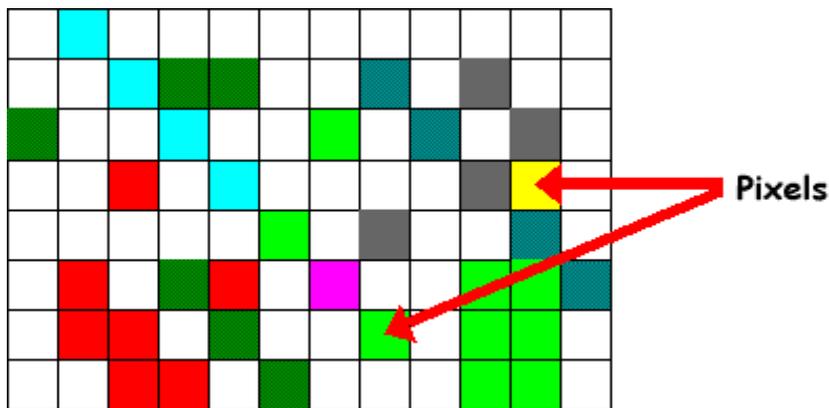


Figure1-7-11 A pixel is the minium unit of drawing

DMA (Direct Memory Access)

DMA is used to directly transfer data from a device to memory (or from memory to a device) without using the CPU. It is an effective way to transfer a lot of data. The advantage of this is that after DMA starts, the CPU can do other processing until the data transfer ends.

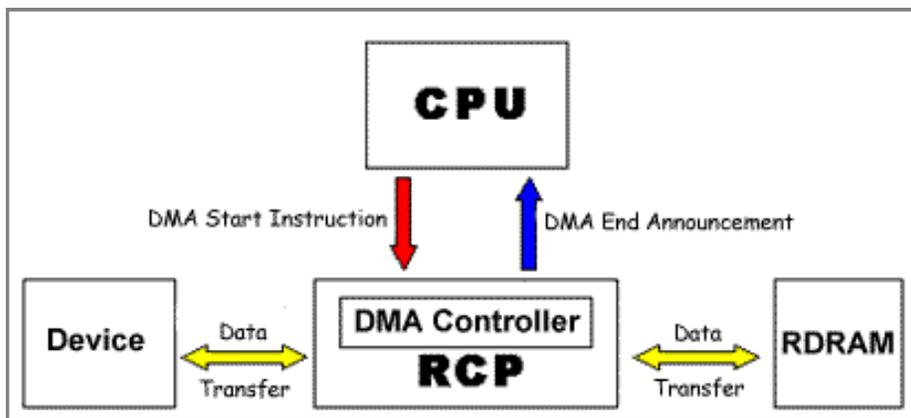


Figure 1-7-12 The flow of the DMA transfer

2-1 Hardware Architecture

The following figure gives a summary overview of the N64 hardware architecture.

This chapter went no further than describing the summary about hardware architecture. For details, please see the Chapter 3 of the N64 Programming Manual. Basically, the RCP (in the center of the following figure) is the primary workhorse of the N64. All data passes through the RCP.

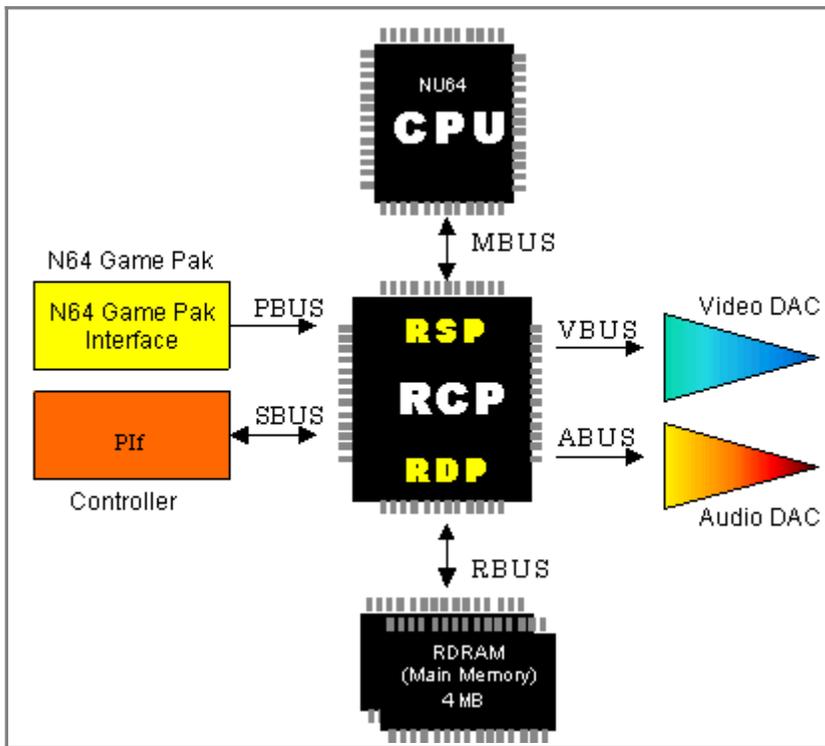


Figure 2-1-1 N64 Hardware Block Diagram

2-2 CPU

The N64 CPU is a very high-speed CPU with a clock speed of 93.75 MHz. Because an integer pipeline and a floating-point pipeline are shared, integer arithmetic and floating-point arithmetic are not processed simultaneously. However, floating-point arithmetic can be executed by the hardware.

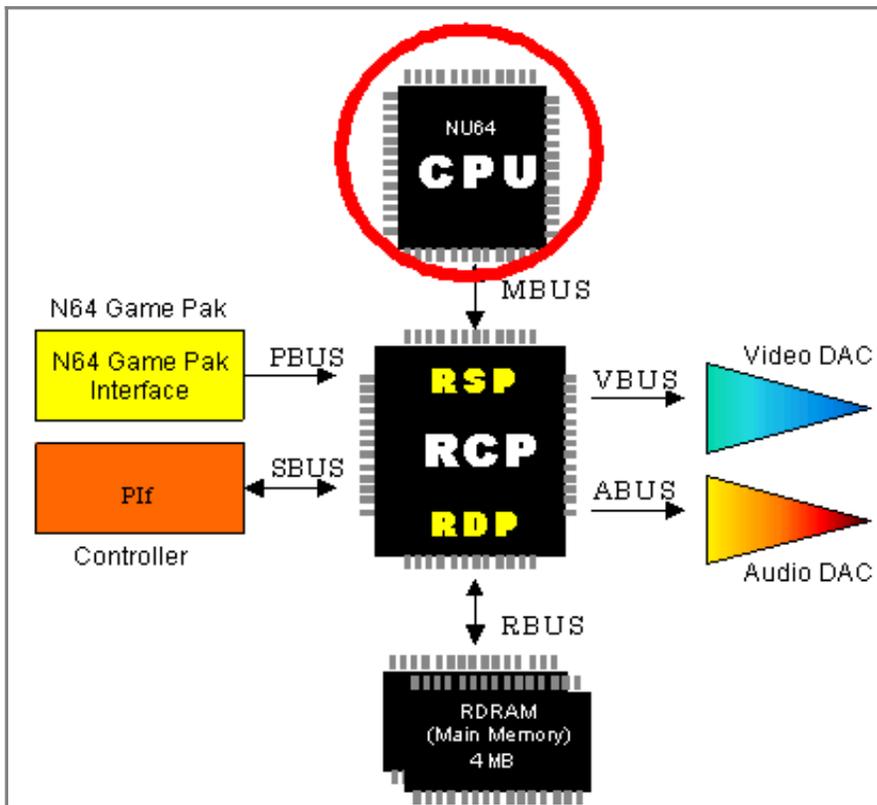


Figure 2-2-1 The N64 Hardware Block (CPU)

2-2-1 CPU Features

- Belongs to the R4000 family of processors
- Is more than 100 times the speed of NES
- Provides an execution unit that equips the 64-bit register file for integer or floating-point arithmetic.
- Provides a 16K-byte instruction cache on the CPU chip.
- Provides a data cache that uses an 8K-byte write-back system on the CPU chip.
- Provides a memory management unit that uses the high-speed translation lookaside buffer (TLB) to convert [virtual addresses](#) to physical addresses.

*TLB : Translation Lookaside Buffer

- This is a register for mapping virtual addresses to physical addresses.
- The TLB has 32 entries. Each entry maps a virtual address onto the page of two physical addresses.
- Page addresses are variable (4kbyte, 16kbyte, 64kbyte, 256kbyte, 1MByte, 4MByte, or 16MByte) and can be set independently in each entry.

2-2-2 CPU Specifications

Item	Specification
------	---------------

system clock :93.75MHz
bus width :64bit
instruction cache :16kbyte
data cache :8kbyte

2-3 RCP (Reality Co-Processor)

The RCP has two processors (RSP and RDP) and I/O interfaces. The RCP is the most important component of the N64 hardware system as shown in the following illustration. All data passes through the RCP, and the RCP serves as a memory controller for the CPU.

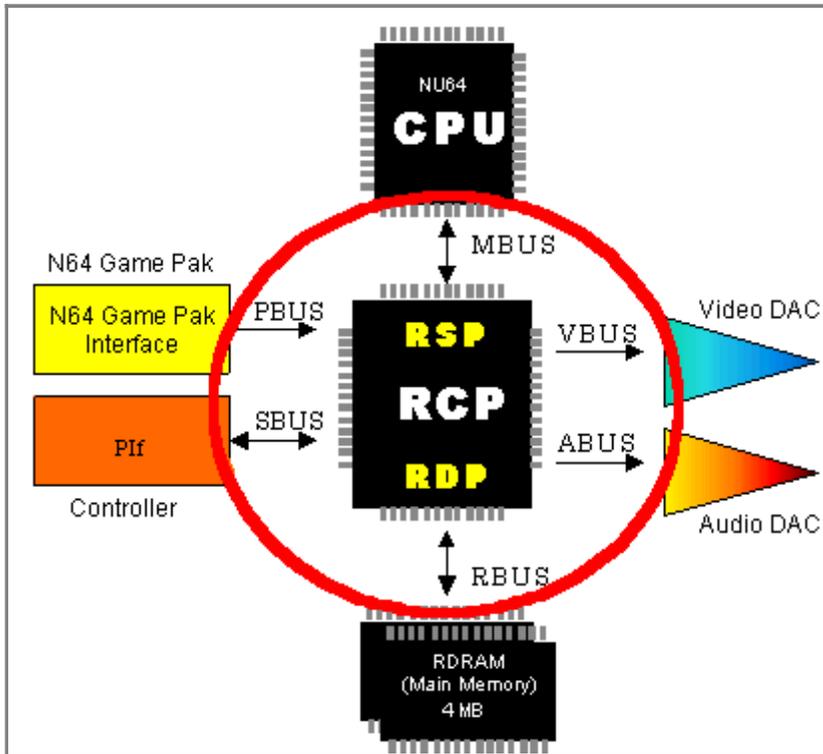


Figure2-3-1 The N64 Hardware Block(RCP)

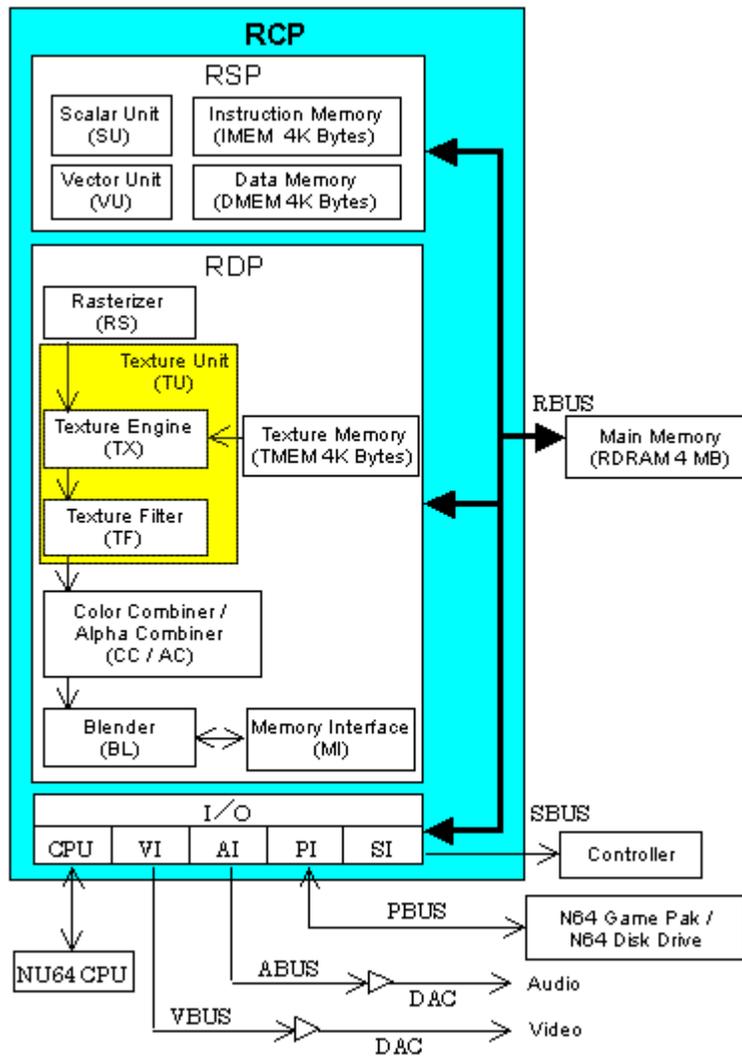


Figure 2-3-1 The RCP Process Blocks

2-3-1 RSP

*RSP (Reality Signal Processor)

The RSP executes graphics and audio tasks. It works based on [microcode](#).

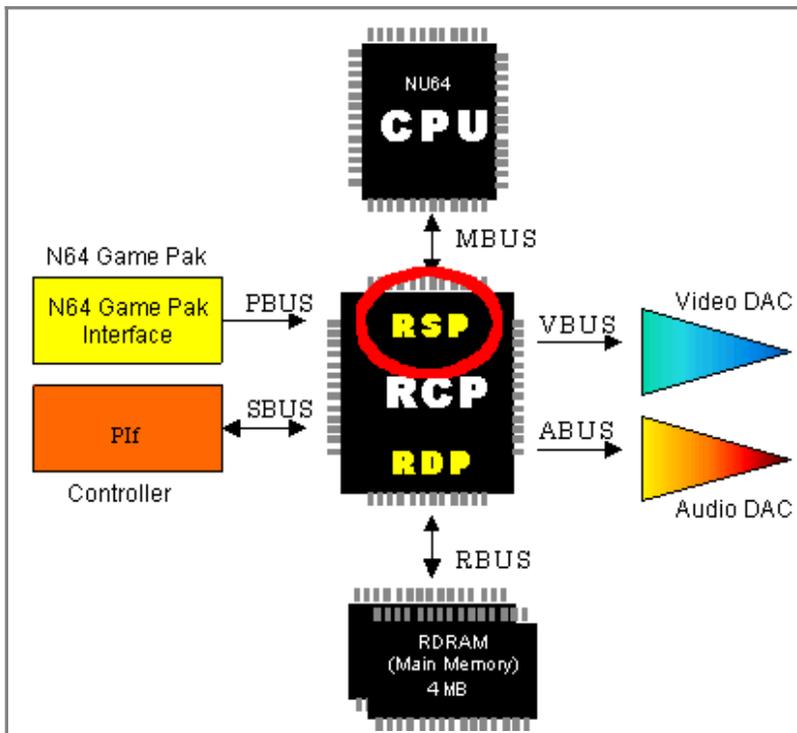


Figure 2-3-3 The N64 Hardware Block(RSP)

*RSP Process Units

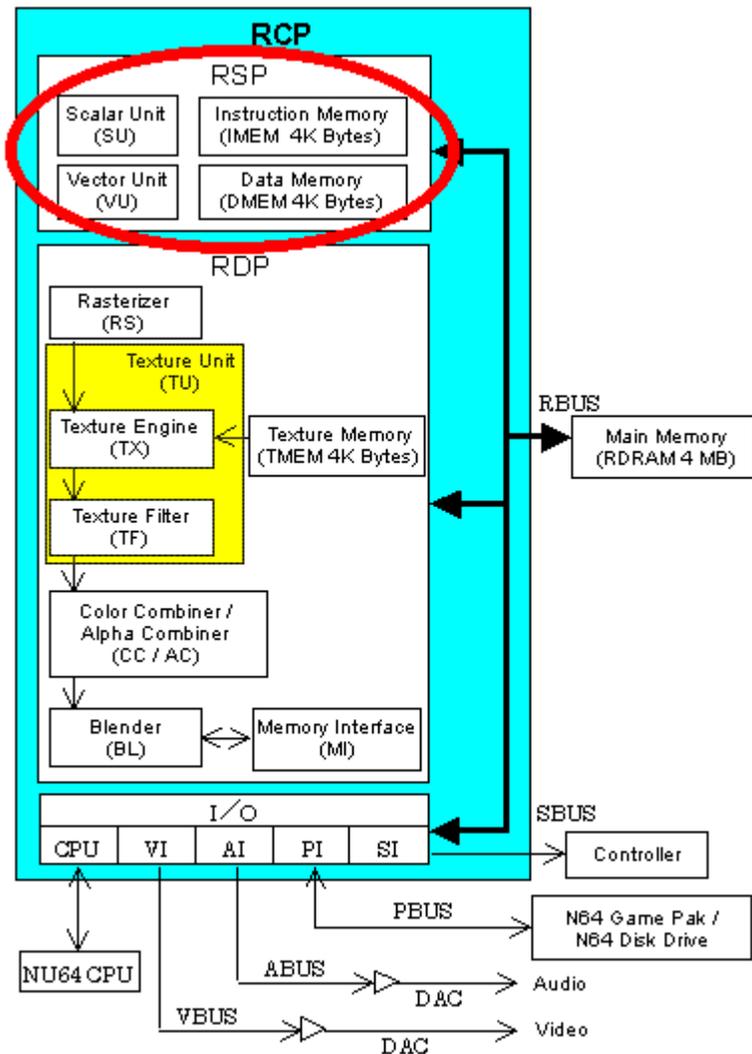


Figure 2-3-4 RSP Process Units

SU :Scalar Unit

The Scalar Unit (SU) uses a subset of R4000 instructions for execution.

VU :Vector Unit

The Vector Unit (VU) has eight 16-bit product-sum operation mechanisms.

IMEM :Instruction Memory

Instruction Memory (IMEM) is the memory that stores microcode.

DMEM:Data memory

Data memory (DMEM) is the internal working memory for the RSP microcode.

***Processes Executed by the RSP**

(Graphics)

Most of the processes provided by the RSP are executed when vertex data is loaded into the [vertex cache](#). The following are the main processes:

Geometric transformation:

This is necessary when three-dimensional objects move or must be scaled. The RSP does all necessary geometric transformations as needed. The RSP uses the 32-bit fixed-point vertex calculations to perform these transformations.

Clipping:

The clipping process cuts off [polygons](#) and pieces of polygons that are out of view of the screen as demonstrated in this illustration:

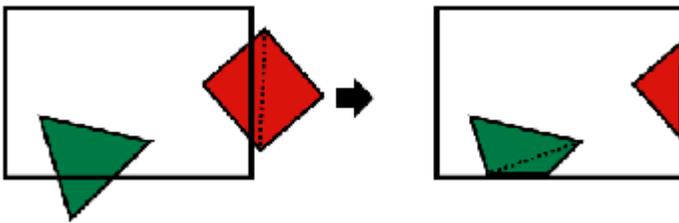


Figure 2-3-5 Example of [clipping](#)

Culling:

The [culling](#) process culls data that is not needed from the graphics pipeline. For example, data to draw the back of an object is unnecessary because it cannot be seen, so it is culled. The N64 supports two types of culling:

-Back-face culling

Back-face culling to cull the unseen back of objects

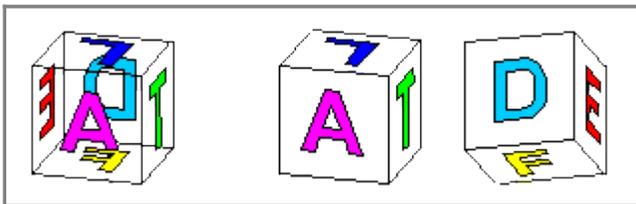


Figure 2-3-6 Image of Back-face culling

-Volume culling

Volume culling to cull items from the [display list](#) that draw objects that lie completely outside the current visual field as demonstrated in this illustration:

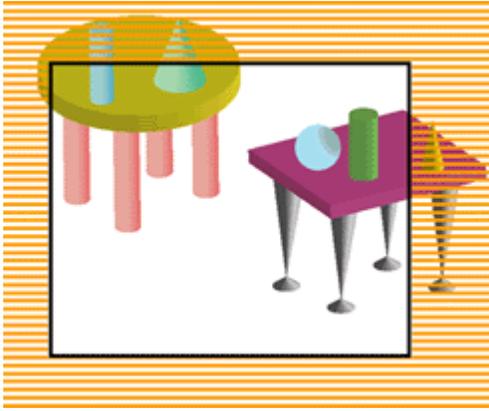


Figure 2-3-7 Example of **Volume culling**

Lighting calculations:

Used to calculate lighting.

(Audio)

The RSP processes waveform synthesis by using ABI (Audio Binary Interface) commands.

2-3-2 RDP

***RDP (Reality Display Processor)**

The RDP processes the display list created by the RSP and CPU to create the graphics data. The RDP works only with graphics; it does nothing with audio. In other words, the RDP draws the graphics in the frame buffer and processes several drawing-related operations.

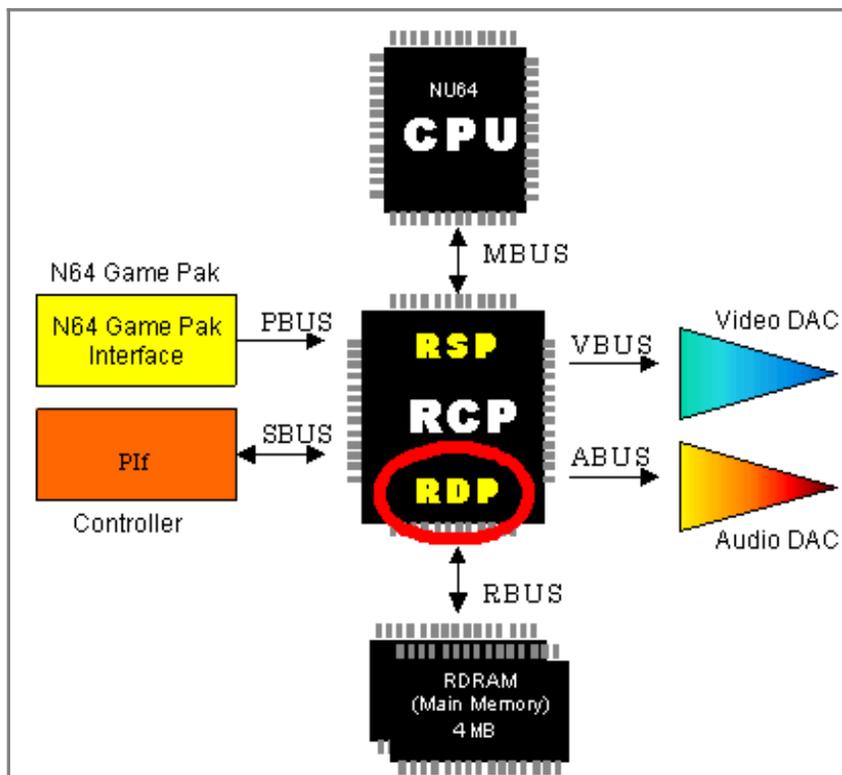


Figure 2-3-8 The N64 Hardware Block(RDP)

*RDP Process Units

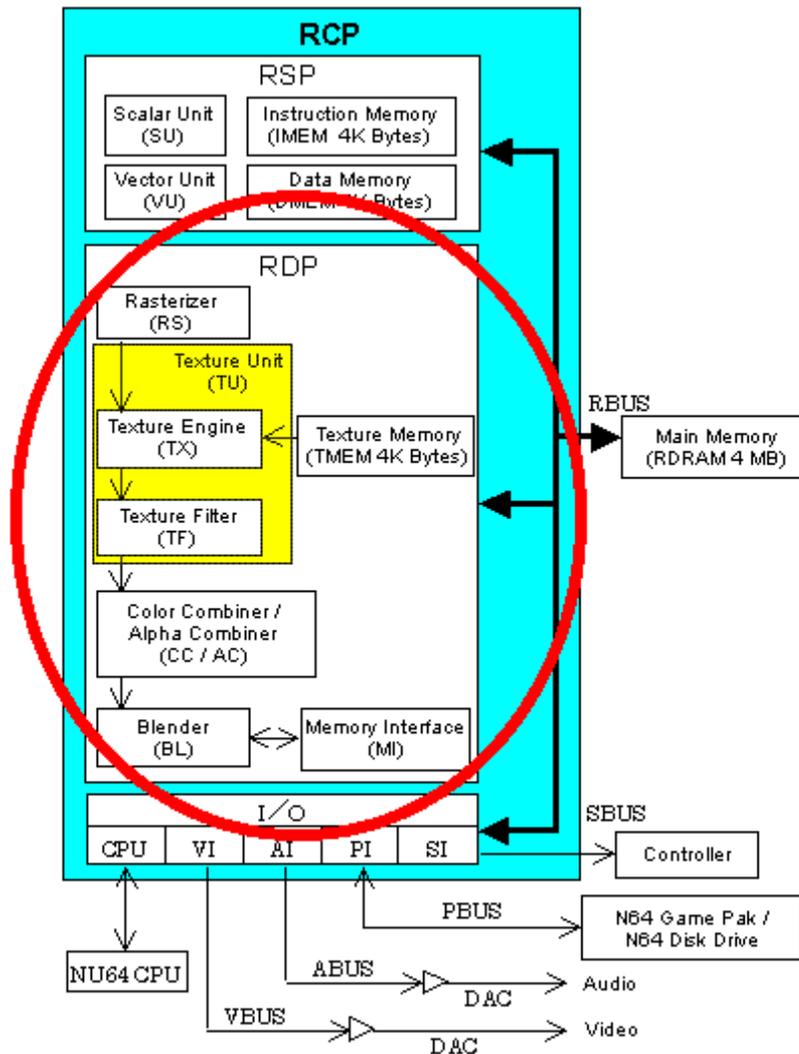


Figure 2-3-9 RDP Process Units

RS :Rasterizer

The Rasterizer (RS) changes triangles and rectangles into pixels

TX :Texture Engine

The Texture Engine (TX) provides sampling for texels(picture elements) by using TMEM(Texture Memory).

TF :Texture Filter

The Texture Filter (TF) provides filtering for texels created by TX.

CC/AC :Color Combiner/Alpha Combiner

The Color Combiner/Alpha Combiner (CC/AC) combines two colors of pixels created by RS and texels created by TF and interpolates between these two colors.

BL :Blender

The Blender (BL) blends the pixel color set from CC, the color in the frame buffer, the fog color, and so on. It draws the resulting color into the frame buffer. At this time, it can also provide [Z-buffering](#) for the first part of the anti-aliasing process.

MI :Memory Interface

The Memory Interface (MI) processes pixel information in the frame buffer including read, modify, and write operations.

***RDP Drawing Cycle Modes**

The RDP has the following four cycle modes:

Fill mode (FILL):

In FILL mode, the RDP writes pixels set in the fill color register. Four pixels in 16-bit frame buffer mode and two pixels in 32-bit frame buffer mode are written per cycle.

Copy mode (COPY):

In COPY mode, the RDP transfers the pixels in TMEM to the frame buffer. Four 16-bit pixels or two 32-bit pixels are copied per cycle.

One-cycle mode (1CYCLE):

In one-cycle mode, the RDP uses each process in the RDP pipeline once to write a pixel per cycle.

Two-cycle mode (2CYCLE):

In two-cycle mode, the RDP uses each process in the RDP pipeline twice, except the RS (rasterizer), to write one pixel per two cycles.

2-3-3 VI (Video Interface)

The VI reads data from the frame buffer using a fixed time interval, and sends it to the DA (digital-to-analog) converter (video DAC) to produce the video output.

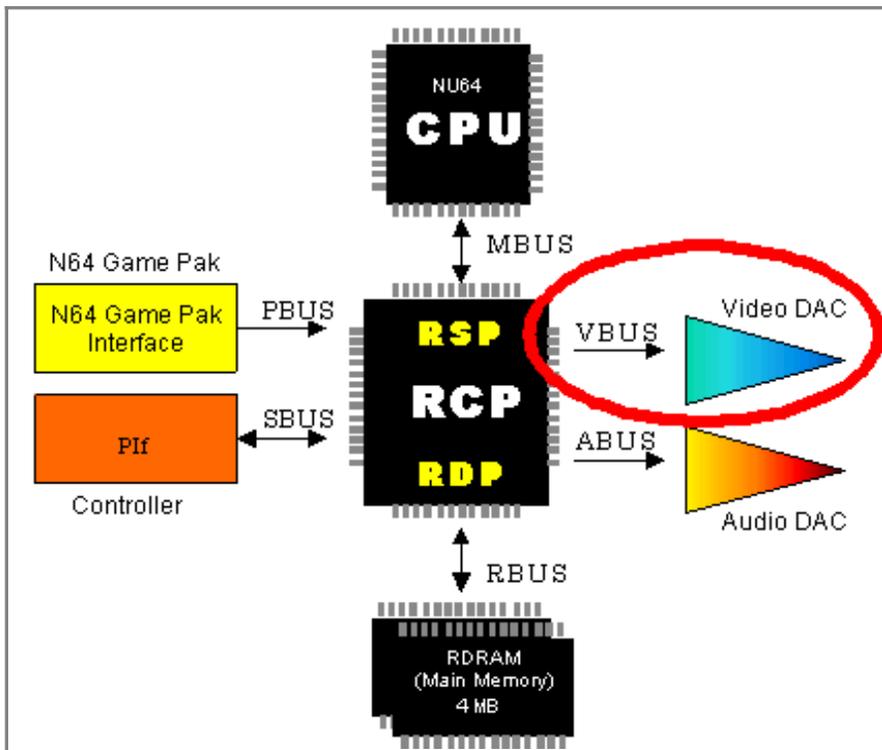


Figure 2-3-10 The N64 Hardware Block(VI)

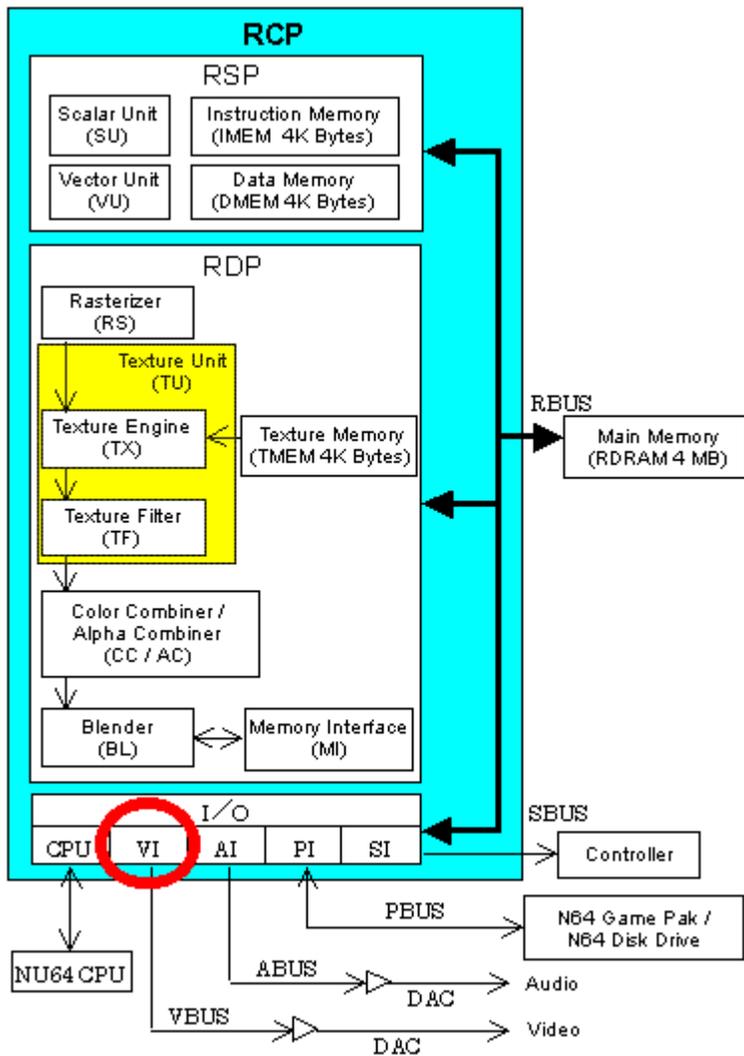


Figure 2-3-11 The N64 Hardware Block(VI)

2-3-4 AI (Audio Interface)

The AI reads data from the audio buffer using a fixed time interval, and sends it to the DA (digital-to-analog) converter (audio DAC) to produce the sound output.

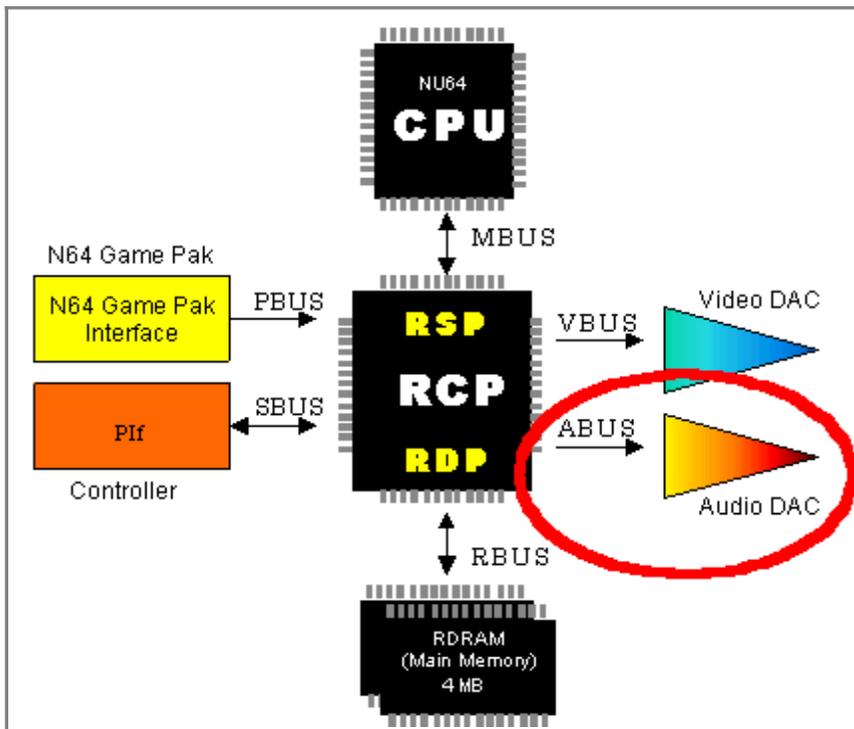


Figure 2-3-12 N64 Hardware Block(AI)

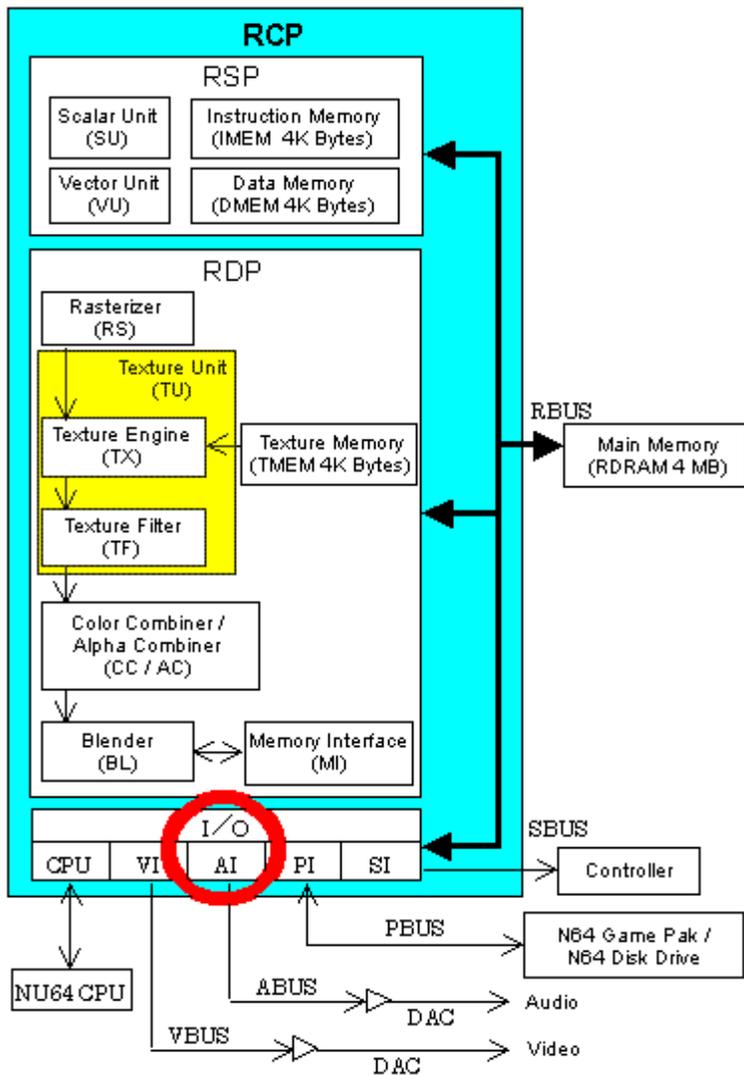


Figure 2-3-13 The N64 Hardware Block(AI)

2-4 RDRAM (N64 Main Memory)

RDRAM is the N64 main memory. It provides a total of 4 megabytes by 9 bits of memory in two chips of 2 megabytes each. The CPU and the RCP processing chips share all available RDRAM as shown in the following illustration:

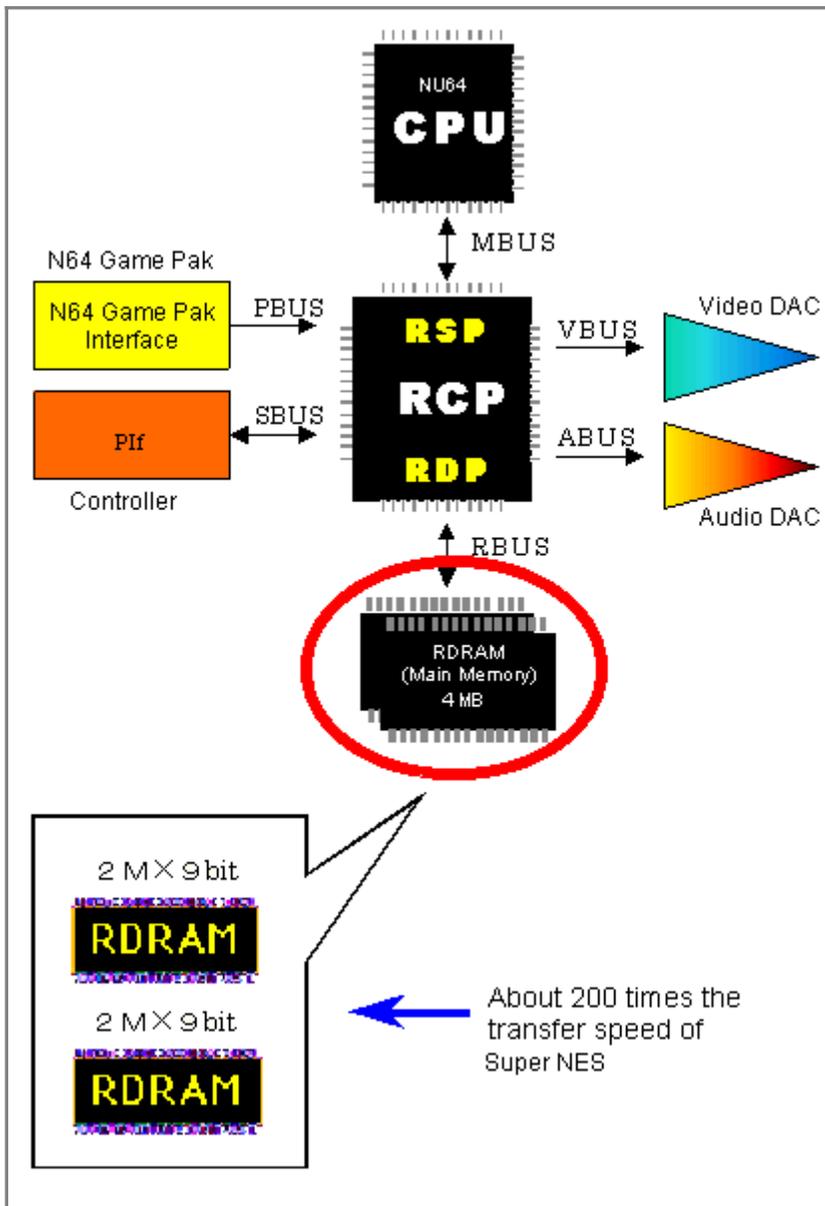


Figure 2-4-1 The N64 hardware block(RDRAM)

2-4-1 Features of RDRAM

- It provides 4 megabytes by 9 bits of space. (The 9th bit is used for [anti-aliasing](#) and Z-buffering.)
- It consists of several banks in units of 1 megabyte by 9 bits. Each bank has separate active page registers.
- The system clock operates at 250 MHz.
- It can provide high-speed data transfer of 500 megabytes per second to read or write consecutive data per the Rambus standard.
- The transfer speed is about 200 times that of Super NES.
- This high speed is not attainable for random access.
- It can be used simultaneously by the CPU, RSP, RDP, and the RCP I/O interfaces.

2-4-2 RDRAM Block Definition

As a game developer, you can map RDRAM as appropriate for each game application. Usually the following [buffers](#) and areas are specified in RDRAM:

- Audio buffer
- Frame buffer
- [Z buffer](#)
- Vertex buffer
- Texture buffer
- Audio command list buffer
- [Display list](#) buffer
- Program area
- Other data areas

Frame Buffer
Audio Buffer
Texture Buffer
Other Buffers
.
.

Figure 2-4-2 RDRAM Block Definition

2-5 SI (Serial Interface)

The SI is the interface used for exchanging data with a Controller. Each Controller is designed so that you can add a Controller Pak or a Rumble Pak, so the SI is also used for exchanging data with the Controller Pak or Rumble Pak.

2-5-1 Devices connected to the SI

- Controller
- Controller Pak for additional data storage
- Rumble Pak for vibration



Figure 2-5-1 Controller, Controller Pak, and Rumble Pak

2-6 PI (Parallel Interface)

The PI is the 16-bit parallel bus connection the N64 Game Pak or the N64 Disk Drive. The average transfer rate for the Game Pak is about 5 megabytes per second. (The peak performance is about 50 megabytes per second.) The PI uses DMA transfer to transfer information from the Game Pak or N64 Disk Drive to RDRAM.

2-6-1 Devices Connected to the PI

The PI is the DMA engine that connects the following devices with RDRAM.

- N64 Game Pak
- N64 Disk Drive

*The transfer rate

Game Pak

Average :5MByte/sec

N64 Disk Drive

Maximum :1MByte/sec

*The actual game application is stored in the ROM of the N64 Game Pak. When you use the N64 emulator board you use the RAM (called virtual ROM or RAMROM) of the developmental board. The communication between the host machine and the developmental board is provided through the RAM device on the developmental board.

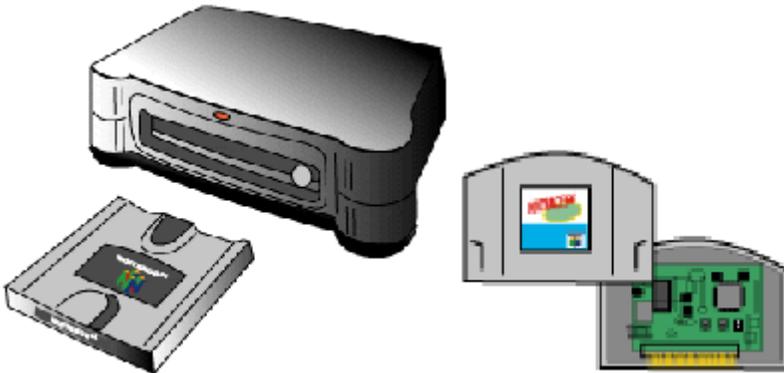


Figure 2-6-1 N64 Disk Drive and N64 Game Pak Connected to the PI

2-6-2 N64 Game Pak Specification

Mask ROM : 32M bytes (256M bits)

EPROM : 4K bits or 16K bits

SRAM(battery backup) : 256K bits

2-6-3 N64 Disk Drive

Features

- Large capacity magnetic disk for changing Character and Data.
- Built-in RTC with backup battery for real-time applications.
- Doubled RDRAM with the addition of the required Memory Pak which adds another 4 megabytes by 9 bits to N64 main memory.

N64 Disk Specification

Capacity : About 64.45M bytes

Writable capacity (the RAM part) : 0 ~ 38.44M bytes

Non-writable capacity (the ROM part) : 26.01 ~ 64.45M bytes

Store the data that users do not rewrite, such as the game program or the character data, in the ROM part. Store the saved data in the RAM part. An identification number (ID) is written on the Disk to identify each individual Disk when it is shipped from the factory. This ID is used when the Disk must be changed such as with multiple Disk game programs.

2-7 N64 Physical Memory Map

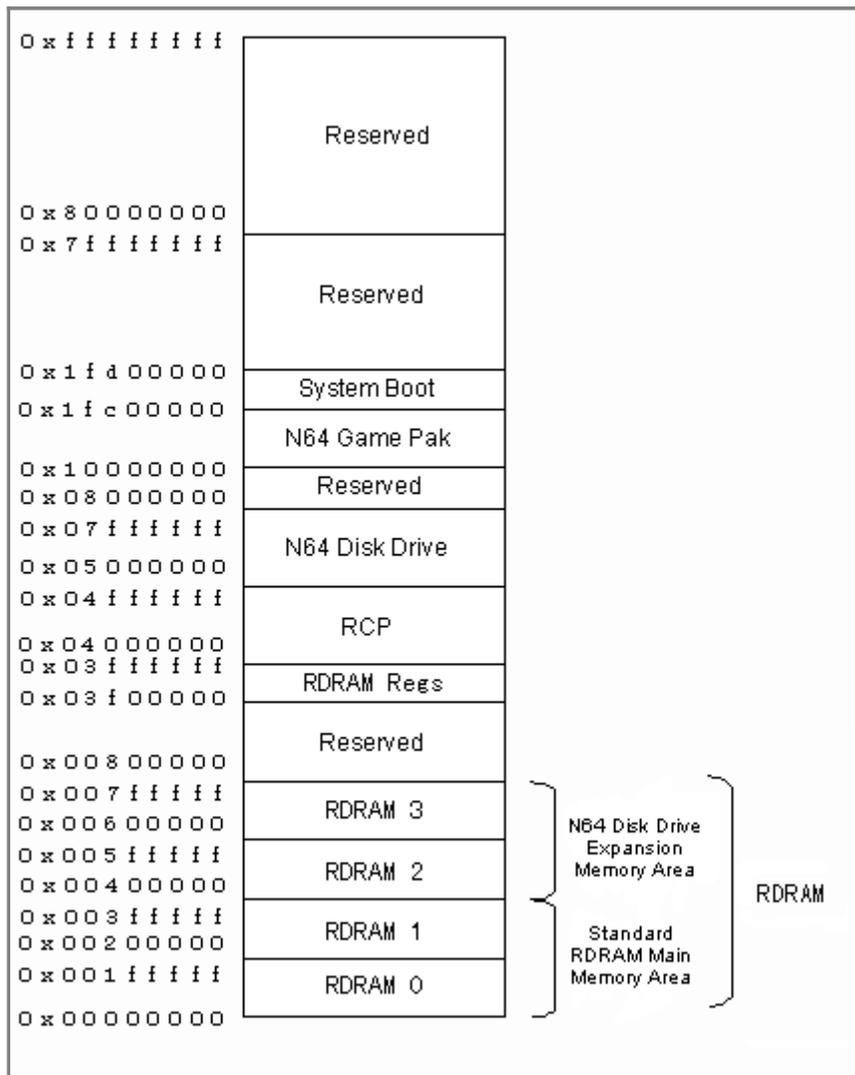


Figure2-7-1 N64 Physical Memory Map

2-8 Graphics and Audio Scheduling

An N64 game has both a graphics and an audio process.

On the N64, a **graphics process flows from the CPU to the RSP to the RDP**, and an **audio process flows from the CPU to the RSP**.

Also, while the graphics process sometimes requires more than one frame to draw one scene, the audio process must be provided in each and every frame. Otherwise, the sound pauses inappropriately or causes popping noises. If a graphics process is intensive, the RSP may be kept occupied by it and therefore fail to yield to the audio process.

The **Scheduler thread** prevents this problem from occurring. When the **Scheduler** thread receives a VI retrace (vertical synchronization interrupt) **message** it analyzes the current RSP status. If the RSP is running a graphics process, the Scheduler saves the current state of the graphics process, and then the Scheduler yields the RSP to the audio process. When the audio process finishes, the Scheduler restores the graphics process at the point where it was interrupted.

In other words, **the Scheduler is a thread that controls RSP traffic**.

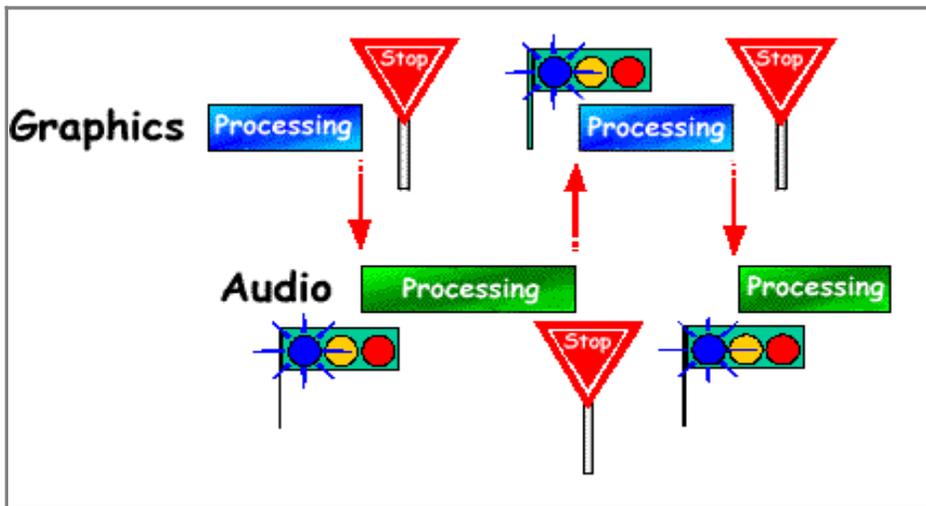


Figure 2-8-1 Graphics and Audio **Scheduling**

Introduction to **NINTENDO⁶⁴**

Chapter 1 N64 Programming

This chapter explains the features and basic concepts of N64 Programming.



1-1 Basics of the N64 Program

N64 game applications are written in the C programming language. However, an N64 game program is structured somewhat differently from a general C language program. Here, we will explain the differences between the general C language program and the N64 program.

1-1-1 Different Programming Styles

In the flow of an ordinary C program, each process executes sequentially to ensure that no processes are executed simultaneously. However, in the flow of an N64 C program, several processes may execute in parallel.*

***Processes are not executing simultaneously they are executing together by taking turns -- pausing and restarting.**

The flow of an ordinary C program continues as if one person is doing the whole job by completing several tasks in a series. The N64 C program flow continues as if several people are working on different parts of the job at the same time by sharing the tools. Because of this, you must design your N64 game programs carefully to manage all the workers and to manage the tool sharing process.

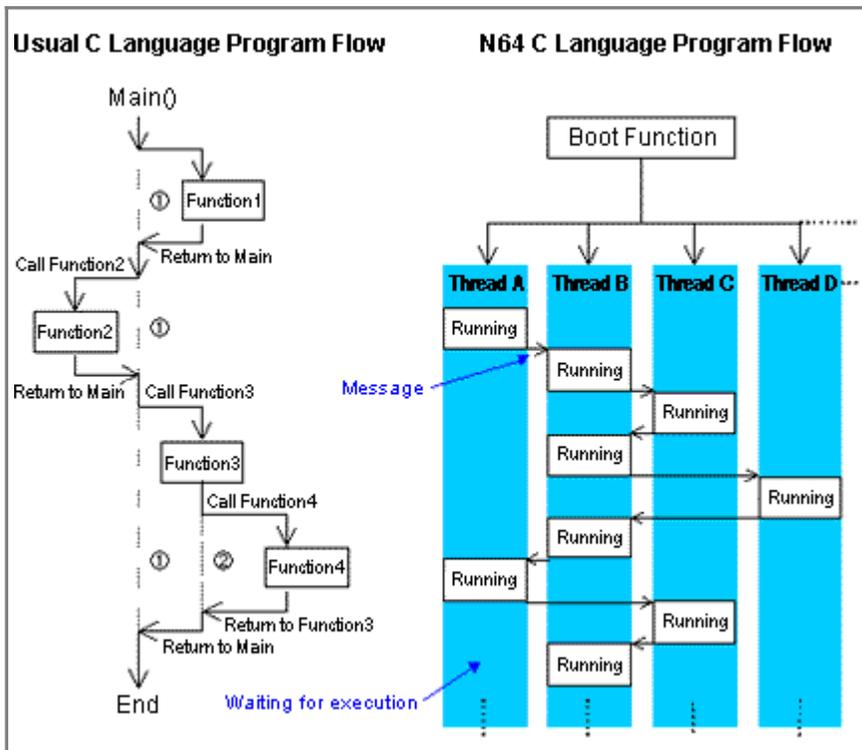


Figure 1-1-1 Standard C program and N64 game program

1-1-2 No main() Function in N64 Programs

A typical C program starts from the beginning of the main() function and proceeds by calling library functions or originally defined functions. An N64 C program, on the other hand, has no main() function. The boot function (a function specified in the spec file) begins the processing. The boot function may or may not call all the other functions the program uses. After providing various initializations and settings, the boot function may turn control over to a thread that takes charge of the main part of the program. In the case of calling the order of the boot function -> thread A -> thread B, thread A continues to process during the thread B process because, in N64 programming, threads appear to be parallel processed.

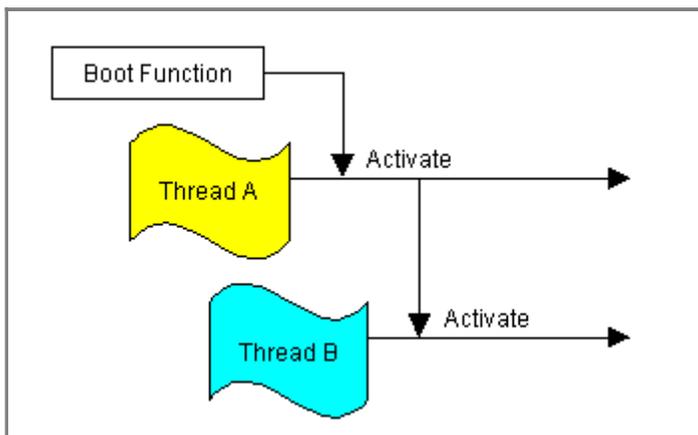


Figure 1-1-2 The parallel processing of threads

1-1-3 No Standard Output (stdout)

N64 does not have a standard output device that can use functions like `printf()`. Instead, it has the `osSyncPrintf()` function that outputs on the debugger console.

1-1-4 No Memory Control Functions

The N64 operating system supports some, but not all, of the functionality for the `malloc()` and `free()` standard C library functions. All of their functionality will be supported in the future.

1-1-5 Initialization Is Always Required

In N64 C programming, you must initialize all variables and arguments; defining them without initializing them is not sufficient. Even if an initial value was previously set, do not create a program that assumes the initialization has already occurred, because once an initialized value is assigned, it changes. You need to initialize all variables in each and every program. Also, it is impossible to guess when a low priority thread will be paused to allow a higher priority thread to execute. It is possible that a referenced working value has changed by the time the low-priority thread resumes execution. Therefore, it is crucial that you initialize each and every variable. Even though a typical C program can rely on the system to handle initializations, an N64 C program must specifically provide all variable initialization and control.

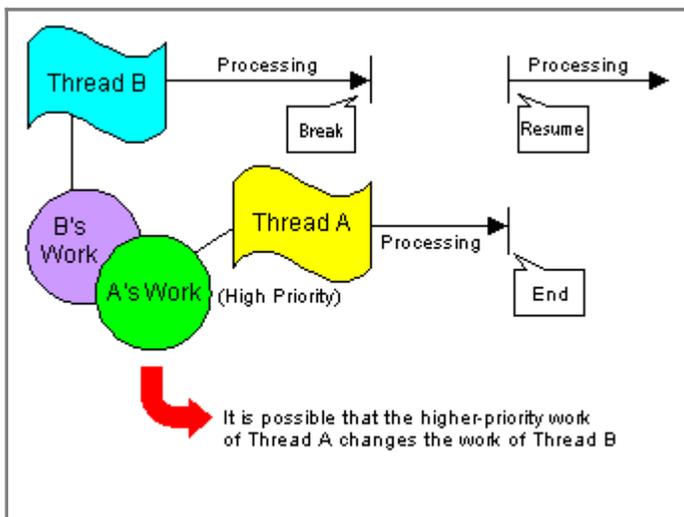


Figure 1-1-3 Initialization Is Always Required

1-1-6 Data Cache & RDRAM Consistency

- In the N64, the program data in the N64 Game Pak ROM is moved to RDRAM overwriting whatever is already there. The code is then executed there in RDRAM. As a result, if you've written the data in the program area incorrectly, there is a very high possibility that processing will enter an endless loop and your program will hang. With N64 programming, you need to keep track of addresses carefully.
- Writeback Data Cache
In N64 processing, the CPU prepares the GBI command, which the RSP provides to the graphics process in RDRAM. Because of this and the fact that the N64 cache adopts the write back method, it is possible

that the data will remain in the cache and never be written to RDRAM even if the CPU writes the data. If this happens, the RSP cannot gain access to the data written by the CPU. Therefore, after writing the data, you need to move the data cache contents to RDRAM as occasion demands by using the [osWritebackDCache](#) or the [osWritebackDCacheAll](#) function. This ensures that RDRAM holds the current contents of the data cache.

In the other direction, new RDRAM contents put there by a DMA transfer are not automatically reflected to the cache. Therefore, before doing a DMA transfer to RDRAM, you need to nullify the data cache by using the [osInvalDCache](#) function and newly make the state of data transferable from the memory to the cache.

If you think your program code is correct but it is not working correctly, make sure the cache and the memory are not working with different data.

1-2 Boot

At the beginning of every N64 game application, the boot function specified by a programmer is activated. (For convenience, it is called the boot function, but strictly speaking, whatever is specified by the entry command in the [spec file](#) becomes the [bootfunction](#).) This boot function is similar to the typical C program's main function in that it is the first function called in the program.

1-2-1 The Boot Function

The boot function initializes the operating system and passes control to the first [thread](#) to begin processing (thread A in the following illustration). Note that control migrates as threads pass control and end. When thread A finishes, it becomes the [idle thread](#).

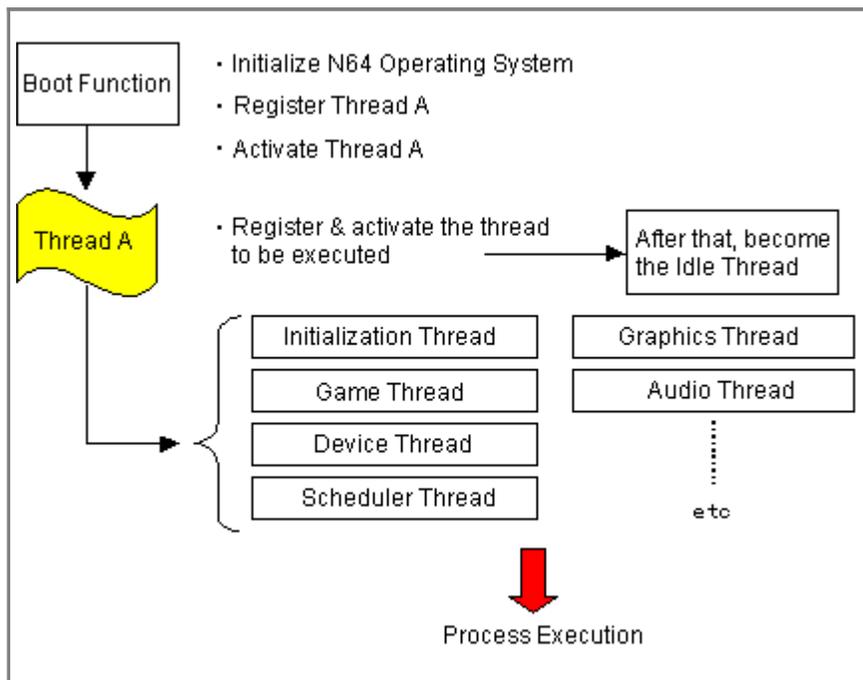


Figure 1-2-1 Boot flow

The registration of the thread that is actually operating can also be carried out by the initialization thread where the execution has been transferred from thread A. Also, you can end thread A and create a new idle thread. In either case, note that you always need the idle thread.

1-3 Thread

A [thread](#) is a single control unit of a CPU process. Under the N64 operating system, the code for all threads exists and is processed in main memory (RDRAM). A thread is like a small sub-program. N64 game programs use threads, [messages](#), [events](#) and tasks. The priority of the thread determines thread execution order. (The kinds of threads differ depending on the applications.)

1-3-1 Kinds of Threads

Each programmer is free to program each thread as is appropriate for a given game application. However, in most cases, a programmer creates the following kinds of threads:

- Initializing thread
- Game thread
- Device thread
- Scheduler thread
- Graphics thread
- Audio thread
- [Idle thread](#), etc.

1-3-2 State of the Thread

- **Running state**
Only one thread in a game program can be executing at a given time. Therefore, among those threads in the "ready condition" state, the thread with the highest priority is executed.
- **Runnable state**
A thread is in the ready condition if it is ready to begin executing as soon as it becomes the highest priority thread in the ready condition. A thread can move into the ready condition when its processing is interrupted by a higher priority thread or when it has been waiting (in the waiting condition) for resources and those resources become available.
- **Waiting state**
A thread is in the queued state when it is waiting for a message or event. Upon receiving the message (or when the event occurs), the thread changes state to the execution state or the ready condition depending on its priority.
- **Stopped state**
A thread in the halt condition has no standing in the execution [schedule](#). That is, a halted thread does not automatically become a candidate for execution unless the program specifically places it in the ready condition. For example, a newly created thread is automatically placed in the halt condition. A thread can change anytime to the halt condition and the ready condition. Use the [osStartThread\(\)](#) function to place a thread in halt condition. Use the [osStopThread\(\)](#) function to place a thread in the ready condition

The following illustration shows the relationships between the thread conditions:

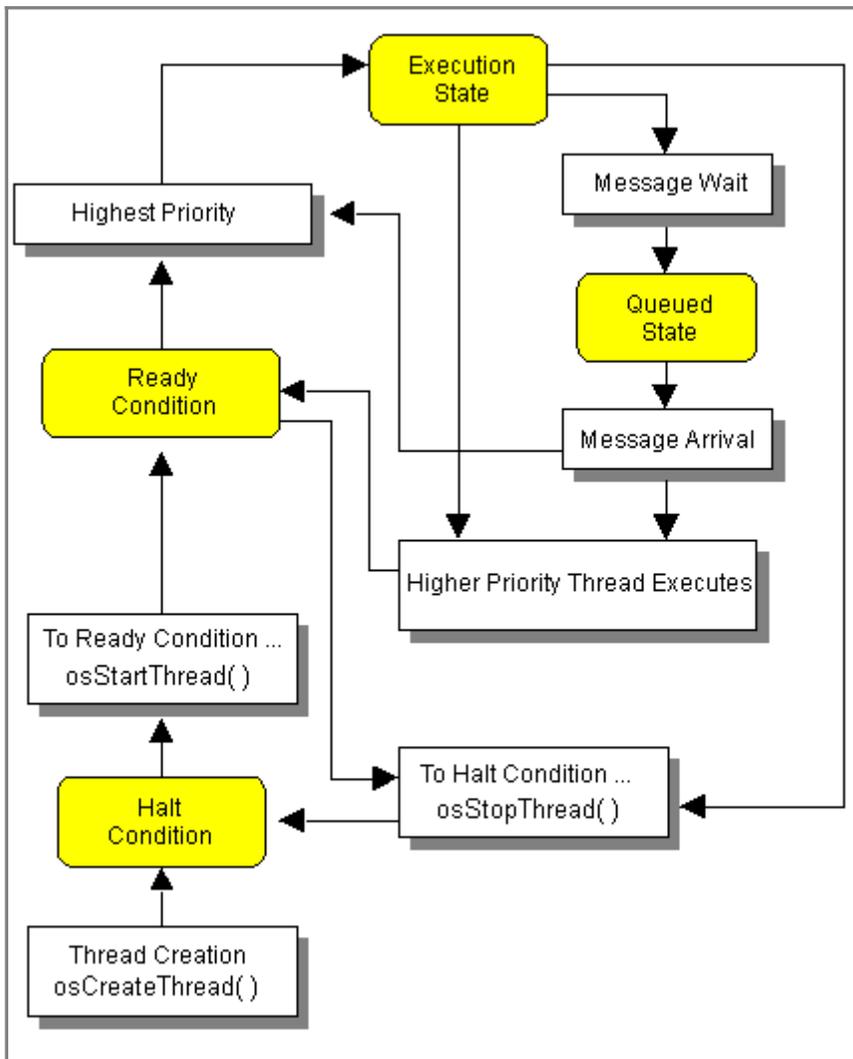


Figure 1-3-1 State of the Thread

Considering the fact that all threads **share RDRAM**, the following points need to be kept in mind:

- If every thread accepts data from the same address, each thread has the same data content.
 - Switching from thread to thread is a very fast process.
 - You must keep careful track of addresses to prevent any serious impact on other threads.
 - A lower-priority thread is always suspended by a higher priority thread during execution. Therefore, each thread must have a [stack](#) and its own control table so that it can be suspended at any time. This should be prepared with the thread accession in the N64 OS.
-
- If you raise the optimization level by adding the `-O` option when you compile your code, it is possible that each thread variable will not be updated. Therefore, you have to use the `volatile` keyword when using a variable that is common to two threads (threads A and B for example).

1-3-3 Necessity of the Idle Thread

The idle thread is the lowest priority thread. This thread is executed when the CPU does not have any other threads to process. If this thread did not exist, the CPU could not do anything. Even an idle thread has an important role. Make sure your game program creates an idle thread.

1-4 Messages

[Threads](#) send and receive messages to transfer information, to ensure synchronization, or to control thread execution and processor sharing. [Messages](#) are used to transfer control of the processor to higher priority threads. To do this, a message is sent that switches the state of the higher priority thread to the execution state, and a message is sent that switches the state of the currently running, lower-priority thread to the ready condition. Messages are also sent to and received by threads to synchronize processes. There are many ways to use messages. Their use is limited only by the imagination of the game developer.

1-4-1 Typical Kinds of Messages

1. Messages issued from other threads:

- End
- Wait
- Start

2. Messages issued for internal thread use:

- Prevent a function dual call
- Make reservations in preparation for the next process

3. Messages issued from the system event:

- Provide information about the state of a device (Controller, Rumble Pak, and so on)

4. Messages issued from the system event to the Scheduler thread and then reissued to the thread:

- Handle a VI retrace (the vertical synchronous interruption)
- Manage the current state of the audio DMA [buffer](#)
- Act as a timer
- Communicate the current RSP situation

Whereas a typical C program might provide these kinds of processes during an interruption, in the N64, the N64 operating system takes care of them by sending messages.

1-4-2 Methods for Sending and Receiving Messages

As the send/receive message has some methods, the usage depends on the application system design.

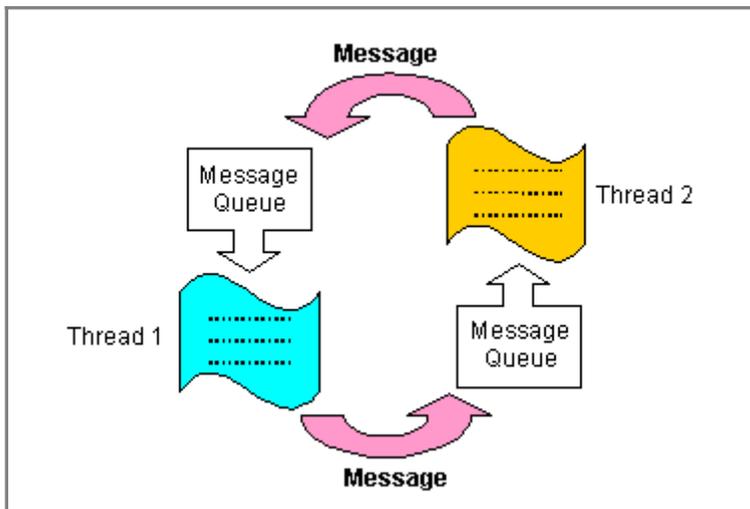


Figure 1-4-1 Messages are used to ensure thread synchronization and communication

1. Sending Method

•Non-blocked mode

If the intended receiver has space in its [message queue](#) buffer, the message is sent. If there is no space, the information of "no space in the message queue buffer", is returned the message is not sent.

•Blocked mode

If the intended receiver is blocked, the sending thread yields the CPU to another thread and enters the "queued state" while it (the sending thread) waits for the block on the intended receiver to be removed.

2. Receiving Method

•Non-blocked mode

If the receiving thread is not blocked, it verifies message arrival while performing its process. If the message queue buffer does not have the message, the receiving thread continues to process.

•Blocked mode

If the receiving thread is in blocked state, it yields the CPU to another process and enters the "queued state" while it waits for the specific message it needs. When the message it needs arrives, the receiving thread removes the block, takes back the CPU if it has the priority to do so, and runs the process.

1-4-3 Notes Regarding Sending & Receiving Messages

Thread execution order depends on thread priority and the message sending method. Therefore you need to be careful to handle priority differences between a sending thread and a receiving thread. If Thread B, executing a process, sends a message to Thread A which has a higher priority than Thread B, processing moves to Thread A, and Thread B is suspended until Thread A processing has completed. However, for this to work, Thread A must be in the "queued state" waiting for the message from Thread B so that processing can instantly move to Thread A as soon as it gets the message.

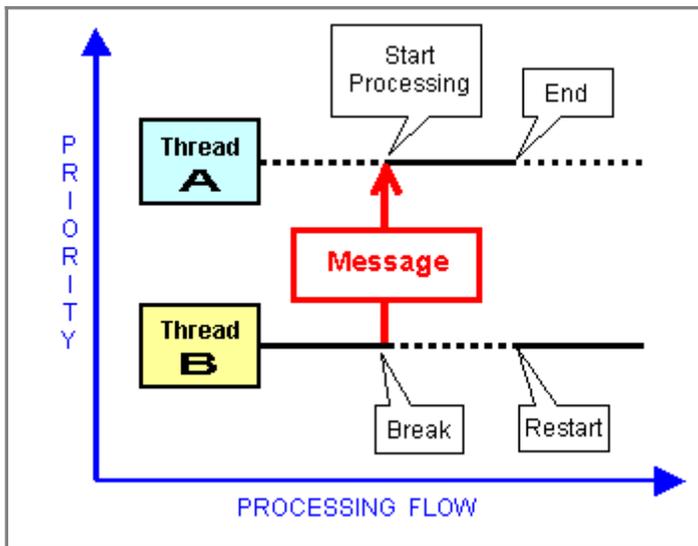


Figure 1-4-2 Thread priority and message sending (high->low)

Also, be particularly careful when a high-priority Thread A sends a message to a lower-priority Thread B to get a result that Thread A needs to continue its process. If Thread A monopolizes the process, Thread B will not be able to run anything.

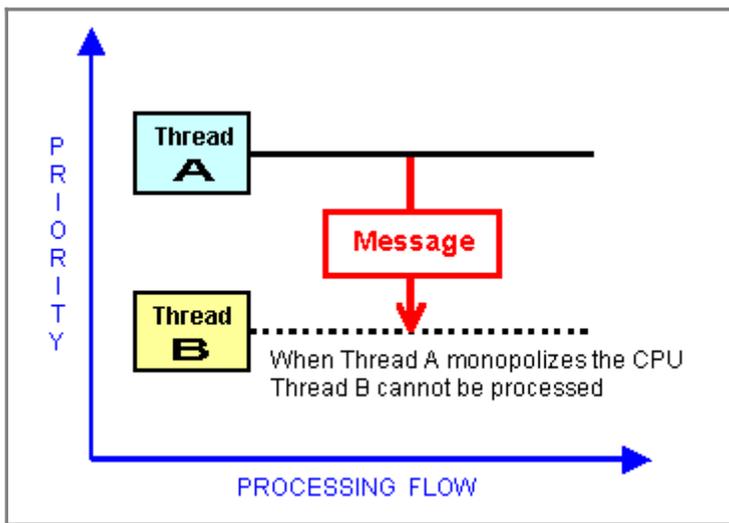


Figure 1-4-3 Thread priority and message sending (high->low)

To switch from one thread to another, always have the current thread yield the CPU by entering the "queued state" to wait for a reply. Then execution can move to another thread.

1-5 Tasks

In the N64 operating system, a "task" is a process control unit provided by the RSP in the RCP. A task is for the RCP similar to what a thread is for the CPU. However, the addressing scheme and the access methods for tasks processed by the RCP are different from the addressing scheme and access methods for threads processed by the CPU. There are basically two kinds of tasks, an audio task and a graphics task.

To execute a task, you need to set the necessary information for the task execution in the OSTask data structure defined in the N64 operating system. When you execute the task, the process follows these steps:

1. The CPU stops the RSP.
2. The CPU transfers boot [microcode](#) to IMEM in the RSP.
3. The CPU transfers the task header to DMEM in the RSP.
4. The CPU initializes the RSP program counter to zero (0) and releases the RSP stop.
5. The boot microcode operates on the RSP, transfers the microcode to IMEM, and transfers the necessary data to DMEM.
6. Execution begins at the [GBI](#) or ABI command list pointed to by the data pointer written in the task header.

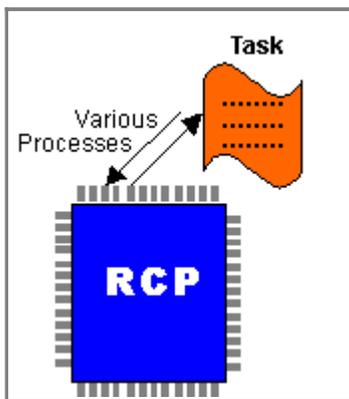


Figure 1-5-1 RCP task processing

1-6 Scheduler

The biggest job of the Scheduler is to coordinate between audio and graphics processing.

1-6-1 RSP Scheduling

The RSP executes both graphics and audio processes. A graphics process sometimes extends over one frame, but an audio process must be provided in each frame to prevent inappropriate pausing. Therefore, in each frame, if the graphics task is executing, the Scheduler suspends it and saves the task state. Then it starts the audio task executing and prepares the RSP for the restart of the graphics task execution when the audio task finishes. This series of breaks and process restarts is called the yield process.

Other jobs managed by the Scheduler include:

- [Scheduling](#) graphics.
- Sending a [message](#) to a client (a [thread](#) waiting for the message) by using the timing of a VI retrace.
- Processing messages for hardware [events](#).

Introduction to **NINTENDO**⁶⁴

Chapter 2 Drawing Structure and Samples

This chapter explains about the graphics structure.



2-1 Architecture of the Graphics Process

To render graphics, the N64 CPU constructs a [display list](#) (GBI command list) and designates the RSP as the processor to use to render graphics. The RSP outputs the resulting graphic drawing into the frame buffer in RDRAM by simply reading and executing the display list provided by the CPU. The accumulated content of the frame buffer is output as a video signal from the video DAC (digital-to-analog converter) by way of the VI (video interface).

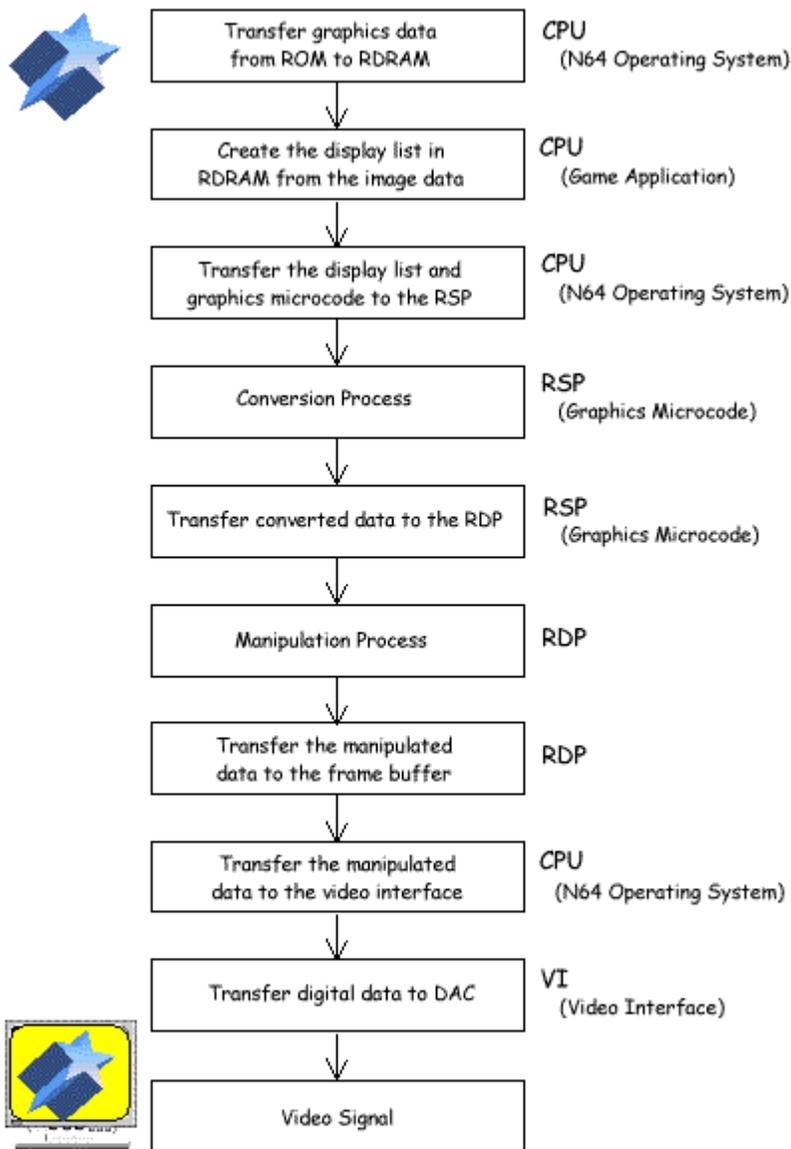


Figure 2-1-1 The flow of graphics process

2-2 Hardware Processes

2-2-1 RCP Process

- Create the [display list](#) in RDRAM.
- Post the pointer to the display list to the RSP and load the graphics [microcode](#) into the RSP.
- Designate the transfer of the data, transferred to the frame buffer from the RCP, to the VI.

2-2-2 RCP Process

Receive the necessary parameter for drawing from the CPU as the display list and execute the coordinate

calculation and the drawing calculation. These drawing processes execute inside of the RCP separating into the following blocks:

- RSP (Reality Signal Processor)
- RDP (Reality Display Processor)
- VI (Video Interface)

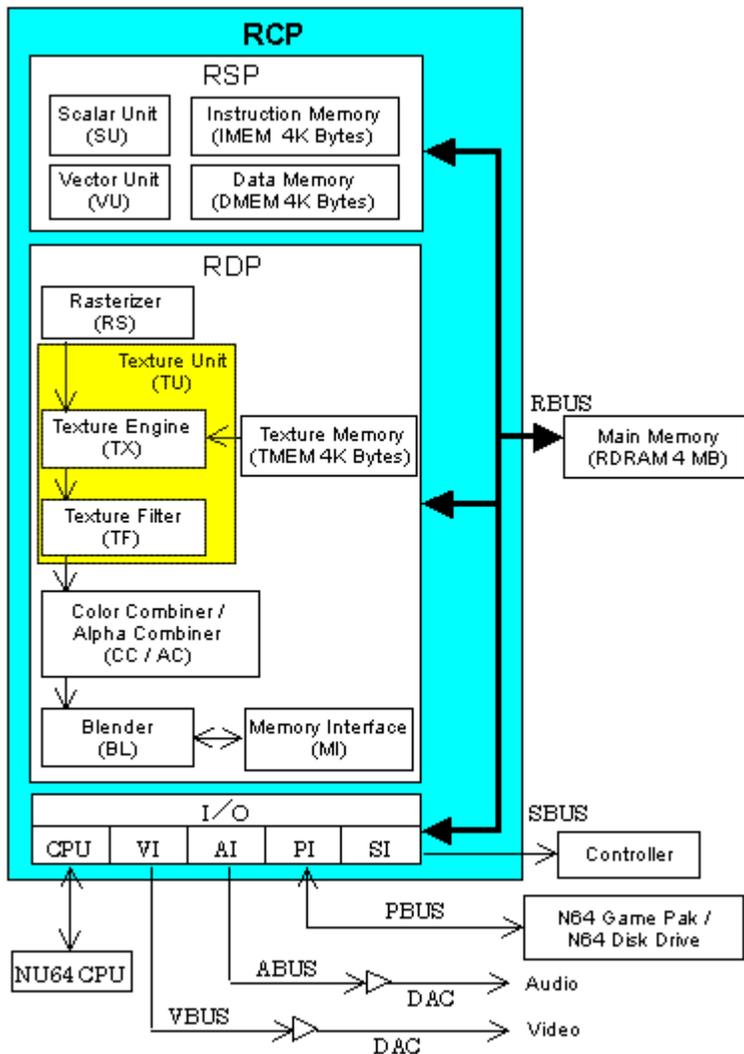


Figure 2-2-1 RCP Process

2-2-3 RSP Process

- Execute the graphics microcode transferred from RDRAM by the CPU.
 - Accept the display list within RDRAM and perform vertex conversions and other calculations.
 - Transfer the drawing instructions to the RDP.
- **Subprocesses performed as part of the calculations and conversions:**
 - [matrix stack](#) manipulation process
 - 3D [geometry](#) conversion process
 - Lighting process

- [Clipping](#) process to remove objects and parts of objects that lie outside the view and clipping the backs of objects
- Illumination and the [reflection mapping](#) process
- Setup process for [polygons](#) and line [rasterization](#) process

2-2-4 RDP Process

The RDP manipulates the data that the RSP converted to form the actual display data. Then it writes the display data to the frame buffer in RDRAM.

- **Subprocesses as part of its manipulation process:**
 - Rasterizing
 - Texturing
 - Filtering
 - Color blending
 - Anti-aliasing
 - Drawing translucent polygons
 - Processing the [surfaces](#) that lie between polygons
 - Applying fog
 - [Dithering](#)
 - Completing [scissoring](#) calculations

2-2-5 VI Process

The VI process sends the data currently residing in the frame buffer to the video DAC.

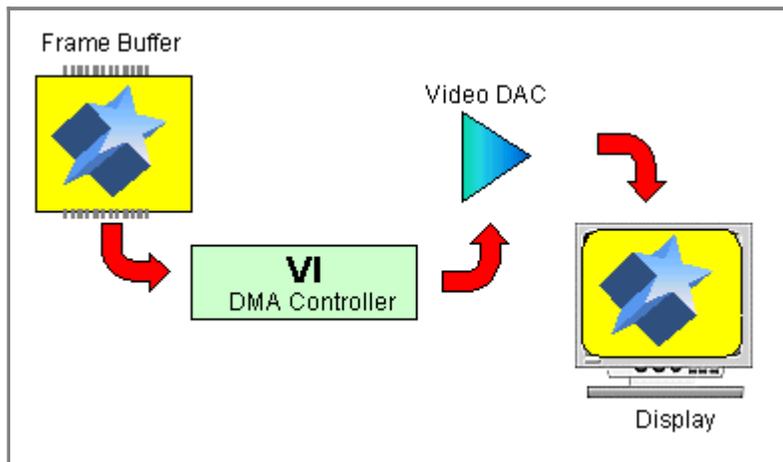


Figure2-2-2 VI Process

2-2-6 Video DAC Process

The video DAC is a piece of hardware that actually outputs the displayable video signal on the TV monitor by converting the digital data transferred from the VI into analog data that can be viewed on the screen.

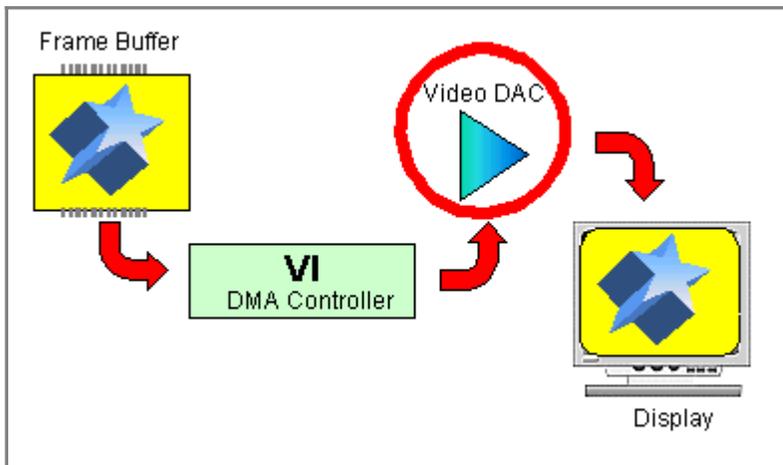


Figure 2-2-3 Video DAC Process

2-3 Software Processes

2-3-1 Library Process

Graphics library functions interpret the image data, create the [display list](#) and transfer the display list to RSP in the RCP.

2-3-2 Graphics Microcode Process

Graphics [microcode](#) transferred from RDRAM to the RSP performs image transformations in the RSP.

2-3-3 Application Process

The game application itself sets up the RCP and controls the image drawing process by specifying such things as image placement and special effects.

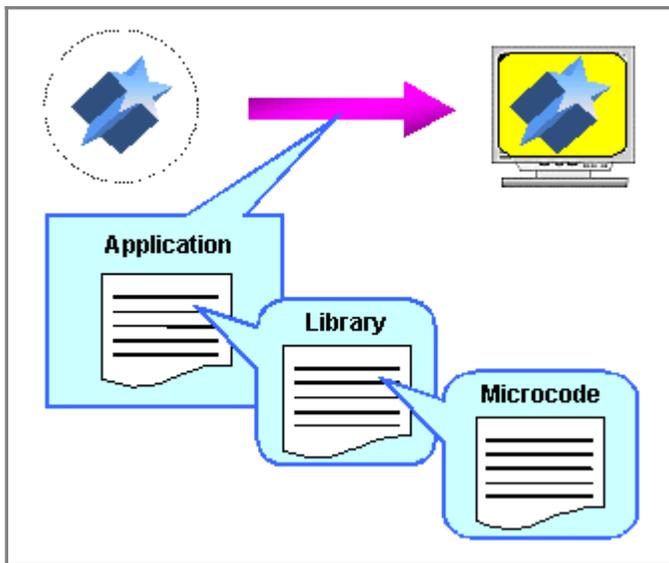
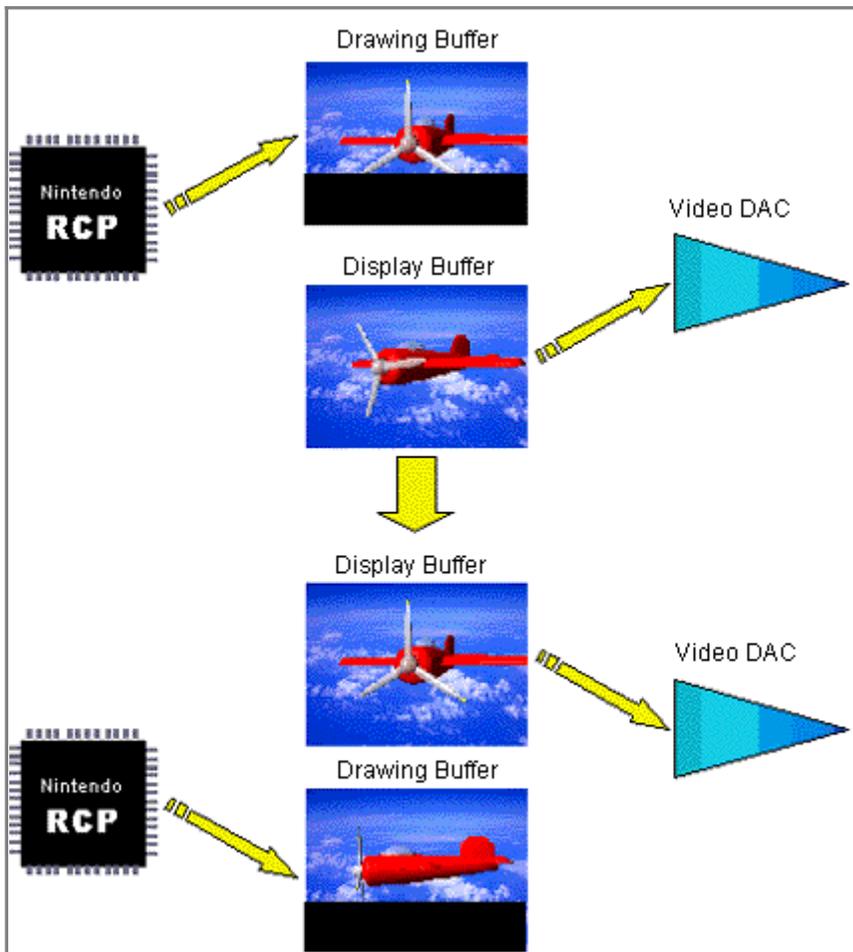


Figure 2-3-1 Software Processes

2-4 Use of Frame Buffer

2-4-1 Double-Buffering Frame Buffers

N64 accommodates two frame buffers. When one is busy displaying, the other is collecting drawing data for the next frame. This is called [double buffering](#).



(The above pictures are extreme examples of this motion)

Figure 2-4-1 Double-Buffering

As shown here, because no writing is occurring in the frame [buffer](#) that is currently displaying, the N64 can display a complete image while at the same time, in the other buffer, it is writing (buffering) new display data for the next frame. Note that the N64 does not dictate how many frame buffers you can have. You can have as many or as few frame buffers as your game application requires.

2-4-2 Rendering 3D Images

The N64 hardware is optimized for the 3D graphics process. Therefore, the 3D image drawing process automatically provides [geometry](#) conversion of the vertex information and executes the process to place the 3D object in the virtual viewing space.

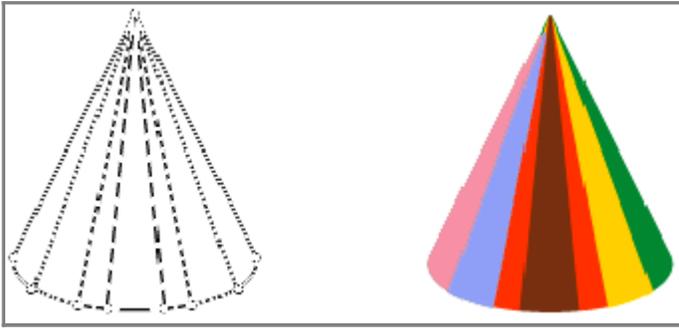


Figure 2-4-2 3D Object

2-5 2D Rendering Process

The N64 displays 2D images using the same process it uses to display [polygons](#). In reality, 2D images are just polygons with texture data pasted on them.

2-5-1 Kinds of 2D Images

To display a 2D image, the N64 uses two kinds of library functions:

- Bitmap display functions**

Bitmap display functions that draw bitmaps by using drawing parameters specified in the game application.

- Sprite display functions**

Sprite display functions that use built-in hardware capabilities to draw four-sided figures (sprites) using a small number of parameters.

The 2D images drawn can be moved in the X or Y directions or be rotated. Note: Rotation is currently not supported.

2-5-2 Procedure



The actual procedure is best illustrated with a sample program (2d.c). See [\[Appendix C Sample Description\]](#) about the sample program. Note that the gfx function is not a standard N64 function. It is defined within the 2D sample for use by that sample.

1. Main 2D Drawing Routine

The main drawing routine forms the executing core for the 2D image drawing sample. As a process flow, first construct the [display list](#) and enter (transfer) it into the RCP. The RCP creates the actual displaying data and outputs the video signal via the frame buffer.

```
void entry(void)
{
    Gfx *gp; /* points to display list */
    u16 w, h;

    w=64; h=64; /* sets up sprite size */
    while(1){
```

- **Construct the display list**

The following code reserves the necessary memory area for the construction of the display list, sets up the RCP execution process for sprite drawing in the reserved area, and provides a termination process for the constructed display list.

```
/* Start to construct a display list */
gp = gfxBegin(1024);
```

This code checks to see if a display list has already been constructed. If a display list has not been completed, this code reserves the GBI command area for new construction. If it has been completed, this code moves on to step 2, the process that transfers the display list to the RCP.

```
/* Set the drawing mode for RSP and RDP */
gp = setup_SP_DP(gp);
```

This code constructs a command that sets the necessary RCP drawing mode, and then adds that command to the display list.

```
/* Accept the texture pattern */
gp = load_texture(gp,w,h);
```

This code sets up a texture pattern loading command in the display list.

```
/* Write the texture pattern */
gp = draw_texture(gp,124,92,w,h);
```

This code sets up a texture drawing command in the display list.

```
/* End the construction of display list */
gfxEnd(gp);
```

This code terminates the display list.

- **Caution 1:** Watch out for an unterminated display list. If it is transferred to the RCP, it will cause the RCP to hang (stop responding).
- **Caution 2:** be sure to put the [gDPFullSync](#) function at the end of each display list. Otherwise, the RDP end [message](#) will not ever come.

- **Transfer the display list to the RCP to execute the drawing process**

The following code transfers the display list to the RCP where the display list is interpreted and executed.

```
/* Transfer display list to RCP */
gfxFlush( );
```

This code transfers the display list to the RCP where it is executed. This function also provides the frame buffer switch that writes the created image data into the frame buffer.

- **Synchronize the CPU with the RCP**

The following function ensures coordination between the CPU and RSP:

```
/* Wait for the retrace */
gfxWaitSync( );

}

}
```

2. Techniques for Construction of the Display List

The actual construction of the display list uses one of these processes:

- Set the RSP and RDP Drawing Modes
- Set and Read the Texture
- Set the Drawing Sequence of the Bitmap Pattern

- **Set the RSP and RDP Drawing Modes**

The following routines set the RCP drawing mode that actually creates the commands that render the drawing reflected in the display list. The [gSP](#) and [gDP](#) functions are included in the N64 library. For more information, please see the online ["N64 Function Reference Manual"](#) (HTML manual pages).

```
static Gfx *setup_SP_DP(Gfx *gp)
{
/* Set all sorts */

/* Set the texturing parameter */
gSPTexture(gp++,0x8000,0x8000,0,G_TX_RENDERTILE,G_ON);

/* The synchronous setting between the rendering and sub-attribute*/
gDPPipeSync(gp++);

/* Set the RDP cycle type */
gDPSetCycleType(gp++,G_CYC_COPY);

/* Set the rendering mode of the blender within RDP */
gDPSetRenderMode(gp++,G_RM_NOOP,G_RM_NOOP2);

/* Set the textureLOD */
gDPSetTextureLOD(gp++,G_TL_TILE);

/* Set the perspective of the texture map */
gDPSetTexturePersp(gp++,G_TP_NONE);

/* Set the detail type */
gDPSetTextureDetail(gp++,G_TD_CLAMP);

/* Set the texture filter type */
gDPSetTextureFilter(gp++,G_TF_BILERP);

/* Set the conversion mode of the color space */
gDPSetTextureConvert(gp++,G_TC_FILT);

/* Set the compare mode of the alpha value */
gDPSetAlphaCompare(gp++,G_AC_NONE);
```

```

/* Set the dithering mode of the color data */
gDPSetColorDither(gp++,G_CD_DISABLE);

/* Set the dithering mode of the alpha value */
gDPSetAlphaDither(gp++,G_AD_NOISE);
return gp;
}

```

- **Set and Read the Texture**

The [gDP functions](#) are included in the N64 library. For more information, please see the online "[N64 Function Reference Manual](#)" (HTML manual pages).

```

static Gfx *load_texture(Gfx *gp,u16 w,u16 h)
{
/* Read TLUT */

/* Set the texture look-up table */
gDPSetTextureLUT(gp++,G_TT_RGBA16);

/* Read the texture look-up table */
gDPLoadTLUT_pal16(gp++,0,texturetlut);

/* Read the bitmap pattern */
gDPLoadTextureTile_4b(gp++,texture,G_IM_FMT_CI,w,h,
0,0,w-1,h-1,0,
G_TX_WRAP | G_TX_NOMIRROR,G_TX_WRAP |
G_TX_NOMIRROR,
G_TX_NOMASK,G_TX_NOMASK,G_TX_NOLOD,
G_TX_NOLOD);

return gp;
}

```

- **Set the Drawing Sequence for a Bitmap Pattern**

The following code sets the drawing sequence for an accepted texture image. The [gSP functions](#) are included in the N64 library. For more information, please see the online "[N64 Function Reference Manual](#)" (HTML manual pages).

```

static Gfx *draw_texture(Gfx *gp,u16 left,u16 top,u16 w,u16 h)
{
/* The bitmap pattern drawing */
gSPTextureRectangle(gp++,
left<<2,top<<2,((left+w)<<2)-1,((top+h)<<2)-1,
G_TX_RENDERTILE,
0,0,4<<10,1<<10);
return gp;
}

```

2-6 3D Rendering Process

N64 displays 3D images as textured polygons. The actual 3D image is an aggregate of polygons provided with [geometric](#) conversion information at each vertex and placed in the virtual view space.

2-6-1 Kinds of 3D Images

Each 3D image is classified as either a 3D object or a 3D model depending on the number of the vertices and how the object is used.

- **3D object**

3D object is a single closed object that treats and displays all vertex information as such. Here is an example of a 3D object:

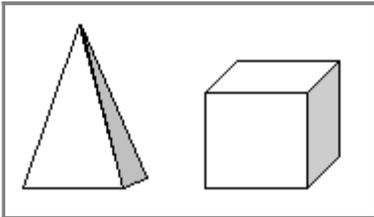


Figure 2-6-1 3D object

- **3D model**

A 3D model is a combination of several separate 3D objects. Each of the 3D objects is a part of the 3D model.

For example, when you draw a car, you create the body and each of the wheels as separate parts (3D objects). Then you combine the parts to create the car (3D model).

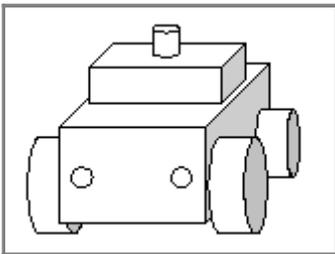


Figure 2-6-2 3D model

2-6-2 Procedure



The procedure is best illustrated with a sample program. See [\[Appendix C Sample Description\]](#) about the sample program. Note that the `gfx` function used in this 3D sample is not part of the standard N64 library of functions; it is defined within the sample for use in this sample.

1. Main 3D Drawing Routine

Below is a routine list which becomes the executing core of the 3D image drawing. As the process flow, first the [display list](#) and entry (transfer) the display list to RCP. RCP creates the actual displaying data and outputs the video signal via the frame buffer.

```
void entry(void)
{
    Gfx *gp; /* points to the display list */
    float r,v,s;

    r=v=0.0;
```

```

s=PI2/50000.0;
while(1)
{
/* Move one unit of the camera (to set the
   rotational rate of the 3D model) */
v += s;
r += v;
if (ABS(v)>PI2/100.0) s=-s;

```

- **Construct the display list**

The following code snippets reserve the necessary memory area for the construction of the display list, set up the RCP executing process in the reserved area (to draw the 3D model) and provide a termination process for the display list.

```

/* Start to construct the display list */
gp = gfxBegin(1024);

```

This code checks to see if execution setup has already been completed. If not, the code reserves the GBI command area for a new construction. If setup has already been completed, this code jumps to the RCP transfer process.

```

/* Set the drawing mode of RSP and RDP */
gp = setup_SP_DP(gp);

```

This code constructs the command that sets up the RCP drawing mode (needed for 3D drawing) and stores it in the display list.

```

/* The projection matrix setting */
gp = gfxPerspective(gp, 30, (float) SCREEN_WD/SCREEN_HT, 64,
1000, 1.0);

```

This code sets up the projection matrix for use by the coordinate transformation process in the display list.

```

/* The model view matrix setting */
gp = gfxLookAt(gp, sinf(r)*400, 220, cosf(r)*400, 0, 0, 0, 0, 1, 0);

```

This code sets up the model view matrix for use by visual effect calculations in the display list.

```

/* Polygon drawing */
gp = draw_polygon(gp);

```

This code creates a display list for polygon drawing. It calls the actual display polygon as a separately accommodated display list.

```

/* End construction of the display list */
gfxEnd(gp);

```

This code provides the display list termination process.

- **Caution 1:** Watch out for an unterminated display list. If it is transferred to the RCP, it will cause the

RCP to hang (stop responding).

- **Caution 2:** Be sure to put the [gDPFullSync](#) function at the end of each display list. Otherwise, the RDP end [message](#) won't ever come.

- **Transfer the display list to the RCP to execute the drawing process**

The following code transfers the display list to the RCP where the display list is interpreted and executed.

```
/* Transfer completed display list to RCP */
gfxFlush( );
```

This code transfers the completed display list to the RCP and makes the RCP execute the process based on the display list. It also provides for the use of the frame buffer that writes the created image data.

- **Synchronize the CPU with the RCP**

The following function uses the CPU retrace message process to ensure that the CPU and RCP processing are in sync with each other:

```
/* Wait for the retrace*/
gfxWaitSync( );
}
}
```

2. Display List Construction Process

The 3d.c sample code constructs the display list by using four techniques. The following explanation describes the two main techniques (the RSP and RDP drawing modes and the RSP 3D model data (polygon) drawing process).

- **RSP and RDP Drawing Modes**

The following routines set up the RCP drawing mode that actually creates the commands that render the drawing reflected in the display list. The [gSP](#) and [gDP](#) functions are included in the N64 library. For more information, please see the online "[N64 Function Reference Manual](#)" (HTML manual pages).

```
/* Set up lighting information */
static Lights1 lightset = gdSPDefLights1(
    127, 127, 127, /* ambient light color */
    255, 255, 255, /* diffuse light color */
    24, 70, 66); /* diffuse light position */
```

Set the light structure.

```
EXCONTPAD *pad;

static Gfx *setup_SP_DP(Gfx *gp)
{
/* Set all sorts*/
/* Clear the geometry pipeline mode */
gSPClearGeometryMode(gp++, G_SHADE |
    G_SHADING_SMOOTH | G_CULL_BOTH
    | G_FOG | G_LIGHTING | G_TEXTURE_GEN
    | G_TEXTURE_GEN_LINEAR | G_LOD );

/* Set the geometry pipeline mode */
```

```

    gSPSetGeometryMode(gp++,G_SHADE | G_LIGHTING |
        G_SHADING_SMOOTH
        | G_ZBUFFER | G_CULL_BACK);

/* Set the number of lights used */
gSPNumLights(gp++,2);

/* Set the light structure */
gSPSetLights1(gp++,lightset);

/* Set the texturing parameter */
gSPTexture(gp++,0,0,0,0,G_OFF);

/* Set the RDP cycle type */
gDPSetCycleType(gp++,G_CYC_1CYCLE);

/* Set the color combine mode */
gDPSetCombineMode(gp++,G_CC_SHADE,G_CC_SHADE);

/* Set the rendering mode of the blender (within the RDP) */
gDPSetRenderMode(gp++,G_RM_ZB_OPA_SURF,
    G_RM_ZB_OPA_SURF2);

/* Set the compare mode of the alpha value */
gDPSetAlphaCompare(gp++,G_AC_NONE);

/* Set the dithering mode of the color data */
gDPSetColorDither(gp++,G_CD_DISABLE);

/* Set the dithering mode of the alpha value */
gDPSetAlphaDither(gp++,G_AD_NOISE);
return gp;
}

```

- **RSP 3D Model Data (Polygon) Drawing Process**

The following code calls another (second) display list to draw (output) the polygon model. The second display list provides the geometry conversion from the vertex data in that the RSP can turn it into polygon data that the RDP can recognize. The [gSP](#) functions are included in the N64 library. For more information, please see the online "[N64 Function Reference Manual](#)" (HTML manual pages).

```

/* Polygon drawing */
static Gfx *draw_polygon(Gfx *gp)
{

/* Call another (second) display list for polygon drawing */
gSPDisplayList(gp++,n64_model0);
return gp;
}

```

Now the second display list (with the converted data) is inside the display list currently being constructed.

2-7 Drawing Cycle Mode

N64 has four kinds of drawing cycle modes as below. The [gDPSetCycleType function](#) is used for setting the drawing cycle mode.

#Fill mode (FILL)

In FILL mode, the RDP writes pixels into the frame buffer using the color specified in the fill color register. At maximum speed (usually only attained with rectangles), two 32-bit pixels or four 16-bit pixels are written per cycle. In FILL mode, you cannot use polygon drawing functions such as [gSP1Triangle](#).

#Copy mode (COPY)

In COPY mode, the RDP transfers the accepted textured pixels in [TMEM](#) to the frame buffer. At maximum speed (usually only attained with rectangles), two 32-bit pixels or four 16-bit pixels are copied per cycle. In COPY mode, you cannot use polygon drawing functions such as the [gSP1Triangle](#) function. However, you can use texturing functions such as the [gSPTextureRectangle](#) function.

#One-cycle mode (1CYCLE)

In 1CYCLE mode, the RDP fills a maximum of one pixel per clock cycle. In this mode, the RDP can fill a fairly high-quality pixel. Using 1CYCLE mode, you can generate pixels that are perspective corrected, bilinear filtered, modulate/decal textured, transparent, and z-buffered at a maximum bandwidth of one pixel per cycle. This mode uses each process unit in the RDP pipeline (RS, TX, TF, CC, BL, and MI) once.

#Two-cycle mode (2CYCLE)

In 2CYCLE mode, the RDP pipeline's process units are reconfigured for additional functionality at a slower maximum speed of one pixel per two clocks. Each process unit in the RDP pipeline is used twice except for the RS (the rasterizer).

2-8 3D Geometry Calculations

Geometry calculations, which mainly calculate coordinate transformations, play a leading role in the 3D process. These calculations are explained in detail in the STEP 3 [N64 Audio Architecture and Samples]

2-8-1 Matrix

A [matrix](#) (two-dimensional array) forms the core of the coordinate transformation process. It is used to convert from one coordinate system to another, and is specified in this manner:

```
float mf[4][4];
```

This matrix consists of 16 elements (4 lines by 4 columns), so it can hold all the modulation elements of the coordinate translation, rotation, projection, and scaling at the same time. The [geometry](#) calculations that use these matrices would take a long time if they were calculated by the CPU, so when N64 needs to do a geometry calculation, the RCP takes over. The RCP does the actual calculation and processes the coordinate transformation much more quickly than the CPU could. After completing the calculations, the RCP puts the result back into the [display list](#) for use by the CPU.

2-8-2 Coordinate System

The 3D rendering process ultimately loops through the coordinate transformation process several times to process the two-dimensional coordinate system of the frame buffer image. N64 uses the following three coordinate systems:

- Local Coordinate System
 - ↓ A. Model View Matrix
- World Coordinate System
 - ↓ B. Projection Matrix
- Frame Buffer Coordinate System

The coordinate data is converted in order from the beginning. Two transformation matrices are used. The local to world coordinate transformation uses a model view matrix (A), and the world to frame buffer coordinate transformation uses a projection matrix (B).

The following are the contents of the transformation matrices:

1. Model View Matrix

A model view matrix defines where to place the model (the object to be displayed) in the world coordinate system. The matrix storage holds 10 columns of the matrix-[stack](#). Note: The depth of the matrix stack varies depending on which [microcode](#) is being used.

2. Projection Matrix

A projection matrix defines how each object placed in the world coordinate system is actually reflected in the frame buffer. The RCP has a "state" area where it stores the model view matrix and the projection matrix. The RCP is continually updated during the analysis of the display list, and it performs the vertex calculations. As the RCP completes these operations, the CPU constructs the display list.

2-9 Use of Model View Matrix Stack

When you place an object in the 3D world coordinate system, you may find it convenient to keep the relative object-to-object placement information. Using the following diagram, we will cover the placement method of objects.

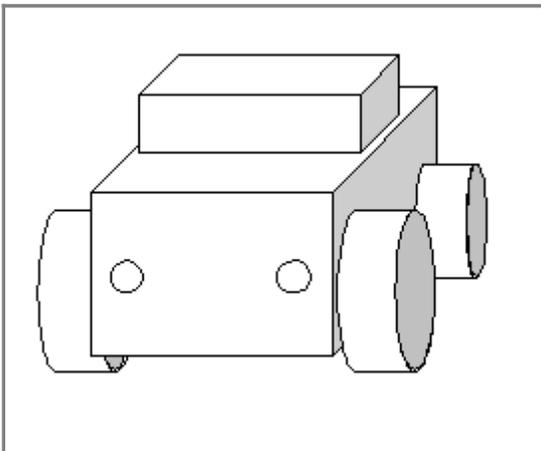


Figure 2-9-1 Example of object placement

1. Place the car in the world coordinate system and store the information about the car's direction of

movement in the model view matrix.

2. Register the car body in the [display list](#).
3. [Push](#) the current model view matrix to the [stack](#).
4. Register the new wheel position in the display list.
5. Return the model view matrix.

After that, repeat operations of 3 through 5, and the other three wheels will be placed and the car will be registered in the display list. Also, if you often display the same objects, leave the calculation to the RCP, so that you can allocate the CPU processing time to other processes. This makes for a more efficient and faster construction of the display list.

Introduction to **NINTENDO⁶⁴**

Chapter 3 N64 Audio Architecture and Samples

This chapter explains about the N64 audio structure.



3-1 N64 Audio Architecture

To provide for audio in your games, the software (game application, N64 operating system, audio library, and audio [microcode](#)) controls each piece of hardware (CPU, RDRAM, RSP, AI, and audio DAC). Waveform synthesis occurs through the cooperative efforts of the CPU and RSP. Ultimately, the AI (audio interface) reads the waveform data in the specified area of RDRAM known as the audio buffer as 16-bit stereo waveform data with a constant time interval. Then the AI sends it to the audio DAC, which outputs the audio signal.

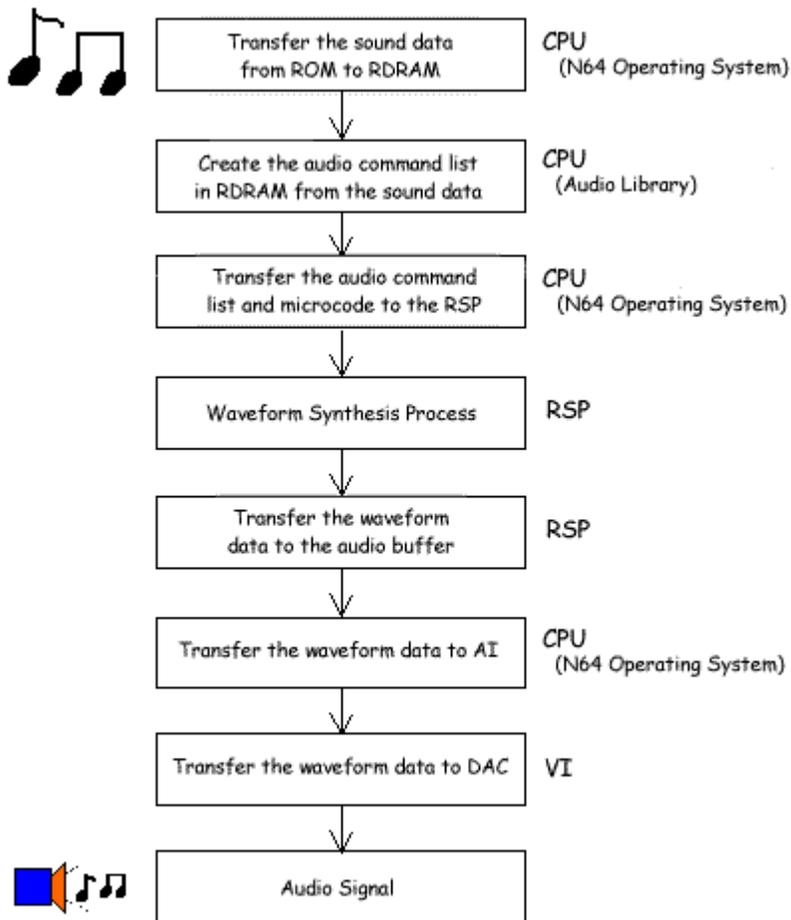


Figure 3-1-1 The flow of the audio process

3-2 Hardware Process

3-2-1 CPU Process

- Construct the audio command list in RDRAM for waveform synthesis.
- Transfer the audio command list and audio [microcode](#) to the RSP.
- Transfer the waveform data (linear PCM) from the RSP to RDRAM to the AI.

3-2-2 RSP Process

- Execute the microcode transferred from RDRAM.
- Accept the audio command list in RDRAM and synthesize the waveform data.
- Transfer the synthesized waveform data to the audio buffer in RDRAM.

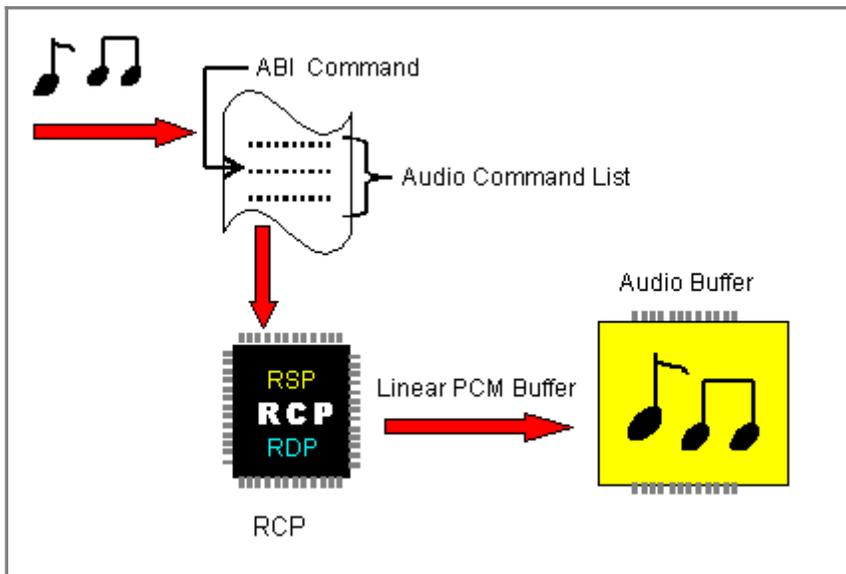


Figure 3-2-1 The audio data process

3-2-3 AI Process

The AI process transfers the waveform data from the audio buffer to the audio DAC (digital-to-analog converter).

3-2-4 Audio DAC Process

In the audio DAC, the digital waveform data is converted to an analog audio signal and output.

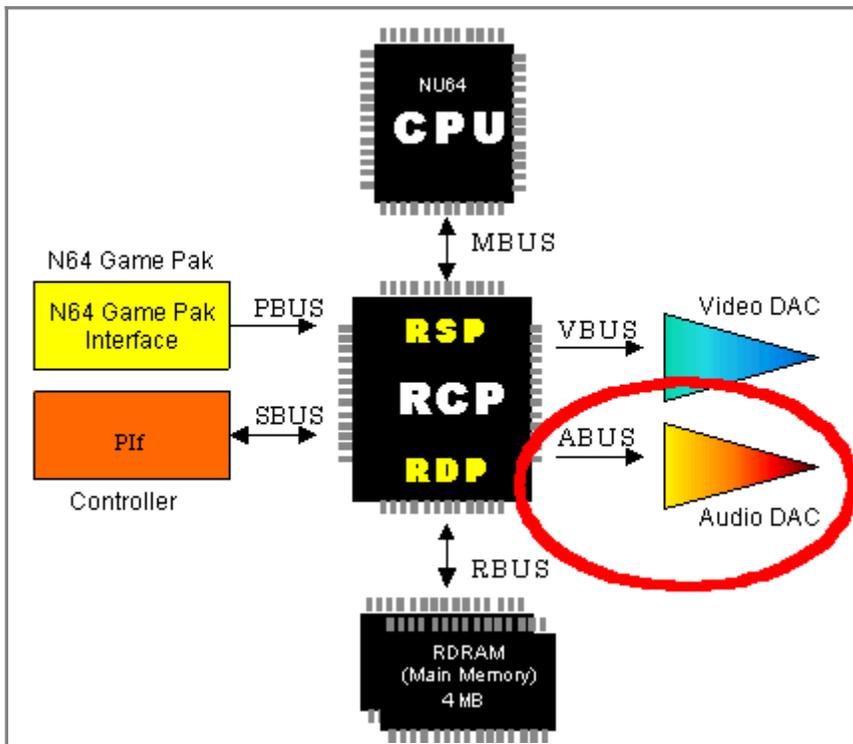


Figure 3-2-2 Audio DAC

3-3 Software Processes

3-3-1 Audio Library

The audio library interprets the sound data and constructs the audio command list.

Library Types

- **Sound Player**
Use the sound player library to interpret [sound effects](#). It can reproduce [ADPCM](#) compression, 16-bit non-compression (linear PCM), looped sound, and so on.
- **Sequence Player**
Use the sequence player to interpret [MIDI](#) files. It executes the allocation of the sequencer, [instrument bank](#), synthesizer [resource](#), and sequence interpretation, and it [schedules](#) the MIDI messages.
- **Synthesizer Driver**
Use the synthesizer driver library to make the audio command list. You can register several players in this driver. Though you will usually register the [sound player](#) or the [sequence player](#), you can register a player of your own making.
- **Audio Microcode**

Use the audio [microcode](#) for waveform synthesis. The data is DMA-transferred from the audio buffer to the RSP. Then the audio microcode operates the RSP as the processor for waveform synthesis.

3-4 Method of the Audio Data Development Process

3-4-1 Audio Data Development Process

To develop audio for your games, use the following processes in combination with sound development tools like N64 Sound Tools for the PC.



- **How to Make Wave Table Audio Data (SGI)**

1. [Sample](#) the voice source data by using any of the many available third-party tools to make the [AIFF](#) format file.
2. Make a [code book](#) (a [predictive coefficient](#) table) used for the [ADPCM](#) encode by using the N64 [tabledesign](#) audio development tool.
3. Compress the AIFF format file to the [AIFC](#) (ADPCM) format by using the N64 [vadpcm_enc](#) audio development tool.
4. Make the source file (.inst file) used by the N64 [ic](#) audio development tool. The ic source file has defined objects that compose the [bank](#) by using a language similar to the C language.
5. Make the [bank files](#) (.tbl, .ctl, and .sym files) by using the N64 ic audio development tool. The bank files are the informational data files used for audio synthesis by the audio library.

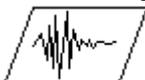


- **How to Make MIDI Sequence Audio Data (SGI)**

1. Make the [MIDI](#) file by using any of the many available third-party tools.
2. If the MIDI file is [Type1](#), convert it to [Type0](#) by using the N64 [midicvt](#) audio development tool.
3. When you generate the [compressed MIDI](#) data generally used for a game program, convert the Type 0 MIDI file to the compact MIDI file by using the N64 [midicomp](#) audio development tool.
4. Make the sequence bank file (.sbk file) by using the N64 audio development tools, [sbc](#).
5. Pass the bank files (.tbl, .ctl, and .sym files) to the programmer.

See each respective tool manual for details on the N64 audio development tools.

3-4-2 Setting the Sampling Rate



- Establish the standard output rate for your game between 22.05kHz and 44.1kHz. With the N64, if the rate is too high, the processing time in the RSP takes longer. If you establish a low rate, the processing time in the RSP is shortened, and you can increase the number of voices.
- With an output rate greater than 32kHz, the human ear notices little difference. A rate less than 22.05kHz produces sampling noise that significantly reduces sound quality.

- Develop the audio data using a sampling rate that is as close as possible to the N64 reproduction output rate to ensure good sound quality. If you cannot decide on the exact output rate in advance, develop the audio using a higher sampling rate first, and then convert it to a lower rate if necessary. This will minimize any decline in sound quality.

3-5 Method of Audio Playback Process

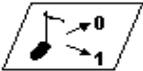
3-5-1 Playing Back Audio Step by Step



The audio playback is provided by the following procedure:

1. Create the audio heap.
2. Set the hardware output frequency.
3. Create a synthesizer.
4. Create a [message queue](#) for receiving the signal that adjusts the timing of the audio process.
5. Create players ([sound player](#), [sequence player](#), etc.) to be added to the synthesizer.
6. Initialize the [resources](#) allocated to each player.
7. Create the audio command list.
8. In RSP, execute the audio task based on the audio command list.
9. Set the audio DAC.

3-5-2 Setting Up the Synthesizer Step by Step



Before playing back the audio, you first need to set the synthesizer by following this procedure:

1. Initialize the audio heap

For each audio library function, N64 allocates required memory dynamically from the memory area called the audio heap. Therefore, to use the audio library, you must first initialize the audio heap area by using the [alHeapInit](#) function.

2. Set the hardware playback rate

To set the hardware [playback rate](#), use the [osAiSetFrequency](#) function.

3. Set up the synthesizer driver configuration structure

Before creating the synthesizer, set the parameters such as maxUpdates, in the synthesizer driver configuration structure.

4. Create the synthesizer

Use the [allnit](#) function to create the synthesizer.

5. Set up the [DMA call back](#) routine

As the occasion demands, to transfer the waveform data in ROM to RDRAM, you must set up the DMA callback routine brought up from the synthesizer driver. To do this, you need to set the pointer to the DMA initialization routine in the synthesizer driver's configuration structure. This DMA initialization routine is brought up once for each [physical voice](#); it initializes the DMA buffer the first time and is set to return the pointer (ALDMAproc) to the function called up when the waveform data actually becomes necessary. ALDMAproc receives the address, length, and state pointer of the required data and returns the pointer to the [buffer](#), storing its data as the return value. Nintendo uses this method in order to give you, the programmer, more freedom. This way, you can customize the algorithm by choosing to externally place either the cache process or the resident process.

3-5-3 Playing Back Sound Effects Step by Step



[Sound effects](#) are played back by using a sound player which is a client of the synthesizer. To playback a sound effect, use this procedure:

1. Accept the audio bank from ROM

To play the sound, you need to first use DMA to transfer the control file of the wave table to RDRAM. The following is a typical procedure you can use to complete the DMA transfer of the audio [bank](#).

- Create the PI Manager by using the [osCreatePiManager](#) function. (In the actual game program, you create the PI Manager in the main program, so there is no need to recreate it here. On the audio side it is only being used.)
- Create the message queue to verify the end of the DMA transfer by using the [osCreateMesgQueue](#) function.
- Reserve the [heap](#) area for the audio bank by using the [alHeapAlloc](#) function.

***The first three steps are required only with the first setting.**

- Before the DMA transfer, provide for the write back of the CPU cache by using the [osWritebackDCache](#) or [osWritebackDCacheAll](#) function.
- Initiate the DMA transfer by using the [osPiStartDma](#) function.
- Verify the end of the DMA transfer by using the [osRecvMesg](#) function.

2. Activate the sound player

Use the [alSndpNew](#) function to activate the sound player. This also automatically registers the client player in the synthesizer driver.

3. Initialize the audio bank

Use the [alBnkfNew](#) function to initialize the audio bank used to playback the sound effect. Here, "initialize", actually means, "convert from the offset to the pointer" and "convert from the offset to the address."

4. Allocate resources to the sound

Allocate the sound to the player by using the [alSndpAllocate](#) function.

5. Choose the sound

Before setting the [pan](#), [volume](#), and so on, you need to choose the sound by using the [alSndpSetSound](#) function.

6. Playback the sound

Use the [alSndpPlay](#) function to playback the sound.

7. Stop the sound

Use the [alSndpStop](#) function to stop the sound.

8. Delete the sound player

If you no longer need the sound player, delete the client from the synthesizer driver by using the [alSndpDelete](#) function.

3-5-4 Playing Back Sequences Step by Step



To playback MIDI sequences, you need to use a sequence player which is a client of the synthesizer. Use the audio library, and follow these steps:

1. Accept the audio bank from ROM

This is the same method as for the playback of sound effects.

2. Accept the MIDI sequence from ROM

Before playing back a [MIDI](#) sequence, you need to use DMA to transfer the sequence from ROM to RDRAM, which you can do by using the following typical procedure: (In addition, the PI Manager and message queue for the DMA transfer should be already created.)

- Read the first four bytes of the sbk file header. The first two bytes hold the version and the next two bytes hold the number of the sequence. To do this step, you need to first reserve the header area by using the [alHeapAlloc](#) function. Then use DMA to transfer the first four bytes.
- Use DMA to transfer the entire header including the ALSeqData structure which holds the sequence number.
- Initialize the sequence [bank file](#) by using the alSeqFileNew function.
- Use DMA to transfer the required MIDI sequence information stored in ALSeqData structure.

Activate the sequence player

Use the [alSeqNew](#) function (for a [Type0](#) MIDI file) or the [alCSPNew](#) function (for a [compressed MIDI](#) file) to activate the appropriate sequence player.

4. Initialize the sequence structure

The sequence structure stores the sequence data information needed to playback the MIDI sequence. Use the [alSeqNew](#) function (for a Type 0 MIDI file) or the [alCSeqNew](#) function (for a compressed MIDI file) to initialize the structure.

5. Set up the sequence in the player

To set up the sequence in the player, use the [alSeqpSetSeq](#) function (for a Type 0 MIDI file) or the [alCSPSetSeq](#) function (for a compressed MIDI file).

6. Initialize the audio bank

This method is the same as the one for the playback of sound effects.

7. Set up the audio bank

Specify the [instrument](#) bank to be used by the sequence player by using the [alSeqpSetBank](#) function (for a Type 0 MIDI file) or the [alCSPSetBank](#) function (for a compressed MIDI file).

8. Play the sequence

To initiate the sequence playback, use the [alSeqpPlay](#) function (for a Type 0 MIDI file) or the [alCSPPlay](#) function (for a compressed MIDI file).

9. Stop playing the sequence

To stop playing the sequence, use the [alSeqpStop](#) function (for a Type 0 MIDI file) or the [alCSPStop](#) function (for a compressed MIDI file).

10. Delete the sequence player

When a sequence player becomes unnecessary, delete it as a client of the synthesizer driver by using the [alSeqpDelete](#) function (for a Type 0 MIDI file) or the [alCSPDelete](#) function (for a compressed MIDI file).

3-5-5 Executing Audio Tasks Step by Step



1. Ensure that the audio buffers are ready

An audio process generally reserves three audio buffer areas in RDRAM. It also reserves two buffers for the audio command list so that one buffer is always ready to be played. This double buffering helps to prevent sound from pausing by never having to wait for an audio process.

2. Synchronize the retrace and RSP events

To execute an audio task, N64 game programmers usually prepare two message queues -- one to post the VI retrace, and the other to post the completion of the RSP task. Set up the former by using the [osViSetEvent](#) function, and set up the latter by using the [osSetEventMesg](#) function. Store the [event](#) type in OS_EVENT_SP. The VI retrace event is essential to ensure that each audio process is executed for each frame.

3. Regulation of the sample to ensure compatibility with the frame size

To avoid clicking sounds in the music, you need to regulate the processing sample number to ensure that it is compatible with the situation. To do this, use IO_READ (AL_LEN_REG) or the [osAiGetLength](#) function. This detects the sample number remaining in the audio buffer without using it, and it regulates the sample so that the newly processing sample number is compatible with the available space and can have a little more room. Because the sample number having room or not varies depending on the program or sound, you need to regulate it by continually calculating the actual amount remaining.

4. Create the audio command list

To create the audio command list, use the [alAudioFrame](#) function.

5. Execute the audio task

To have the RSP execute the audio task based on the audio command list and create the waveform data in the audio buffer, use the [osSpTaskStart](#) function.

6. Set up the audio DAC

Use DMA to transfer the audio to the audio DAC by using the [osAiSetNextBuffer](#) function.

3-6 Audio Playback Samples

The following code demonstrates how to play a MIDI sequence in your game. Another audio sample on the CD demonstrates how to play [sound effects](#). See [Appendix C Sample Description] about the sample program. [os-fuctions](#) and [al-functions](#) are included in libraries. You can look them up in the "N64 Function Reference Manual" for more information. All other functions are defined for this sample only; they are not part of the os or al libraries.

3-6-1 Sequence Playback Sample Code



```
/*-----  
The main process  
-----*/  
void mainproc(void* arg)  
{  
    /* Initialize each type */  
    InitMain( );  
  
    /* The main loop */  
    while(1){
```

```

switch(main_no){
  case GM_MAIN:
    main_no = main00(&sched);
    break;
}
}
}

/*-----
Initialize each type
-----*/
void InitMain(void)
{
  /* Initialize the audio player */
  auAudioInit( );

  /* Initialize the sequence player */
  auSeqPlayerInit( _midibankSegmentRomStart,
    _midibankSegmentRomEnd,
    _seqSegmentRomStart,
    _seqSegmentRomEnd,
    _miditableSegmentRomStart);

  /* Create the scheduler thread */
  nnScCreateScheduler(&sched, OS_VI_NTSC_LAN1,1);

  /* Create the audio thread */
  auCreateAudioThread(&sched);

  main_no = GM_MAIN;
}

/*-----
Allocate each buffer and initialize the
parameter used in the audio library
-----*/
void auAudioInit(void)
{
  /*initialize audio heap */
  alHeapInit(&audio_heap,audio_heap_buf,AUDIO_HEAP_SIZE);

  /* create command list buffer */
  audio_cmdlist_ptr[0] = alHeapAlloc(&audio_heap,1,
  AUDIO_CLIST_SIZE_MAX*sizeof(Acmd));
  audio_cmdlist_ptr[1] = alHeapAlloc(&audio_heap,1,
  AUDIO_CLIST_SIZE_MAX*sizeof(Acmd));

  /* create task list buffer */
  audio_tlist_ptr[0] = alHeapAlloc(&audio_heap,1,
  sizeof(OSTask));
  audio_tlist_ptr[1] = alHeapAlloc(&audio_heap,1,
  sizeof(OSTask));

  /* create audio buffer */
  audio_buffer_ptr[0] = alHeapAlloc(&audio_heap,1,
  sizeof(s32)*AUDIO_BUFFER_MAX);
  audio_buffer_ptr[1] = alHeapAlloc(&audio_heap,1,
  sizeof(s32)*AUDIO_BUFFER_MAX);
  audio_buffer_ptr[2] = alHeapAlloc(&audio_heap,1,
  sizeof(s32)*AUDIO_BUFFER_MAX);
}

```

```

/* create message queue for DMA */
osCreateMesgQueue(&audioDmaMessageQ,
&audioDmaMessageBuf,1);
osCreateMesgQueue(&audioRomMessageQ,
&audioRomMessageBuf, 1);

/* Clear initialization flag of DMABuffer */
dmaState.initialized = 0;

/* initialize audio library */
audio_config.outputRate=
osAiSetFrequency(AUDIO_OUTPUT_RATE);
audio_config.maxVVoices =
AUDIO_VVOICE_MAX; /* max virtual voices */
audio_config.maxPVoices =
AUDIO_PVOICE_MAX; /* max physical voices */
audio_config.maxUpdates = AUDIO_UPDATE_MAX;
audio_config.dmaproc = &dmaNew; /* DMA callback function */
audio_config.fxType = AL_FX_SMALLROOM; /* effect type */
audio_config.heap = &audio_heap; /* audio heap */
audio_config.params = 0; /* custom effect */

/* Create the synthesizer (initialize the audio library) */
alInit(&audio_global,
&audio_config);
}

/*-----
Initialize the sequence player
input
midi_start: The header ROM address of the bank file(.ctl)
midi_end: THE end ROM address of the bank file(.ctl)
seqheader_start:The header ROM address of
the sequence file (.sbk)
seqheader_end:The end ROM address of the
sequence file (.sbl)
midi_table_star:The header ROM address of
the wave table file (.tbl)
-----*/
void auSeqPlayerInit(u8* midi_start, u8* midi_end,
u8* seqheader_start, u8* seqheader_end,
u8* midi_table_start)
{
ALBank *midiBank_ptr;
u32 size;

/* create sequence data buffer */
seqplayer[0].seqdata_ptr =
alHeapAlloc(&audio_heap,
1,AUDIO_SEQDATA_SIZE_MAX);
seqplayer[1].seqdata_ptr =
alHeapAlloc(&audio_heap,
1,AUDIO_SEQDATA_SIZE_MAX);

/* initialize sequence player 0 */
seqplayer[0].seqconfig.maxVoices =
AUDIO_SEQ_VVOICE_MAX; /* max virtual voices */
seqplayer[0].seqconfig.maxEvents =
AUDIO_SEQ_EVTCOUNT_MAX; /* max inside events */

```

```

seqplayer[0].seqconfig.maxChannels =
    AUDIO_SEQ_CHANNEL_MAX; /* max MIDI channels */
seqplayer[0].seqconfig.heap = &audio_heap; /* audio heap */
seqplayer[0].seqconfig.initOsc = 0;
seqplayer[0].seqconfig.updateOsc = 0;
seqplayer[0].seqconfig.stopOsc = 0;
seqplayer[0].seqconfig.debugFlags = 0;

#ifdef _AUDIO_COMPACTMIDI_
    alCSPNew(&seqplayer[0].seqplayer,
&seqplayer[0].seqconfig);
#else
    alSeqpNew(&seqplayer[0].seqplayer, &seqplayer[0].seqconfig);
#endif /* _AUDIO_COMPACTMIDI_ */

/* initialize sequence player 1 */
seqplayer[1].seqconfig.maxVoices =
    AUDIO_SEQ_VVOICE_MAX; /* max virtual voices */
seqplayer[1].seqconfig.maxEvents =
    AUDIO_SEQ_EVTCOUNT_MAX; /* max inside events */
seqplayer[1].seqconfig.maxChannels =
    AUDIO_SEQ_CHANNEL_MAX; /* max MIDI channels */
seqplayer[1].seqconfig.heap = &audio_heap; /* audio heap */
seqplayer[1].seqconfig.initOsc = 0;
seqplayer[1].seqconfig.updateOsc = 0;
seqplayer[1].seqconfig.stopOsc = 0;
seqplayer[1].seqconfig.debugFlags = 0;

#ifdef _AUDIO_COMPACTMIDI_
    alCSPNew(&seqplayer[1].seqplayer, &seqplayer[1].seqconfig);
#else
    alSeqpNew(&seqplayer[1].seqplayer, &seqplayer[1].seqconfig);
#endif /* _AUDIO_COMPACTMIDI_ */

/* read seqfileheader data */
auReadSeqFileHeader((u32)seqheader_start);
alSeqFileNew(seqHeaderfile_ptr,
seqheader_start);

/* read midi bank data */
size = (u32)midi_end-(u32)midi_start;
midi_buffer_ptr = alHeapAlloc(&audio_heap, 1,
size);
auRomRead((u32)midi_start, midi_buffer_ptr, size);

/* Specify the instrument bank
used by the sequencer */

alBnkfNew(midi_buffer_ptr,
midi_table_start);
midiBank_ptr = midi_buffer_ptr->bankArray[0];

#ifdef _AUDIO_COMPACTMIDI_
    alCSPSetBank(&seqplayer[0].seqplayer, midiBank_ptr);
    alCSPSetBank(&seqplayer[1].seqplayer, midiBank_ptr);
#else
    alSeqpSetBank(&seqplayer[0].seqplayer, midiBank_ptr);
    alSeqpSetBank(&seqplayer[1].seqplayer, midiBank_ptr);
#endif /* _AUDIO_COMPACTMIDI_ */

```

```

}

/*-----
Create and activate the audio thread
-----*/
void auCreateAudioThread(NNSched *sched)
{
    osCreateThread(&audioThread,
AUDIO_THREAD_ID,audioProc,
    (void *)sched,
    (audioThreadStack+AUDIO_STACKSIZE/sizeof(u64)),
    AUDIO_THREAD_PRI);
    osStartThread(&audioThread);
}

/*-----
Main Sequence Playback
-----*/
int main00 (NNSched* sched)
{
    OSMesgQueue msgQ;
    OSMesg msgbuf[MAX_MESGS];
    NNScClient client;
    u32 seqno;

    /* create message queue for VI retrace */
    osCreateMesgQueue(&msgQ,
msgbuf, MAX_MESGS);

    /* create message queue for VI retrace */
    auSeqPlayerSetFile(0,2);

    /* add client to shceduler */
    nnScAddClient(sched, &client, &msgQ);

    seqno = 0;
    auSeqPlayerPlay(seqno);
    while(1){
        (void) osRecvMesg(&msgQ,
NULL, OS_MESG_BLOCK);
        /* If the sequence has stopped, repeat it */
        if(auSeqPlayerState(0) == AL_STOPPED){
            auSeqPlayerPlay(seqno);
        }
    }

    /* exit label */
    return MAIN_01;
}

/*-----
Playback the sequence
Input:
    seqplayer_no:
The sequence player number that
starts the reproduction.
If the sequence player is not in AL_STOPPED
state,
return FALSE.
-----*/

```

```

int auSeqPlayerPlay(u32 seqplayer_no)
{
#ifdef _AUDIO_COMPACTMIDI_
    s32 seqdata_len;
    u32 seqno;
#endif /* _AUDIO_COMPACTMIDI_ */

#ifdef _AUDIO_DEBUG_
    if(seqplayer_no > 1){
        osSyncPrintf("seqplayer_no
over!!\n");
        return FALSE;
    }
#endif /* _AUDIO_DEBUG_ */

    /* check seqplayer state. */
    if(seqplayer[seqplayer_no].seqplayer.state!= AL_STOPPED){
#ifdef _AUDIO_DEBUG_
        osSyncPrintf("sequence
player %d is playing\n", seqplayer_no);
#endif /* _AUDIO_DEBUG_ */
        return FALSE;
    }

#ifdef _AUDIO_COMPACTMIDI_
    alCSeqNew(&seqplayer[seqplayer_no].sequence,
    seqplayer[seqplayer_no].seqdata_ptr);
    alCSPSetSeq(&seqplayer[seqplayer_no].seqplayer,
    &seqplayer[seqplayer_no].sequence);
    alCSPPlay(&seqplayer[seqplayer_no].seqplayer);
#else
    seqno = seqplayer[seqplayer_no].seqno;
    seqdata_len = seqHeaderfile_ptr->seqArray[seqno].len;
    alSeqNew(&seqplayer[seqplayer_no].sequence,
    seqplayer[seqplayer_no].seqdata_ptr, seqdata_len);
    alSeqpSetSeq(&seqplayer[seqplayer_no].seqplayer,
    &seqplayer[seqplayer_no].sequence);
    alSeqpPlay(&seqplayer[seqplayer_no].seqplayer);
#endif /* _AUDIO_COMPACTMIDI_ */
    return TRUE;
}

```

Introduction to **NINTENDO⁶⁴**

Chapter 4 Tips and Techniques

This chapter contains extracts of useful techniques that developers sent in at the time of creation, in answer to questions and also to provide a higher level of efficiency.



4-1 Coordinating Audio and Graphics

The CPU, RSP, and RDP processors each have a leading role in the N64 when it comes to executing audio and graphics. The CPU starts the task, and the RSP and RDP actually execute it. These hardware processors take partial control of the work and can enhance processing efficiency dramatically by making processes operate in parallel. For efficient parallel processing, it is best to create a scheduler [thread](#) to manage and coordinate the execution of the audio and graphics threads.

4-1-1 Thread Management with a Scheduler Thread



The audio command list and graphics display list are usually created just before the frame, but a problem occurs if the process does not end within one frame (1/60 second). In the case of audio, the sound may pause inappropriately or popping noises may occur. In the case of graphics, because the frame buffer is not updated, it continues to show the last image.

If the audio process does not end within one frame, you can avoid inappropriate pauses and noises by giving priority to the audio process even if you have to interrupt the graphics process. Therefore, give audio threads a higher priority than graphics threads, and manage the thread by using a scheduler. A scheduler is simply a thread that has a higher priority than both the audio and the graphics threads.

A program about the coexistence of audio and graphics is provided: see [[Appendix C Sample Description](#)] about the sample program.

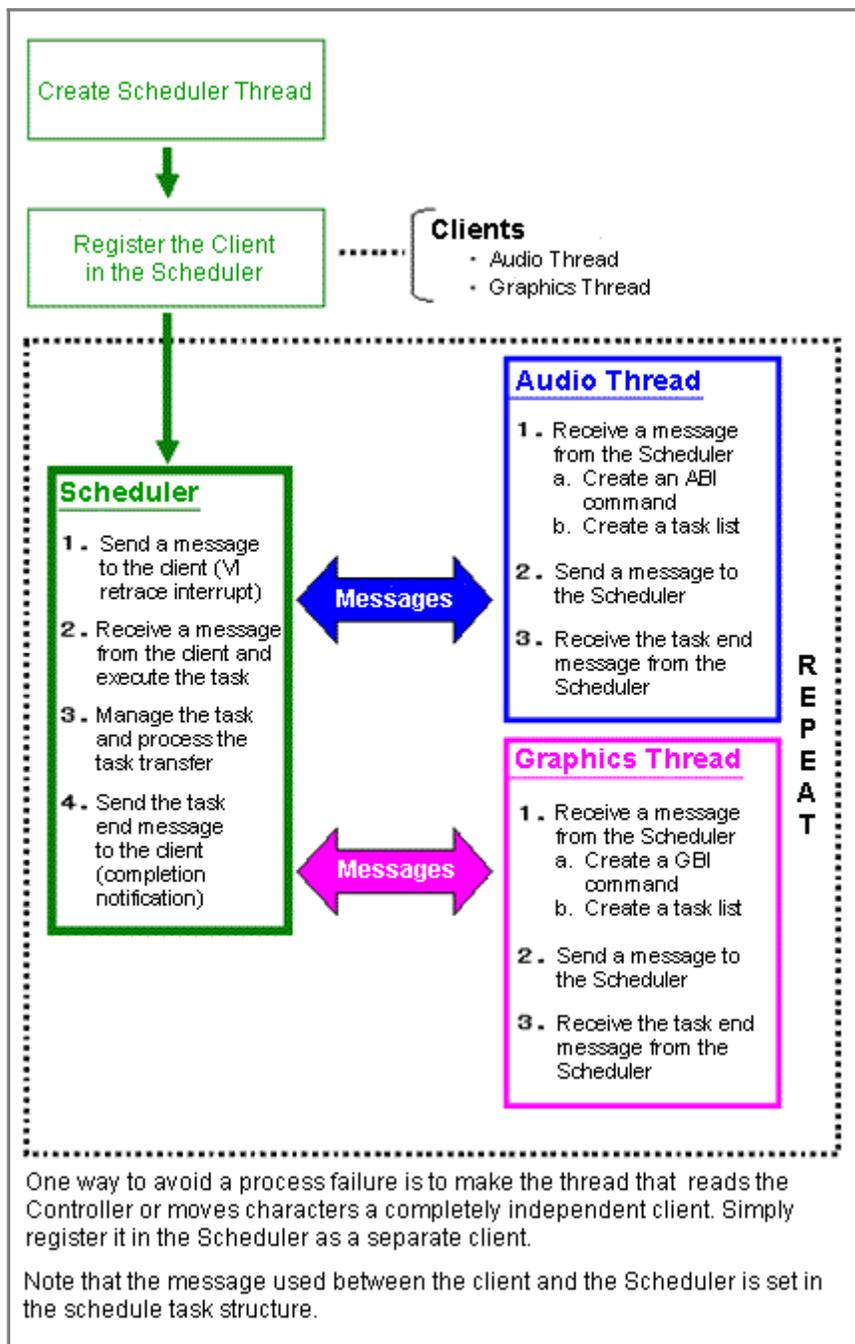


Figure 4-1-1 Scheduler Process Flow



Figure 4-1-2 Thread Priority

4-2 Tuning Performance

4-2-1 Double Buffering the Display List

Usually only one [buffer](#) is used to hold the [display list](#) that is constructed by the CPU and executed by the RCP. However, if you use two display list buffers instead of one, the RCP can execute the drawing process for one frame at the same time as the CPU is constructing the next frame's display list. This technique is called "double buffering." Use this technique to speed up your game's graphics if the graphics are complex and you have plenty of memory.

-A display list is a graphics command list that holds the commands necessary to render one frame of graphics. The CPU constructs the display list and then passes it to the RCP to execute it.

The longer it takes the CPU to construct each display list, the more speed you can add by using the double-buffering technique. However, remember that although double-buffering makes it appear as though the display list construction time is zero, it really is not. The CPU still has to construct each display list and that means the CPU is not available for other processes that might require it. Therefore, you should devise an efficient algorithm to minimize the processing time needed to construct the display list. This can ultimately lead to faster overall game processing. A weak point of this method is that double buffering requires twice as much memory to make the two buffers, so if you are short on memory, this technique may not work for you.

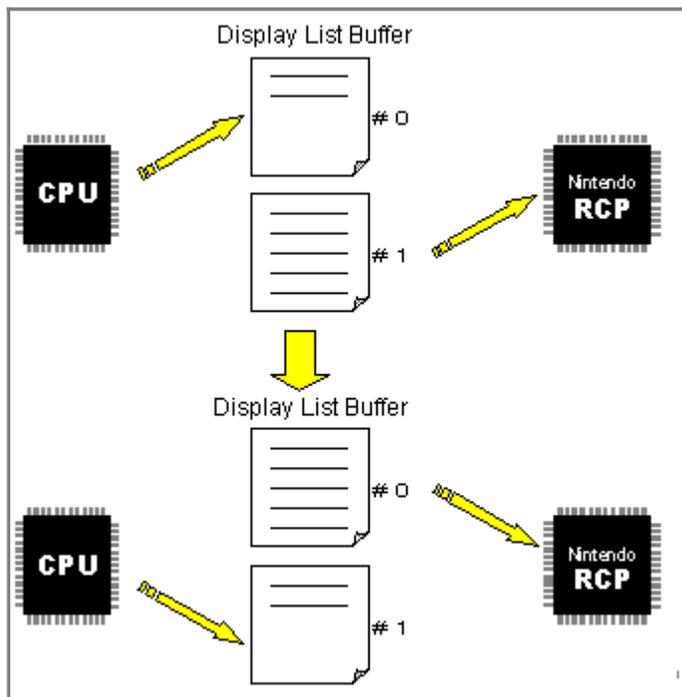


Figure 4-2-1 Double Buffering the Display List

4-2-2 Triple Frame Buffering

Usually, you use two frame buffers (double buffering) as explained in Section [2-4 Use of Frame Buffer](#). As the RCP is drawing the next frame into one buffer, the video DAC is displaying the previously drawn frame. However, switching between the two frame buffers occurs only at the vertical synchronization point. Therefore, if the RCP hasn't finished drawing a new frame when the next vertical synchronization occurs, you won't be able to use the buffers together for the next frame. In cases like this where it takes longer to draw each frame, you can make the RCP more efficient by using triple frame buffering. One buffer for "displaying," one for "drawing completed and waiting for switch", and one for "drawing".

The speed-up effect of this method can be huge when the drawing time of a frame is frequently out of sync with the vertical synchronization timing, because the waiting time without triple buffering is long. On the other hand, if the drawing time of a frame is usually in sync with the vertical synchronization, this method has little value because each drawn frame has little waiting time. You need to weigh the advantages against the disadvantages. Triple buffering uses a lot of memory because each of the three frame buffers are quite large even in low resolution. Also, there is another disadvantage in that the TV display is always two frames behind instead of just one.

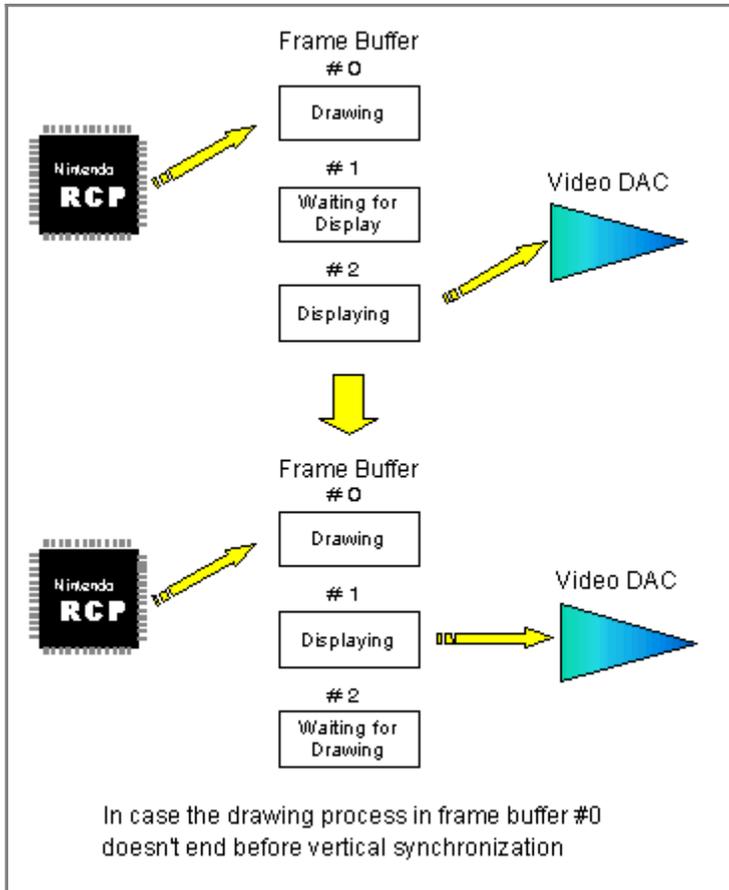


Figure 4-2-2 Triple Frame Buffering

4-2-3 Using LOD (Level Of Detail)

LOD means the level of detail. By providing different levels of detail, you can significantly improve performance. For example, objects that are viewed as fast moving or far away need much less detail than do stationary objects that are close. When you display a lot of objects on the screen, the RCP processing time increases. The RCP processing time is determined by the time it takes to do vertex coordinate transformations, lighting, and so on done by the RSP microcode and the polygontexturization process of the RDP. When you increase the number of objects to be displayed, the processing time for the vertex coordinate transformation or lighting is sometimes going to be a problem.

When displaying 3D objects that are close, you need to provide a lot of precise detail. On the other hand, when an object is small and far away, you can provide very little detail. Therefore you can prepare in advance, several versions of a model, each with a varying level of detail. Then switch the display model based on the distance of the model from the viewer. This very effectively reduces processing time. The disadvantages of this technique are that it takes a long time to prepare several LOD versions of each model in order to make the switching appear natural.

Also, you need to use memory to store all those models. However, because models that have little detail use very little room, the impact on memory is not too bad.

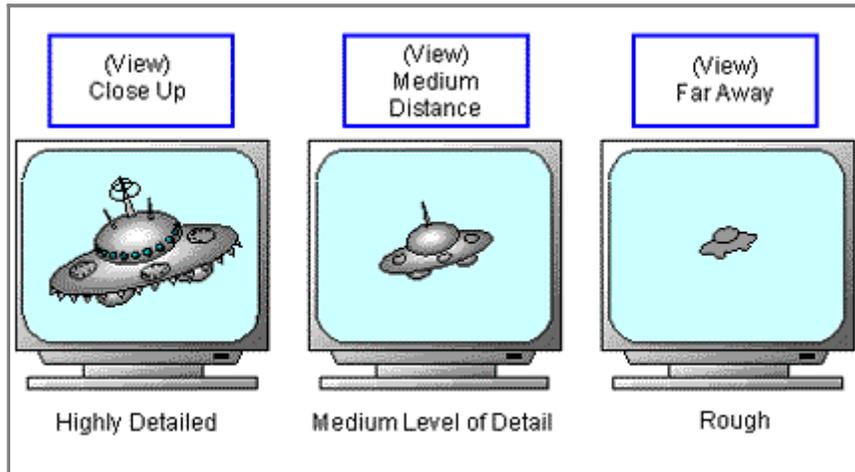


Figure 4-2-3 Images of the three levels of LOD

When a model reaches a certain distance from the viewer, you can greatly improve performance without affecting quality by not showing the model in 3D. Just make it one piece of a picture pasted as pre-rendered image data. This makes it possible to produce good resolution at a fast pace.

This method is most effective when the processing capability of the RSP microcode is saturated, but it has no effect when the RDP performance is saturated.

4-2-4 Volume Culling

If the RCP simply displayed all objects on file, it would waste a lot of time processing coordinate transformations for vertices and models that lie outside the current view. To speed up processing, do not process data that is not displayed on the screen. [Volume culling](#) simply means removing those commands from the display list that apply to vertices or models that lie outside the current view. See the [gSPCullDisplayList](#) function for details. Of course, it is most effective not to send unused drawing instructions for things outside of the view in the CPU process. Volume culling is very effective when there are many models or vertices and the processing capability of the RSP microcode is saturated, but it has no effect when the RDP performance is saturated.

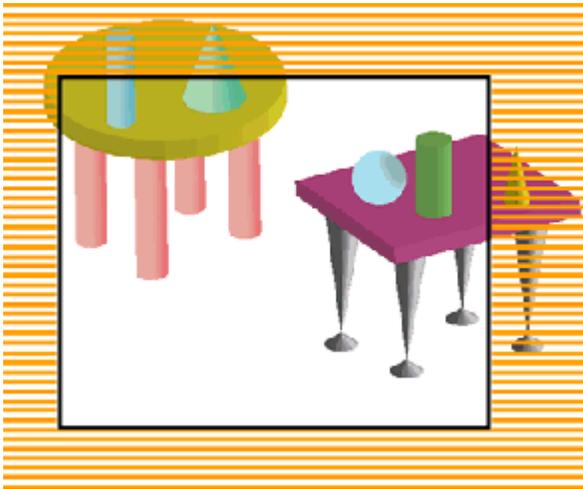


Figure 4-2-4 Do not process data that is not displayed on the screen.

4-2-5 Anti-Aliasing

Anti-aliasing is one of the strongest features of the N64. It smoothes out the jagged edges on lines. However, anti-aliasing takes time. It reduces the pixel fill-rate performance of the RDP because the anti-aliasing process needs to update the [coverage value](#) of each pixel, read the frame buffer, and write the update. Therefore, memory access amount of the frame buffer increases by a factor of two. When the RDP fill-rate performance is saturated, you should **turn off anti-aliasing**. You need to consider the trade off and determine which is most important; the image quality or the drawing speed.

4-2-6 Z-Buffering

When you use the [Z-buffer](#), **draw the closest object first** and then move into the background to get the best speed. If you draw the farthest object first, you have to repeatedly write the entire Z-buffer. Drawing the closest object first is faster because for subsequent objects, you need only write that part of the Z-buffer that is not "covered up" by the foreground object. This is a very effective technique when the RDP fill-rate performance is saturated. See the Z-buffer of STEP 1 [[1-7 Basic Terminology \(Thread, Message, etc.\)](#)]

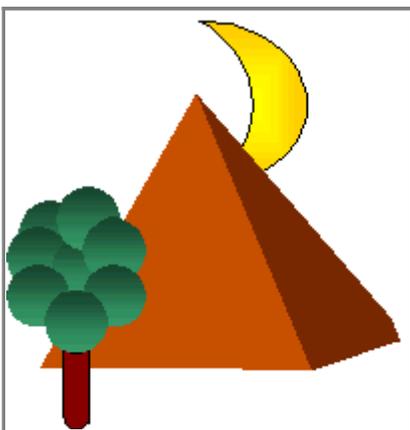


Figure 4-2-5 Draw from the closest picture in the Z-buffer

4-2-7 Optimizing GBI Commands

-The [gSPVertex](#) function loads vertex information. It uses vectorization so it can provide coordinate transformations for two vertexes at a time. Therefore, it is better to load an even number of vertices rather than an odd number so that no display list processing time is wasted.

-If you use the [gSPModifyVertex](#) function, you can directly load values into a previously loaded [vertex cache](#). When a vertex cannot be shared because it has the same xyz coordinates but a different texture coordinate, you can use this function and load only the new texture coordinate, thus optimizing the display list. Additionally, this command directly embeds the value into a 64-bit GBI command, so it operates at a very high-speed. However, you need to provide the multiplication value for the scaling value set up by the [gSPTexture](#) function; it is needed for the texture coordinate.

-After you load a modeling matrix or projection matrix, the MP matrix is recalculated. For example, if you give matrices of scaling, rotation, and translation to the RSP separately and multiply them, you will cause an unnecessary recalculation of the MP matrix internally. Try to **complete this with a single matrix multiplication** if possible. For example, if you know you are going to reuse a [matrix](#) multiplication, you can improve performance by doing it once in advance and then provide the result as a constant in all the places it is needed.

-Lighting can have a significant effect on the performance of the vertex coordinate transformation process. Try to do as much as possible with **lighting off**. For example, when using objects that are fixed in the field and that don't operate the light, turn the lighting off. Also, if you create a texture that appears lighted, the lighting process can be omitted. However, if you have too many textures, the performance will be sacrificed. In addition, do volume culling with lighting off. Obviously, culled vertices don't need lighting. Depending on the microcode process, use of the [gSPVertex](#) function can double the performance.

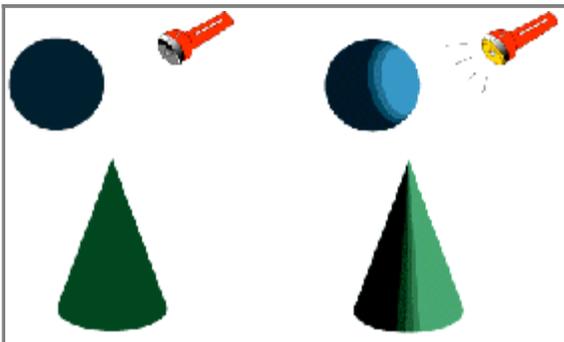


Figure 4-2-6 Lighting

-In graphics microcode, the command that RSP processes most is not the [gSPVertex](#) function, but the [gSP1Triangle](#) function. This is because to require the RDP to draw triangles, it must set up the command with close to 180 bytes and send it. This causes a data sending delay to the RDP. In most microcode, the buffer memory allocates 1K byte from DMEM inside of the RSP, but this becomes full with a drawing command of six triangles. If these triangles are too small, it is not a problem because the RDP completes the drawing process at once and they are removed one after another. However, if the triangles are big, this command process is slow and the RSP has to wait for the output. If you use the **.fifo microcode** or **.dram microcode** for this, the process may speed up.

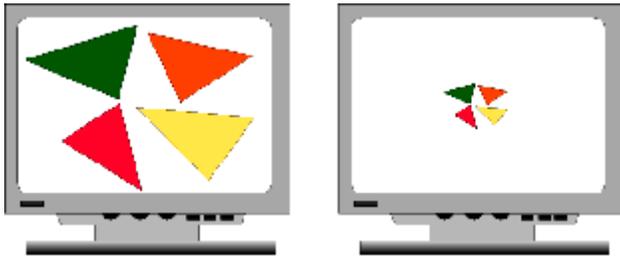


Figure 4-2-7 Drawing Triangles

-If you do not use the texture, one of the triangle commands to the RDP becomes unnecessary and the length of the RDP command becomes 64 bytes short. Therefore, you should avoid giving color to a monochromatic object by using texture. Instead, turn the texture off, and **add color by applying the primitive color or the vertex color**. The vertex color cannot directly add color so if you need lighting, use the primitive color or switch lights. However, because the RSP processing time takes longer to switch lights, it is more effective to use the primitive color. The RDP drawing time does not change, therefore, this method is effective only when you have many vertices and a problem with the RSP processing time, not the drawing time.

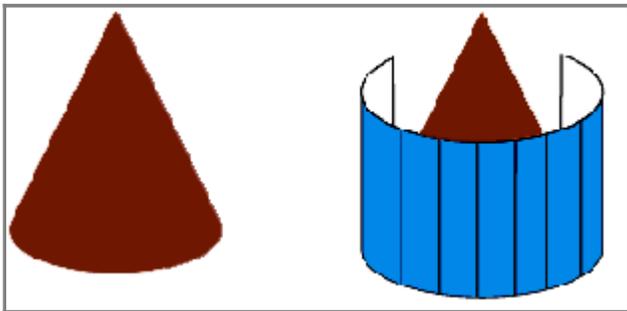


Figure 4-2-8 Setting color with the texture

4-2-8 Optimizing the Display List

Commands to the RSP microcode are provided by the display list. The processing rate of the microcode varies depending on the way the display list is constructed. The key here is to reuse the vertex cache effectively, acquiring the vertex data as needed. It is particularly important to optimize the display list when the game application creates model data dynamically.

4-2-9 Speeding up the Audio Process

Audio processing time must never be ignored. The audio process needs both CPU and RSP processing time.

To decrease the audio processing time, pay attention to the following:

-Lower the sampling frequency of the audio playback as low as sound quality permits.

The lower the sampling frequency, the fewer the number of samples that must be processed per unit time. The direct effect is a faster audio process. Refer to the [osAiSetFrequency](#) function.

-Decrease the number of physical voices (the number of simultaneous pronunciations) given to the synthesizer driver to as few as possible.

Because the time for the waveform synthesis process in the CPU and RSP is almost proportional to the number of [physical voices](#), decreasing the number of physical voices is effective in speeding up the process. You can change the number of physical voices with each scene. Therefore, it may be most effective to use the most fantastic music that uses a lot of physical voices at the beginning or end of the game, thus decreasing the number of physical voices during the game when processing time is most crucial.

-Minimize the resampling (pitch shift) process or the [ADPCM](#) process as it takes a relatively long time.

Try to record the sound using a [sampling rate](#) as close to the playback rate as possible, so that you can avoid the resampling process. Or, when the N64 Game Pak ROM capacity can afford it, try to use raw voice source data that is not ADPCM compressed. This improves RSP processing time. However, in actuality, a resampling process is likely to be necessary because, for example, the numerical value becomes 32006Hz even if you set AI to 32000Hz.

-Minimize sound [effects](#); these take a long time to process.

The more you use the effect primitive, the more it takes time to process. Try to keep the number of effect primitives to an absolute minimum.

4-3 Using Microcode

Currently, N64 has two types of [microcode](#), graphics microcode and audio microcode.

4-3-1 Graphics Microcode

The commands used in the graphics microcode are called [GBI](#) (Graphics binary interface) commands. Each GBI command uses 64 bits.

(gspF3DEX Series)

- gspF3DEX.fifo.o/gspF3DEX.NoN.fifo.o:

spF3DEX.fifo.o and gspF3DEX.NoN.fifo.o both have an increased [vertex cache](#) of 32 based on the Fast 3D microcode and have packaged the 2Triangles command. Also, the number of links in the [display list](#) has increased from 10 to 18.

gspF3DEX.fifo.o : with near clip
gspF3DEX.NoN.fifo.o : without near clip

- gspF3DLX.fifo.o/gspF3DLX.NoN.fifo.o:

spF3DLX.fifo.o and gspF3DLX.NoN.fifo.o both have improved performance. This was accomplished by simplifying the [sub-pixel](#) calculation on F3DEX, so sometimes the texture has little creases. The GBI in F3DLX is compatible with F3DEX. Also, F3DLX has an additional feature in that it can turn [clipping](#) on or off. When clipping is off, performance improves a little.

gspF3DLX.fifo.o : with near clip
gspF3DLX.NoN.fifo.o : without near clip

- gspF3DLX.Rej.fifo.o/gspF3DLP.Rej.fifo.o:

gspF3DLX.Rej.fifo.o and gspF3DLP.Rej.fifo.o both omit the usual microcode clipping process and use the "reject" process instead. The reject process simply means that the microcode **[draws only triangles whose three vertices are within a fixed area inside the screen. If one of the vertices lies outside the screen, the entire triangle is rejected (not drawn)]**. In the F3DLX.Rej and F3DLP.Rej microcode, the processing rate for the 2Triangles command has been significantly improved by adopting this reject process. Therefore, you should use the [gSP2Triangles](#) function as often as possible when creating the display list.

gspF3DLX.Rej.fifo.o : with texture correction
gspF3DLP.Rej.fifo.o : without texture correction

Note that the F3DLP.Rej microcode is slightly faster than the F3DLX.Rej microcode. However, this faster speed applies only to the RSP process. There is no effect on the RDP process. Therefore, the RDP process may not follow the RSP process, in which case, you may need to change the render mode to the RA mode (G_RM_RA_ZB_OPA_SURF). Also note that this microcode does not support G_CULL_BOTH or G_CULL_FRONT.

- gspL3DEX.fifo.o:

gspL3DEX.fifo.o is a modified version of the Line3D microcode adapted to a vertex cache size of 32. It is the same as gspLine3D and displays polygons (such as 1Triangle and 2Triangles) rendered by F3DEX series of microcode to be displayed using wireframe.

- **GBI Compatibility**

At the binary level, there is no compatibility between the GBI of the gspF3DEX series of microcode and the GBI of the usual Fast3D microcode. However, the gspF3DEX series has been designed so that the differences are absorbed by gbi.h. Even if both types of microcode are in the same display list, a compile option can distribute each command to its proper GBI. Specifically, you can ensure that the GBI corresponding to the gspF3DEX microcode is output by defining the "F3DEX_GBI" keyword before including the gbi.h file.

After Release 0.96, all microcode of F3DEX, F3DLX, F3DLX.Rej, and F3DLP.Rej became compatible at the GBI binary level, so each display list was able to use it. However, because of the reject process, operations are slightly different even under the same command; as a result, you need to be careful that the same screen is not always drawn even when in the same display list.

- **Microcode of the Fast3D series**

Although microcode of the Fast3D series is used in many samples, they are really obsolete now that the new and improved F3DEX series is out. You should use the improved F3DEX series for your actual game making.

(Sprite Microcode)

- S2DEX

S2DEX is the microcode that makes it possible to use [sprites](#) in N64 development. This microcode makes it possible for you to manage drawn objects using the separate concepts of the sprite and the BG. This makes it easy for you to get used to the usual sprite game development methods.

- Microcode of the Sprite2D Series**

Although microcode of the Sprite2D series is used in many samples, you should use the S2DEX microcode in actual game development. The S2DEX microcode is an improved version of the Sprite2D microcode.

(JJPEG Microcode)

The N64_JPEG library has two types of microcode, one for encoding and one for decoding.

- njpgespMain.o

njpgespMain.o provides [YUV conversion](#), culling, DCT, quantization, and a zigzag scan for RGBA image data. The output is entropy encodable data.

- njpgdspMain.o

njpgdspMain.o provides a reverse zigzag scan, reverse quantization, and reverse DCT for entropy decoded data. The output is [YUV](#) format data.

4-3-2 Audio Microcode

The audio microcode is made up of ABI (audio binary interface) commands. These ABI commands are strung together to form the audio command list. Each ABI command uses 64 bits.

- aspMain : audio microcode
Audio microcode is used only for waveform synthesis. This microcode makes the RSP operate as a waveform synthesis processor. The audio microcode processes each game application task and synthesizes the 16-bit sample data of an L/R stereo.
- n_aspMain : audio microcode for n_audio
n_aspMain is an enhanced version of the original N64 audio library. This microcode helps speed up the audio process.

Introduction to **NINTENDO⁶⁴**

Chapter 5 N64 Emulator Set

This chapter explains about setting up the "N64 Emulator Set".



5-1 Outline of the Tools

The purpose of the N64 simulator set is to develop/debug the N64 application on the Indy workstation.

5-1-1 The configuration of the N64 Emulator Set

1. N64 OS

N64 OS 1(the DAT tape)

2. N64 emulator board

The N64 emulator board

The N64 controller

The N64 controller conversion adapter



3.

4. **Figure 5-1-1 The N64 emulator board, the controller and the conversion adapter**

5-2 Operation Environment

You need the following environment to use the N64 Emulator Set.

1. System : The Indy Workstation
2. OS : IRIX 5.3

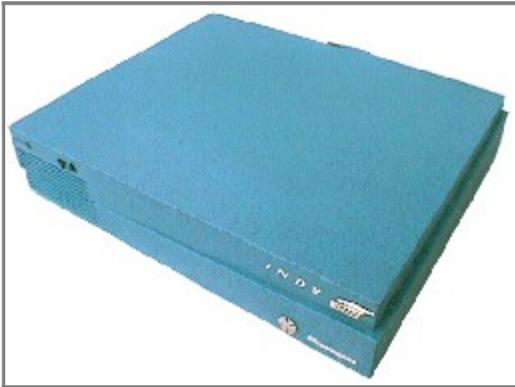


Figure 5-1-2 External view of Indy

5-3 Setting up

5-3-1 Setting up the Hardware

1. Mounting the N64 Emulator Board

After verifying that the **power** of the Indy workstation is **OFF**, remove the upper cover and connect the GIO connector.

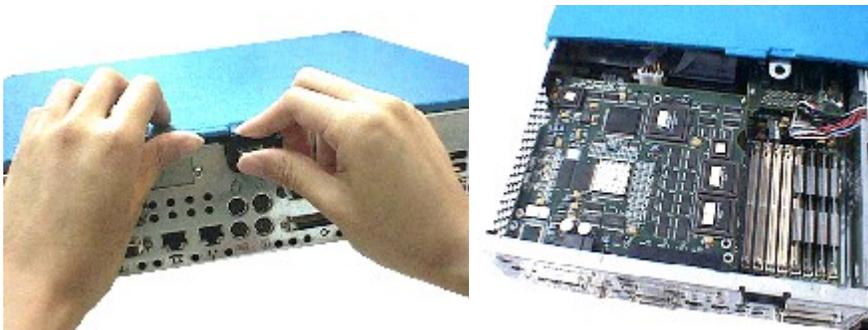


Figure 5-3-1 Removing the Indy cover and mounting the emulator board

2. Mounting the N64 Controller Conversion Adapter

Insert the N64 Controller conversion adapter into the game controller port of the N64 emulator board. Insert the connector of the game controller port from left to right (viewed from the back of the Indy workstation) so that the cables do not crisscross. (The following figure is the way it should look, with the right-most modular connector vacant as viewed from the back.)

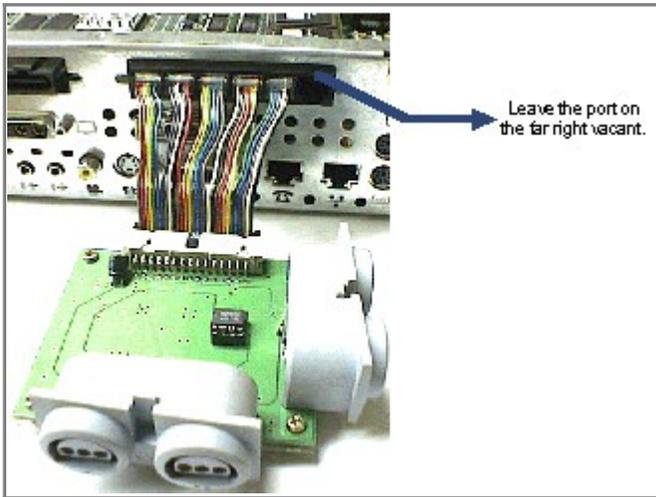


Figure 5-3-2 Connect the connector of the conversion adapter to the controller port

3. Mounting the N64 Controller and the Video Cable

Connect the N64 controller to the N64 controller conversion adapter and the video cable (purchased separately) to the AV output port of the N64 emulator board. Connect the pin plug (RCA) side of the video cable to the television.

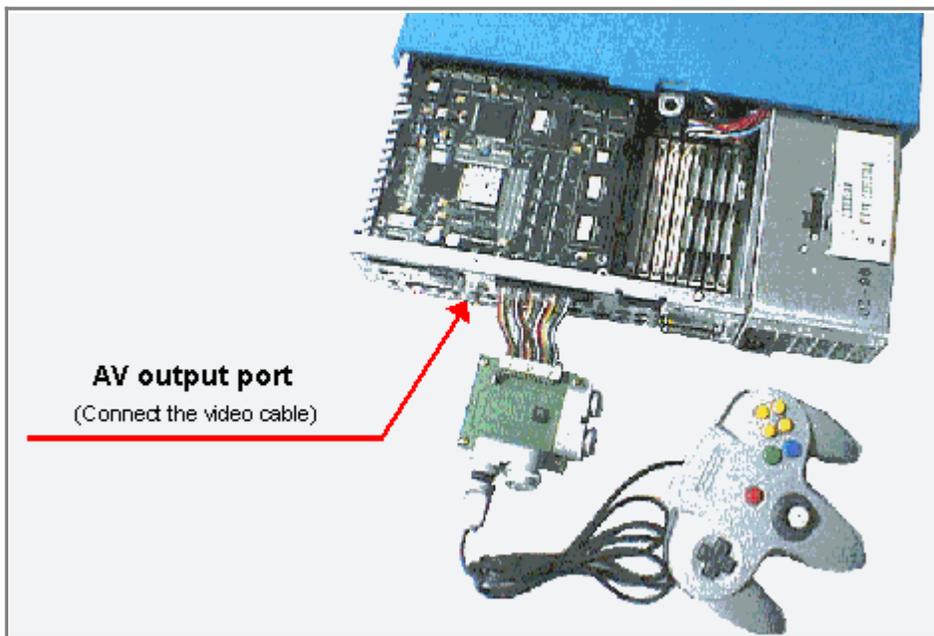


Figure 5-3-3 Connect the adapter, the controller and the cable

4. Alter the N64 Emulator Board required for use with PAL

The following alteration is required when you use the N64 emulator board to create an application for Europe (corresponding to PAL).

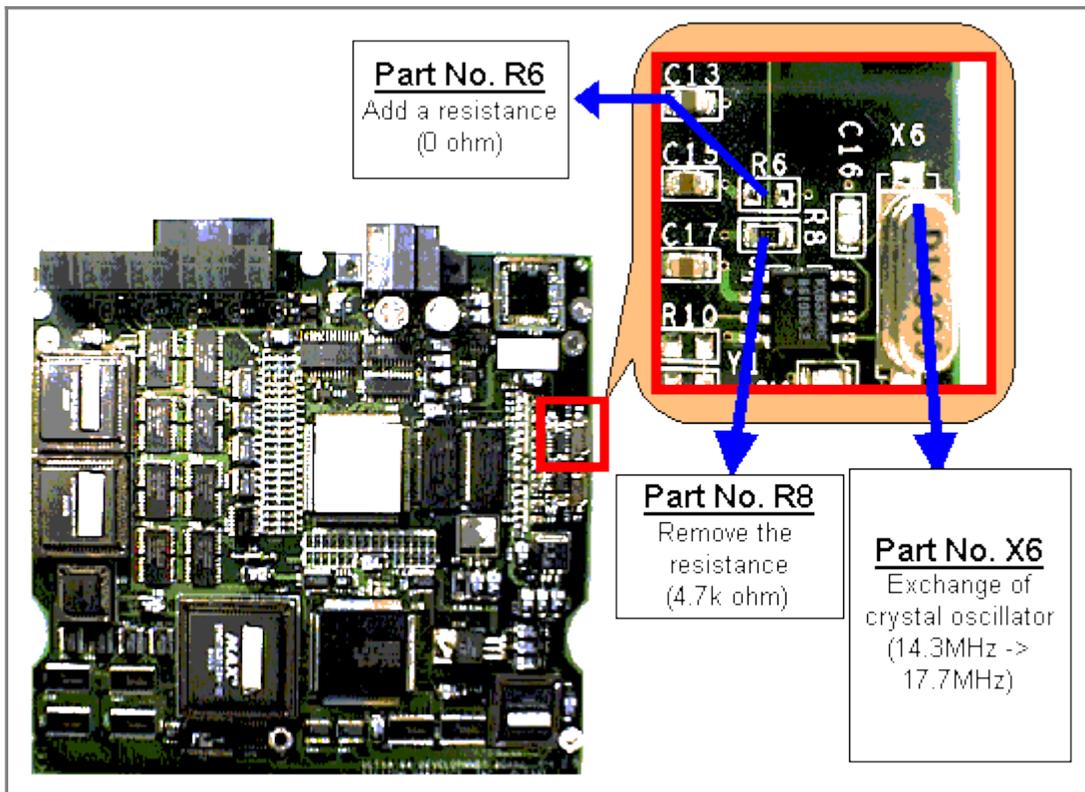


Figure 5-3-4 Alteration of the N64 Emulator Board required for use with PAL

-Alter the N64 Emulator Board

- Part number X6 : Change the crystal oscillator (14.3MHz -> 17.7MHz)
- Part number R8 : Remove the resistance (4.7k ohm)
- Part number R6 : Add resistance (0 ohm)

-Alter the Flash ROM Cartridge

- Change CIC (for NTSC -> for PAL)

5-3-2 Setting up the Software

This manual assumes that the compiler (cc), assembler 9as), and linker 9ld) required to develop the application, and the debugger, ([gvd](#)), required to debug the created application, have already been installed. See Section 2 of Chapter 1, "[Install the N64 Emulator Software](#)" in the programming manual for details.

1. The installation of N64 OS

Install the included files library, [microcodes](#) and sample program required to use the N64 OS in the N64 application. After inserting the DAT tape, move to the appropriate directory on the console and enter

```
% tar xv
```

(contents of DAT are expanded to the directory). Next, click "Software Manager" from the menu, "System," with the left mouse button and activate "Software Manager." After activation, designate the directory the contents of DAT were expanded to and click, "Customize Installation", with the left mouse button.

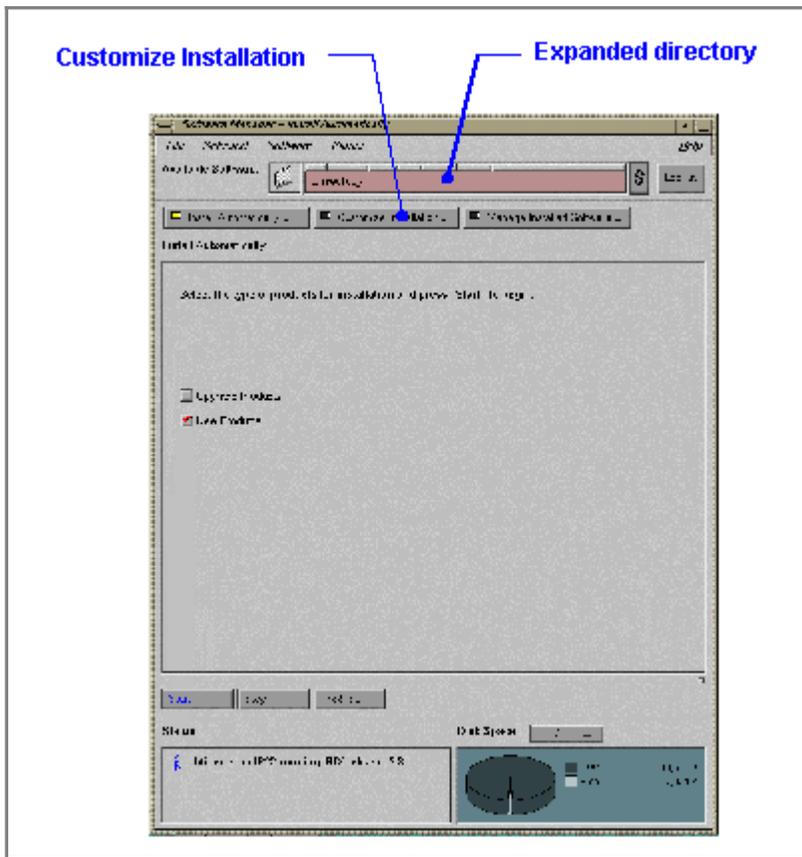
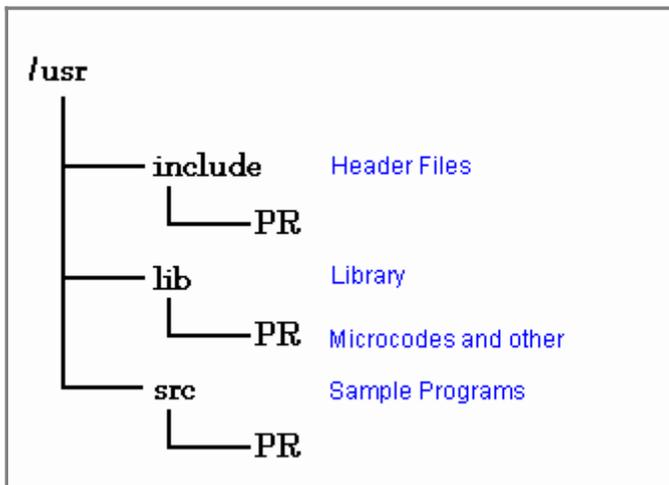


Figure 5-3-5 Installation of the N64 OS

The installable contents are displayed, so after selecting all (check the Install field), push "Start." When the installation ends without error, the Status field changes from "New" to "Same Version."

The directory configuration after the installation becomes as follows (this manual assumes these directories):



-If the disk capacity becomes insufficient during the installation, create disk space by adding hard disks or deleting unnecessary applications.

5-4 Tutorial

This section describes the program compilation/link, creating the ROM image file and debugging methods by using an actual sample. Before you use the sample, we will describe the procedure for creating the ROM image file. Once you understand this you can proceed.

5-4-1 Procedure for Creating the ROM Image File

The ROM image file is created by designating the object file and the picture/sound data to the ROM image creation tool after you have created the **relocationable object file** with the compiler/linker. (**Relocationable object file: Object files where the function and argument addresses have not been determined.**) The ROM image creation tool outputs the ROM image file and the debugger symbol file. The default name of the ROM image file is "rom" if not specified otherwise. Specify the symbol file name between "beginwave" and "endwave" of the spec file designated to the ROM image creation tool.

5-4-2 Compilation/link/creation of the ROM image using an actual sample

Now let us do the compilation/link/creation of the ROM image with an actual sample. Here, we will use the sample called "simple." "simple" has been installed in the directory, "/usr/src/PR/simple." Copy it to the appropriate directory from the console. For example, you can create the directory, "simple", copied into the current directory by executing:

```
% cp -r /usr/src/PR/demos/simple.
```

1. Makefile

The compilation/link/ROM image creation in "simple" are provided by using the program, "make." "make" provides the process based on the contents defined in the text file, "Makefile" (See the outline manual of "make" for details).

Now we will describe the contents defined in "Makefile." Open the text file, "Makefile" in the directory, "simple", with your editor (the actual "Makefile" does not describe the following **#comments**).

[Makefile]

```
#!/smake
include $(ROOT)/usr/include/make/PRdefs # Files reserved in the system [defining the
# compiler name and
# dependency, etc.]

# to make the tags file do "make simpletags"
```

```

# to make just the simple_d directory do "make SUBDIRS=simple_d"

SUBDIRS=simple_d simple simple_rom # Specify the subdirectory
# In "simple," you create the following three types of ROM image files
# and symbol files:
# 1. For debugging simple_d
# 2. For ordinary use simple
# 3. For master submitting simple_rom

COMMONPREF = simple #unused

APP = simple #Specifies the symbol file name
TARGETS = rom #Specifies the ROM image file name

HFILES = \ #Specifies the header file name for "simple"
:
CODEFILES = \ # Specifies the .c (the program code) file name
:
CODEOBJECTS = $(CODEFILES:.c=.o)
#Specifies (the program code) file name
#(The name is replaced from .c to .o, which is
#the file name specified in CODEFILES)
#Example: test.c -> test.o

CODESEGMENT = codesegment.o #Specifies the relocationable
#object file name created as a
#result of linking the program code
# Data files that have their own segments:

DATAFILES = \ #Specifies the .c (data) file name
:
DATAOBJECTS = $(DATAFILES:.c=.o)
#Specifies the .o(data code) file
#name (The name as CODDOBJECTS above)

OBJECTS = $(CODESEGMENT)$(DATAOBJECTS)
#Specifies all relocationable object file names
#specifies to the ROM image creation tool

LCINCS = -I. -I$(ROOT)/usr/include/PR
#Specifies the pass of the include file specifying to the compiler (for local use)
LCOPTS = $(DFLAG) -fullwarn -non_shared -G0 -Xcpluscomm
# Specifies the option of specify to the compiler (for local use)
LCDEFS =
# Specifies the symbol definition to specify to the
# compiler (for local use). Unused in "simple"

LDIRT = load.map
# Specifies the map file name

LDFLAGS = $(MKDEPOPT) -nostdlib -L$(ROOT)/usr/lib
-L$(ROOT)/usr/lib/PR -I$(ULTRALIB)
# Specifies the option to specify to the linker
# MKDEPOPT is a reserved name and the file defined the dependent
# relation of each object file.
# "-nostdlib" is a specification which is not use the standard library.
# "-L$(ROOT)/usr/lib-L$(ROOT)/usr/lib/PR -I$(ULTRALIB)" is the

```

option to link the N64 OS library.

.PATH: .. # The path specification of "simple"
"simple" is to execute "make" in
each subdirectory
include locdefs
Accept the setting of each
subdirectory. The error does not return even if it fails.

#include \$(COMMONRULES)
Note that "#" is a comment-out.

default: **# Specifies to provide the following**
process with default in this
for i in \$(SUBDIRS) ; do \ **# "Makefile." Specifies to move to**
echo ==== \$\$i === ; \ **# each subdirectory and execute "make"**
cd \$\$i ; \
\$(MAKE) -f ../Makefile loc_\$\$i ;\
cd .. ; \
done

\$(COMMONTARGETS) : **# Specifies to provide the process when it specifies**
the reserved command in the system to "make"
Example)make clean
for i in \$(SUBDIRS) ; do \ **#default: the same**
echo ==== clobber \$\$i === ; \
cd \$\$i ; \
\$(MAKE) -f ../Makefile loc_\$\$i ; \
cd .. ; \
done

include \$(COMMONRULES) **# Accept the reserved file in the**
System

install: default **# Specifies to install "simple"**
The files below are installed.
Execute "default" before executing
this. If it has not been executed,
first execute "default."
\$(INSTALL) -m 444 -F /usr/src/PR/demos/simple \
\$(HFILES) \$(CODEFILES) \
\$(DATAFILES) Makefile spec \
simple/log.fmt simple_d/locdefs \
simple/locdefs simple_rom/locdefs

\$(CODESEGMENT): \$(CODEOBJECTS)
Specifies the dependent relation between "code segment.o" and the .o file.
If the .o file is updated, provide the following process.
\$(LD) -m -o \$(CODESEGMENT) -r \$(CODEOBJECTS) \
\$(LDFLAGS) > load.map
Specifies to create the relocationable object file with using the linker
"-m" specifies to output the map file.
"-o" \$(CODESEGMENT) is an option to specify the output file name.
"-r" is the option to create the relocationable object file name.
"\$(CODEOBJECTS)" specifies the linking object file name.
"\$(LDFLAGS)" specifies to pass other options to the linker.

```

rom: ../spec$(OBJECTS)
# Specifies the ROM image file, all ".o" files and the dependent relation of spec file.
# Provide the following process when the .o file and the spec file are updated
$(MAKEROM) $(MAKEROMDEFS) ../spec
# Specifies to create the ROM image file with using the ROM image
# "$(MAKEROSDEFS)" is the standard option specified to the image creation too.
# "spec" is the text file to specify the ROM image to the ROM image creation, too.
# We will describe this later.

```

The contents mentioned above are just one example. The compiler/linker/ROM image creation tool has a lot of other convenient functions. Apply them based on the program usage.

2. Specifying the ROM Image

Next, we will describe the contents defined in the script file to specify the ROM image. Open the text file, "spec," in the directory, "simple", with your editor (In the actual "spec," the */*Comments*/* below are not described). Read the "[makerom](#)" online manual

[spec]

```

/* The ROM image manages in units of segments.*/
/* Define the segments having the program code attributes.*/
beginseg /* Initiate to define the segments*/
name "code" /* Specify the segment names*/
flags BOOT OBJECT
/* Designate the boot attribute and the object attribute*/
entry boot /* Specify the boot function */
stack bootStack + STACKSIZEBYTES
/* Specify the stack used by the boot function*/
include "codesegment.o"
/* Specify the object file mapping within the segment */
include "$(ROOT)/usr/lib/PR/rspboot.o"
/* Specify the boot microcode*/
include "$(ROOT)/usr/lib/PR/gspFast3D.o"
/* Specify the graphic microcode */
include "$(ROOT)/usr/lib/PR/gspFast3D.dram.o"
/* Specify the graphic microcode */
include "$(ROOT)/usr/lib/PR/aspMain.o"
/* Specify the sound microcode */
endseg /* End the segment definition */
/* The following is the description only about the parts which */
/*don't overlap with the "code" segments */
beginseg
name "gfxdlists"
flags OBJECT /* Designate the object attribute */
after code /* Specify mapping right after the "code" segment. */
include "gfxdlists.o"
endseg
beginseg
name "zbuffer"
flags OBJECT /* Designate the object attribute */
address 0x801da800 /* Specify mapping on the 0x801da800 address */
include "gfxzbuffer.o"
endseg

```

```

beginseg
name "cfb"
flags OBJECT /* Designate the object attribute */
address 0x80200000 /* Specify mapping on the 0x80200000 address */
include "gfxcfb.o"
endseg
beginseg
name "static"
flags OBJECT /* Designate the object attribute */
number STATIC_SEGMENT /* Specify the static segment number */
include "gfxinit.o"
include "gfxstatic.o"
endseg
beginseg
name "dynamic"
flags OBJECT /* Designate the data attribute */
number DYNAMIC_SEGMENT /* Specify the dynamic segment number */
include "gfxdynamic.o"
endseg
beginseg
name "bank"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/soundbanks/GenMidiBank.ctl"
/* Specify the sound bank data */
endseg
beginseg
name "table"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/soundbanks/GenMidiBank.tbl"
/* Specify the sound table data. */
endseg
beginseg
name "seq"
flags RAW /* Designate the data attribute*/
include "$(ROOT)/usr/lib/PR/sequences/BassDrive.seq"
/* Specify the sound sequence data */
endseg

beginwave /* Initiate to define waves*/
name "simple" /* Specify the symbol file name (ignore ".out") */
include "code" /* The following are the specification of mapping segments.*/
include "gfxdlists"
include "static"
include "dynamic"
include "cfb"
include "zbuffer"
include "table"
include "bank"
include "seq"
endwave /* End the wave definition */

```

The above is the required procedure to provide the compilation/link/ROM image creation to the sample with "make."

3. Executing "make"

Now, let us actually execute "make". Execute "make" using the directory, "simple".

% make

With the completion of "make," the ROM image file, "rom", and the symbol file, "simple", are created in three subdirectories. Verify this with the "ls" command, etc.

5-4-3 Executing the Sample

Execute the sample using the ROM image and symbol files which were created in "5-4-2: The Compilation/Link/ROM Image Creation of the Sample." (You use "simple" again here.) Provide the execution of the sample using the command, "gload". Continue using the following procedure:

1. Moving the Directory

Move to the subdirectory "simple_d" of "simple."

```
% cd simple_d
```

2. Executing "gload"

Input "gload" in the moved directory. (If there is no specification of the ROM image file name, "gload" recognizes the file "rom" as the ROM image file.)

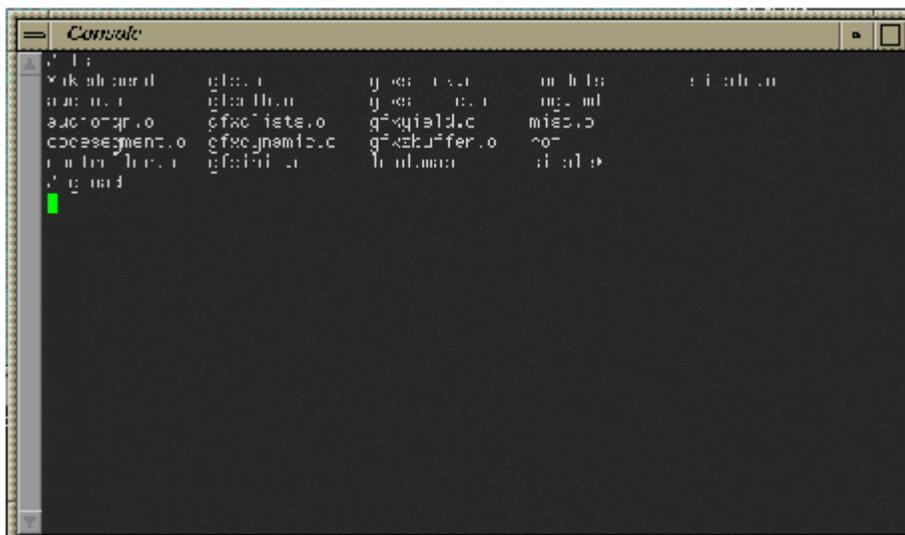


Figure 5-4-1 Executing "gload"

3. The end of the execution

End with "^c"(Ctrl+C).

5-4-4 Debugging the Sample

This section describes some debugging examples using the sample. Because you use "simple" here again, move to the subdirectory "simple_d" of "simple" in the same way as in, "5-4-3 : Executing the Sample."

1. Activation "gvd"

You need some procedures for using the debugger (the rest is "gvd").
Input commands in the following order on the console:

%[dbgif](#) & : Specify the connection of the N64 emulator board

%[gload](#) -a "-d" & : Specify the software down_load

%[gvd](#) simple & : Specify "simple" to "gvd"

Note: To operate each command on a background, specify "&" at the end. To end this command, verify the process ID of each command with the "ps" command, and specify the process ID to the "kill" command. The "-a" option of "gload" specifies the passing of arguments to the program. In "simple," if the argument "-d" is specified, it provides the process corresponding to "[gvd](#)" (see "simple"-misc.c").

2. Connecting to Targets

The debugging by "gvd" is provided in units of [threads](#). This manual debugs the thread called "simple" - "game Thread". To provide a choice of threads, select "Admin" - "Switch Thread" from the "gvd" main menu with the left mouse button. After you select, the window to input the thread ID is displayed. Input the thread ID here ("game Thread" is 6).

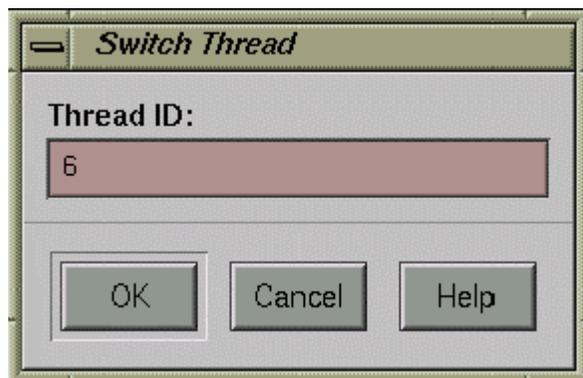


Figure 5-4-2 Selecting the thread

3. The Setting & Cancellation of the Breakpoint

This section describes the setting method of the breakpoint. Set the breakpoint in "simple" - "gfx.c" - "createGFxTask()". Click "Source" - "Open" from the "gvd" main menu with the left mouse button.

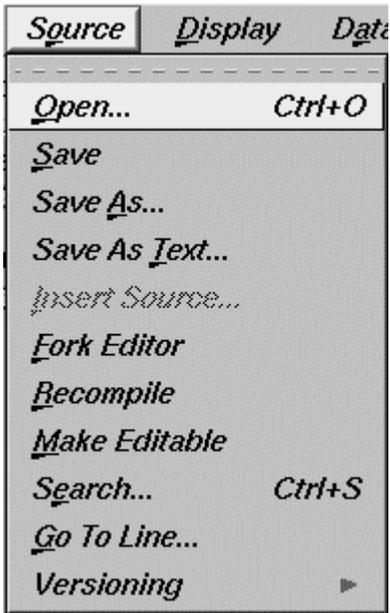


Figure 5-4-3 Open the source

After you select, the file list box is displayed. Select "gfx.c" and click the "OK" icon with the left mouse button.

Note: The directory (Selection) right after the "gvd" activation is the subdirectory of "simple". Select the directory after correcting it to the directory having "simple".

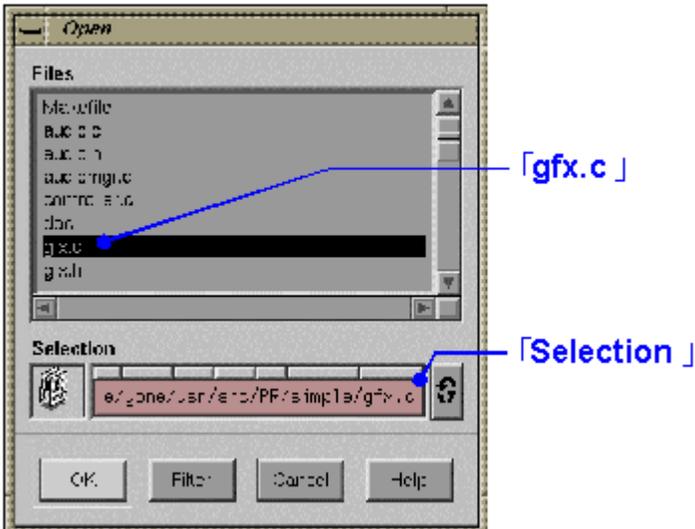


Figure 5-4-4 Select the file

After you have selected the file, the content of "gfx.c" is displayed in the "gvd" source window. Use the scroll bar and scroll the source until "createGfxTask()". The 127th line of "gfx.c" has a place to bring up the "assert()" function.

Move the mouse cursor to the left margin of the source window and click. Then, the line color bringing up the "assert()" function of the source window is highlighted and a downward arrow is displayed in the left margin. Now you have finished setting the breakpoint.

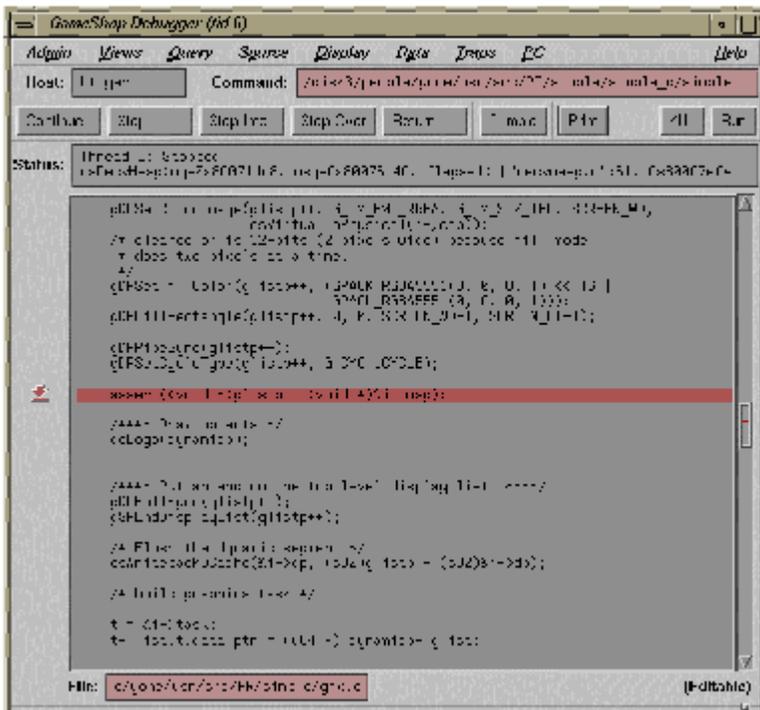


Figure 5-4-5 Select the line calling the "assert ()" function

Next, click the "continue" button with the left mouse button and execute the program.

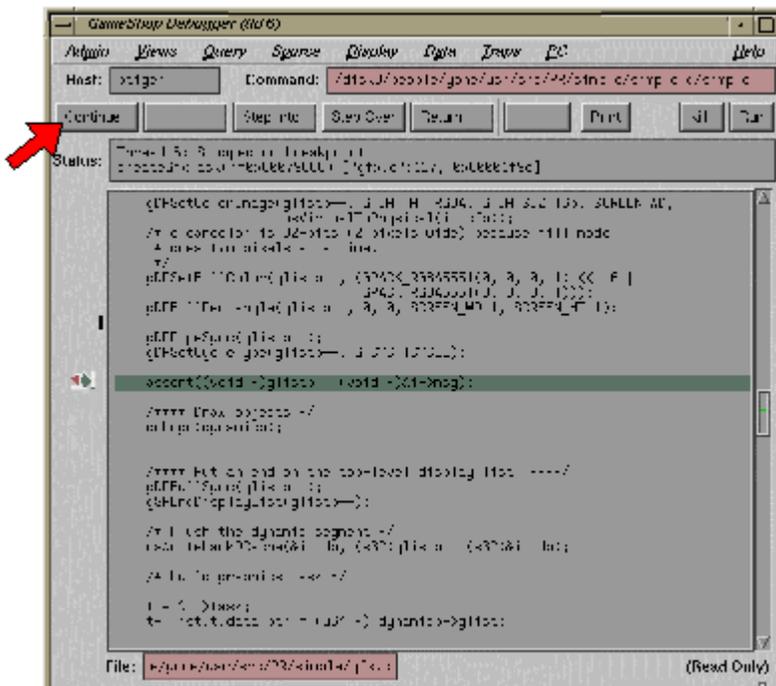


Figure 5-4-6 Setting/cancellation of the breakpoint

After executing, the color of the line set as the breakpoint is highlighted, and a sideways arrow is displayed in the left margin and the program breaks. To free the breakpoint, click the left margin with the left mouse button in the same way as with setting. After clicking, the downward arrow in the left margin disappears. Now you have freed the breakpoint.

4. Step Execution & Trace Execution

This section describes step execution and trace execution of the program. First, complete the steps in "3. The Setting & Cancellation of the Breakpoint". After the break, provide the step execution. For the step execution, click the "Step Over" button with the left mouse button, or input "n" on the command input window. After clicking, the line which was highlighted on the source window moves along through the program process, and the sideways arrow in the left margin moves with it. This shows that the step execution is being carried out.

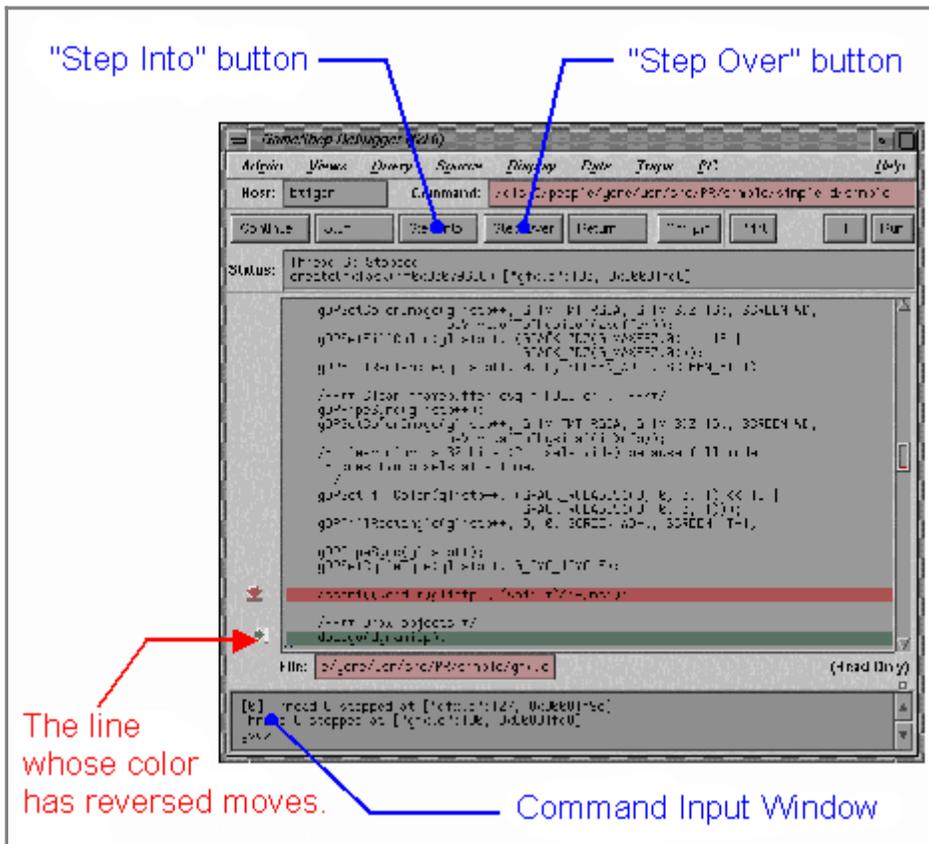


Figure 5-4-7 Step Execution

For the trace execution, click the "Step Into" button with the left mouse button, or input "s" in the command input window. The difference between the trace execution and the step execution is that if a function is brought up, the trace execution goes into the function, but the step execution just provides the function call and does not go into the function.

5. The Dump & Editing

This section describes the [dump](#) and editing functions. First, complete the steps described in "3. The Setting & Cancellation of the Breakpoint". After the break, carry out the dump.

In "createGfxTask()", the data setting is provided to the argument "glistp". Dump this content. Select "Views" - "Memory View" from the "gvd" main menu with the left mouse button. After the selection, the "Memory View" window is displayed. Input "glistp" to "Address" and press the "Enter" key. After the input, the dump data is displayed in the window.

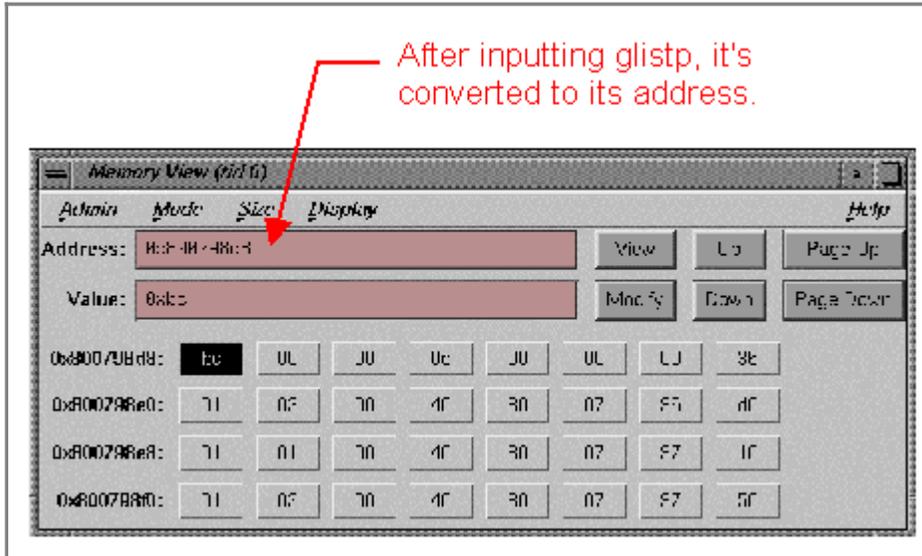


Figure 5-4-8 Display the dump data

This section describes the dump methods in units of byte. However, in "Memory View" you can change the notation of the numeric values(decimal notation and hexadecimal notation, etc.), depending on the menu "Mode", or the dump unit depending on the menu "Size".

Next, the editing will be done.

The editing also uses "Memory View" in the same way as the dump.

First, provide the same content as described for the dump.

Next, input "0xff" in "value" and click on the "modify" button with the left mouse button.

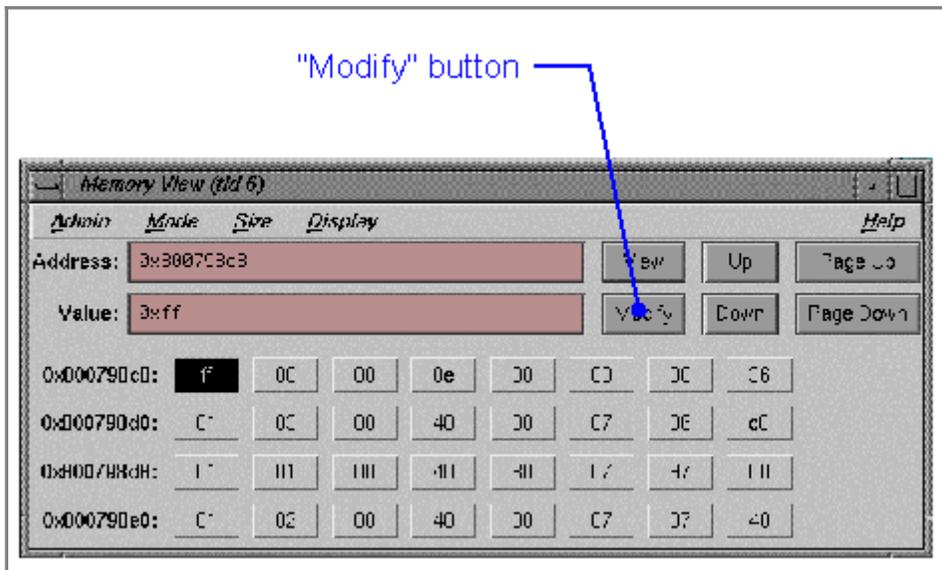


Figure 5-4-9 The editing

After the click, try to dump "glistp" again and you will notice that the first byte has been changed to "0xff"

6. Refer to Arguments of C

This section describes about the source code debugging function referring to arguments of C. First, provide the content described in "3. Setting & Cancellation of the Breakpoint". After the break, provide the reference of arguments. Next, provide the reference using the "p" command from the command input window. In "createGfxTask()", the local argument "dynamicp" is passed to the "doLogo()" function. Input the following command in the command input window:

```
gvd> p *dynamicp
```

Because "dynamicp" is a pointer type, "*" specified at the beginning of "dynamicp" is placed in order to reference its content. After the input of the command, the content of "dynamicp" is displayed in the command input window.

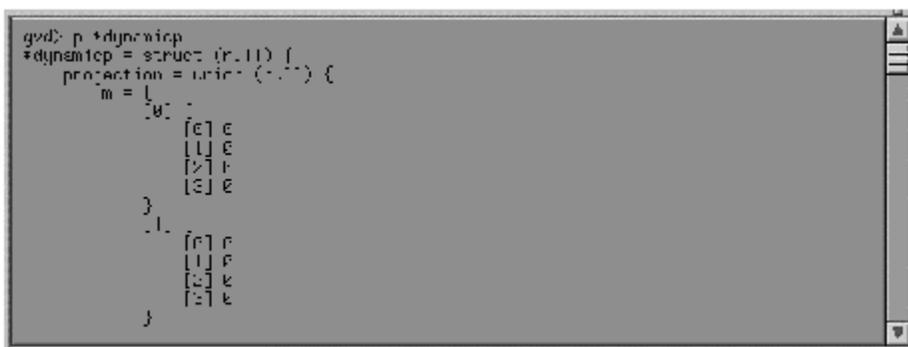


Figure 5-4-10 Display the content of "dynamicp"

If it hard to see the screen, adjust the command input window by enlarging.

Next, provide the reference by using "Structure Browser." ("Structure Browser" is a function to refer to the structure

content.) This time, refer to the argument "glistp". Select "Views" - "Structure Browser" from the "gvd" main menu with the left mouse button. After selection, the "Structure Browser" window is displayed. Input "glistp" in "Expression" and press the "Enter" key. After input, the content of "glistp" is displayed in the window.

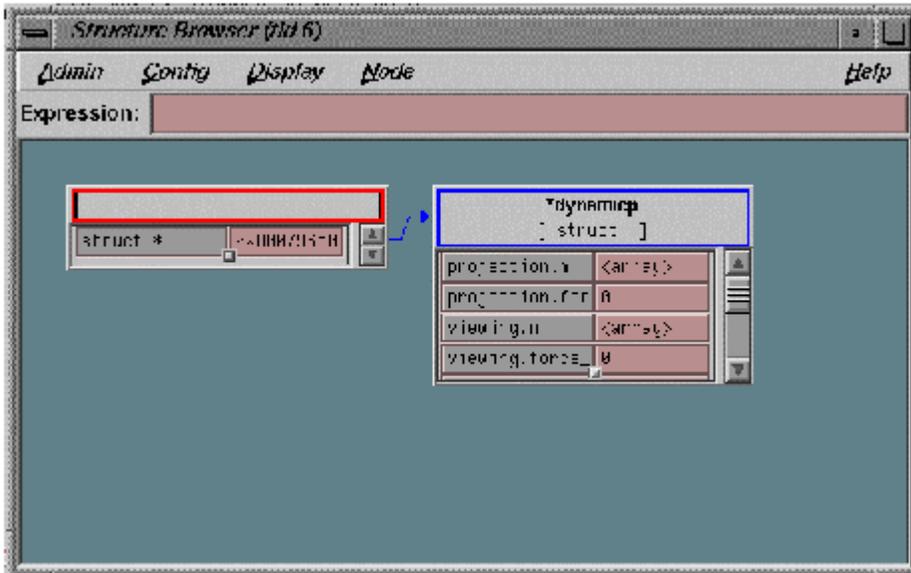


Figure 5-4-11 Display the content of "glistp"

Finally, you can look up the local argument by using, "Variable Browser", if so desired. Select "Views"- "Variable Browser" from "gvd" main menu with the left mouse button. After the selection, the "Variable Browser" window is displayed and you can look up the content of the local argument.

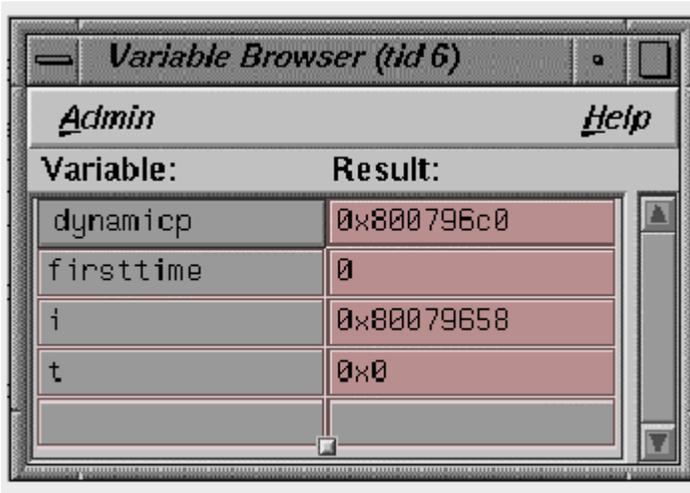


Figure 5-4-12 Looking up the content of the local argument

7. The Debug Print

On the "N64 Emulator", you can look up the debug print in the program. The display the debug print, "[osSyncPrintf\(\)](#)" set up in the N64 OS function is used. The output result of "[osSyncPrintf\(\)](#)" is displayed in the

console which has executed the "gload" command. "simple" is a program which is supposed to display the debug print when the A, B or C buttons of the controller are input; so execute "simple" by using the "gload" command, without using "gvd". When you input the controller button while executing the program, the debug print is displayed in the console window.

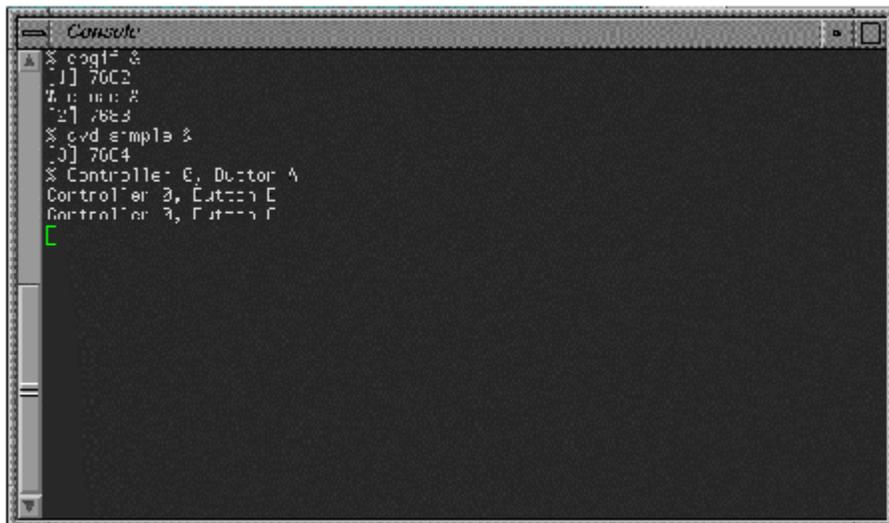


Figure 5-4-13 Looking up the debug print

The above is a simple example of the debug. "gvd" has various other functions beside this. Find the best debug method from among these.

Introduction to **NINTENDO⁶⁴**

Chapter 6 PARTNER-N64 PC

This chapter explains about the setup of the "PARTNER-N64 PC."



PARTNER-N64 Product System

	PARTNER-N64 NW		PARTNER-N64 PC	
The host machine	SGI/INDY SGI/O2		DOS/V PC-98 series	
OS	IRIX5.3 IRIX6.2, 6.3		Windows 3.1 Windows 95	
The compiler	exeGCC N64 for SGI IDO5.3 IDO7.1 for 6.2 IDO7.1 for 6.3		exeGCC N64 for PC	
The connecting method	10BASE-T 10BASE-2		The special purpose parallel interface	
The model	Model 10	Model 11	Model 10	Model 11
The emulation ROM	16 MByte	32 MByte	16 MByte	32 MByte



6-1 Outline of Tools

The purpose of PARTNER-N64 PC is to develop/debug the N64 application on DOS/V or the PC-98 series.

6-1-1 Configuration of the PARTNER-N64 PC

1. N64 OS for PC

N64 OS for PC, 1 piece (CD-ROM)

2. PARTNER-N64 PC and exeGCC (for N64 use)

PARTNER-N64 PC Debugger

Modified N64 Control Deck (AC adapter and Controller included)

PARTNER-N64 Interface Board

PARTNER-N64 Interface Board Cable

PARTNER-N64 Debugger and exeGCC Software (1 CD)

PARTNER-N64 PC User's Guide, 1 volume

exeGCC User's Guide, 1 volume

6-2 Operating Environment

The following is the operating environment of the PARTNER-N64 PC:

1. Machine - DOS/V or the PC-98 series

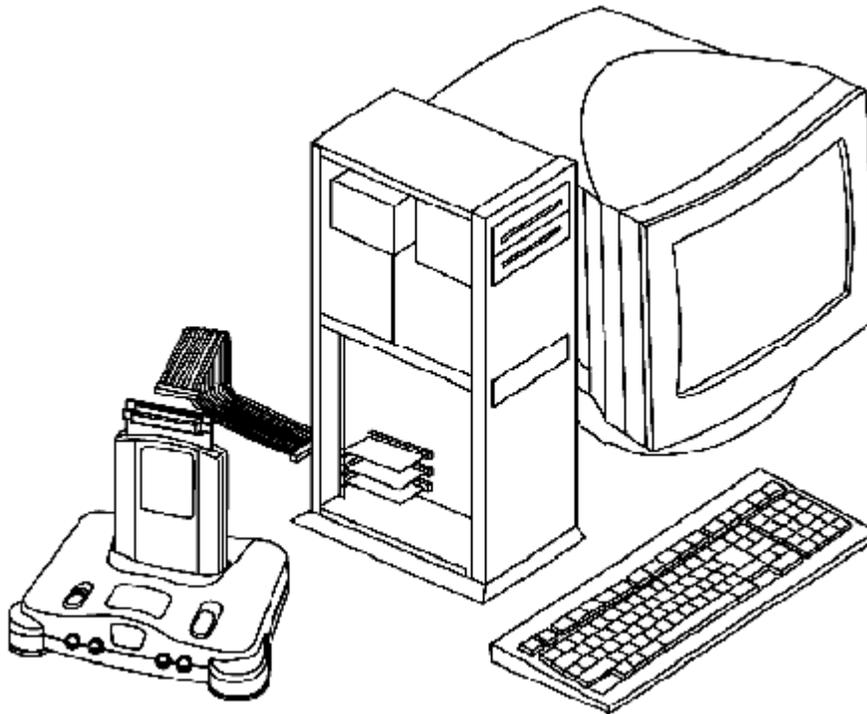
2. Slot - ISA bus of C bus

3. CPU - 80386 or above

4. Memory - 16 Mbytes or more

5. OS - MS-Windows 3.1/95

6. Hard drive - 100 Mbytes or more available space



7.

Figure 6-2-1 The PARTNER-N64 PC

6-3 Setting up

6-3-1 Setting up the Hardware

1. Install the PARTNER-N64 Interface Board

Connect the interface board provided with PARTNER-N64 to the ISA slot of the host machine (DOS/V) or to the C bus slot of PC-98.

2. Install the PARTNER-N64 Debugger

Insert PARTNER-N64 Debugger into the Game Pak slot of the Modified N64 Control Deck.

3. Install the Interface Board Cable

Connect between the 40-pin connector of the PARTNER-N64 Interface Board and the PARTNER-N64 I/F connector (40-pin) using the special cable provided.

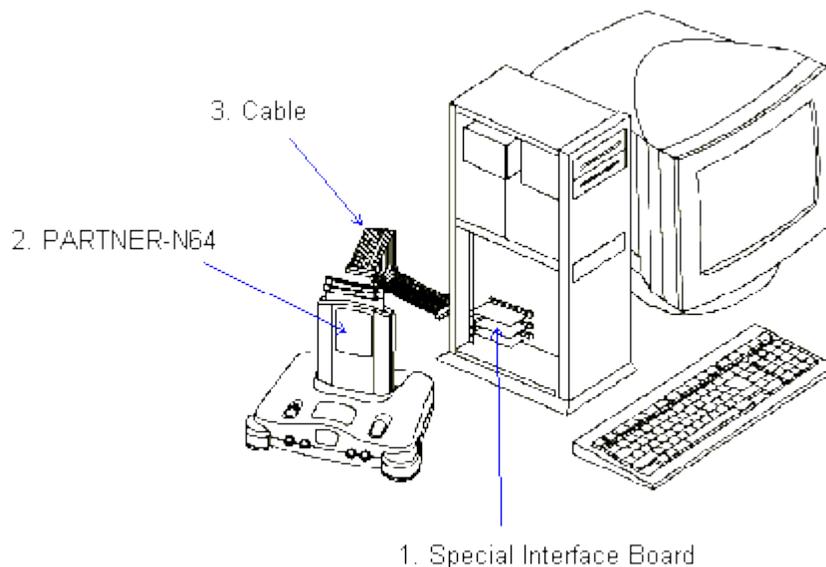


Figure 6-3-1 The hardware setup

4. Install the Controller, video cable and AC adapter

After checking that the N64 **Power switch is OFF**, connect the Controller, a video cable (purchased separately) and the AC adapter.

6-3-2 Setting up the software

Carry out the software installation using the following procedure:

1. Install N64 OS (PC)

The N64 OS must be installed prior to installing PARTNER-N64 Debugger software. If you have not installed the N64 OS software on your PC, use the CD and installation instructions included in your N64 Software Development

Kit to do so now. The [microcode](#) and sample program which are required for using N64 OS in the N64 application are contained on this CD.

2. Install PARTNER-N64PC Debugger and exeGCC Compiler

Install the PARTNER-N64PC Debugger software and exeGCC by inserting the CD into your CD-ROM drive. If Setup does not run after a few seconds, double-click Setup.exe in your CD-ROM directory.

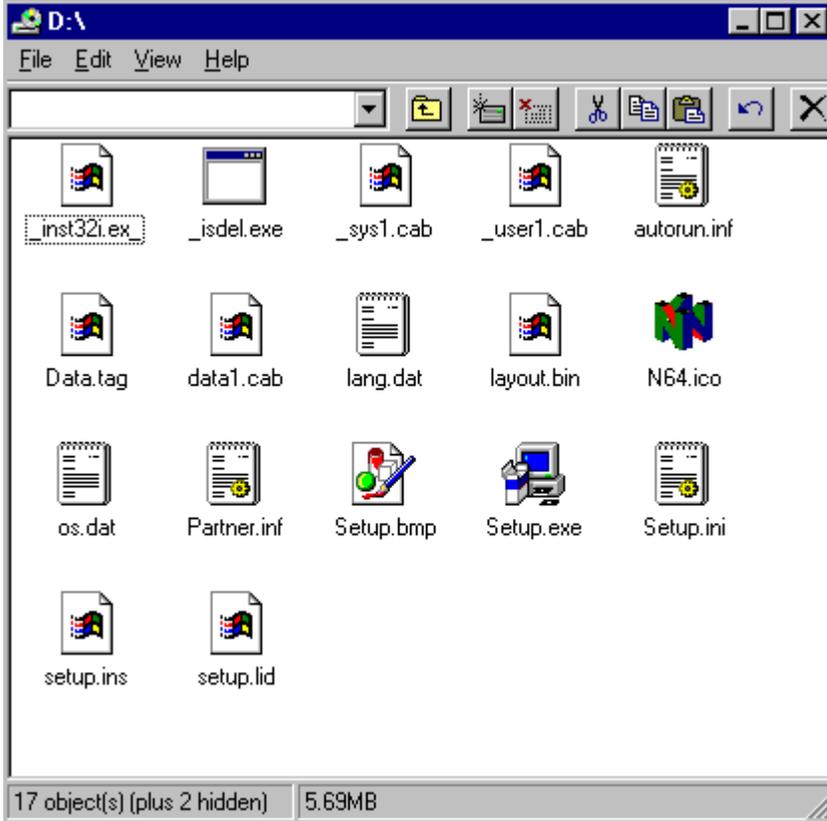


Figure 6-3-1 The contents of CD-ROM directory

PARTNER-N64PC files are installed in "c:\ultra\WPTN64" by default.

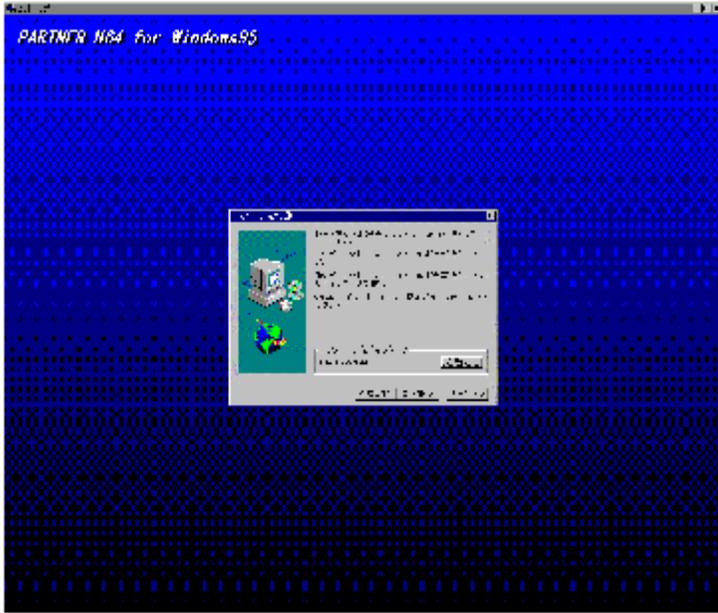
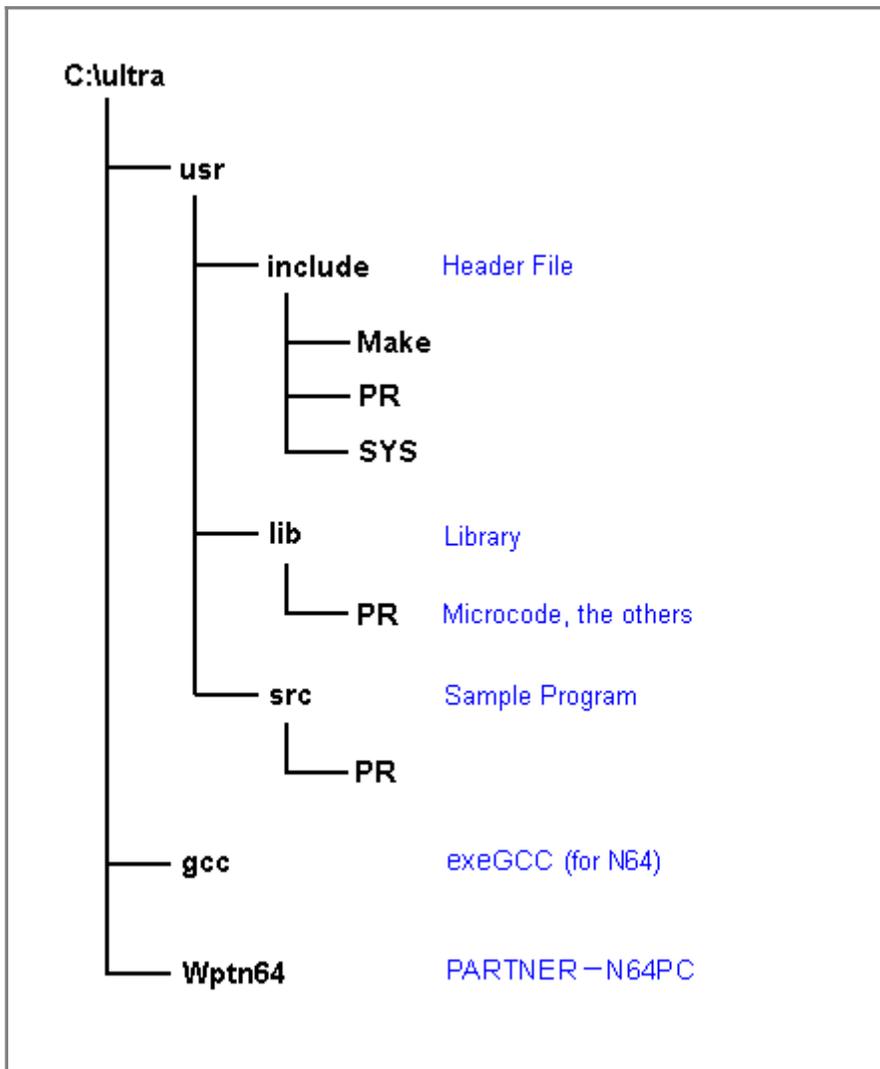


Figure 6-3-2 The PARTNER-N64 install screen

The following is the directory configuration after the installation:



6-4 Tutorial

By using an actual sample this section describes the program compilation/link, creation of the ROM image file, and the debugging method. Before you use the sample, we will describe the procedure for creating the ROM image file. Proceed to the next step after you have an understanding of this.

6-4-1 Procedure for creating the ROM Image File

The ROM image file is created by specifying the object file and the picture/sound data to the ROM image creation tool after you have created the **Relocationable object file** one time using the compiler/linker. (**Relocationable object file: Object files where the function and argument addresses have not been determined**)

The ROM image creation tool outputs the ROM image file and the debugger symbol file. If no file name has been specified, use the following for output:

The ROM image file : rom.n64
The symbol file : rom.out

If you specify the file name, do not change the extensions: specify each of them to make a pair.

[Example]

The ROM image file : sample.n64
The symbol file : sample.out

6-4-2 The compilation/link/creation of the ROM image, using an actual sample.

Now let us do the compilation/link/creation of the ROM image, using an actual sample. Here, we will use the sample called, "simple." In this manual, it has been set up in the directory, "c:\ultra\src\PR\simple". Activate "MS-DOS prompt" from "Windows 95", and go to the directory, "simple"

1. Makefile

The compilation/link/ROM image creation in "simple" are done using the program, "make." "make" provides the process based on the contents defined in the text file, "Makefile" (See "The exeGCC (N64) handling manual" for details).

Now we will describe the contents defined in "makefile." Open the text file, "Makefile", in the directory of "simple" with your editor. (The actual "Makefile" does not describe the following **#comments**.)

[Makefile]

```
LIB = $(ROOT)/usr/lib # Specify the pass of the N64 OS library
LPR = $(LIB)/PR # Unused in "simple".
INC = $(ROOT)/usr/include # Specify the pass pf the N64 OS header file.
CC = gcc # Specify the compiler
LD = ld # Specify the linker
MAKEROM = mild # Specify the ROM image creation tool.

.c.o: # Specify the dependent relation between the .c and .o file.
# Provide the following process if .c is updated.
$(CC) -g -G 0 -c -I. -I$(INC)/PR -I$(INC)-D_LANGUAGE_C
-D_MIPS_SZLONG=32 -D_MIPS_SZINT=32 -D_DEBUG $<
# Specify the creation of the object file by using the compiler.
# Output the object file having the debugger source code information
# by combining "-g" and "-c".
# -G 0 -I$(INC)/PR -I$(INC) -D_LANGUAGE_C -D_MIPS_SZLONG=32 -
```

D_MIPS_SZINT=32 is the required option when you use the N64 OS.
-I.-D_DEBUG is the option for simple.

APP = simple.out # Specifies the symbol file name for debugger
TARGETS = simple.n64 # Specifies the ROM image file name

HFILES = \ # Specifies the header file name for "simple"

:

CODEFILES = \ # Specifies the .c (the program code) file name

:

CODEOBJECTS = \$(CODEFILES:.c=.o)

Specifies the .o (the program code) file name

(The name is replaced from .c to .o, which is

the file name specified in CODEFILES)

Example) test.c->test.o

CODESEGMENT = codesegment.o

Specifies the relocationable object file name created

as a result of linking the program code

DATAFILES = \ # Specifies the .c (data) file name

:

DATAOBJECTS = \$(DATAFILES:.c=.o)

Specifies the .o (data code) file name

(Same as CODEOBJECTS above)

OBJECTS =\$(CODESEGMENT) \$(DATAOBJECTS)

Specifies all relocationable object file names

Specifies to the ROM image creation tool

**LDFLAGS = \$(MKDEPOPT) -L\$(LIB) -lgultra_d-L\$(GCCDIR)/mipse/
-liblkmc**

Specifies the option to specify to the linker

"MKDEPORT" is a reserved name and required option when you create

the relocationable object file by using the linker.

"-L\$(LIB)-lgultra_d" is the option to link the N64 OS library.

"-L\$(GCCDIR)/mipse/lib-lkmc" is the required option when you link

the object file created by the compiler

default: \$(TARGETS) # This "make" is the specification of creating

the ROM image file by default

\$(CODESEGMENT): \$(CODEOBJECTS)makefile

Specify the dependent relation among "codesegment.o", the .o file and "Makefile"

Provide the following process when ".o" and "Makefile" are updated:

\$(LD) -o \$(CODESEGMENT) -r \$(CODEOBJECTS)\$\$(LDFLAGS)

Specify the creation of the relocationable object file by using the linker.

"-o \$(CODESEGMENT)" is the option to specify the output

file name.

"-r" is the option to create the relocationable object file.

\$(CODEOBJECTS) Specifies the linking object file name

\$(LDFLAGS) is the specification of passing other options to the linker

\$(TARGETS): \$(OBJECTS)

Specify the dependent relation between the ROM image file and all ".o" files.

Provide the following process if ".o" is updated:

\$(MAKEROM) spec -r \$(TARGETS) -e \$(APP)

Specify the creation of the ROM image file by using

```
# the ROM image creation tool.
# "spec" is the text file to specify the ROM image to the ROM
# image creation tool. This will be mentioned later.
# "-r $(TARGETS)" is the option to specify the ROM image
# file name.
# "-e $(APP)" is the option to specify the symbol file name.
```

The contents described here are just one example. The compiler/linker/ROM image creation tool has many other convenient functions. Utilize them based on what the program can be used for.

2. Specifying the ROM Image

Next, we will cover the contents defined within the script file to specify the ROM image. Open the text file, "spec", in the directory, "simple", with your editor (In the actual spec, the following */*comments*/* are not described.) Read "The [ROM Packer-specfile format](#)" from "The exeGCC (N64) handling manual" along with this description.

[spec]

```
/* The ROM image manages in units of segments.*/
/* Define the segments having the program code attributes.*/
beginseg /* Initiate to define the segments */
name "code" /* Specify the segment names */
flags BOOT OBJECT /* Designate the boot attribute and the object attribute */
entry boot /* Specify the boot function */
stack bootStack + STACKSIZEBYTES
/* Specify the stack used by the boot function*/
include "codesegment.o"
/* Specify the object file mapping within the segment */
include "$(ROOT)/usr/lib/PR/rspboot.o"
/* Specify the boot microcode. */
include "$(ROOT)/usr/lib/PR/gspFast3D.o"
/* Specify the graphics microcode */
include "$(ROOT)/usr/lib/PR/gspFast3D.dram.o"
/* Specify the graphics microcode */
include "$(ROOT)/usr/lib/PR/aspMain.o"
/* Specify the sound microcode */
endseg /* End defining the segment */
/* The following is the description only about the parts which don't overlap with the "code" segments
*/
beginseg
name "gfxdlists"
flags OBJECT /* Designate the object attribute */
after code /* Specify mapping right after the "code" segment. */
include "gfxdlists.o"
endseg
beginseg
name "zbuffer"
flags OBJECT /* Designate the object attribute */
address 0x801da800
```

```

/* Specify mapping on the 0x801da800 address */
include "gfxzbuffer.o"
endseg

beginseg
name "cfb"
flags OBJECT /* Designate the object attribute */
address 0x80200000
/* Specify mapping on the 0x80200000 address */
include "gfxcfb.o"
endseg
beginseg
name "static"
flags OBJECT /* Designate the object attribute */
number STATIC_SEGMENT /* Specify the static segment number */
include "gfxinit.o"
include "gfxstatic.o"
endseg
beginseg
name "dynamic"
flags OBJECT /* Designate the data attribute */
number DYNAMIC_SEGMENT /* Specify the dynamic segment number */
include "gfxdynamic.o"
endseg
beginseg
name "bank"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/soundbanks/GenMidiBank.ctl"
/* Specify the sound bank data */
endseg
beginseg
name "table"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/soundbanks/GenMidiBank.tbl"
/* Specify the sound table data. */
endseg
beginseg
name "seq"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/sequences/BassDrive.seq"
/* Specify the sound sequence data */
endseg

beginwave /* Initiate to define waves */
name "simple" /* Specify the symbol file name (Ignore ".out") */
include "code" /* The following are the specification of mapping segments. */
include "gfxdlists"
include "static"
include "dynamic"
include "cfb"
include "zbuffer"
include "table"
include "bank"
include "seq"

```

```
endwave /* End the wave definition */
```

The above is the required procedure to provide the compilation/link/ROM image creation to the sample with "make".

3. Executing "make"

Now, let us actually execute "make". Execute "make" from the directory, "simple".

```
C:\ULTRA\USR\SRC\PR\SIMPLE>make
```

After the end of "make", the ROM image file, "simple.n64", and the symbol file, "simple.out" will have been created. Verify it with the DIR command, etc.

6-4-3 Executing the sample

Try to execute the sample by using the ROM image file and the symbol file which were created in "6-4-2: The compilation/link/ROM image creation of the sample" (we use "simple" here again). Carry out the execution of the sample by using the debugger (See "PARTNER" below). Proceed using the following procedure.

1. Activating "PARTNER"

Click the icon, "N64 WRT64", from "Windows 95" - "Start" - "Program(P)" - "PARTNER-N64", with the left mouse button.

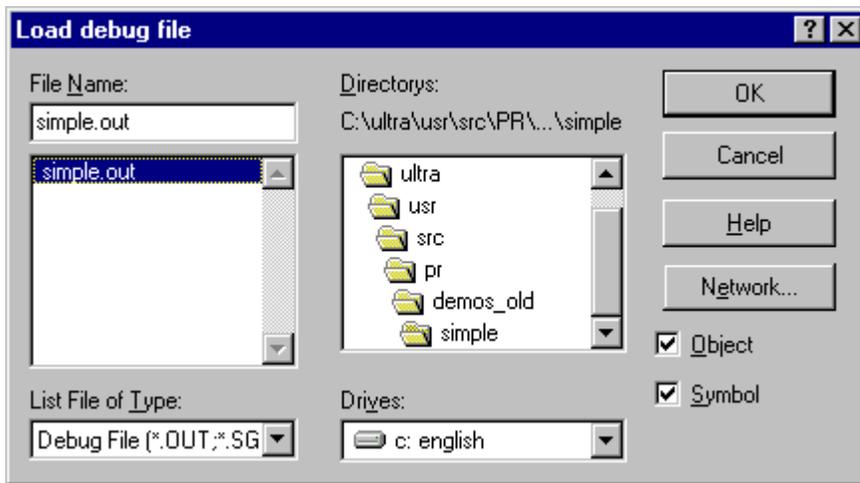


2. Load the file

Click "File(E)" - "Load(L)" of the PARTNER main menu or the load icon on the tool bar with the left mouse button.



The dialog box for the file load is displayed. After going to the directory which contains, "simple", select the file, "simple.out", with the left mouse button and click the "OK" button with the left mouse button.



3. The execution

Click "Run(R)" - "Program run(G) F5" from the PARTNER main menu, or the execute icon on the tool bar with the left mouse button.



4. Ending the program

Press the "ESC" key, or click the stop icon on the tool bar with the left mouse button.



6-4-4 Debugging the sample

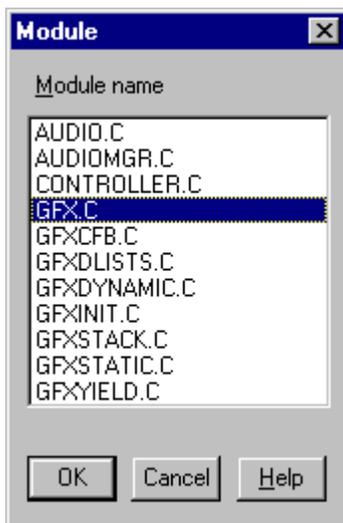
This section describes some debug examples by using the sample (the operating method of PARTNER). We use, "simple", here again.

1. Setting/freeing the breakpoint

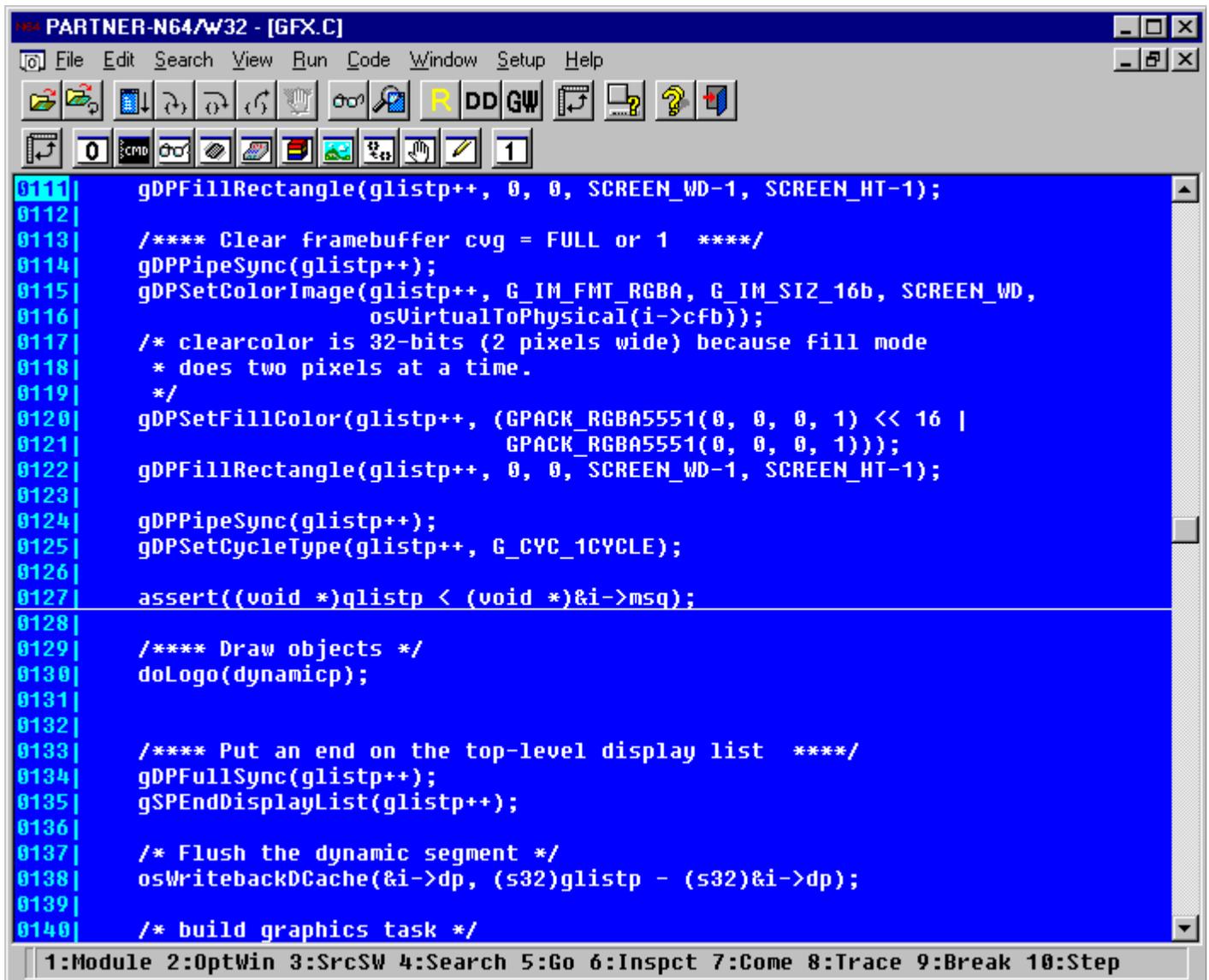
First, we will describe the setting method of the breakpoint. Set the breakpoint within "simple" - "gfx.c" - "createGfxTasc()". After loading the sample, click on, "Code(C)" - "Module(M) F1", in the PARTNER main menu with the left mouse button.

Code	Window	Setup	Help
I	n	s	p
W	a	t	c
C	o	m	e
D	i	s	a
M	o	d	u
L	i	n	e
T	o	p	
B	o	t	t
P	C	L	o
F	i	e	F
E	n	t	i
F	l	i	p

After clicking, the modules list box is displayed. Select "GFX.C" with the left mouse button, and click the "OK" icon with the left mouse button.



After selecting the module, the contents of "GFX.C" are displayed in the PARTNER command window. Use the scroll bar and scroll down in sources until "createGfxTasd()" (The line numbers are displayed on the left side of the code window, so scroll to between the 110th and 130th line). The 127th line of "GFX.C" contains a portion for bringing up the assert() function. Bring the mouse cursor to this line number and click on it with the left mouse button. A line is displayed under the line which is bringing up the assert() function of the code window. Now we have set the breakpoint.

A screenshot of a debugger window titled "PARTNER-N64/W32 - [GFX.C]". The window has a menu bar with "File", "Edit", "Search", "View", "Run", "Code", "Window", "Setup", and "Help". Below the menu bar is a toolbar with various icons, including a red 'R' icon, 'DD', and 'GW'. The main area of the window is a blue background with white text showing C code. The code includes function calls like gDPFillRectangle, gDPPipeSync, gDPSetColorImage, gDPSetFillColor, gDPFullSync, gSPEndDisplayList, and osWritebackDCache. Line numbers 0111 through 0140 are visible on the left side of the code. At the bottom of the window, there is a status bar with the text "1:Module 2:OptWin 3:SrcSW 4:Search 5:Go 6:Inspct 7:Come 8:Trace 9:Break 10:Step".

```
0111| gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0112|
0113| /**** Clear framebuffer cvg = FULL or 1 ****/
0114| gDPPipeSync(glistp++);
0115| gDPSetColorImage(glistp++, G_IM_FMT_RGBA, G_IM_SIZ_16b, SCREEN_WD,
0116|                 osVirtualToPhysical(i->cfb));
0117| /* clearcolor is 32-bits (2 pixels wide) because fill mode
0118|  * does two pixels at a time.
0119|  */
0120| gDPSetFillColor(glistp++, (GPACK_RGBA5551(0, 0, 0, 1) << 16 |
0121|                          GPACK_RGBA5551(0, 0, 0, 1)));
0122| gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0123|
0124| gDPPipeSync(glistp++);
0125| gDPSetCycleType(glistp++, G_CYC_1CYCLE);
0126|
0127| assert((void *)glistp < (void *)&i->msq);
0128|
0129| /**** Draw objects */
0130| doLogo(dynamicp);
0131|
0132|
0133| /**** Put an end on the top-level display list ****/
0134| gDPFullSync(glistp++);
0135| gSPEndDisplayList(glistp++);
0136|
0137| /* Flush the dynamic segment */
0138| osWritebackDCache(&i->dp, (s32)glistp - (s32)&i->dp);
0139|
0140| /* build graphics task */
```

1:Module 2:OptWin 3:SrcSW 4:Search 5:Go 6:Inspct 7:Come 8:Trace 9:Break 10:Step

Next, run the program.

```

0111| gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0112|
0113| /**** Clear framebuffer cvg = FULL or 1 ****/
0114| gDPPipeSync(glistp++);
0115| gDPSetColorImage(glistp++, G_IM_FMT_RGBA, G_IM_SIZ_16b, SCREEN_WD,
0116|                 osVirtualToPhysical(i->cfb));
0117| /* clearcolor is 32-bits (2 pixels wide) because fill mode
0118|  * does two pixels at a time.
0119|  */
0120| gDPSetFillColor(glistp++, (GPACK_RGBA5551(0, 0, 0, 1) << 16 |
0121|                          GPACK_RGBA5551(0, 0, 0, 1)));
0122| gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0123|
0124| gDPPipeSync(glistp++);
0125| gDPSetCycleType(glistp++, G_CYC_1CYCLE);
0126|
0127| assert((void *)glistp < (void *)&i->msg);
0128|
0129| /**** Draw objects */
0130| doLogo(dynamicp);
0131|
0132|
0133| /**** Put an end on the top-level display list ****/
0134| gDPFullSync(glistp++);
0135| gSPEndDisplayList(glistp++);
0136|
0137| /* Flush the dynamic segment */
0138| osWritebackDCache(&i->dp, (s32)glistp - (s32)&i->dp);
0139|
0140| /* build graphics task */

```

1:Module 2:OptWin 3:SrcSW 4:Search 5:Go 6:Inspct 7:Come 8:Trace 9:Break 10:Step

After running, the color of the line which set the breakpoint is highlighted and the program breaks. When you free the breakpoint, click the line number with the left mouse button in the same way as with the setting. After clicking, the underline of the line set to the breakpoint disappears. Now we have removed the breakpoint.

2. The step execution/ the trace execution/ the trace execution of the program.

First, execute the contents explained in "1. Setting/freeing the breakpoint." After the break, do the step execution.

For the step execution click "Run(R)" - "Step(S) F/O" in the PARTNER main menu, or the step icon on the tool bar with the left mouse button.



After clicking, the line that was highlighted in the code window proceeds in the program process. This shows that the STEP 2 execution is being done.

```
0111| gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0112|
0113| /**** Clear framebuffer cvg = FULL or 1 ****/
0114| gDPPipeSync(glistp++);
0115| gDPSetColorImage(glistp++, G_IM_FMT_RGBA, G_IM_SIZ_16b, SCREEN_WD,
0116|                 osVirtualToPhysical(i->cfb));
0117| /* clearcolor is 32-bits (2 pixels wide) because fill mode
0118|  * does two pixels at a time.
0119|  */
0120| gDPSetFillColor(glistp++, (GPACK_RGBA5551(0, 0, 0, 1) << 16 |
0121|                           GPACK_RGBA5551(0, 0, 0, 1)));
0122| gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0123|
0124| gDPPipeSync(glistp++);
0125| gDPSetCycleType(glistp++, G_CYC_1CYCLE);
0126|
0127| assert((void *)glistp < (void *)&i->msq);
0128|
0129| /**** Draw objects */
0130| doLogo(dynamiccp);
0131|
0132|
0133| /**** Put an end on the top-level display list ****/
0134| gDPFullSync(glistp++);
0135| gSPEndDisplayList(glistp++);
0136|
0137| /* Flush the dynamic segment */
0138| osWritebackDCache(&i->dp, (s32)glistp - (s32)&i->dp);
0139|
0140| /* build graphics task */
```

1:Module 2:OptWin 3:SrcSW 4:Search 5:Go 6:Inspct 7:Come 8:Trace 9:Break 10:Step

Highlighted line moves

The trace execution is also done in the same way as the STEP 2 execution. The difference between this and the STEP 2 execution is that if the function call is provided, the trace execution goes into the function. However, the STEP 2 execution carries out the function call and does not enter the function.

3. The dump/editing

This section describes the [dump](#)/editing functions. First, execute the contents explained in "1. Setting/freeing the breakpoint". After the break, do the dump. The dump has two kinds of methods, using "d command" from the command window and using the memory window.

Next, try to dump from the command window. In "createGfxTask()", the data is set up in the argument, "glistp". Proceed with dumping this content. Input the following in the command window.

>d glistp

After inputting, the 16-byte dump data is displayed in the command window. If you do a dump follow this by simply entering the "d command". The dump data is displayed in units of 16-bytes.

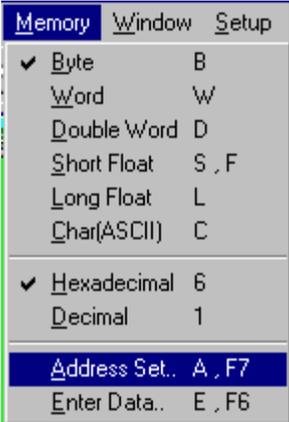
```

Command
>d glistp
800A6890  80 08 30 70 00 00 00 00  00 00 00 00 00 00 00 00  ..0p.....
>d
800A68A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>d
800A68B0  00 00 00 00 00 00 02 E0  80 02 39 60 80 02 39 60  .....à.9`..9`
>d
800A68C0  00 00 00 01 00 00 00 00  00 00 00 01 80 03 07 70  .....p
>d
800A68D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>d glistp
800A6890  80 08 30 80 00 00 00 00  00 00 00 00 00 00 00 00  ..0.....
>d
800A68A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>d
800A68B0  00 00 00 00 00 00 02 E0  80 02 39 60 80 02 39 60  .....à.9`..9`
>d
800A68C0  00 00 00 01 00 00 00 00  00 00 00 01 80 03 07 70  .....p
>d
800A68D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>

```

Though we described here only the dump in units of bytes, you can dump with various sizes using "d command". Try out some of these. Next, we will do a dump from the memory window. Open the memory window. Bring the mouse cursor into the memory window and right click. A pull-down menu is displayed. Select "Setting address(A)A, F7" and click it. (Either the right or left button is OK.)

After clicking, the address-specify window is displayed. Input "glistp" in "Address (A)" and click the "OK" icon with the left mouse button.



After clicking, the dump data is displayed in the memory window.

Memory																	
800A6870	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6880	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6890	80	08	30	80	00	00	00	00	00	00	00	00	00	00	00	00	..0.....
800A68A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A68B0	00	00	00	00	00	00	02	E0	80	02	39	60	80	02	39	60 9` .9`
800A68C0	00	00	00	01	00	00	00	00	00	00	00	01	80	03	07	70 p
800A68D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A68E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A68F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6900	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6910	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6920	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6930	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6940	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6950	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6960	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6970	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6980	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6990	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A69A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A69B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A69C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A69D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A69E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A69F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
800A6A00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Next, do editing. Editing, similar to the dump, also has two kinds of methods, using "e command" from the command window and using the memory window. First, edit from the command window. With the editing, we will use the argument "glistp" as well. Input the following in the command window.

```
>e glistp
```

After inputting, the input status changes to wait. Input "ff", and input "." next ("." specifies the completion of editing)

```

Command
>d
800A68B0  00 00 00 00 00 00 02 E0  80 02 39 60 80 02 39 60  .....à.9`.9`
>d
800A68C0  00 00 00 01 00 00 00 00  00 00 00 01 80 03 07 70  .....p
>d
800A68D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>d glistp
800A6890  80 08 30 80 00 00 00 00  00 00 00 00 00 00 00 00  ..0.....
>d
800A68A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>d
800A68B0  00 00 00 00 00 00 02 E0  80 02 39 60 80 02 39 60  .....à.9`.9`
>d
800A68C0  00 00 00 01 00 00 00 00  00 00 00 01 80 03 07 70  .....p
>d
800A68D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
>e glistp
address  asc oct  dec hex data
800A6890  '.' 200 -128 80 ff
800A6891  '.' 010   8 08 .
>

```

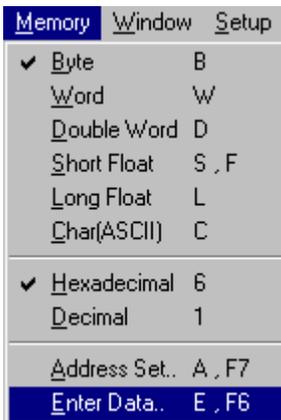
Though we have only described the editing in units of bytes here, you can edit with various sizes using "e command". Try out some of these. Next, we will do editing from the memory window. Open the memory window in the same way as with the dump. Here, make sure the first byte of "glistp" is "FF". The reason for this is that the contents of "glistp" were changed by using "e command" in the command window before opening the memory window.

```

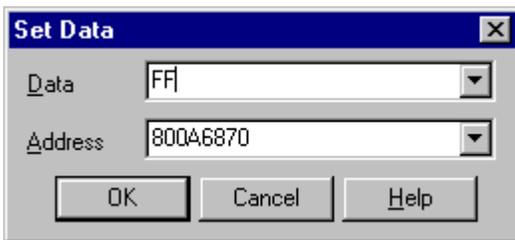
Memory
800A6890  FF 08 30 68 00 00 00 00  00 00 00 00 00 00 00 00  ..0h.....
800A68A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A68B0  00 00 00 00 00 00 02 E0  80 02 39 60 80 02 39 60  .....à.9`.9`
800A68C0  00 00 00 01 00 00 00 00  00 00 00 01 80 03 07 70  .....
800A68D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A68E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A68F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6900  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6910  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6920  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6930  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6940  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6950  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6960  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
800A6970  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....

```

Here, to return "glistp" to its original content, we will edit in the memory window. Bring the mouse cursor to the first byte in the memory window and click the right button of the mouse. A pull-down menu is displayed. Select "Data change(E) E, F6" and click. (Either the right or left button is OK)



After clicking, the data set window is displayed and the value "FF" is entered in "Data (D)."



Input "80" in "Data (D)" and click the "OK" icon with the left mouse button. After clicking, try to dump "glistp" again. You will find that the first byte is "80".

4. Refer to the Argument of C

This section describes about the source code debug function to reference the arguments of C. First, execute the contents described in "1. Setting/freeing the breakpoint." After the break, we will reference the argument.

First of all, let's try to provide the reference using the inspect window.

In "createGfxTask()", the local argument "dynamicp" is passed to the function "doLogo()". Set the mouse cursor on "dynamicp" in the code window, and click the right button of the mouse. A pull-down menu is displayed. Select "Inspect(I) F6, Ctrl+I" and click. (Either the right or left button is OK.)

```

0123 |
0124 | gDPPipeSync(glistp++);
0125 | gDPSetCycleType(glistp++, G_CYC_1CYCLE);
0126 |
0127 | assert((void *)glistp < (void *)&i->msg);
0128 |
0129 | /* Draw objects */
0130 | doLogo(d
0131 |
0132 |
0133 | /* Put an end on the top-level display list */
0134 | gDPFullSync(glistp++);
0135 | gSPEndDisplayList(glistp++);
0136 |
0137 | /* Flush the dynamic segment */
0138 | osWritebackDCache(&i->dp, (s32)glistp - (s32)&i->dp);
0139 |
0140 | /* build graphics task */
0141 |
0142 | t = &i->task;
0143 | t->list.t.data_ptr = (u64 *) dynamiccp->glist;
0144 | t->list.t.data_size = (s32)(glistp - dynamiccp->glist) * sizeof (Gfx);
0145 |
0146 | t->list.t.flags = 0x0;

```

Inspect	F6, Ctrl+I
Watch	Ctrl+W
Compare	F7
Disassemble	F3
Module..	F1
Line no.	Ctrl+QL
Top	Ctrl+QR
Bottom	Ctrl+QC
PC Location	Ctrl+QP
File Focus	
Function Focus	
Flip	

```

GFX.C
0121 | gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0122 |
0123 |
0124 | gDPPipeSync(glistp++);
0125 | gDPSetCycleType(glistp++, G_CYC_1CYCLE);
0126 |
0127 | assert((void *)glistp < (void *)&i->msg);
0128 |
0129 | /* Draw objects */
0130 | doLogo(d
0131 |
0132 |
0133 | /* Put an end on the top-level display list */
0134 | gDPFullSync(glistp++);
0135 | gSPEndDisplayList(glistp++);
0136 |
0137 | /* Flush the dynamic segment */
0138 | osWritebackDCache(&i->dp, (s32)glistp - (s32)&i->dp);
0139 |
0140 | /* build graphics task */
0141 |
0142 | t = &i->task;
0143 | t->list.t.data_ptr = (u64 *) dynamiccp->glist;
0144 | t->list.t.data_size = (s32)(glistp - dynamiccp->glist) * sizeof (Gf

```

```

Inspect(@8002C8C8)
(struct *)dynamiccp = *80082DC8
[0] = @80082DC8 {projection= {m= { {0,0,0,0}, {0,0,0,0},

```

After clicking, the inspect window is displayed and you can look up the contents of "dynamiccp". The inspect window is displayed in horizontal lines. If it's hard to see, double-click the @ address in the inspect window with the left mouse button. You can display them vertically.

```

0121 |                                     GPACK_RGBA5551(0, 0, 0, 1));
0122 | gDPFillRectangle(glistp++, 0, 0, SCREEN_WD-1, SCREEN_HT-1);
0123 |
0124 | gDPPipeSync(glistp++);
0125 | gDPSetCycleType(glistp++, G_CVC_1CYCLE);
0126 |
0127 | assert((void *)glistp < (void *)&i->msg);
0128 |
0129 | doLogo(dynamicp);
0130 |
0131 |
0132 |
0133 | /* **** Put an end on the top-level display list **** */
0134 | gDPFullSync(glistp++);
0135 | gSPEndDisplayList(glistp++);
0136 |
0137 | /* Flush the dynamic display list to the GPU */
0138 | osWritebackDCache(&i->dp);
0139 |
0140 | /* build graphics task */
0141 |
0142 | t = &i->task;
0143 | t->list.t.data_ptr = (u64 *) dynamicp->glist;
0144 | t->list.t.data_size = (s32)(glistp - dynamicp->glist) * sizeof (Gfx);

```

Inspect[@8002CBC8]

```
(struct *)dynamicp = *80082DC8
[0] = @80082DC8 {projection= {m= { {0,0,0,0}, {0,0,0,0},
```

Inspect[@80082DC8]

```
(struct *)dynamicp = @80082DC8 {projection= {m= { {0,0,0,0},
(union )projection = @80082DC8 {m= { {0,0,0,0}, {0,0,0,0},
(union )viewing = @80082E08 {m= { {0,0,0,0}, {0,0,0,0},
(union )bg_model = @80082E48 {m= { {0,0,0,0}, {0,0,0,0},
(union )logo_scale = @80082E88 {m= { {0,0,0,0}, {0,0,0,0},
(union )logo_rotate = @80082EC8 {m= { {0,0,0,0}, {0,0,0,0},
(union )logo_trans = @80082F08 {m= { {0,0,0,0}, {0,0,0,0},
(union [512])glist = @80082F48 { {words= {w0=BC000006,w1
```

Next, we will provide the reference using the watch window. The operation of the watch window is basically the same as the inspect window. Set the cursor on "glistp" in the code window and click the right button of the mouse. A pull-down menu is displayed. Select "Watch (W) Ctrl+W", and click. (Either the right or left button is OK.)

```

0129 | /* **** Draw objects **** */
0130 | doLogo(dynamicp);
0131 |
0132 |
0133 | /* **** Put an end on the top-level display list **** */
0134 | gDPFullSync(glistp++);
0135 | gSPEndDisplayList(glistp++);
0136 |
0137 | /* Flush the dynamic display list to the GPU */
0138 | osWritebackDCache(&i->dp);
0139 |
0140 | /* build graphics task */
0141 |
0142 | t = &i->task;
0143 | t->list.t.data_ptr = (u64 *) dynamicp->glist;
0144 | t->list.t.data_size = (s32)(glistp - dynamicp->glist) * sizeof (Gfx);
0145 | t->list.t.type = M_GFX;
0146 | t->list.t.flags = 0x0;
0147 | t->list.t.ucode_boot = 0;
0148 | t->list.t.ucode_boot_start = 0;

```

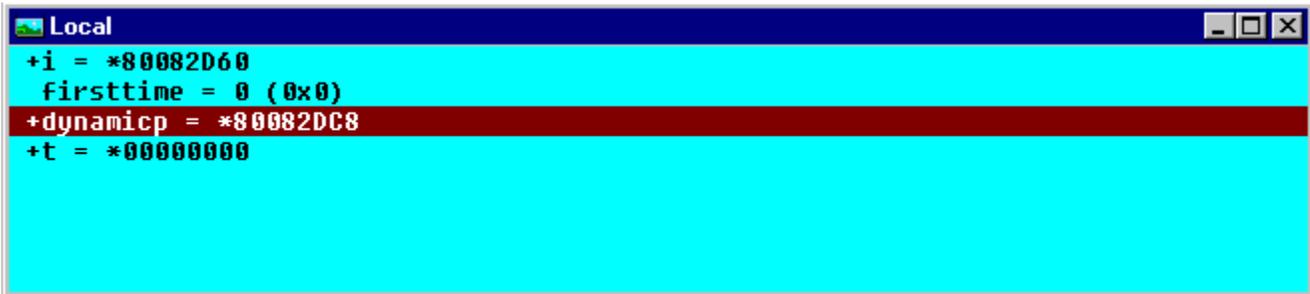
Inspect	F6, Ctrl+I
Watch	Ctrl+W
Come	F7
Disassemble	F3
Module..	F1
Line no.	Ctrl+QL
Top	Ctrl+QR
Bottom	Ctrl+QC
PC Location	Ctrl+QP
File Focus	
Function Focus	

After clicking, the watch window is displayed, and you can look up the contents of "glistp"



The watch window can also be displayed vertically, in the same way as the inspect window, by double-clicking the @ address or * address in the window with the left mouse button.

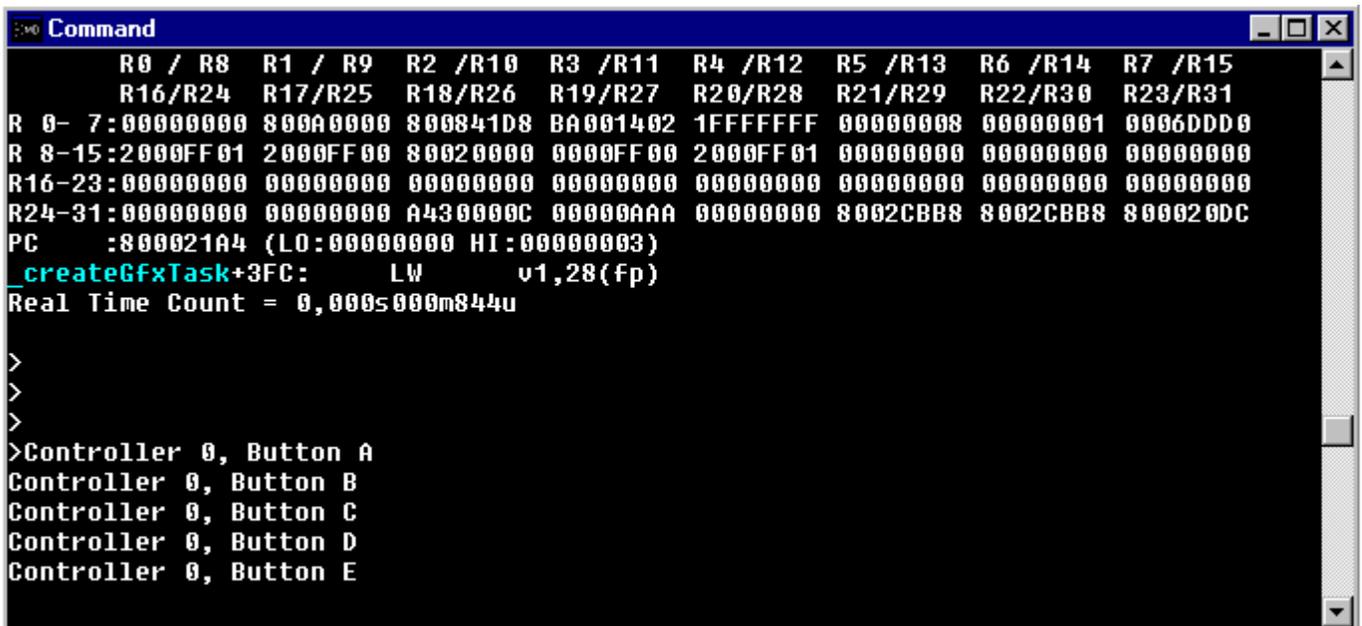
Finally, if you want to look up the local argument, you can do so by opening the local window. It is not necessary to do the inspect and watch.



The local window can also be displayed vertically, in the same way as the inspect window, by double-clicking the @ address or * address in the window with the left mouse button.

5. The Debug Print

In PARTNER, you can look up the debug print during programming. To display the debug print, "osSyncPrintf()", set up in the N64 OS functions, is used. The output result of "osSyncPrintf()" is displayed in the command window. "simple" is the program which displays the debug print when the A, B and C buttons of the controller are entered. After loading "simple", execute it without setting the breakpoint. When you enter the controller button during the execution of the program, the debug print is displayed in the command window.



All of the above are examples of simple debugs. PARTNER has other various functions, so you will be able to find the most suitable debugging method from among them.

Introduction to **NINTENDO⁶⁴**

Chapter 7 PARTNER-N64 NW

This chapter explains about the setup of "PARTNER-N64 NW."



PARTNER-N64 Product System

	PARTNER-N64 NW		PARTNER-N64 PC	
The host machine	SGI/INDY SGI/O2		DOS/V PC-98 series	
OS	IRIX5.3 IRIX6.2, 6.3		Windows3.1 Windows95	
The compiler	exeGCC N64 for SGI IDO5.3 IDO7.1 for 6.2 IDO7.1 for 6.3		exeGCC N64 for PC	
The connecting method	10BASE-T 10BASE-2		The special purpose parallel interface	
The model	Model 10	Model 11	Model 10	Model 11
The emulation ROM	16MByte	32MByte	16MByte	32MByte



7-1 Outline of Tools

The purpose of Partner-N64 NW is to develop/debug the N64 application on Indy or O2 workstations. (This manual covers "Model 11.")

7-1-1 The configuration of PARTNER-N64 NW

1. N64 OS

N64 OS, 1 piece (DAT tape) <provided by NINTENDO>

2. PARTNER-N64 NW

The PARTNER-N64 NW debugger body

The specialized N64 alteration machine (includes the AC adapter and controller)

The network adapter

The cable for the network adapter

The power supply for the network adapter

The power cable for the network adapter

PARTNER-N64 (SGI), 1 piece (CD)

PARTNER-N64 NW user's manual, 1 volume

The user attribute card, 1 piece

7-2 Operating Environment

You need the following environment to use the PARTNER-N64 NW.

1. Machine Indy or the O2 workstation

2. OS Indy : IRIX 5.3, 6.2

O2 : IRIX 6.3

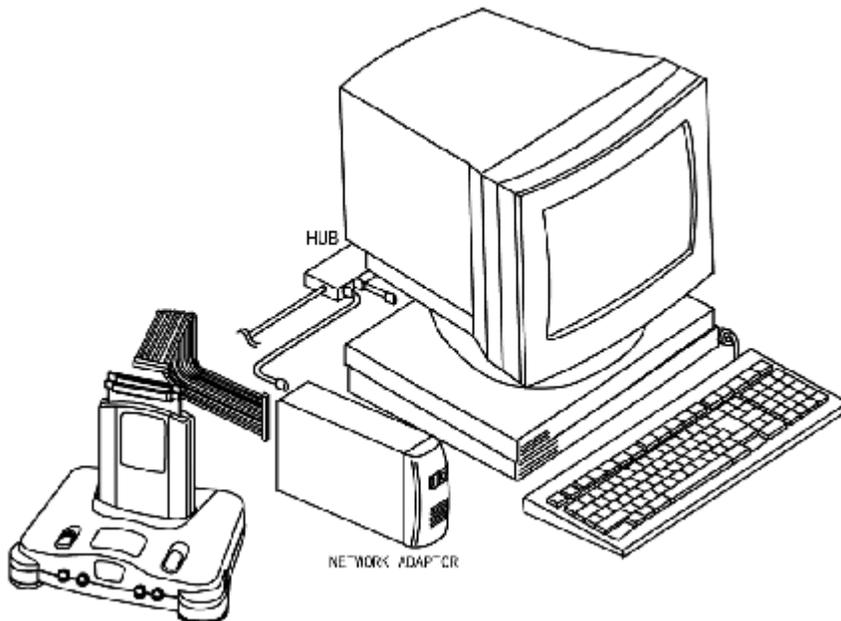


Figure 7-2-1 The PARTNER-N64 NW

7-3 Setting up

7-3-1 Setting up the Hardware

1. Setting the Network Adapter

Connect the specialized power supply and power cable to the network adapter. Next, set up the IP address to the network adapter by using the serial terminal. After verifying that **the power** of the network adapter and the serial terminal **are OFF**, connect them. Apply power to the network adapter and the serial terminal in that order.

After setting the IP address, **turn power** of the network adapter **to OFF** and connect it with Indy or the O2 workstation with the network cable.

2. Mounting PARTNER-N64

Insert PARTNER-N64 into the cassette mouth of the N64 alteration machine.

3. Connecting the Network adapter

Connect the 40-pin connector of the network adapter with the I/F connector (40-pin) of PARTNER-N64 using the specialized cable for the network adapter.

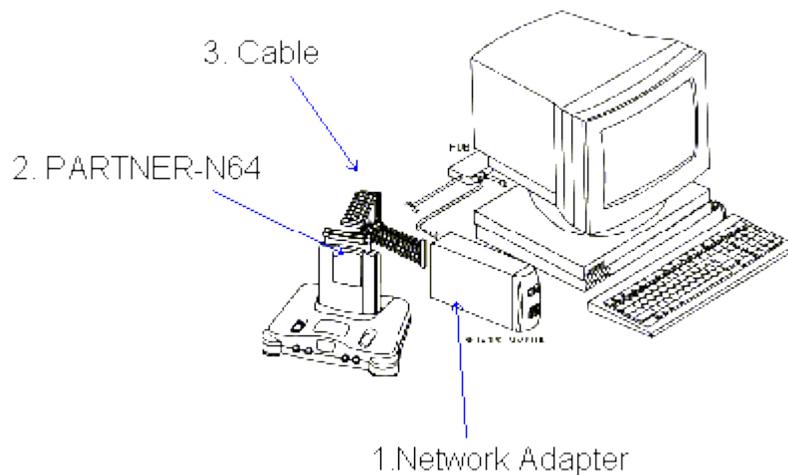


Figure 7-3-1 Setting up the hardware

4. Mounting the Controller, Video Cable and AC adapter

After verifying that the N64 **power switch is OFF**, connect the controller, the video cable (purchased separately) and the AC adapter. Connect the pin plug (RCA) side of the video cable to the television, and the AC plug side of the AC adapter to the outlet (AC Outlet).

7-3-2 Setting up the Software

In this manual, the compiler (cc), the assembler (as) and the linker (ld) required to develop the application should be already installed. For details, see Chapter 1, Section 2 [N64 Emulator software installation] from the Programming Manual. (However, you do not need to install gvd debugger.)

1. The installation of N64 OS

Install the include file, library, [microcodes](#) and sample program required to use the N64 OS on the N64 application. After inserting the DAT tape, go to the appropriate directory on the console and enter

```
% tar xv
```

(contents of DAT are expanded to the directory).

Next click "Software Manager" from the "System" menu with the left mouse button and activate "Software Manager". After activation, designate the directory the contents of DAT were expanded to and click "Customize Installation" with the left mouse button.

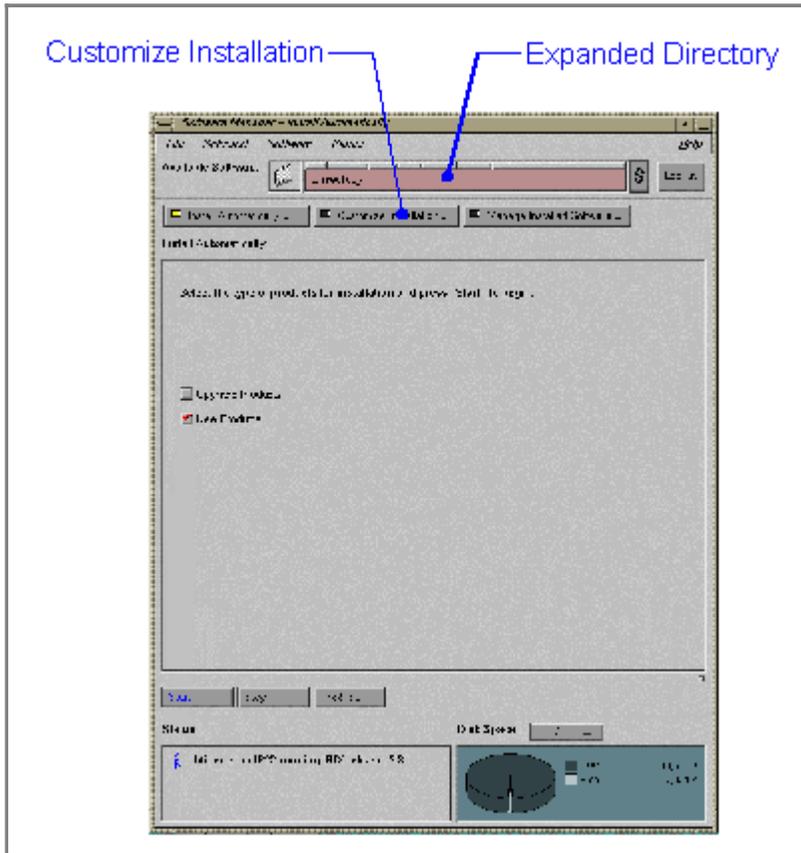
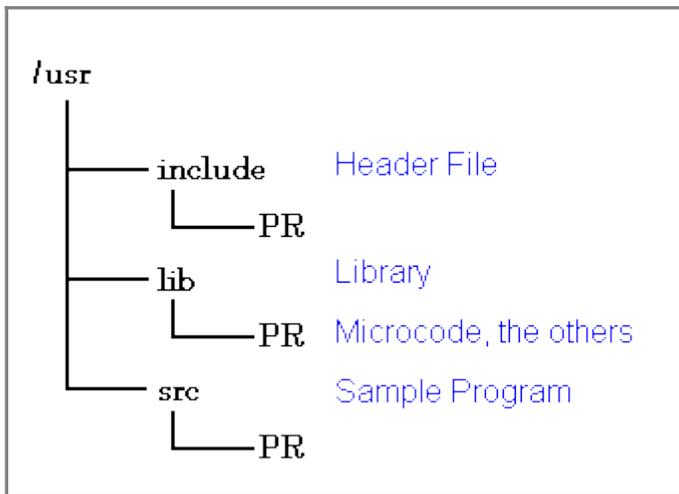


Figure 7-3-2 Installation of the N64 OS

The installable contents, will be displayed so after selecting all (check the Install field), push "Start." When the installation ends without error, the Status field changes from "New" to "Same Version."

The directory configuration after the installation will appear as seen below:



1. The Installation of PARTNER-N64 (SGI)

Install debugger for debugging the created software.

Insert the PARTNER-N64 (SGI) Disk into the CD drive. On the CD, the "tar" format has been set up. Decompress the file within the appropriate directory. After decompressing, the shell script file "ptinst" for the installation is created. Execute "ptinst" from the console (the standard place for the installation is "/usr/local/partner-net/n64").

% ptinst

PARTNER-N64 (SGI) is compatible with "Software Manager" starting from version 1.06.

7-4 Tutorial

By using an actual sample this section describes the program compilation/link, creation of the ROM image file, and the debugging method. Before you use the sample, we will describe the procedure for creating the ROM image file. Proceed to the next step after you have an understanding of this.

7-4-1 Procedure for creating the ROM image file

The ROM image file is created by designating the object file and the picture/sound data to the ROM image creation tool after the **relocationable object file** is created one time using the compiler/linker. (**Relocationable object file: Object files where the function and argument addresses have not been determined**). The ROM image creation tool outputs the ROM image file and the debugger symbol file. When a name for the ROM image file is not specified it is output using the name, "rom". Specify the symbol file name within "beginwave" and "endwave" of the spec file specified to the ROM image creation tool.

7-4-2 The compilation/link/creation of the ROM image using an actual sample.

Now let us do the compilation/link/creating the ROM image of the actual sample. Here, we will use the sample called, "simple". "simple" has been installed in the directory, "/usr/src/PR/demos/simple". Copy it into the appropriate directory from the console. For example, you can make the directory, "simple". copied into the current directory by executing

```
% cp -r /usr/src/PR/demos/simple .
```

1. Makefile

The compilation/link/ROM image creation on "simple" are provided by using the program, "make".

"make" provides the process based on the contents defined in the text file, "Makefile" (see the online manual, "make", for details).

Now, we will describe the contents defined in "Makefile". Open the text file, "Makefile", in the directory, "simple", with your editor (the actual "Makefile" does not describe the following **#comments**).

[Makefile]

```
#!/smake
include $(ROOT)/usr/include/make/PRdefs # Files reserved in the system [defining
# the compiler name and
# dependency, etc.

# to make the tags file do "make simpletags"
# to make just the simple_d directory do "make SUBDIRS=simple_d"

SUBDIRS=simple_d simple simple_rom # Specify the subdirectory
# In "simple", you create the following three types of ROM image files
# and symbol files
# 1. For debugging simple_d
# 2. For ordinary use simple
# 3. For master submitting simple_rom

COMMONPREF = simple # unused

APP = simple # Specifying the symbol file name
TARGETS = rom # Specifies the ROM image file name

HFILES = \ # Specifies the header file name for "simple"
:
CODEFILES = \ # Specifies the .c (the program code) file name
:
CODEOBJECTS = $(CODEFILES:.c=.o)
# Specifies (the program code) file name
# (The name is changed from .c to .o, which is
# the file name specified in CODEFIELES
# Example) test.c -> test.o

CODESEGMENT = codesegment.o # Specifies the relocationable
# object file name created as a
# result of linking the program code
```

```

# Data files that have their own segments:

DATAFILES = \ # Specifies the .o(data) file name
:
DATAOBJECTS = $(DATAFILES:.c=.o)
# Specifies the .o (data code) file name
# (Same as CODDOBJECTS above)

OBJECTS = $(CODESEGMENT)$$(DATAOBJECTS)
# Specifies all relocationable object file names
# Specifies to the ROM image creation tool

LCINCS = -I. -I$(ROOT)/usr/include/PR
# Specifies the passing of the include file specified to the compiler (for local use)
LCOPTS = $(DFLAG) -fullwarn -non_shared -G 0 -Xcpluscomm
# Designates the option to specify to the compiler (for local use)
LCDEFS =
# Specifies the symbol definition to specify to the compiler (for local use)
# Unused in "simple"

LDIRT = load.map # Specifies the map file name

LDFLAGS = $(MKDEPOPT) -nostdlib -L$(ROOT)/usr/lib
-L$(ROOT)/usr/lib/PR -I$(ULTRALIB)
# Designates the option to specify to the linker
# MKDEPOPT is the reserved name and the file defined to dependent
# relation of each object file.
# "-nostdlib" is a specification which is not use the standard library
# "-L$(ROOT)/usr/lib -L$(ROOT)/usr/lib/PR -I$(ULTRALIB)" is the
# option to link the N64 OS library.

.PATH: .. # The path specification of "simple"
# "simple" is to execute "make" in each subdirectory.
# The file reserved in the system..
sinclude locdefs

#include $(COMMONRULES) # Note that "#" is a comment.

default:z # Specifies to provide the following
# process with default in this "Makefile".
for i in $(SUBDIRS) ; do \ # Specifies to go to each subdirectory
echo ==== $$i ==== ; \ # and execute "make".
cd $$i ; \
$(MAKE) -f ../Makefile loc_$$i ; \
cd .. ; \
done

$(COMMONTARGS) : # Specifies to provide the process when it specifies
# the reserved command in the system to "make"
# Example) make clean
for i in $(SUBDIRS) ; do \ # default: the same
echo ==== clobber $$i ==== ; \
cd $$i ; \
$(MAKE) -f ../Makefile loc_$$@ ; \
cd .. ; \
done

```

```
include $(COMMONRULES) # Accept the reserved file in the
# system
```

```
install: default # Specifies to install "simple"
# The files below are installed.
# Before executing this, execute the default
# if it is not executed.
$(INSTALL) -m 444 -F /usr/src/PR/demos/simple \
$(HFILES) $(CODEFILES) \
$(DATAFILES) Makefile spec \
simple/log.fmt simple_d/locdefs \
simple/locdefs simple_rom/locdefs
```

```
$(CODESEGMENT): $(CODEOBJECTS)
# Specifies the dependent relation between "codesegment.o" and the .o file.
# If the .o file is updated, provide the following process.
$(LD) -m -o $(CODESEGMENT) -r $(CODEOBJECTS) \
$(LD_FLAGS) > load.map
# Specifies to create the relocationable object file by using the linker.
# "-m" specifies to output the map file.
# "-o" $(CODESEGMENT) is an option to specify the output file name.
# "-r" is the option to create the relocationable object file.
# "$(CODEOBJECTS)" specifies the linking object file name.
# "$(LD_FLAGS)" specifies to pass other options to the linker.
```

```
rom: ../spec $(OBJECTS)
# Specifies the ROM image file, all ".o" files and the dependent relation of the spec file
# Provide the following process when the .o file and the spec file are updated
$(MAKEROM) $(MAKEROMDEFS) ../spec
# Specifies to create the ROM image file by using the ROM image creation tool.
# "$(MAKEROMDEFS)" is the standard option specified to the image creation tool.
# "spec" is the text file to specify the ROM image to the ROM image creation tool.
# We will describe this later.
```

The contents mentioned above are just one example. The compiler/linker/ROM image creation tool has many other convenient functions. Utilize them based on what the program can be used for.

2. Specifying the ROM Image

Next, we will describe the contents defined in the script file to specify the ROM image. Open the text file, "spec", in the directory, "simple", with your editor (in the actual "spec", the */*Comments*/* below are not described). Read the "makerom" online manual along with this explanation.

[spec]

```
/* The ROM image manages in units of segments.*/
/* Define the segments having the program code attributes.*/
beginseg /* Initiate to define the segments*/
name "code" /* Specify the segment names */
flags BOOT OBJECT /* Designate the boot attribute and the object attribute */
entry boot /* Specify the boot function */
stack bootStack + STACKSIZEBYTES
/* Specify the stack used by the boot function */
include "codesegment.o"
/* Specify the object file mapping within the segment */
```

```

include "$(ROOT)/usr/lib/PR/rspboot.o"
/* Specify the boot microcode*/
include "$(ROOT)/usr/lib/PR/gspFast3D.o"
/* Specify the graphic microcode */
include "$(ROOT)/usr/lib/PR/gspFast3D.dram.o"
/* Specify the graphic microcode */
include "$(ROOT)/usr/lib/PR/aspMain.o"
/* Specify the sound microcode */
endseg /* End segment definitions */
/* The following is a description only about the parts which do not overlap with the "code" segments.
*/
beginseg
name "gfxdlists"
flags OBJECT /* Designate the object attribute */
after code /* Specify mapping right after the "code" segment. */
include "gfxdlists.o"
endseg
beginseg
name "zbuffer"
flags OBJECT /* Designate the object attribute */
address 0x801da800 /* Specify mapping on the 0x801da800 address*/
include "gfxzbuffer.o"
endseg

beginseg
name "cfb"
flags OBJECT /* Designate the object attribute */
address 0x80200000 /* Specify mapping on the 0x80200000 address*/
include "gfxcfb.o"
endseg
beginseg
name "static"
flags OBJECT /* Designate the object attribute */
number STATIC_SEGMENT /* Specify the static segment number */
include "gfxinit.o"
include "gfxstatic.o"
endseg
beginseg
name "dynamic"
flags OBJECT /* Designate the data attribute */
number DYNAMIC_SEGMENT /* Specify the dynamic segment number */
include "gfxdynamic.o"
endseg
beginseg
name "bank"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/soundbanks/GenMidiBank.cti"
/* Specify the sound bank data */
endseg
beginseg
name "table"
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/soundbanks/GenMidiBank.tbl"
/* Specify the sound table data. */
endseg
beginseg
name "seq"

```

```
flags RAW /* Designate the data attribute */
include "$(ROOT)/usr/lib/PR/sequences/BassDrive.seq"
/* Specify the sound sequence data */
endseg

beginwave /* Initiate to define waves */
name "simple" /* Specify the symbol file name (ignore ".out") */
include "code" /* The following are the specification of mapping segments. */
include "gfxdlists"
include "static"
include "dynamic"
include "cfb"
include "zbuffer"
include "table"
include "bank"
include "seq"
endwave /* End the wave definition */
```

The preceding is the required procedure to do the compilation/link/ROM image creation for the sample with "make".

3. Executing "make"

Now, we will actually execute, "make". Execute "make" from the directory, "simple".

```
% make
```

After completion of "make" in three subdirectories, the ROM image file, "rom" and the symbol file, "simple" are created. Verify this with the "ls" command, etc.

7-4-3 Executing the Sample

Execute the sample by using the ROM image and symbol files which were created in "5-4-2: The Compilation/Link/ROM Image Creation of the Sample" (you use "simple" again here). Do the execution of the sample by using the debugger (The rest is called "PARTNER"). Continue with the following procedure:

1. Move the Directory

Move to the subdirectory "simple_d" of "simple".

```
% cd simple_d
```

2. Activate PARTNER

In the moved directory, enter the shell script "ptn64" for the "PARTNER" activation.

```
% ptn64
```

Note: Before the "PARTNER" activation, specify the directory of, "simple", to the environment argument, "PTSRC".

3. Load the File

Enter the "l" command" in the command window.

```
>l simple
```

4. The Execution

Enter the "g command" or the F5 key in the command window.

5. End the Program

Press the ESC key.

7-4-4 Debug the Sample

Here we will describe some debugging examples (the operation methods of PARTNER) using the sample. We use "simple" here again.

1. Setting/Freeing the Breakpoint

We will describe the setting method of the breakpoint. Set the breakpoint in "simple" - "gfx.c" - "createGfxTask()".

After loading the sample, press the F1 key. After you press the key, the window, "Select reference file name", is displayed. Select "gfx.c" with the cursor key, and press the "Enter" key,

```
Select reference file name
epirawread.c    epirawwrite.c    error.c          errorasm.s
event.c        exceptasm.s      filter.c         getactivequeue.c
getcause.s     getcount.s       getsr.s          getthreadpri.c
gettime.c      gfx.c            heapalloc.c      heapinit.c
```

After the end of the module selection, the contents of "gfx.c" are displayed in the command window of PARTNER. Scroll sources until you reach, "createGfxTask()", by using the PageDown key or the scroll key. Line numbers are displayed at the left side of the code window. Scroll until you reach between the 110th and 130th lines.

On the 127th line of "gfx.c", there is a place which is bringing up the assert() function. Bring the mouse cursor to the line number and click on it with the left button of the mouse. Then a line is displayed under the line bringing up the assert() function from the code window. Now the breakpoint is set.

```

PARTNER
Code
0102
0103  /*** clear zbuffer, z = max z, dz = 0 ***/
0104  glDepthRange(glistp, osVirtualToPhysical(zbuffer));
0105  glDepthFunc(glistp);
0106  glDepthFunc(glistp, GL_DEPTH_FILL);
0107  glDepthRange(glistp, GL_DEPTH_RANGE, GL_DEPTH_BITS, SCREEN_WD,
0108               osVirtualToPhysical(zbuffer));
0109  glClearColor(glistp, (GLfloat)(G_MAXFZ, 0) << 16 |
0110                 GLfloat(G_MAXFZ, 0));
0111  glClearColor(glistp, 0, 0, SCREEN_WD-1, SCREEN_HF-1);
0112
0113  /*** Clear framebuffer cvg = FULL or 1 ***/
0114  glDepthFunc(glistp);
0115  glDepthRange(glistp, GL_DEPTH_RANGE, GL_DEPTH_BITS, SCREEN_WD,
0116               osVirtualToPhysical(1.0f));
0117  /* Clearcolor is 32-bits (2 pixels wide) because fill mode
0118   * does two pixels at a time.
0119   */
0120  glClearColor(glistp, (GLfloat)RGBE51(0, 0, 0, 1) << 16 |
0121               GLfloat)RGBE51(0, 0, 0, 1));
0122  glClearColor(glistp, 0, 0, SCREEN_WD-1, SCREEN_HF-1);
0123
0124  glDepthFunc(glistp);
0125  glDepthFunc(glistp, GL_DEPTH_FILL);
0126
0127  assert((void *)glistp < (void *)0);
0128
0129  /*** Draw objects */
0130  doLogo(dynamic);
0131

```

Next, execute the program.

```

PARTNER
Software break point NO. 1
0102
0103  /*** clear zbuffer, z = max z, dz = 0 ***/
0104  glDepthRange(glistp, osVirtualToPhysical(zbuffer));
0105  glDepthFunc(glistp);
0106  glDepthFunc(glistp, GL_DEPTH_FILL);
0107  glDepthRange(glistp, GL_DEPTH_RANGE, GL_DEPTH_BITS, SCREEN_WD,
0108               osVirtualToPhysical(zbuffer));
0109  glClearColor(glistp, (GLfloat)(G_MAXFZ, 0) << 16 |
0110                 GLfloat(G_MAXFZ, 0));
0111  glClearColor(glistp, 0, 0, SCREEN_WD-1, SCREEN_HF-1);
0112
0113  /*** Clear framebuffer cvg = FULL or 1 ***/
0114  glDepthFunc(glistp);
0115  glDepthRange(glistp, GL_DEPTH_RANGE, GL_DEPTH_BITS, SCREEN_WD,
0116               osVirtualToPhysical(1.0f));
0117  /* Clearcolor is 32-bits (2 pixels wide) because fill mode
0118   * does two pixels at a time.
0119   */
0120  glClearColor(glistp, (GLfloat)RGBE51(0, 0, 0, 1) << 16 |
0121               GLfloat)RGBE51(0, 0, 0, 1));
0122  glClearColor(glistp, 0, 0, SCREEN_WD-1, SCREEN_HF-1);
0123
0124  glDepthFunc(glistp);
0125  glDepthFunc(glistp, GL_DEPTH_FILL);
0126
0127  assert((void *)glistp < (void *)0);
0128
0129  /*** Draw objects */
0130  doLogo(dynamic);
0131

```

After execution, the line which set the breakpoint is highlighted and the program breaks. When you free the breakpoint, click on the line number with the left mouse button in the same way as with the setting. After clicking, the underline of the line which set the breakpoint disappears. Now you have freed the breakpoint.

2. The Step Execution/ The Trace Execution

Next we will describe the step execution/the trace execution of the program. First execute the contents explained in "1. Setting/ Freeing the Breakpoint". After the break, provide the step execution.

The trace execution is done by entering the "t command" or F8 key in the command window. The difference

out some of these.

Next we will do editing. The dump is done by using the "e command" from the command window. The editing also uses the argument called "glistp". Do the following input in the command window.

>e glistp

After inputting, it will enter the input wait status. Enter "ff" and enter "." next. ("." specifies the end of editing).

```
Command
R24-31:8007A88 8007AA4C A43C000C 0C000AFA 0C0C0000 80076C08 0C0C0000 80001EC4
PC      :80001F8C (LI:000C0000 HI:000C0002)
cras:00070000+428:  LN      n25,70(n29)
Real Time Count = 0,000000n793u

>e glistp
address asc oct dec hex data
80071D24 'f' 200 -128 8C ff
80071D25 ' ' 007 7 07 .
>
```

After clicking, when you dump "glistp" again, you will find that the first byte becomes "Oxff".

Though we have only described the editing in units of bytes here, you can edit in various sizes with the "e command". Try out some of these.

4. Referring to the Argument of C

Next we will describe the source code debug function which refers to the arguments of C. First execute the contents explained in "1. Setting/Freeing the Breakpoint." After the break, we will carry out the argument reference.

First, let us try to reference using the inspect window. In "createGfxTask()", the local argument "dynamicp" is passed to the function "doLogo()". Set the mouse cursor to "dynamicp" on the code window, and double-click the left mouse button.

```
0127 | assert((void *)glistp < (void *)&i->msg);
0128 |
0129 | /**** Draw objects */
0130 | do_logo(@dynamicp);
0131 |
0132 |      inspect (@80076314)
0133 |      (struct *)dynamicp = *80079E30
0134 | /**** P [0] = @80C79800 ;projection= {m= : {C,0,0,C}, {C,0,0
0135 | gD-Full
0136 | gS-EndDisp Lay_list(glistp++);
```

After double-clicking, the inspect window is displayed and you can look up the contents of "dynamicp". The inspect window is displayed in horizontal lines. If it's hard to see, double-click the @ address in the inspect window with the left mouse button. You can display them vertically. You can also do the inspect by using the "ins command" from the command window.

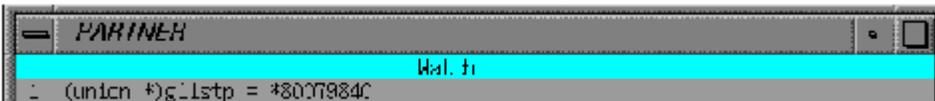
>ins dynamicp

```

0127| assert((void *)glistp < (void *)&);
0128|
0129| /*** Draw objects */
0130| dtLogo(dynamicp);
0131|      Inspect (@80078074)
0132|      (struct *)dynamicp = *80079630
0133|      [C] = @80079630 {projection= {m= { {0,0,0,0}, {0,C,0
0134|      zIFFull      Inspect (@80079630)
0135|      (<local>)*dynamicp = @80079970 {projection= {m= { {0,0
0136|      (union)projection = @80079630 :m= { {0,0,0,0}, {0,0
0137|      (union)viewing = @80079670 :m= { {0,C,0,0}, {C,0,0,
0138|      (union)kg_model = @800793BC :m= { {0,0,0,0}, {C,0,0
0139|      (union)kg_scale = @80079FF0 :m= { {0,0,0,0}, {0,0
0140|      /* build (union)logo_rotate = @80079730 :m= { {0,0,0,0}, {0
0141|      (union)logo_trans = @80079770 :m= { {0,0,0,0}, {0,0
0142|      t = &t (union)E12 :glist = @800737E0 :wsize= |u0=BCC0000
0143|      t->list
0144|      t->list.t.data_size = (s32)(glistp - dynamicp->glist) * sizeof (Gf*);

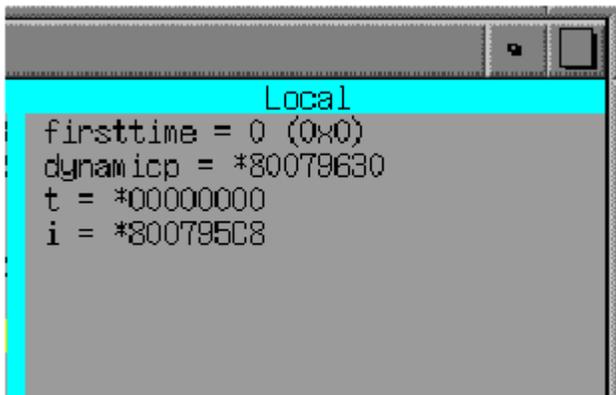
```

Next, let us provide the reference using the watch window. The operation of the watch window is basically the same as the inspect window. This time, try to look up the argument "glistp". Set the mouse cursor on "dynamicp" in the code window, and enter CTRL+W or CTRL+F7. After clicking, the watch window is displayed, and you can look up the contents of "glistp".



You can also carry out the watch by using the "w command" from the command window.

Finally, if you want to look up the local argument, you can do so by operating the local window of the option window. You do not have to do the inspect and the watch.



5. The Debug Print

In PARTNER, you can look up the debug print during programming. To display the debug print, "[osSyncPrintf\(\)](#)" which is compatible with the N64 OS functions, is used. The output result of "[osSyncPrintf\(\)](#)" is displayed in the command window. "simple" is the program which displays the debug print when the A, B and C button of the controller are entered. After loading "simple", execute it without setting the breakpoint. When you enter the button of the controller while executing the program, the debug print is displayed in the command window.

```
Command

PARTNER-V64(SBT) ver.1.04 Copyright (c) 1996 Kyroto Micro Computer Co., LTD
Monitor Code address=0xBFF00000 size=0x2000(8192)
Emulation ROM size= 16 MB
>l simple
>g
>Controller 0, Button A
Controller 0, Button B
Controller 0, Button C
File OptWin SrcSh Search Gz Inspect Cons Trace Break Stop (16)
```

All of the above are simple examples of debugging. PARTNER has various other functions. Find the most suitable debugging method from among them.

Introduction to **NINTENDO**⁶⁴

Appendices

1. N64 Function List (Extract)
2. Glossary
3. Sample Description



Appendix A N64 Function List (Extract)

Display List Control

<GBI Commands>

gSPSegment	Set the segment register and base address.
gSPBranchLessZ	Process the model's LOD.
gSPBranchLessZrg	Process the model's LOD.
gSPBranchList	Branch the display list.
gSPDisplayList	Branch execution with the current display list.
gSPEndDisplayList	Terminate the display list command.
gSPLoadUcode	Load specified microcode.
gSPLoadUcodeL	Load specified microcode.
gDPPipeSync	Synchronize the RDP attribute change which is on the primitive drawing.
gDPFullSync	Signal the end of a frame.
gDPLoadSync	Synchronize the texture load which is on the primitive drawing.
gDPTileSync	Synchronize the texture tile change which is on the primitive drawing.

Drawing Primitives

<GBI Commands>

gSPVertex	Load the vertex to the on-chip vertex cache.
-----------	--

gSPModifyVertex	Update the vertex data after the RSP accepts it.
gSP1Triangle	Draw a triangle. (The single triangle surface command.)
gSPLine3D	Draw a line. (Single Line Drawing Command.)
gSPLineW3D	Draw a line that includes a width specification.
gSP2Triangles	The triangle surface command. (Draw two triangles.)
gSPSprite2Ddraw	Specify the screen coordinate of the sprite and initiate the rendering.
gDPFillRectangle	Draw a single-color rectangle using the screen coordinate.
gDPScisFillRectangle	Draw a single-color rectangle using the screen coordinate.
gSPTextureRectangle	Draw a textured 2D rectangle.
gSPTextureRectangleFlip	Draw a rectangle using the s/t inversion on the texture coordinate.
gSPScisTextureRectangle	Draw a rectangle using the scissoring on the texture coordinate.

Perspective Transformation, Clipping and Scissoring

<Utility Functions>

guFrustum	Calculate a perspective projection matrix.
guFrustumF	Calculate a perspective projection matrix.
guLookAt	Calculate a 'lookat' view matrix.
guLookAtF	Calculate a 'lookat' view matrix.
guOrtho	Calculate a positive projection matrix.
guOrthoF	Calculate a positive projection matrix.
guPerspective	Calculate a perspective drawing projection matrix.
guPerspectiveF	Calculate a perspective drawing projection matrix.

<GBI Commands>

gSPViewport	Load the view port using the projection parameter.
gSPPerspNormalize	Normalize the perspective projection.
gSPClipRatio	Specify the relative ratio between clipping and scissoring.
gDPSetScissor	Specify the scissoring box using the screen coordinate.
gdSPDefLookAt	Load the x and y screen space coordinate directions to the lookat structure.

Culling

<GBI Commands>

gSPSetGeometryMode	Set the geometry pipeline mode.
gSPClearGeometryMode	Clear the geometry pipeline mode.
gSPCullDisplayList	Cull volumes from the display list if they lie outside the screen.

Lighting

<Utility Functions>

guPosLight Pseudo-represent positional light.
guPosLightHilite Pseudo-represent positional light.

<GBI Commands>

gSPSetGeometryMode Set the geometry pipeline mode.
SPClearGeometryMode Clear the geometry pipeline mode.
gSPLight Load light to the RSP.
gSPLightColor Change the light color of the RSP quickly.
gSPNumLights Specify the number of the light structure that loads to RSP.
gSPSetLights Load the light structure to the RSP.
gdSPDefLights Load the parameter that defines a light to the light structure.

Specular Highlight

<Utility Functions>

guLookAtHilite Calculate the lookat viewing matrix, and set the light structure that sets the default color and the viewing direction. Then calculate the texture offsets of two specular highlights.
guLookAtHiliteF <the same as the above>

<GBI Commands>

gSPLookAt Load the 'lookat' structure to the RSP for specular highlights or reflection mapping.
gSPLookAtX <the same as the above>
gSPLookAtY <the same as the above>
gDPSetHilite1Tile Set the texture parameter in the RDP to be used when rendering the specular highlight.
gDPSetHilite2Tile <the same as the above>

Reflection Mapping

<Utility Functions>

guLookAtReflect Calculate the 'lookat' viewing matrix, and set up the 'lookat' structure for the reflection mapping.
guLookAtReflectF <the same as the above>

<GBI Commands>

gSPSetGeometryMode Set the geometry pipeline mode.
gSPClearGeometryMode Clear the geometry pipeline mode.
gSPLookAt Load the 'lookat' structure to the RSP for the specular highlight or reflection mapping.
gSPLookAtX <the same as the above>
gSPLookAtY <the same as the above>

Matrix Operations

<Utility Functions>

guAlign	Calculate vector alignment for the modeling matrix.
guAlignF	Calculate vector alignment for the modeling matrix.
guLookAtStereo	Calculate the lookat view matrix for the stereo graphics display.
guLookAtStereoF	Calculate lookat view matrix for the stereo graphics display.
guMtxCatF	Concatenate two 4x4 floating-point matrices using multiplication.
guMtxCatL	Concatenate two 4x4 fixed-point matrices using multiplication.
guMtxF2L	Convert the 4x4 floating-point matrix to the fixed-point matrix.
guMtxIdent	Create the 4x4 fixed-point identity matrix.
guMtxIdentF	Create the 4x4 floating-point matrix.
guMtxL2F	Create the 4x4 fixed-point matrix to the floating-point matrix.
guMtxXFMF	Provide the point coordinate transformation using the 4x4 floating-point matrix.
guMtxXFML	Provide the point coordinate transformation using the 4x4 fixed-point matrix.
guNormalize	Normalize the vector.
guPosition	Calculate the modeling matrix.
guPositionF	Calculate the modeling matrix.
guRotate	Rotate the modeling matrix.
guRotateF	<the same as the above>
guRotateRPY	<the same the above>
guRotateRPYF	<the same as the above>
guScale	Calculate the scaling matrix.
guScaleF	Calculate the scaling modeling matrix.
guTranslate	Calculate the conversion modeling matrix.
guTranslateF	Calculate the conversion modeling matrix.

<GBI Commands>

gSPMatrix	Load the matrix, concatenate, and push the stack.
gSPPopMatrix	Pop the matrix stack.
gSPForceMatrix	Load the new MP matrix.
gSPInsertMatrix	Update the matrix element without using multiplication.

RDP Setting

< GBI Commands>

gDPPipelineMode	Turn ON/OFF the coherent pipeline mode of the span cache buffer.
gDPSetCycleType	Set the RDP cycle type.
gDPSetDepthSource	Set the type of the source Z to use for comparing the depth buffer.
gDPSetColorImage	Set the color of the frame buffer area.
gDPSetDepthImage	Set the Z-Buffer area.

gSPSetOtherMode Set or clear the RDP othermode.

Textures

<Utility Functions>

guLoadTextureBlockMipMap Calculate the MIP-map pyramid from the original texture, and return the MIP-map texture with the required display list for drawing.

<GBI Commands>

gSPTexture Make the texture effective and specify the scaling of the texture.

gDPLoadBlock Load TMEM using this low-level TMEM load macro.

gDPLoadTLUT_pal16 Load the CI4 (16 colors) pallet to texture memory.

gDPLoadTLUT_pal256 Load the CI8 (256 colors) pallet to texture memory.

gDPLoadTextureBlock Load the consecutive texture block to TMEM.

gDPLoadTextureBlock_4b Load the consecutive 4-bit texture block to TMEM.

gDPLoadTextureTile Load a part of a large texture image to TMEM.

gDPLoadTextureTile_4b Load a part of a large 4-bit texture image to TMEM.

gDPSetTextureImage Set the texture area.

gDPSetTextureConvert Control and convert the color texture filter.

gDPSetTextureDetail Turn ON/OFF the detail texture.

gDPSetTextureFilter Set the filter type on the sampling of the texture tile.

gDPSetTextureLOD Turn ON/OFF LOD for the MIP-map texture.

gDPSetTextureLUT Set the full color texture or the textualizing of the table look-up

gDPSetTexturePersp Turn ON/OFF texture map perspective transformation correction.

gDPSetTile Set the parameter for one of the eight tile descriptors.

gDPSetTileSize Set the parameter for one of the eight tile descriptors.

Using the Color Combiner

<GBI Commands>

gDPSetCombineMode Set the color combine mode.

gDPSetPrimColor Set the RDP primitive color.

gDPSetEnvColor Set the RDP environment color.

gDPSetConvert Set the matrix coefficient to convert the YUV pixel to RGB.

Using the Chroma Key

<GBI Commands>

gDPSetCombineKey Turn ON/OFF chroma keying.

gDPSetKeyR Select the color of the chroma key operation.

gDPSetKeyGB <the same as the above>

Using the Blender

<GBI Commands>

gDPSetBlendColor	Set the RDP blend color.
gDPSetRenderMode	Set the rendering mode of the blender.
gDPSetAlphaCompare	Set the alpha compare mode of the writing condition to the frame buffer.
gDPSetAlphaDither	Turn ON/OFF the alpha dither.
gDPSetColorDither	Turn ON/OFF the RGB color dither.
gDPSetPrimDepth	Set the primitive depth.

Using Fog

<GBI Commands>

gSPSetGeometryMode	Set the geometry pipeline mode.
gSPClearGeometryMode	Clear the geometry pipeline mode.
gSPFogPosition	Specify the thickness of the fog.
gDPSetFogColor	Set the RDP fog color.

Mathematical Calculation Utilities

sinf, sins	Calculate the sine of the angle using floating or fixed-point.
cosf, coss	Calculate the cosine of the angle using floating or fixed-point.
sqrtf	Calculate the square root.
guRandom	Calculate a 32-bit integer random number.

N64 Operating System Initialization

osInitialize	Initialize the hardware and software.
--------------	---------------------------------------

Threads

osCreateThread	Create a thread.
osDestroyThread	Delete a thread.
osGetThreadId	Get the debugger ID of a thread.
osGetThreadPri	Get the priority of a thread.
osSetThreadPri	Reset the priority of a thread.
osStartThread	Start or restart the execution of a thread.
osStopThread	Stop a thread.
osYieldThread	Yield CPU access and activate the thread dispatcher.

Messages

- osCreateMesgQueue Create a message queue.
- osSendMesg Send a message and synchronize.
- osJamMesg Send a message and synchronize.
- osRecvMesg Receive a message and synchronize.
- osSetEventMesg Register the message queue and message each time an event occurs.

Scheduler

- osCreateScheduler Create the RCP task scheduler.
- osScAddClient Add the client to the RCP task scheduler.
- osScGetTaskQ Get the message queue used for the communication with the scheduler thread.
- osScRemoveClient Delete the client from the RCP task scheduler.

Tasks

- osSpTaskLoad Load the received task to the SP.
- osSpTaskStart Start the SP task.
- osSpTaskStartGo Load and start the SP task.
- osSpTaskYield Request to yield the SP task.
- osSpTaskYielded Validate the break of the SP task.

Display Processor

- osDpGetCounters Get the inside execution counter of the display processor (DP).
- osDpGetStatus Get the status of the display processor (DP).
- osDpSetStatus Set the status of the display processor (DP).
- osDpSetNextBuffer Set the DMA transfer of the display processor (DP).

Controller

- osContInit Detect and initialize the Controller.
- osContReset Reset all Controllers.
- osContSetCh Set the number of accepting Controllers.
- osContStartQuery Issue the create command to acquire the status and type of the Controller.
- osContGetQuery Get the state and type of the Controller.
- osContStartReadData Issue the lead command to get the state of the Controller.
- osContGetReadData Get the 3D stack and the button status.

Controller Pak

- osPfsAllocateFile Create the specified file.

osPfsDeleteFile	Delete the specified file.
osPfsFindFile	Browse the specified file.
osPfsChecker	Check and repair the file system of the Controller Pak.
osPfsFileState	Get the file information of the Controller Pak.
osPfsNumFiles	Get the file number of the Controller Pak.
osPfsFreeBlocks	Get the space size (in number of bytes) of the Controller Pak.
osPfsIsPlug	Detect the Controller Pak.
osPfsReSizeFile	Make the file the specified length.
osPfsReadWriteFile	Read and write the file data.
osPfsSetLabel	Write the label of the Controller Pak.
osPfsGetLabel	Get the label of the Controller Pak.
osMotorInit	Initialize the Rumble Pak.
osMotorStart	Work the Rumble Pak.
osMotorStop	Stop the Rumble Pak.
osPfsInitPak	Initialize the file handle of the Controller Pak.
osPfsRepairId	Correct the ID area of the Controller Pak.

EEPROM

osEepromLongRead	Read the data of several EEPROM blocks.
osEepromLongWrite	Write the data of several EEPROM blocks.
osEepromProbe	Detect EEPROM.
osEepromRead	Read one EEPROM block.
osEepromWrite	Write one EEPROM block.

AI (Audio Interface)

osAiGetStatus	Get the status of the audio interface (AI).
osAiGetLength	Get the DMA remaining transfer amount of the audio interface (AI).
osAiSetFrequency	Get the frequency of the audio interface (AI).
osAiSetNextBuffer	Set the DMA transfer of the audio interface (AI).

PI (Parallel Interface)

osPiRawStartDma	Set the DMA transfer of PI (low level).
osPiRawWritel0	Write PI I/O (low level).
osPiRawReadl0	Read PI I/O (low level).
osPiGetStatus	Get the status of PI.
osPiStartDma	Set the DMA transfer of PI using the PI Manager.
osPiWritel0	Write PI I/O using the PI Manager.

osPiReadIo	Read PI I/O using the PI Manager.
osCreatePiManager	Create the PI Manager.
osPiGetCmdQueue	Get the command queue of the PI Manager.
osCartRomInit	Return information about hardware setting that are required by the Game Pak.
osDriveRomInit	Return information about hardware setting that are required by the ROM in the N64 Disk Drive.
osEPIWriteIo	Provide "32bit IO Write" to the PI device address (devAddr), and store its value to "data".
osEPIReadIo	Provide "32 bit IO Read" from the PI device address (devAddr), and store the value in "data."
osEPIRawStartDma	Obtain low-level access to EPI without using the PI Manager.
osEPIStartDma	Obtain access to EPI by using the PI Manager.

VI (Video Interface)

osTvType	Get the television system.
osViGetStatus	Get the VI status.
osViGetCurrentMode	Get the current VI mode.
osViGetCurrentLine	Get the VI half line.
osViGetCurrentField	Get the VI field number.
osViGetCurrentFramebuffer	Get the current frame buffet address.
osViGetNextFramebuffer	Get the next frame buffer address.
osViSetMode	Set the VI mode.
osViSetEvent	Register the vertical synchronization event message to the VI Manager.
osViSetSpecialFeatures	Set the VI special features.
osViSetXScale	Set the VI horizontal expanding ratio.
osViSetYScale	Set the VI vertical expanding ratio.
osViSwapBuffer	Register the frame buffer displayed in the next frame.
osViBlack	Black out the VI screen.
osViFade	Fade out the VI screen.
osViRepeatLine	Display the first scan line to all frames.
osCreateViManager	Create the VI Manager.

Timer

OS_NSEC_TO_CYCLE S	Convert nanoseconds (ns) to the cycle number of the CPU count register.
OS_USEC_TO_CYCLE S	Convert microseconds (ms) to the cycle number of the CPU count register.
OS_CYCLES_TO_NSEC C	Convert cycle number of the CPU count register to the time in nanoseconds (ns).
OS_CYCLES_TO_USEC C	Convert the cycle number of the CPU count register to the time in microseconds (ms).

osGetTime	Get the real time counter value.
osSetTime	Set the real time counter value.
osSetTimer	Start the interval or count-down timer.
osStopTimer	Stop the interval or count-down timer.

Administration of Virtual and Physical Addresses

OS_K0_TO_PHYSICAL	Convert the CPU virtual address (KSEG0), which is direct-mapped with cache, to the physical address.
OS_K1_TO_PHYSICAL	Convert the CPU virtual address (KSEG1), which is direct-mapped without cache, to the physical address.
OS_PHYSICAL_TO_K0	Convert the physical address to the direct-map CPU virtual address (KSEG0) with cache.
OS_PHYSICAL_TO_K1	Convert the physical address to the direct-map CPU virtual address (KSEG1) without cache.
osVirtualToPhysical	Convert the CPU virtual address to the physical address.
osPhysicalToVirtual	Convert the CPU physical address to the virtual address.
osMapTLB	Set up the CPU virtual address mapping.
osUnmapTLB	Free the CPU virtual address mapping.
osUnmapTLBAll	Free all the CPU virtual address mapping.
osSetTLBASID	Set the ID setting of the CPU virtual address mapping.
osGetRegionBufSize	Get the size of the memory buffer.

Cache Management

OS_DCACHE_ROUNDUP_ADDR	Round up address or size values to adapt to the data cache line size, and make the address the physical address.
OS_DCACHE_ROUNDUP_SIZE	<the same as the above>
osInvalDCache	Nullify the CPU data cache line.
osInvalICache	Nullify the CPU instruction cache line.
osWritebackDCache	Write back the CPU data cache line to the physical memory.
osWritebackDCacheAll	Write back the CPU data cache line to the physical memory.

Memory Region Library

osCreateRegion	Initialize the memory allocation region.
osMalloc	Allocate the memory region.
osFree	Free the memory region.
osGetRegionBufCount	Get the buffer count created in the region.
osGetRegionBufSize	Get the size allocated to each buffer in the region.

Emulator Board and Host Communications

osReadHost Read data from the host.
osWriteHost Write data to the host.
osTestHost Get the data transfer state from the host.
uhCloseGame Close the data communication facility between the host (Indy) and the emulator board.
uhOpenGame Open the data communication facility between the host (Indy) and the emulator board.
uhReadGame Read data from the game to send it to the host (Indy).
uhReadRamrom Read data from RAMROM.
uhWriteGame Write data from the host (Indy) to the game.
uhWriteRamrom Write data to RAMROM.

Debugging

gDPNoOpTag NO-OP command of the RDP.
gDPNoOp NO-OP command of the RDP.
guParseGbiDL Display using the decodable format of the GBI display list.
guParseRdpDL Display using the decodable format of the low-level RDP display list.
guDumpRawRdpDL <the same as the above>
osSyncPrintf Output the formatted text to the debug board.

Logging

osCreateLog Initialize logging.
osLogEvent Add the entry of logging.
osFlushLog Output logged data to the host.

Profiler

osProfileInit Initialize the profiled segment count buffer.
osProfileStart Start the counter between profilers.
osProfileStop Stop the counter between profilers.
osProfileFlush Transfer the profiler data to the host.
gperf Analyze the profiler data.

00>OS Global Variables

osTvType Type configuration.
osMemSize Size of the main memory (DRAM).
osResetType Type of the system reboot.

Manage CPU Registers and Error Handler

osGetCount	Get the count register of CPU.
osGetIntMask	Get the interrupt mask.
osSetIntMask	Set the interrupt mask.
osSetErrorHandler	Set the error handling routine of the debug library.

Sprite Library

spColor	Set the sprite color.
spDraw	Create the display list to display the sprite on the screen.
spFinish	Reset the graphics mode to default at the end of the sprite drawing.
splnit	Set the graphics mode required for the sprite drawing.
sp, splntr	Introduce the sprite library.
spMove	Specify the position of the top left-hand angle on the sprite screen.
spScale	Change the size and shape of the sprite.
spScissor	Set the border area drawn.
spSetAttribute	Set the designated attribute.
spClearAttribute	Clear the designated attribute.
spSetZ	Set the depth value (the Z value) of the sprite.
mksprite, mksprite32, mkisprite	Convert RGB file to the sprite data structure of the C language.

Audio Library

alAudioFrame	Creates an audio command list for one frame.
alBnkfNew	Initializes a bank file for use on the Nintendo 64.
alCents2Ratio	Converts an s32 cents value to an f32 ratio.
alClose	Shuts down the N64 Audio Library.
alCSeqGetLoc	Initializes a marker with the current sequence location for use with the compressed MIDI sequence player.
alCSeqGetTicks	Returns the number of MIDI clock ticks of the compressed MIDI sequence location.
alCSeqNew	Initializes an N64 compressed MIDI sequence structure.
alCSeqNewMarker	Initializes a sequence marker at a given location for use with the compressed MIDI sequence player.
alCSeqNextEvent	Returns the next MIDI event from the compressed MIDI sequence.
alCSeqSecToTicks	Converts from seconds to MIDI clock ticks.
alCSeqSetLoc	Sets the current sequence location within the compressed MIDI sequence.
alCSeqTicksToSec	Converts from MIDI clock ticks to seconds.
alCSPDelete	Deallocates a MIDI sequence player.
alCSPGetChlFXMix	Returns the effect mix for the given MIDI channel.
alCSPGetChlPan	Returns the pan position for the given MIDI channel.
alCSPGetChlPriority	Returns the priority for the given MIDI channel.

alCSPGetChlProgram	Returns the MIDI program number assigned to a MIDI channel.
alCSPGetChlVol	Returns the volume for the given MIDI channel.
alCSPGetSeq	Returns the sequence currently assigned to the compressed MIDI sequence player.
alCSPGetTempo	Returns the tempo of the current sequence.
alCSPGetVol	Returns the overall sequence volume.
alCSPNew	Initializes a compressed MIDI sequence player.
alCSPPlay	Starts the target sequence playing.
alCSPSetBank	Specifies the instrument bank for the sequence player to use.
alCSPSetChlFXMix	Sets the effect mix on the given MIDI channel.
alCSPSetChlPan	Sets the pan position for the given MIDI channel.
alCSPSetChlPriority	Sets the priority for the given MIDI channel.
alCSPSetChlProgram	Assigns a MIDI program to a MIDI channel.
alCSPSetChlVol	Sets the volume for the given MIDI channel.
alCSPSetSeq	Sets the compressed MIDI sequence player's target sequence.
alCSPSetTempo	Specifies the tempo for the sequence player to use.
alCSPSetVol	Sets the overall sequence volume.
alCSPStop	Stops the target compressed MIDI sequence.
alHeapAlloc	Allocates memory from an Nintendo 64 audio heap.
alHeapCheck	Checks the consistency of an N64 audio heap.
alHeapInit	Initializes an audio heap for use with the Nintendo 64 Audio Library.
alInit	Initializes the N64 Audio Library.
alSeqGetLoc	Initializes a marker with the current sequence location.
alSeqGetTicks	Returns the number of MIDI clock ticks of the sequence location.
alSeqNew	Initializes an N64 MIDI sequence structure.
alSeqNewMarker	Initializes a sequence marker at a given location.
alSeqNextEvent	Returns the next MIDI event in the sequence.
alSeqpDelete	Deallocates a MIDI sequence player.
alSeqpGetChlFXMix	Returns the effect mix for the given MIDI channel.
alSeqpGetChlPan	Returns the pan position for the given MIDI channel.
alSeqpGetChlPriority	Returns the priority for the given MIDI channel.
alSeqpGetChlProgram	Returns the MIDI program number assigned to a MIDI.
alSeqpGetChlVol	Returns the volume for the given MIDI channel.
alSeqpGetSeq	Returns the sequence currently assigned to the ALSeqPlayer.
alSeqpGetTempo	Returns the tempo of the current sequence.
alSeqpGetVol	Returns the overall sequence volume.
alSeqpLoop	Sets sequence loop points.
alSeqpNew	Initializes a Type 0 MIDI sequence player.
alSeqpPlay	Starts the target sequence playing.

alSeqpSendMidi	Sends the given MIDI message to the sequence player.
alSeqpSetBank	Specifies the instrument bank for the sequence player to use.
alSeqpSetChlFXMix	Sets the effect mix on the given MIDI channel.
alSeqpSetChlPan	Sets the pan position for the given MIDI channel.
alSeqpSetChlPriority	Sets the priority for the given MIDI channel.
alSeqpSetChlProgram	Assigns a MIDI program to a MIDI channel.
alSeqpSetChlVol	Set the volume for the given MIDI channel.
alSeqpSetSeq	Sets the sequence player's target sequence.
alSeqpSetTempo	Specifies the tempo for the sequence player to use.
alSeqpSetVol	Sets the overall sequence volume.
alSeqpStop	Stop the target sequence.
alSeqSecToTicks	Converts from seconds to MIDI clock ticks.
alSeqSetLoc	Sets the current sequence location.
alSeqTicksToSec	Converts from MIDI clock ticks to seconds.
alSndpAllocate	Allocates a sound to a sound player.
alSndpDeallocate	Deallocates a sound from a sound player.
alSndpDelete	Deallocates a sound player.
alSndpGetSound	Gets the identifier of the current target sound in a sound player.
alSndpGetState	Gets the state (playing, stopping, or stopped) of the current target sound.
alSndpNew	Initializes a sound player.
alSndpPlay	Starts playing the current target sound.
alSndpPlayAt	Starts playing the current target sound at a specified time.
alSndpSetFXMix	Sets the wet/dry mix of the current target sound.
alSndpSetPan	Sets the pan position of the current target sound.
alSndpSetPitch	Sets the pitch of the current target sound.
alSndpSetPriority	Sets the priority of a sound.
alSndpSetSound	Sets the current target sound in a sound player.
alSndpSetVol	Sets the volume of the current target sound.
alSndpStop	Stops playing the current target sound.
alSynAddPlayer	Adds a client player to the synthesizer.
alSynAllocFX	Allocates an audio effect processor.
alSynAllocVoice	Allocates a synthesizer voice.
alSynFreeVoice	Deallocates a synthesizer voice.
alSynGetFXRef	Gets the address of an effect.
alSynGetPriority	Requests the priority of a voice.
alSynHeapSize	Get the heap size of the synthesizer
alSynNew	Allocates the specified synthesizer driver.
alSynRemovePlayer	Removes a player from the synthesizer driver.

alSynSetFXMix	Sets the wet/dry effect mix for a voice.
alSynSetFXParam	Sets an effect parameter to the specified value.
alSynSetPan	Sets the stereo pan position of the specified voice.
alSynSetPitch	Sets the pitch of the specified voice.
alSynSetPriority	Sets the priority of the specified voice.
alSynSetVol	Sets the target volume of the specified voice.
alSynStartVoice	Starts synthesizing audio samples with the specified voice.
alSynStartVoiceParam	Starts synthesizing audio samples with the specified voice using the specified parameters.
alSynStopVoice	Stops generating audio samples with the specified voice.

64DD Leo Functions

LeoByteToLBA	Converts a byte size to an LBA number.
LeoLBAToByte	Converts an LBA number to a byte size.
LeoClearQueue	Clears the Leo Manager command queue.
LeoJCreateLeoManager	Start (Game Pak boot, Japanese version).
LeoCCreateLeoManager	Start (Game Pak boot, English version).
LeoCreateLeoManager	Start (disk boot).
LeoGetKAdr	Gets the kanji storage offset address from the Shift JIS code sjis.
LeoGetAAdr	Gets the ASCII character storage offset address from the character code.
LeoGetAAdr2	Gets the ASCII character storage offset address from the character information data code.
LeoInquiry	Checks the version number of the hardware and the software.
LeoModeSelectAsync	Changes the time for switching between 64DD modes.
LeoReadCapacity	Calculates the usable area of the disk.
LeoReadDiskID	Gets the disk ID.
LeoReadRTC	Reads the time of the built-in real-time clock.
LeoSetRTC	Sets the time of the built-in real-time clock.
LeoReadWrite	Reads from and writes to 64DD.
LeoReset	Stops any further execution of commands sent to Leo Manager and clears the command queue.
LeoResetClear	Releases 64DD from the Reset state.
LeoRezero	Recalibrates the 64DD.
LeoSeek	Executes the command to seek on 64DD.
LeoSpdlMotor	Controls the 64DD motor and heads.
LeoTestUnitReady	Checks the 64DD status.

Appendix B Glossary

1. Glossary about the Graphic

A value

->The alpha value.

Aliasing

When you display lines or polygons with the display equipment which has the limited pixels like TV, the stepped-notch is produced on the border of these lines or polygons. zAliasing means these notches.

Alpha compare

To synthesize the drawing, of the translucent polygon, and the pixel, this function compares the alpha value of pixel with the set threshold, and provides the conditional writing.

Alpha dither compare

When the pixel alpha value is larger than the random dither value, it rewrites the frame buffer temporarily. Because of this, it can affix the alpha gradation on the surface and it implements the particle effect.

Alpha value

The level of opacity. The larger the alpha value is, the more opaque it becomes.

Ambient light

The light dispersing in the environment with no direction. It determines the part color of the object where light does not shine on.

Animate

Give something movement.

Anti-aliasing

A technique used to smooth images by reducing the jagged (notches) edge effect.

Aperture angle

The angle defined the visibility area. It is like a caliber of the camera lens.

Attribute

The attribute.

Bi-linear filtering

To provide the bi-linear interpolation for color change of the texture so that dots do not become rough when the texture is extended.

Bi-liner interpolation

The two-dimensional linear interpolation provided to the vertical and horizontal directions.

Bilerp

The bi-liner filtering mode of the texture.

Blend color compare

The alpha value (threshold) used in the alpha compare feature.

Blend texture

To determine the mixed ratio between the shading and texture RGB values with using the texture alpha value.

Blender

The junction to mix the calculated pixels and frame buffers. Used with drawing of the translucent polygon, the anti-aliasing process, the fog process and the dither process, etc.

Blender mux control bit

The information to set the blender hardware. This information defines the blend expression. Stored in the parameter of RDP, "Other Modes."

Bounding volume test

->Volume culling. The test to check if the object bounding volume goes into the object viewing volume. After this test, if the object is completely outside the viewing volume, the display list is not executed and is speeded up instead.

Box filter

One of the modes of the texture filter. Simply balance four texels around the sampling point. Used effectively when the sampling point is placed in the center of four texels.

Chroma-key process

The process which sets a specific color (mainly blue) to a transparent as a key and synthesizes more than two images except parts of the specific color.

CI

->The color index.

Clamp

A technique which simply uses the end data of the texture if you specify outside the texture range.

Clipping

The process which cuts off polygons or lines outside the view. Though clipping relatively takes a long time to process, you can reduce the process time by changing the ratio between the clipping pyramid and screen, and using it together with scissoring.

Clipping pyramid

The three-dimensional area in which is provided clipping. Objects sticking out this area are provided clipping.

Clipping pyramid box

The clipping pyramid.

Color combiner mode

The combining method of color sources.

Color dither

->Dither.

Color index

The mode which has the pallet information at each texel. There are 4-bit and 8-bit formats.

Color index pallet

It is given the index (pallet number) for each color.

Color scheme

The color configuration or the arrangement of colors

Cover value

The information how much of the pixel part are covered by the primitive. It has 16-subpixels at each pixel, gives the checked dither mask and preserves how many of 8-subpixels are covered. Used by the anti-aliasing process.

Cover value wrap

This status that the total of the existing cover value on the memory and the new pixel cover value if larger than 1.0 (when it is 1.0, the pixel is the state totally covered by the primitive). The blender executes anti-aliasing with using the information of the cover value wrap.

Crack

The crack.

Culling

The process which tries not to let the unnecessary data for display (the back of the object, etc.) flow to the graphic pipeline.

cvgbit

->The cover value.

Decalcomania line

Similar to the decull surface, lines drawn on the rendered surface. Exaggerate that an object is composed of polygons. Used for showing it like a high-tech style.

Decalcomania surface

The surface placed on the rendered surface. Because the decalcomania writing is placed under the condition that "it blends only when the cover value does not lap," you can overwrite the original surface.

Decalcomania texture

The texture drawing on the rendered surface. For example, it is used for the wing design of the airplane.

Decalcomania Z algorithm

The algorithm which blends the parts unwrapped (then it is 1.0) the cover value and overwrite the parts lapped.

Depth buffer compare

Compare the depth value of each pixel with the depth value stored in Z buffer if provided the rendering with using the Z-buffer.

Depth value

The depth value. It is provided and decided the interpolation of three vertexes' Z value of a triangle included its pixel. The rectangle, etc., which the Z value does not relate is set one Z value for the whole primitive. Used on the depth buffer compare.

Detail texture

Even if you provide the tri-liner MIP map interpolation, the texture become dim when it approaches more than the maximum level of MIP mapping. This is the texture which expresses the detail design used for protecting it.

Diffused light

->The diffused light

Diffusing light

Light specified the direction. It reflects to all directions with the same strength. The strength of light reflecting depends on the angle of incidence.

Display list

Arrange a series of GBI command as a drawing routine and provide all commands each time without repeating by executing the routine name. Not only a program becomes simple, but also the execution effect will improve because of the initialization.

Dither filter

In the 16-bit format, each color is allocated only in 5-bit. So, to enhance the color data precision, people make it totally 8-bit by allocating the correction data with a special pattern of the lowest three positions. This softens the mach band effect.

Dither matrix

The correction data pattern of the lowest 3-bit used by the dither filter.

Dithering

Generally, a technique used to specify a different color to a close pixel and show it as neutral tints. In N64, it is a manipulation of dither filter.

Divot circuit

A filter which relieves the holes (produced when several border edges overlap in one pixel) of anti-aliased pixels. Effective only on the anti-aliasing video mode. Also, it is not applied for the full-covered area.

Dots drawing transparent process

The process of the level of transparent in units of pixel.

Double buffering

Two parts of image buffer. One image buffer is displayed on the screen while the other is being rendered into. When drawing ends, the roles of two buffers are exchanged.

dz value

It is a value used for the object drawn by the anti-aliasing and the decull render mode, and for the decision if the new pixel is on the same surface as the existing pixel within the memory.

Environment light

->The environment light

Environmental mapping

The texture mapping which apparently makes thing around the object reflect. You can execute it easily by using reflection mapping.

Fill rate

The pixel drawing rate per second.

Flat shading

To fill up the whole primitive with a single color.

Flip

To switch the coordinates like this: the s coordinate goes to y direction and the t coordinate goes to x direction. To move the texture.

Flip book animation

The animation like a flip comic.

Fog process

The process showing the far object misty. Thanks to this, even if the quality of the far object falls, the whole quality of image is not effected so much. Also, it has an effect of handing the pop in or pop out.

Fore (back) ground

Fore (back) ground

Fractal

The mathematical expression of the own analogy (essentially, possible to look the same even from near or far places) used for a complicated object which takes very long time to process to express with polygons.

Frame rate

The frame number of the displayed screen in one second. The maximum is 60 frames in the NTSC system.

Gamma correctness

The color intensity change usually corresponds to its input, and the output corresponds to the nonlinear depending on the feature of the equipment or the perceptual feature of human. The correctness of this nonlinear to the linear correspondence.

Geometric primitive

->Geometry primitive

Geometry

The three-dimensional coordinate data or a graphic from created by it.

Geometry engine

A feature which executes a numeric value calculation of the three-dimensional coordinate transformation.

Geometry primitive

The basic geometric three-dimensional space which composes the complicated and varied screen. The examples are points, line segments and polygons, etc. (a closed area by line segments).

Global state register

A register which specify the configuration and synchronization of the pipeline inside RDP. It sets the cycle type, the synchronization between the pipeline and attribute and the information of atomic primitive mode.

Gouraud shading

A technique of shading which makes vertexes or edges in conspicuous and shows curved surface smooth by getting and interpolating each vertex color.

Graphic Binary Interface (GBI)

The command of the single instruction of the 64-bit length which is executed by the RSP microcode to draw graphics. This GBI command string is called the display list.

Graphics pipeline

Pipelines creating the image which is consisted of a series-unit of the rasterizer, the texture units the texture filter, the color combiner, the blender and the memory interface inside of RDP.

I value

The intensity value.

IA mode

The mode having the I (intensity) and alpha information at each texel. The 4-bit format is 3/1, the 8-bit format is 4/4 and the 16-bit format is 8/8-bit.

Intensity and alpha

->The IA mode.

Intensity mode

The mode only having the intensity and alpha information at each texel. It has the 4-bit and 8-bit formats.

Intensity ramp

(Briticism) The transition difference (slope) of the intensity.

Internal edge

Edges which two visible-displayed polygons overlap and share.

Interpolator

To rewrite. In here, it is a color converter in the color combiner.

Inverse kinematics

A method to look for the angle state of each joint only by specifying the edge of the model having the multi-joint structure on the meaning of the reverse-kinematic system. Suitable for the animation of human or animals.

Jag part

"jag part" (Notched part.)

The law of Nyquist

The sampling theorem that "if you do the sampling with the more than twice of the frequency, the maximum frequency of the input signal, you can reproduce the original signal consistently."

lerp

The line interpolation.

Level of detail

->LOD

Level of pack

The level of complexity of textures' color or shape, etc.

LOD

The abbreviation of "Level Of Detail." Several levels of detail preparing for changing the level of detail between the close and far objects. Because of this, we can lower the level of detail of the far object and the drawing rate improves.

Material

The object quality of the material and quality feeling.

Material characteristic

The difference between the reflection and transparency of light by the object material. This difference is mathematically expressed in the following factors:

The reflection	The transparency
The surface color	The object color (inside color of the material)
The mirror face factor	The object density
The plane roughness factor	The mirror face factor
The metal factor	The inside roughness factor
The diffusion factor	The inflectional ratio
	The diffusion factor

Matrix

Matrices described objects' positions or directions in the three-dimensional space. Also, the matrix calculation for the coordinate transformation. Used for the object rotation or move in the three-dimensional space and for the projection to the screen.

Matrix stack

The stack to implement the complicated matrix operation. There are two types of stack: One is a 10 column of modeling matrix stack corresponding to the object position or direction, etc.; the other is a 1 column of viewing matrix stack corresponding to the view position or direction, etc.

Memory color

Existing pixel color in the frame buffer.

Mesh object

The aggregate of vertexes formed objects and triangle faces.

MIP map

If you reduce the texture, the display quality decreases with the appearance of moiré. So, prepare textures of 1/2, 1/4 and 1/8 of the length and width of the original texture pattern. You can remove moiré by providing the bi-linear or tri-linear interpolations for two suitable pieces of texture to be used with the texture expansion ratio, which is decided depending on distance from the viewpoint. This is the process of MIP map. However, it has weak points. Such as, you must use the 2-cycle mode, and you must store several texture patterns in the small texture memory.

MIP map texture

->MIP map

Mirror

A technique to be able to rotate each texture individually to the vertical or horizontal directions.

Modulation texture

The mix-adjusted texture between the texture and shade colors. Provided by a color combiner.

Morphing

A technique to change the image interpolating information in the specified area from several places or images.

Motion path

A technique to prepare the path for moving objects or camera, and make the animation with moving its path.

MPEG

The abbreviation of "Moving Picture coding Experts Group." The digital compression method of the color animation.

Multi tile texture

A technique to use the maximum eight texture tiles and to display using several tiles. According to this, it is possible to MIP map or Detail texture process if it activates in 2-cycle.

Mutual sticking mode

The mode assumed a surface which is sticking into another surface. You do not have to use this mode with such objects' intersection. But if you set to this mode, the intersection part is correctly anti-aliased.

Near clipping

Clip the object which is closer than the near plan. Usually it is not drawn. But, because it may become a problem visually, there is a technique which does not provide clipping with a certain condition.

near/far plane

The viewing pyramid near plane is the closest plane to the observer of the viewing pyramid. The far plane is the farthest plane.

NURBS

The abbreviation of "Non-Uniform Rational B-Spline." The method used for expressing the curve or curved surface by the high-level function.

Outside edge

->The silhouette edge

Padding

When you load the texture tile, pixel addresses of four angles can be on the byte boundary. But, if the column width of tile is not a 64-bit boundary, the hardware automatically tries to make each column of TMem be the 64-bit boundary.

Painter's algorithm

The method of rendering polygons from far to close places from the view. This provides anti-aliasing suitable.

Particle effect

The apparently show to implement the particle system.

Particle system

A particle is a very small object. The particle system is the modeling which uses many of these particles. This system is an ideal technique to express a thing that many small objects gather based on some general law.

Pers-correctness texture

Execute the perspective correctness to the calculation of texture coordinate value.

Perspective correctness

The correction used for enhancing the calculation precision of the texture coordinate value.

Pixel color

The current pixel color. The manual distinguishes pixel colors by making the pixel color on the existing memory the memory color.

Point sampling

The sampling provided assuming that each texel of the source texture is indicated to 1 pixel on the display. The best case is that the texel and pixel is 1:1. The faithful map is not provided except this.

Polygon

It expresses three-dimensional object by the plane combination of polygon.

Pop-out and Pop-in

The pop-out means that the object goes farther than the far plane from the view and suddenly disappears from the screen. The pop-in means that the object comes to close and suddenly appear on the screen.

Primitive

The basic elements (dots, lines and polygons, etc.) to be drawing objects of 3D graphics.

Random alpha source

To use the random number for the threshold alpha by the compare function. Called the alpha dither comparison and used for the particle effect, etc.

Raster

A scan line. The horizontal line on the display screen (TV, etc.).

Raster image

->Image displayed by using the raster scan line.

Rasterize

To receive each vertex and color of the primitive and create pixels inside the primitive. Each pixel has attributes such as coordinate, depth value, color value, LOD level and cover value, etc. and they are used in the calculations later.

RGBA mode

The mode having the RGB (red, green, blue) and alpha information at each texel. The 16-bit format is 5/5/5/1 and the 32-bit format is 8/8/8/8-bit.

Reflection highlight

->The specular highlight

Reflection mapping

->A texture mapping method with dynamically calculate the texture coordinate that reflects into the position by using the normal vector data of the reflection mapping object. Used for expressing that the surrounding situation reflects into the objects surface.

Rendering

To convert the primitive specified on the object coordinate to the image data for the frame buffer.

Scale-up factor

The setting item of VI, and the scaling factor for enlargement or reduction of images. You can specify from 0.25~1.0 for the X direction and from 0.05~1.0 for the Y direction.

Scissoring

The process which two-dimensionally cuts off parts outside the drawing area.

Sharp texture

Used in the similar status as the "Detail" texture. The texture which is pseudo-created from two textures of the maximum and following levels, when it approaches more than the maximum level of MIP mapping.

Silhouette edge

Parts which particularly share with the background in edges of polygons.

Span

The length.

Span buffer

It has a type of line buffer called span buffer in RDP and can process information together so that it shows a high speed even at the random access which is a weak point of RDRAM.

Specular highlight

A bright spot which appears when light reflects to a shiny object. Because it is implemented by using the texture, you cannot use the specular highlight expression to the texture-mapped object.

Sprite

A rectangle image using the texture. In N64, the sprite is expressed by using the texture.

Sub-pixel

Split one pixel to 16 (4X4). it is used for searching for a cover value.

Surface

A surface of polygon.

Texel

A point (pixel) in a texture.

Texture decalcomania

Provide the decalcomania (design imprint method) with using the texture.

Texture map

Pictures or patterns placed onto the surface of polygon. It is simply called the texture.

Texture mapping

The process of placing the texture onto the surface of polygon. It is simply called the texture.

Texture memory

The 4-byte special memory of texture built-in RPD. Because it is separated to 4 simultaneous accessible banks, it can output 4 texels with 1 clock.

Texture rectangle primitive

"Sprite" which draws the texture rectangle on the screen coordinate.

Texture tile descriptor

TMEM can store the maximum 8 texture tiles, such as the following information that each tile has: the texture size, flags of wrap/clamp/mirror, the format and the TMEM address, etc.

TLUT (Texture Look-Up Table)

The color index pullet stored the color information corresponding to each pallet number which is created on the

latter 2k-byte of TMEM when you use the color index(CI) mode.

TLUT pallet

->TLUT

TMEM

->The texture memory.

Tri-linear interpolation

The three-dimensional linear interpolation provided to the vertical, horizontal and depth directions.

Vertex

Vertexes of the three-dimensional space.

Vertex alpha

The alpha value specified by the vertex.

Vertex cache

Cache stored the vertex data after the coordinate transformation operation. There are totally 16 vertex caches.

Video display logic

VI(the video interface).

Viewing pyramid

The definition of the three-dimensional area inside the view (inside the screen).

Volume culling

A technique to check if a complicated object can be on the screen and to make the rest of the parts of the display list skip if the object is completely outside the screen.

Wrap

A technique to utilize small textures as a large texture by displaying them repeatedly. If you use the wrap, the texture size must be the exponentiation of 2.

YUV

The digital video signal standard consisted of the intensity (Y) and color difference (UV) components. Because human eyes are not so sensitive to the color component, it decreases the sampling of the color component and provides the effective compression. Used for the MPEG image, etc.

YUV-RGB conversion

Convert the YUV pixel to RGB. Provided by the texture filter and the color combiner.

Z-buffer

A buffer which stores the depth value used for only displaying the closest polygon watching from the view.

Zap

To prevent VI from anti-aliasing, it makes a cover value 1.0 by force.

2. Terms about the system

2 way set associative

->The cache system which splits the set associative cache memory into two and puts a certain data on the main memory onto the decided position in the either area. By this, you can change contents of one cache while keeping contents of the other cache.

64-bit boundary sorting

The format which separate data into 64-bit each from the lowest address in the memory.

Active page register

The register which specifies the page position that each bank has; the banks of RDRAM are separated into 4 banks of 1 MB each. When using the frame and Z buffering, you can store them to separate memory banks by switching them. You do not have to change to page register all the time, and it reduces to make mistakes.

Add-on RAM pack

Add-on RAM is which expands the storage capacity. In other words, it is a controller pack.

Address conversion buffer

The address conversion table corresponding to the virtual and physical address to convert from the virtual to physical address in a high speed. Usually called TLB (Translation Look aside Buffer).

Address space identifier (ASID)

The 8-bit value used for expanding the virtual address when you provide the virtual addressing with TLB

Alignment rule

The alignment is a relationship between the information stored in memory and the memory boundary. N64 has a rule to align the memory area to the 16-byte boundary do that it becomes a multiple of cache.

API

Application Programming Interface. The programming interface of functions, commands and utility, etc., which OS supplies for the application program.

Application programming interface

->API

Aspect ratio

The size ratio of the length and width of the TV screen size.

Atomic primitive mode

The mode to avoid the span buffer coherence problem. But this mode reduces the fill-rate very much.

Band width

The data transfer amount per constant time.

big-endian, little-endian

The data storage method to the memory space. For example, each 32-bit data of Oxdeadbeef is stored into the memory space. In other words, the little-endian is a style that starts from the lowest byte and stores in order from the lowest address. The big-endian system starts from the highest byte and stores in order from the lowest address.

Boot

A procedure that accepts the initialization of system or the program to main memory when it activates. It is provided for executing the program outside memory equipment.

Boundary alignment

->The 64-bit boundary alignment.

Bss

Block Started by Symbol. The data area whose initial value is not defined. If you use the bss section for that, you need to clear the initial value (to 0). The register to store the cause to inform it to the system when the CAUSE register interrupt occurs.

Bucket sort

This method provides that you divide numeric values, which you want to sort, into buckets; the numeric values are split some sections (buckets). And it sorts all of them by sorting in the buckets. To range the object numeric values equally makes the process much faster.

Buffer

The temporary memory area used for adjusting the difference of transfer rate between systems at data transferring.

Calling process

The process which calls next process from the current status and continuously executes them.

Command dispatcher

The command operations control.

Command parser

The command analysis section.

Context

(1) Information of addresses, etc., which causes the CPU exception.

(2) Information, controlled by OS, to store the thread state or the register value when it loses the executing right. [thread -] [- switch].

Counter interrupt

An interrupt to post that the inside counter reaches to the final value.

dbx command

The command of the source level debugger (dbx) used in UNIX. The N64 debugger (gdb) supports most dbx commands.

Device dependent system interface

The interface which removes functions such as naming or buffering, etc., which are seen in the device independent interface, and can only do the low-level (direct) operations.

Device driver

A control program used for controlling the peripheral equipment connected to the computer.

Device independent system interface

The interface which has common I/O functions (protecting, blocking and buffering) for various devices. You can operate it without knowing the detail configuration of device.

Device manager

N64 OS uses the high-priority thread for the device control. After registering the event, event message queue and message, the manager provides the device control with doing the I/O operation from the input command queue in order.

Disassemble

An operation which converts from the machine language program to the mnemonic language assembler. This can make difficult machine language program easier one. It is also called the reverses-assemble.

Dispatch

The scheduler allocates the CPU application right to next process on the multi-programming control.

DP command buffer

The buffer which stores the RDP display list.

Dump

To find out problems in the program, it provides the display or print-out of contents of the program or file.

Encoder

A machine or circuit to encode the data.

Entry Hi register

A readable and writable register used for the access to the highest bit of built-in TLB. Stored the information about the exceptional causing address if a TLB exception occurs.

Entry Lo0 and 1 register

A readable and writable register used for the access to the lowest bit of built-in TLB. The entry Lo register is

consisted of the event virtual pages, Lo-0, and the odd virtual page, Lo-1.

Error PC

The exceptional program counter. It indicates the virtual address value of the command which directly cause the exception, or the virtual address value of the just before branch or the jump command.

Event

A structure to manage occurrences of allocations or exceptions. N64 OS is defined occurrences or ends, etc., of SP, DP, VI and AI in advance.

Event flag

A flag which indicates occurrences or end, etc., of events.

Event notifier

Event notifier (Briticism). The event notifier system.

Fault hander.

->The system fault handler

Full associative

->Set associative. This system can place an address data on main memory to the arbitrary position on cache.

Game preamble code

The code added by makerom which provides the Bss clear, stack pointer setting and jumping to the root entry routine, etc.

Global data area

The global data area which is usable for all functions, not only in a certain function.

Grid

The fixed-size two dimensional grid. Split the two-dimensional area into the small area by using this. Reduce the process by changing the grid configuration and canceling a lot of unnecessary geometry.

gvd

The debugger which operates on the developmental workstation in the host and communicates with the game development board through the dbgif program.

Heap

The area which dynamically allocates memory. Heap is a large block of memory area and used by cutting out as occasion demands when the program is executed.

Idle thread

The lowest priority thread which does nothing except when the other threads do not operate. But, N64 OS hangs up if there is no idle thread.

In-line expansion

->In-line model

In-line model

A method which directly includes the concluded commands in the main routine without using subroutines.

Instance

Elements which forms data of mainly created arguments or objects, etc.

Instruction address

The effective address of the program command.

Instruction cache

A cache memory that temporary holds the program commands. This makes the command call speed up.

Interval timer

A timer which sends a signal at each certain constant interval.

Kernel

A core part to provide the allocation of the basic system resources on OS.

Latency

A delay time from inputting the address to memory to outputting the data.

Linkage editor

A tool which concatenates or edits some programs and create one machine language. It decides addresses or provides the process of concatenation of run-time library, etc.

Map

To place data from ROM to main memory and from main memory to cache, etc., and associate them.

Message

A structure to control the information sending/ receiving among several threads or the thread execution. By sending/ receiving messages, the high priority thread in the state of wait message can switch to the executing thread. Threads can communicate by using messages and synchronize.

Message queue

A queue which stores a message. The message specifies and sends a message queue.

Microcode

A micro-program command which is included inside of the processor (RCP) and controls operations of the logic unit, register or control flag, etc. By changing microcodes, it can add or change functions.

Microcode engine

A hardware device which processes with microcodes.

Multi-plexer

To select and output one necessary thing from several inputs.

Mutual exclusion

While it protects that each divide provides the I/O process, it excludes the other I/O process for the former device.

Naming

To name (identifier) the device used in the program.

Octree

Expanded a quad tree to 3D.

Overhead

The process or processing time which are not directly related to the user program, which OS provides the allocation or management of system resources and the process control.

Overlay

A technique to split the program to each function and overwrite, store and execute the only required things in the main memory to execute a large program.

Page mask register

A register to set the page sizes (4K, 16K, 64K, 256K, 1M, 4M, 16M) of each TLB entry.

Paging

By splitting the program into proper sizes called page and loading only the required pages, you can use the memory area effectively. Paging is different from the segment method since it splits a program into fixed-size suitably. So, it is hard for the program to use although the using efficiency of the memory area is high.

Parallelism

(Criticism) The parallel structure.

Performance profiling

->The profiler

Pipeline structure

A technique which speeds up the process by overlapping the process in CPU. Even if each component element takes 1 clock to process, you can get the output of 1-clock per 1-data when flowing the data continuously.

Pop

An operation which takes out data elements from the current stack pointer.

Position independent code

An independent code which can be placed anywhere of the address.

Pre-emption

->Pre-emptive system

Pre-emptive system

When the process which is higher priority than the executing one occurs, this system breaks the execution process and yield CPU for the higher priority process.

PreNMI

It is an interrupt to CUP when the reset switch for N64 is pushed When PreNMI happens, the current executed process will be in the waiting status for the coming NMI about 0.5 second later.

Pre-processor

A program which provides the pre-process of the translation program. Used for expanding functions without altering the compiler.

Procedure

It arranges a series of command and data

Profiler

A tool which measures the performance information, such as the number of times for calling, execution or the execution time, at each resource.

Push

The storing operation of data elements to the current stack pointer. *Stack is a LIFO structure.

Quad tree

Division of two-dimensional area by hierarchical structure consisting of different size grids.

RDB port

The port which couples Indy with the development board.

Real-time pre-emptive

->The pre-emptive system

Resource

A structure and function which are demanded by a thread or a task. For example, it indicates CPU, memory, the I/O device and RCP, etc.

Retrace interrupt

The interrupting process at the occurrence of vertical synchronizing signal.

Lock-up

Same as Hung-up.

ROM spec file

The file which describes the segment configuration of objects or unprocessed data files. Referred when the ROM image is composed.

Round robin scheduling

One of the scheduling methods. It does not give priority. It switches the process roundly at each constant time.

Runtime library function

At the program execution, the function is included to support the operation of aiming program and called by other programs.

Scheduling

To decide the execution order of several threads or the allocation order of system resources of the device, etc.

Segment

It can use the memory area effectively by splitting the program logically (the function module, etc.) and only loading the required segments. It differs from the paging system. Though it is easy for the program to use, the using efficiency of the memory area falls; because it becomes the variable length size.

Segment address

The RDRAM address specification of RSP is provided by using the segment address. The RSP microcode can control 16 segments, and the information, given as a segment, address stores the information of the segment ID and offset. The physical address is searched for adding the segment offset to the base address which found from ID.

Segment base register

A register whose role is a segment table. It is stored the segment base register corresponding to the segment ID.

Segment offset

->The segment address

Segment table

A table used for calculating the physical address of the program which is split into segment.

Semaphore

The integer type argument used for controlling the synthesis between processes in the multi-task system, etc. It has a meaning of the flag semaphore, and it indicates to control the synthesis by signals without any problems.

Service routine

The support program to use the system effectively. The file creation, utility change, library editor which manages the library, linkage editor and debugger, etc.

Set associative

A system to split the cache memory to some groups and place the data of a certain address on main memory to the decided position in the split area. If the number of splitting is 2, it is called the 1-set associative. If the split number is the block number of the data area, it is called the full associative.

Sing-on

Provide the active post

Sorting process

The process which rearranges data in order based on the specified items.

Source level debugger

A debugger which can debug the high-level language like C language verifying with source code.

Span buffer coherence problem

If 2-span continuously provide rendering to the same pixel, the frame buffer value which is unprocessed by the first span becomes the input frame buffer, because the second span does not wait for the first span's process ends. The problem is to read the wrong value and return the incorrect value to the frame buffer.

Stack

The temporary used memory data which has the structure (LIFO) that the final stored data is taken off first.

Stride control

A control to make the boundary aligned memory data a large block by continuously accepting and connecting it.

Subset

Taken off the unnecessary parts to meet the specifications and cut off only required parts from the OS system.

Swap clock

To split the CPU process time and allocate them to the executable state process in order.

Symbol table

A table relating the information of program or data segments, etc., to the symbol (name).

System call

To execute the OS function from the program. Also, the calling name of its each OS function unit.

System exceptional handler

->The system fault handler

System execution queue

A queue to store the executable state thread in the system-forming threads.

System fault handler

Provide the process corresponding to the exception if it occurs. In N64 OS, this handler find out the message queue and message from the corresponding event table and send the message to the applicable message queue, when it receives the exception or interrupt.

System thread

Threads to control DMA or the I/O operation. Device manages such as PI or VI are just higher priority threads.

Thread

The basic unit of the CPU allocation on N64 OS. The thread works with the pre-emptive system.

Topology

(Britishism) The geometric shape

Trade off

(Britishism) The state of "quite unable to satisfy both sides."

Vertical retrace

The vertical synthesis of scan lines of the TV screen.

Virtual address

The address of the virtual memory space which is beyond the packaged memory. Use the memory control unit. Simply use it if the calling data is in the physical memory. Apparently, a user sees that is has a large memory space.

Wave

A wave statement defined in the spec file. In the wave, defined segments are registered, and registered segments, in the wave, themselves can use the symbol.

Writeback cache

When the data writing occurs, this system first writes data to the cache memory and writes back to main memory as the cache memory overflows. Also, the cache memory of the system itself. Because the writing frequency to main memory whose access is slow decreases by this, the process speeds up.

3. Terms about audio

ABI command

A command interpreted and executed by the microcode (audio microcode) for waveform synthesis. Created by the waveform synthesis driver.

ADPCM

The abbreviation of "Adaptive Differential Pulse Code Modulation." This compression method of sampling data predicts the next sampling value from the past sampling value and encodes the difference between its predicted value and the actual one. It can compress the data amount to about 1/4.

ADSR

"Attack time, Decay time, Sustain level, Release time. " One of the configurations of envelope. A (attack time) - first transition time. D (decay time) - decay time until lasting volume. S (sustain level) - the level of lasting sound. R (release time) - the time that sound decays and disappears.

AIFC/AIFF

A form of sound data. AIFC is added the correspondence to compression CODEC on the AIFF form.

Attack time

->ADSR

Bank

->The bank structure

Bank file

Defined that it concludes .tbl file which stores the wave table data that is ADPCM compressed, .ctl file which stores the control information such as key map, envelop or gain, etc., of the wave table and .sym file which stores the symbol information of bank file.

Big room

The sound effect that you hear the sound as if it were produced in a big room.

Call back

->The DMA call back.

Cent

An interval of 1/100 of a semitone. 1 octave is 1200 cents.

Check

The resolution specified to the header of MIDI sequence, and the unit time related to the MIDI clock.

Chorus

To mix a single-sound with delayed and modulated sound, and to give the sound thickness by creating effect like chorus.

chorus/flange form

-> The chorus/flange effect made by mixing chorus/flange sound.

Code book

ADPCM. A table of estimated coefficients used for optimizing the sound quality when providing the estimated-coefficients ADPCM compression.

Compact MIDI

The N64 original MIDI format which is compressed the standard MIDI format. It is played by a compact sequence player. The compact sequence player -> The sequence player.

DC normalization

The abbreviation of "Discrete Cosine normalize." By doing the orthogonal transform (discrete cosine transformation) of input signals, it splits into the direct current and alternating current components, and takes out the direct current component. You can do the discrete cosine transformation and reverse-transformation in real-time by using DSP.

De-tune

To slightly shift the sound frequency

Decay time

->ADSR

Delay line

A device used for providing effect. It provides the frequency modulation or amplitude modulation of sound and implements echo or flange.

Delta time

Displayed the time between events in units of chick.

DMA call back

As occasion demands, it loads the waveform data in ROM to RAM depending on DMA. Because this method does not have to leave the unnecessary data in RAM, the amount of RAM required for audio decreases greatly.

Dry

The state not manipulated by effect. It can be set from 0 to 127. 0 indicates sound dry completely and 127 indicates sound wet completely.

Echo

A phenomenon that the sound from voice source reflects at wall, etc., and you hear it later than the sound which directly reaches. Also, the sound itself.

Effect

To manipulate the encoded sound and create the different sound from original. It has vibration, tremolos or echoes, etc.

Envelope

The information to define the changeable volume with timing. The N64 audio library uses the ADSR envelope.

Event

The definition of sound change occurring with time. Note-on, note-off and control change, etc.

Flange

One of the effect. To make input sound slightly delay and make the frequency component exaggerate by adding it to the original sound.

FX TYPE

Information used as an effect type. It has nothing, the small room, the big room, echo, flange, chorus and custom.

Gain value

IT expresses the transition ratio of the output signal responding to the transition of the input signal in "dB."

Instrument

In the N64 audio library, this is the whole information of sound, envelop, key map or pan, etc.

Instrument compiler (ic)

A tool to create bank files (.tbl, the waveform data file, .ctl, the control file and .syn, the bank file sign information file) from some compressed AIFC sound data.

Key base

The definition of MIDI note number suited to sound played with the standard pitch. It does not have sound from all keys, but provides the pitch shift from the difference between specified keys and the key base, then create sound

of specified keys.

Key map

It defines where of pitch extent and what velocity the object sound replies to. Make up a party of these information and sound, and select the suitable sound depending on the scale and velocity.

Key number

->The note number

Magnitude value

(Bricicism) The volume.

Meta-event

The event that MIDI event does not have but the user can define. It has tempo or end-of-track, etc.

Meta-status byte

Information to identify the meta-event. The top of the meta-event always begins from OxFF.

Middle C

The center C4 (MIDI note number 60).

MIDI

The data transfer standard to exchange sound information such as volume and length, etc., among a synthesizer, sequencer or computer, etc.

MIDI clock time

A timing when it controls MIDI equipment.

MIDI daemon

A tool used for performing the MIDI data by N64 in real-time.

Mixing

To mix several independent notes.

Modulation depth

->A degree of the modulation rate pitch change.

Modulation rate

The delay time from the input sound.

Nest

To nest. To fit small data blocks into a large data block.

Note number

The number indicated the scale. As setting 60 for the center value, it can set 128 stages.

Note-on and note-off

A MIDI command used when sound occurs (note-on) or stops (note-off). Note-on and note-off must always exist as a pair. Instead of note-off, "note-on but the velocity is 0" also used.

Pan

The orientation information of right and left sound.

Physical voice

->Virtual voice. The voice which actually provide the wave form synthesis. It corresponds to the sound process module which is consisted of the ADPCM decompressor, pitch shifter, and the gain unit.

Pitch

The sound interval be frequency.

Pitch bend

To change the pitch continuously up and down. Same effect as with the "choking" of a guitar.

Playing rate

The sample number that AI (Audio Interface) processes per second.

Polyphonic key pressure

Information to provide effect individually for some specific note when it plays several notes at the same time.

Predictive coefficient

When you provide the ADPCM compression, you estimate the current sampling value from the past sampling value and encode the difference between its estimating value and the actual value. This predictive coefficient is used for searching for the estimating value, then.

Prefix

A prefix. The file name attached to *of *.ctl, *.tbl and *.sym when providing the compilation with ic.

Program number

The number used for the program change of the MIDI sequence associated to each instrument.

Redundancy mode

The mode displaying the information, like debugging, if the operation is provided normally, etc. Set it with the option when executing ic or MIDI daemon, etc. IT must be a signal pole primary (one pole).

Release time

->ADSR

Resample

To provide sampling again by shifting the pitch for converting the ADPCM output data to the arbitrary pitch. Also, called the pitch shift process.

Reverb format

The reverb effect created by reflecting sound.

Root pitch

The note pitch based on the code called the root note.

Sampling

To extract the signal size from the timely continuous analogs at each suitable time interval.

Sampling rate

The sampling number provided in one second. The higher the sampling rate is, the more sound quality improves; and the more you need memory.

sbc

A tool creating a sequence bank file from some sequence files.

Semitone

A half tone.

Sequence player

The sequence player has two types; the MIDI sequence file player of Type 0 and the compact MIDI sequence player. These players have the same functions except the compression format and the loop process. While the MIDI sequence player of Type 0 can provide the loop process from outside, the compact MIDI sequence player need to embed the loop point inside of the sequence.

Sequence position marker

It marks on a position of Type 0 sequence data and is used for setting the playing time or loop point.

Sequencer

The device or software which continuously control the data arranged in order.

Small room

The sound effect that you hear the sound as if it were produced in a small room.

Sound effect

The effective sound or music added to raise the atmosphere with the game image.

Sound player

A player to play a single sound effect or streamed audio. It can play the sound of the ADPCM compression or 16-bit non-compression format.

Stealing voice

The low-priority virtual voice, allocated by the physical voice once, is taken over by the high-priority or same-priority virtual voice.

Streaming

To make the constant amount of sound data flow all the time by the real-time process.

Subtype byte

Stored the information which defines the MIDI event type.

Sustain level

->ADSR

Tag

The attached information to identify data.

Tap value

A coefficient (gain) for output of each effect which is output from the delay line.

Tremolo

To provide the low frequency modulation to the sound quantity and change it large and small.

Type 0/Type 1 MIDI

Forms of the MIDI sequence data. A general MIDI sequencer uses a Type 1 MIDI file. As the N64 audio library can only use the sequence data of Type 0 MIDI file, you need to convert to the Type 0 MIDI file by using midicvt.

Velocity

The rate to strike the keyboard. Usually, the information indicating the sound quantity.

Vibration

One of the effect. It makes the sound frequency (pitch) go up and down shake by the low frequency modulation.

Virtual voice

Expressed using the ALVoice structure, this is for the player's convenience. To play the sound, it needs to allocate players' virtual voices to the play-sound. It allocates to the physical voice in order of high priority.

Volume

Loudness.

Wave table

It provides the ADPCM compression to raw sound data and concludes it to a bank.

Wet

The state manipulated by effect. It can set from 0 to 127. 0 indicates sound dry completely and 127 indicates sound wet completely.

Appendix C Samples Description

The following files are attached about the sample programs.
These are parts of sample about the contents of this manual.

- **Graphics Drawing Samples
(contents)**

1. 2D image drawing sample
(/sgi/allman/samples/2d.tar)
(/pc/allman/samples/2d.lzh)
2. 3D image drawing sample
(/sgi/allman/samples/3d.tar)
(/pc/allman/samples/3d.lzh)

- **Audio Playing Samples
(contents)**

1. Sample showing how to reproduce sound effects
(/sgi/allman/samples/effect.tar)
(/pc/allman/samples/effect.lzh)
2. Sample showing how to reproduce a MIDI sequence
(/sgi/allman/samples/sequence.tar)
(/pc/allman/samples/sequence.lzh)

- **Scheduler Sample
(Content)**

1. Sample showing how to effectively mix graphics and audio
(/sgi/allman/samples/mix.tar)
(/pc/allman/samples/mix.lzh)

- **Please unzip the *.tar files under the /sgi/allman/samples/ directory for the samples programs mentioned above.**

STEP 3 [N64 3D Graphics]

In STEP 3, "N64 3D Graphics," we will explain the theory and practical use of the basic matrix, coordinate system, etc. Additionally, we will cover the 3D image, which is the most important factor for N64 application production. Our goal is to provide a basic understanding about the mathematical handling of the 3D image. We have also included illustrations to help explain complicated areas. Refer to these along with "The N64 Function Reference Manual," etc.

A list of resources cited has also been attached at the end of the manual for your reference.



Introduction to **NINTENDO**.⁶⁴

Chapter 1 Basics of 3D Graphics

This chapter explains the basic theory of drawing 3D objects.



1-1 Basic 3D Concepts

Basically, when you are working with 3D computer graphics, you are drawing pictures in a virtual three-dimensional space created in a computer, so your two-dimensional drawings have added depth information.

For example, when you draw a two-dimensional picture on a computer, you draw dots by specifying the color of each pixel (picture element) with graphic tools as in this illustration:

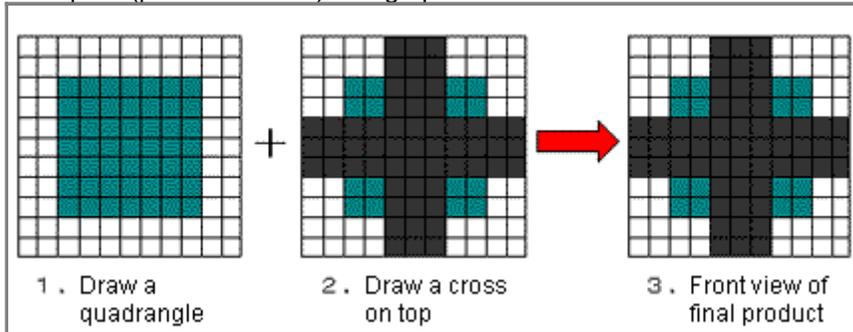


Figure 1-1-1 Two-dimensional drawing

However, when drawing a three-dimensional picture, you draw dots that have both color and depth (z axis) information. Because of this additional depth information, objects that lie behind other objects are partially or completely hidden as illustrated here:

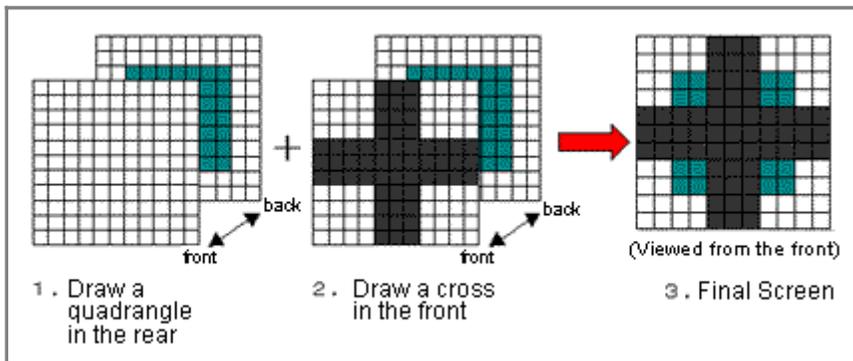


Figure 1-1-2 Three-dimensional drawing

It looks like there is no difference between the two-dimensional final screen and the three-dimensional final screen. But the difference is obvious if you change the drawing order as shown here:

Because the two-dimensional picture does not have the depth information, a picture drawn on top of another picture always wipes out what is underneath it. However, because the three-dimensional picture has depth information, the relationship between two objects holds regardless of the drawing order.

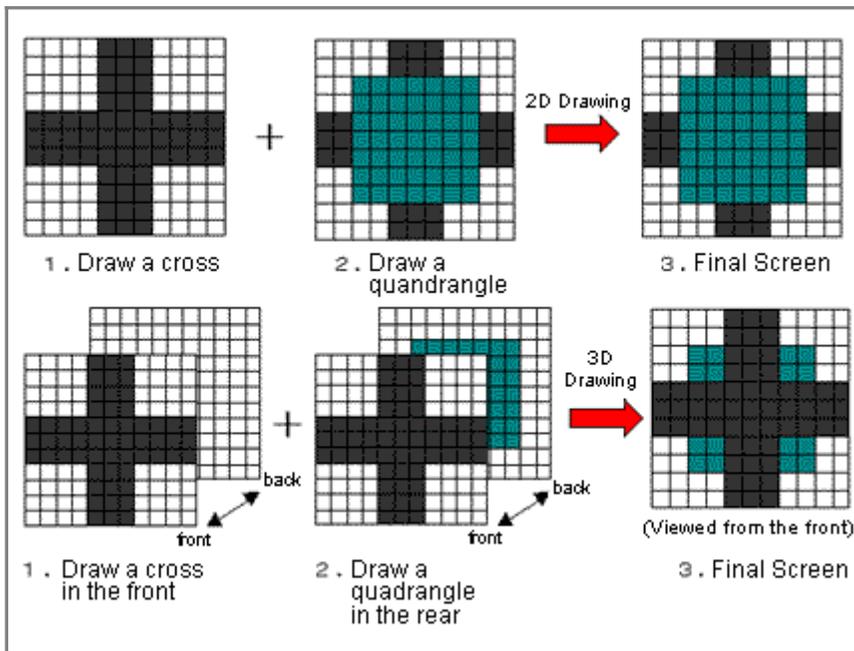


Figure 1-1-3 Differences in the drawing order

1-1-1 More on 3D

Various objects exist in the three-dimensional space surrounding us. As light falls on a three-dimensional object, some parts are lit up and other parts are dark or in the shadows. It all depends on the source and intensity of the light. Also, objects that are far away appear relatively small and appear to grow larger as they move closer and closer. To make objects appear three-dimensional, you need to apply these visual effects.

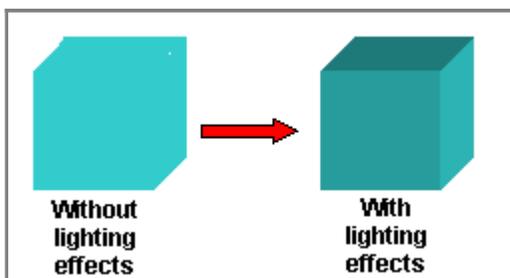


Figure 1-1-4 3D Lighting Effects

Because 3D graphics data includes three-dimensional information, you can use computer calculations to apply appropriate 3D lighting and perspective effects. Moreover, you can use calculations to rotate and scale objects. These kinds of data manipulations by computer calculation are called **transformations**.

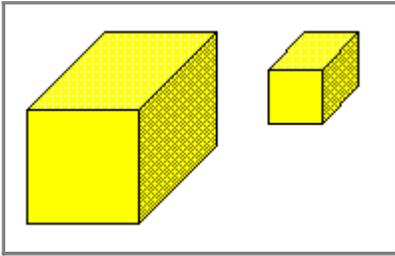


Figure 1-1-5 Perspective by size

1-1-2 General 3D Graphics Process

The basic 3D graphics process is a three-step process.

1. Create the model

First of all, you need to create the data for the object you want to draw. In general, an object to be drawn in computer graphics is called a **model** and the set of coordinates and other data belonging to a model is called **model data**. To create model data effectively, you can use any of the many available 3D graphic tools called 3D Modelers.

You also create model data on graph paper by hand, and then feed it into a computer. However, this method is complicated and requires a lot of work, therefore it is not a recommended way to create a large model.

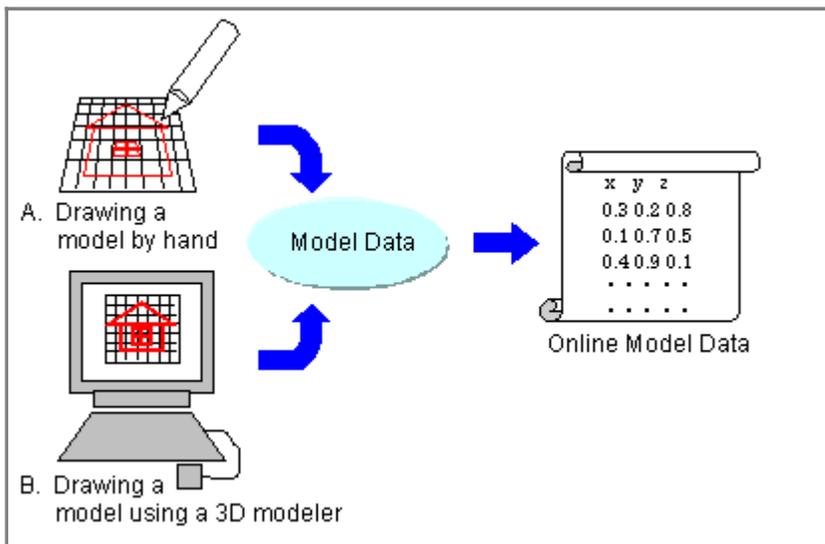


Figure 1-1-6 Creating the model data

2. Convert the model data

Next, you need to convert the model data, and place it in the virtual three-dimensional space of the

computer. Then provide for the conversion calculations to show rotation, scale, movement, lighting, perspective, and so on. As a result of the conversion calculations, you can display a model that will be viewed from all directions. Moreover, it will be a solid image that has depth as shown by the lighting visual effects.

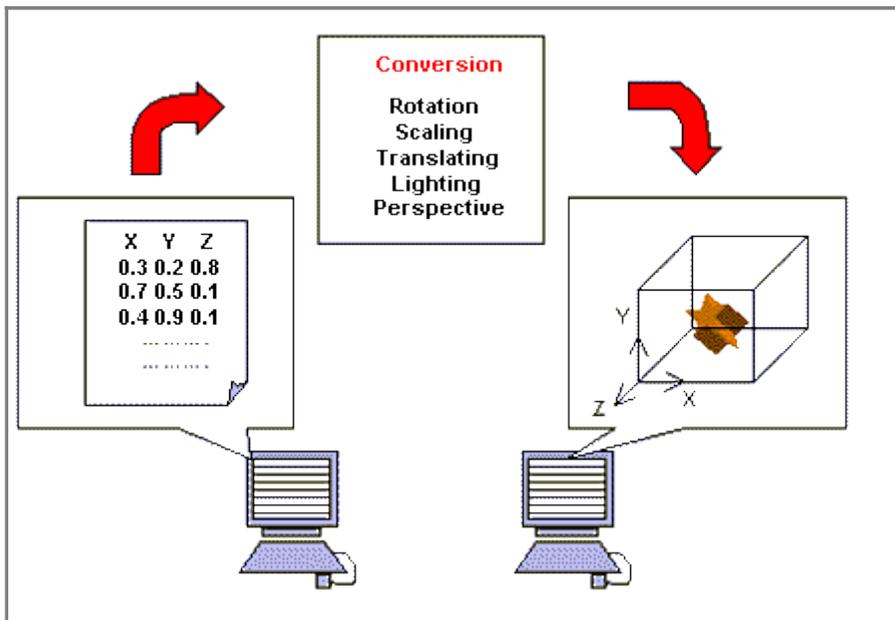


Figure 1-1-7 Conversion of the model data

3. Display model on TV screen

Finally, you need to send the model data in the virtual three-dimensional space of the computer to the TV screen by way of various software and hardware processes. Ultimately, when the converted model is displayed on a two-dimensional TV screen, the depth (z axis) information is no longer needed. Pixels (picture elements) are displayed on the screen.

This conversion by the computer process has various aspects and produces a variety of effects. For example, you can express a translucent object by taking into consideration the level of transparency when composing a screen.

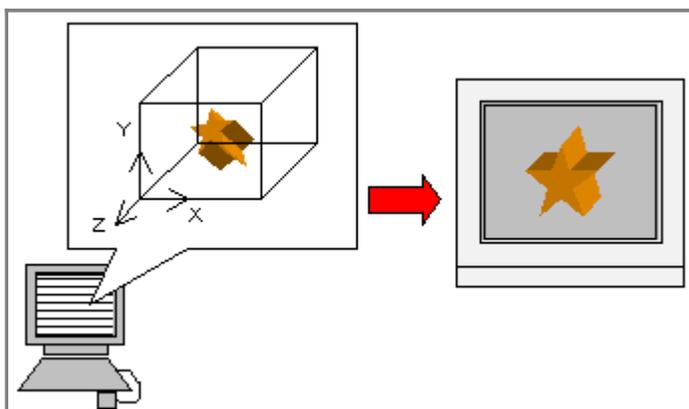


Figure 1-1-8 Display on monitor

1-2 3D Shape Specification

Next, the process of actual drawing when expressing a three-dimensional object on a computer by using triangles or quadrangles will be described.

1-2-1 3D Shape Specification

Each three-dimensional shape is made up of a combination of the following three elements:

- 1.Vertices (corners)
- 2.Edges (lines) that connect vertices
- 3.Planes (surfaces) surrounded by edges

You can render any three-dimensional object on the computer by creating a detailed database of these three elements.

1-2-2 Using Vertices, Edges and Planes

Using vertices, edges, and planes to render three-dimensional models is more complex than it is to use them to render two-dimensional models. To render a cube in three-dimensions, you need information about 8 vertices, 12 edges, and 6 planes as shown here:

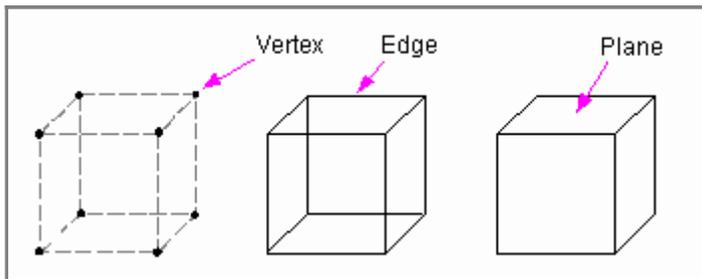


Figure 1-2-1 Using Vertices, Edges and Planes

You have to be careful when connecting vertices. If you simply connect vertices to make edges, you may end up rendering an object different from the one you want, as shown here:

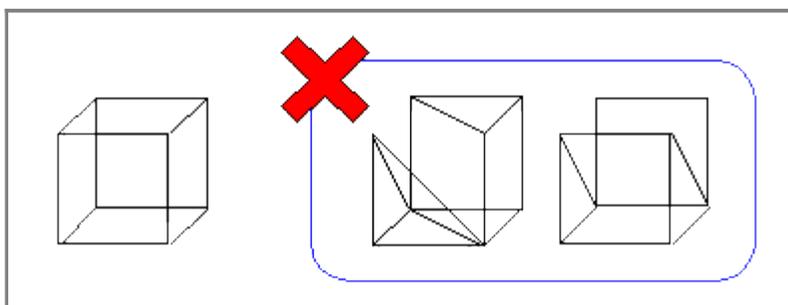


Figure 1-2-2 Example of connecting between vertices

Also, for an object ([surface](#) mode) having planes, many three-dimensional processes have rules for creating these. The following three rules are representative of these.

No Open Sides

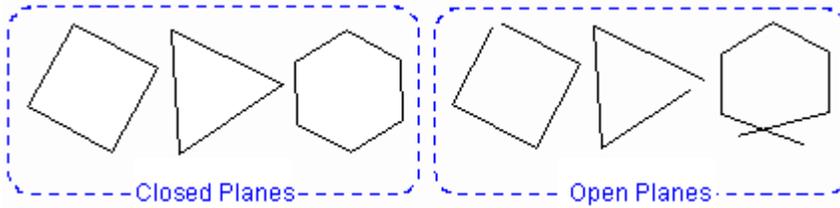


Figure 1-2-3 Rule One About Creating Planes

No Dents

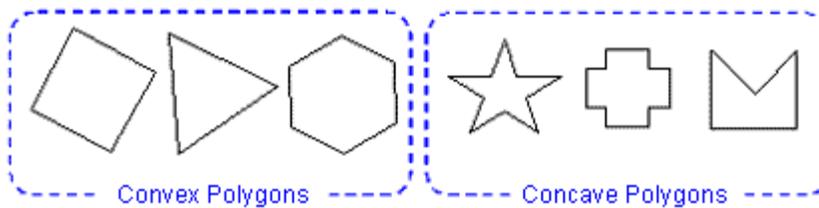


Figure 1-2-4 Rule Two About Creating Planes

No Twisting

Vertices must be on the same plane, and edges must not intersect

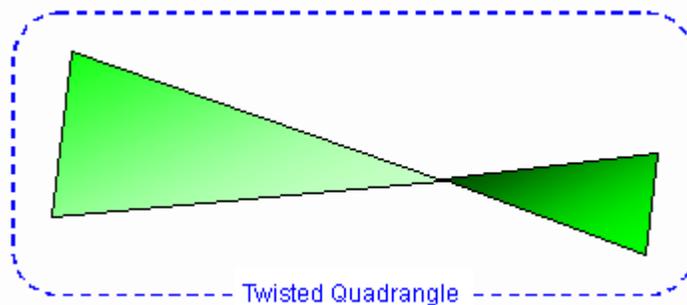


Figure 1-2-5 Rule Three About Creating Planes

If you create and draw planes that do not follow these three rules, you will not get the correct result. Therefore, you need to be careful when creating planes.

Rendering a model that has many planes takes a long time. Therefore, if you try to render a complex model in real-time, you may encounter problems. For real-time rendering, you have to pay close attention to the drawing rate. You need to decide how many surfaces (planes) your game can support. A model with few surfaces, like the cube on the right in the following illustration, requires a relatively "light" drawing process as compared to a model that has many surfaces, like the sphere on the left:

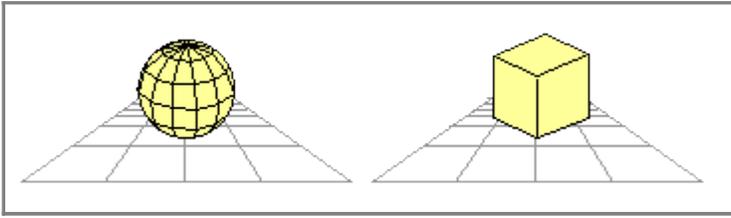


Figure 1-2-6 Model with many surface and Model with few surface

To help solve this problem, there is a technique called "back face culling" which ensures that no unseen back surfaces are drawn. This reduces the number of planes to be drawn and thus lightens the amount of work required.

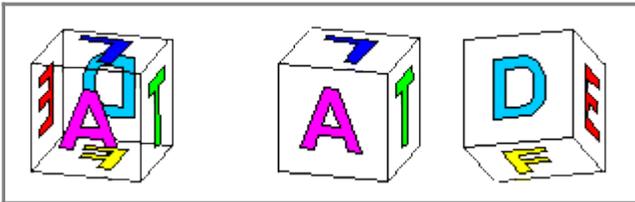


Figure 1-2-7 Back Face Culling

The next problem is how we should judge the front and back of planes. Something obvious to a human being can be a difficult process for a computer. In general, computers judge the front and back by the order of edges created to connect vertices. In the N64 system, planes that are rendered by connecting vertices in a counter-clockwise manner become the front.

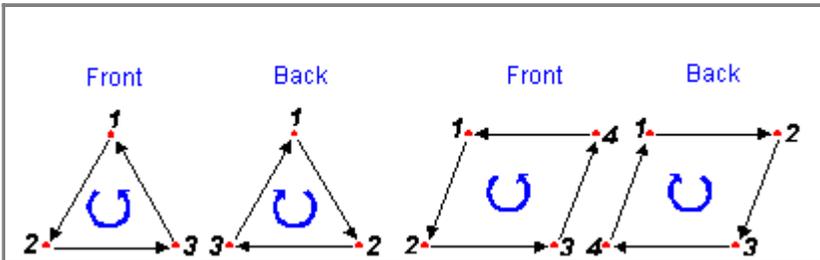


Figure 1-2-8 The order of edges created by connecting vertices

You can distinguish front and back for the computer by using a normal vector. This technique will be explained later. It is important for the computer to recognize the front from the back so that it can improve the drawing rate by not doing the drawing calculations for the back side planes.

Pay attention to all the issues discussed here when creating your planes. Then combine the planes to complete a three-dimensional object as shown here:

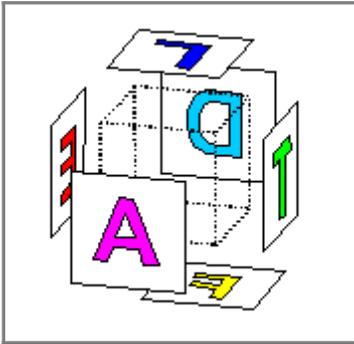


Figure 1-2-9 Combine planes and complete three-dimensional object

Use the following N64 functions to draw three-dimensional objects:

vertices->sides->surfaces

Examples of N64 Functions to Draw 3D Shapes

[gSPVertex](#) (load the vertex data)
[gSPModifyVertex](#) (change the vertex data)
[gSP1Triangle](#) (draw one triangle)
[gSP2Triangles](#) (draw two triangles)
[gSPSetGeometryMode G_CULL_BACK](#)
 (turn on back face culling)
[gSPClearGeometryMode G_CULL_BACK](#)
 (turn off back face culling)

1-3 Coordinate System

1-3-1 Right/Left Hand Coordinate System

Next, the coordinate system for 3D graphics will be described. A three-dimensional coordinate system can be right-handed or left-handed. The Z-axis direction compared to the X-axis and Y-axis directions, is different. In the right-handed system, **the Z axis points out** toward the viewer. In the left-handed coordinate system, **the Z axis points in** away from the viewer. N64 uses the right-handed coordinate system as do most others.

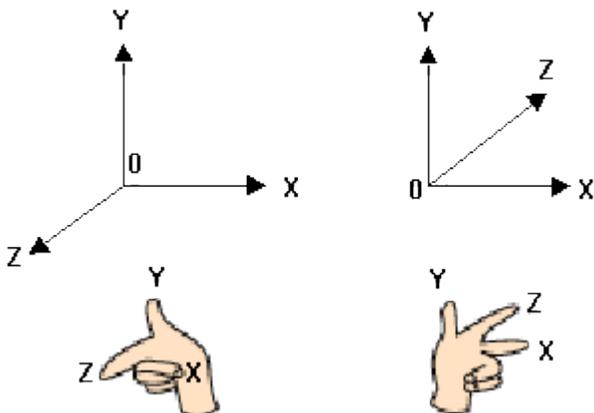


Figure 1-3-1 Right/Left Hand Coordinate System

1-3-2 Model Coordinate System

The model coordinate system is the base system used when creating model data for a specific design of an object. Generally, when modeling several objects, you define the coordinate origin, axes, scale, and so on individually so that it is easy to model each object. For example, when modeling a robot's arm consisting of a forearm and an upper arm, you use an individual coordinate system for each of them. Depending on the modeler, the range used as a model coordinate, may be normalized to $-1.0 \leq x < 1.0$, $-1.0 \leq y < 1.0$, $-1.0 \leq z < 1.0$. In this case, you have to create the model data within this limitation.

You can use the model coordinate system to define each modeling object independently. For example, when you model an airplane like the one shown above, you might define a length of 1 meter as 0.1 and do modeling in a space where one edge is 20 meters (because the range of each axis is from -1.0 to 1.0). When you model a human head, you could define 1 centimeter as 0.05, and then do the modeling in a space where one edge is 40 centimeters. The coordinates defined in the model coordinate system are expressed as (x_m, y_m, z_m) . Note that there is no uniform model coordinate system that all 3D modelers of each company use. Each 3D modeler is free to set up their own model coordinate system.

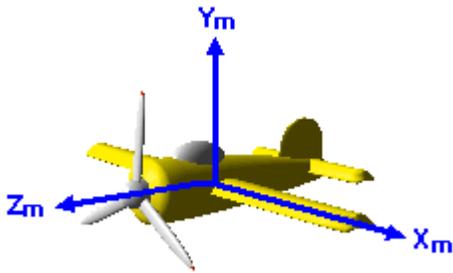


Figure 1-3-2 Model Coordinate System

1-3-3 World Coordinate System

The world coordinate system is like the coordinate system of a photo studio where all the objects are placed together to prepare for a photograph to be taken of the model. In the world coordinate system each three-dimensional object is in a defined location.

The world coordinate system needs a very wide range depending on the world you want to render. For example, in the case of a racing game, you might define 1 meter as 1.0 and render up to 1 kilometer (1,000 meters). In the case of a fighting game, you might define one edge of the world coordinate system to be only 10 meters.

Calculations are necessary when you place each object using a model coordinate system into the world coordinate system. For example, say you define 1 meter as 1.0 for your world coordinate system. When you place the previously illustrated airplane (which was defined with 1 meter as 0.1) into your world coordinate system, you need to reduce the size of the model data to 1/10th its previous size. When you place the human head which was defined with 1 centimeter as 0.05, into the same world coordinate system, you must reduce its size to 1/500th of its previous size. The world coordinate system's coordinates are expressed as (x_w, y_w, z_w) .

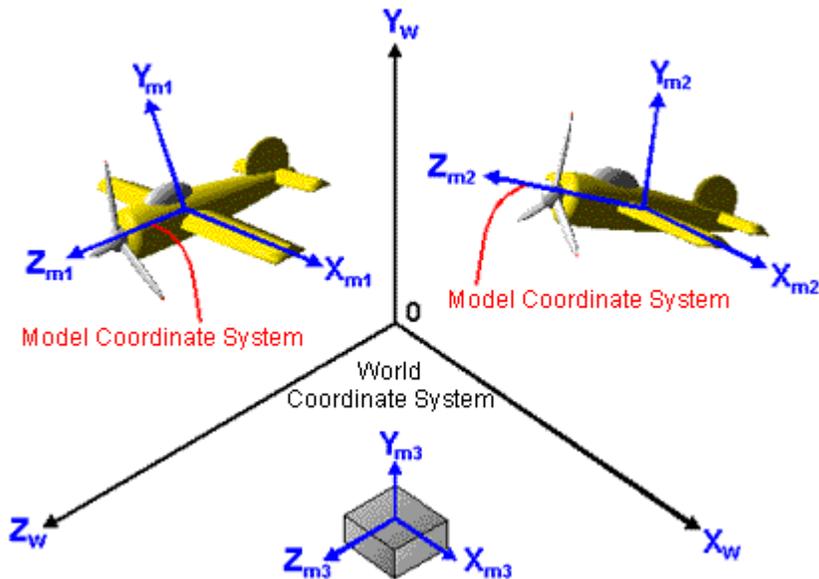


Figure 1-3-3 World Coordinate System

1-3-4 View Coordinate System

The view coordinate system puts the view (camera) at the point of origin and places the direction of the view along the Z-axis. Sometimes it is called the **view point coordinate system**. Because the drawing processor (the RCP) is like a camera, this coordinate system is what the RCP hardware uses to view the virtual 3D world set up in the computer. The range actually seen is the part that falls inside the rectangular pyramid which is called the "view volume." This becomes the visual field, and a vertex of the rectangular pyramid is like the focus of a camera. The distance between the screen and focus is equal to the focal distance.

When the distance between the focus and screen is long, the angle of the visual field is narrow, and the image seems to be taken with a telephoto lens. When the distance is short, the angle of the visual field is wide and the image seems to be taken with a wide-angle lens. (x_v, y_v, z_v) .

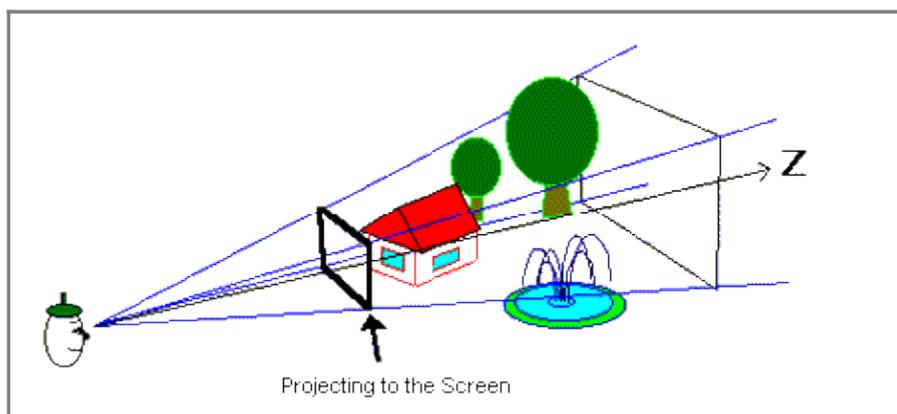


Figure 1-3-4 View Coordinate System

1-3-5 Screen Coordinate System

The normal screen coordinate system takes a picture of a view that is currently represented in the view coordinate system to place that view in much the same way as a camera takes a picture of a view to record on film. In other words, the normal screen coordinate system converts a view volume of a quadrangular pyramid in the view coordinate system into a view volume of a rectangular solid screen by applying conversion algorithms. By using perspective transformation calculations, the conversion process ensures that objects that are located further away appear smaller. Thinking of the normal screen coordinate system as a device for displaying the virtual 3D space created inside your computer with less physical restrictions, such as the size for a frame buffer, makes it easier to comprehend. As with the view coordinate system, the direction of view is the Z-axis and the center of the screen is the origin. Coordinates in the normal screen coordinate system are expressed as (x_s, y_s, z_s) .

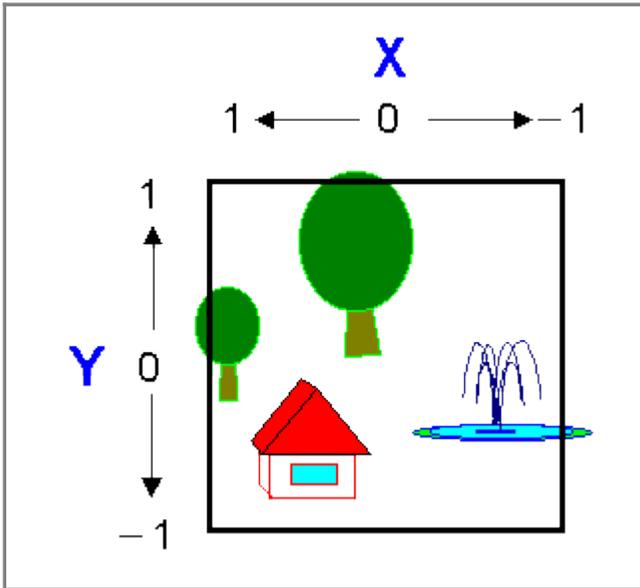


Figure 1-3-5 Screen Coordinate System

1-3-6 Device Coordinate System

The device coordinate system is the coordinate system used to fit an image into the physical size of the frame buffer by using the pixel addresses in the frame buffer. Generally, the physical size of the N64 frame buffer is 320 by 240. The upper-left corner of the screen is (0,0) and the bottom-right is (319.75, 239.75). A Z value factor is also included. The deepest (furthest away) value is expressed as 65532 (0xffff) and the closest value is expressed as 0. The [Z buffer](#) uses a macro that sets the initial value of the Z factor by using the macro-arguments G_MAXFBZ and GPACK_ZDZ. The macro converts an image currently in the normal screen coordinate system into the actual physical screen pixels so that it can be shown on the TV screen. Coordinates in the device coordinate system are expressed as (x_d, y_d, z_d) .

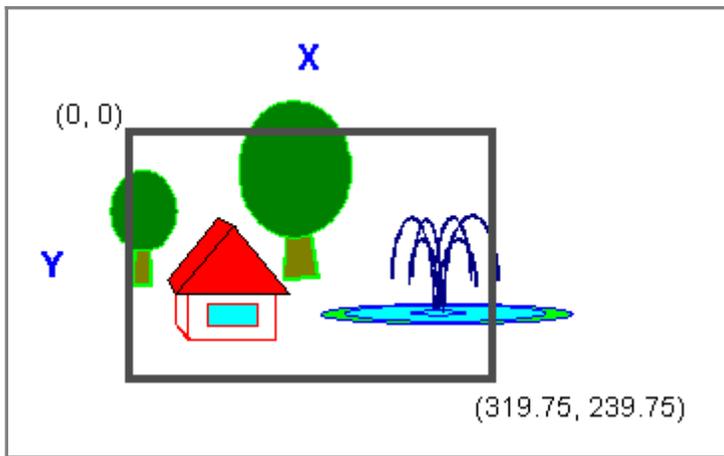


Figure 1-3-6 Device Coordinate System

1-3-7 Coordinate Transformation Flow

A vertex defined in a model coordinate system is converted to other coordinate systems and displayed. The following illustration gives a summary of the flow of this conversion process: Moving from one coordinate system to another is called "coordinate transformation." The coordinate transformation processes are represented by arrows in the illustration. The actual coordinate transformations are easily accomplished by using matrix calculations which will be described later.

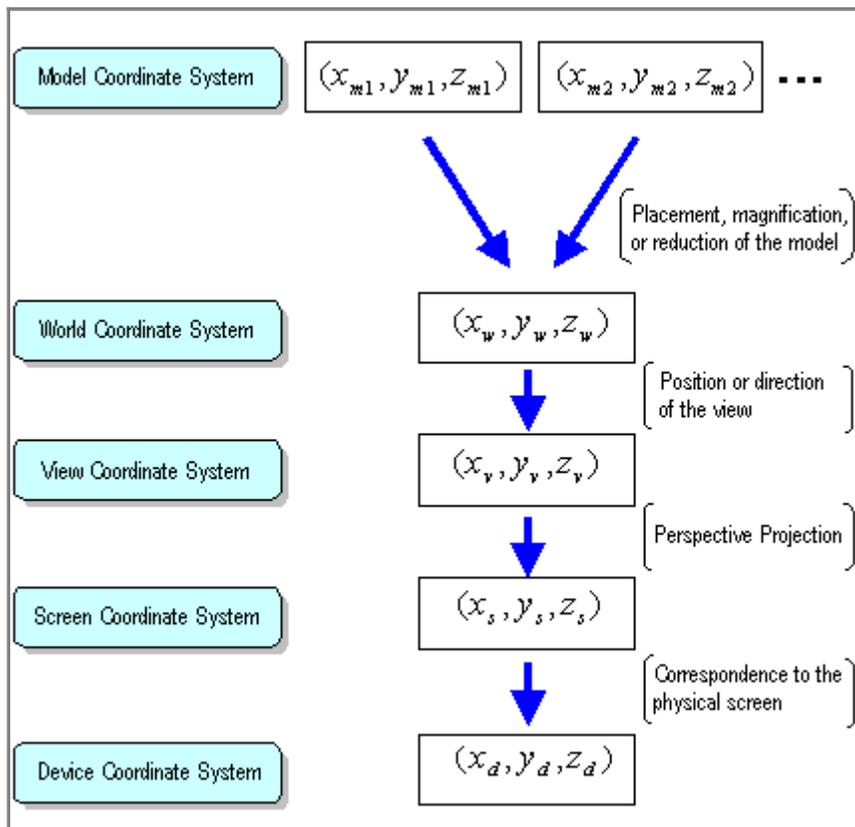


Figure 1-3-7 Coordinate Transformation

1-4 Vector Basics

It is easier to place an object into a three-dimensional space if you first move the vertex coordinate to its starting position. Then in your game, you can apply changes to the objects such as translating, rotating, or scaling them -- all of which are easily accomplished by using vectors and matrices.

1-4-1 Vector

A scalar is a quantity that has only one magnitude on a specific scale such as length, time, or volume. A vector is a quantity that has both magnitude and a direction such as force or rate.

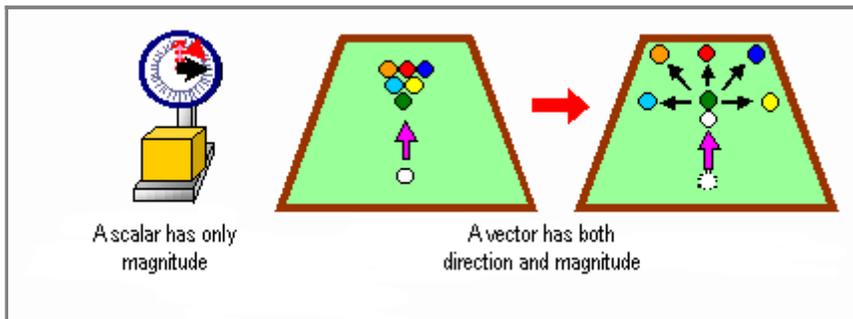


Figure 1-4-1 Scalar and Vector

The following shows a vector moving from point A to point B:

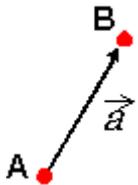


Figure 1-4-2 Vector \vec{a}

The direction of the arrow is the direction of its vector. A is called the initial point and B is called the terminal. This vector is expressed as \overrightarrow{AB} or \vec{a} .

1-4-2 Vector Addition

Given two vectors, \vec{a} and \vec{b} , you can add them together to define a diagonal vector, which in turn specifies a parallelogram as shown here:

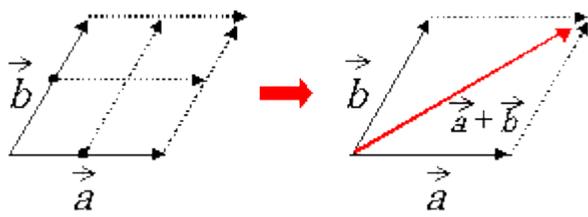


Figure 1-4-3 Vector Addition

The following shows how vector addition is done:

$$\vec{a} = (ax, ay, az) \quad \vec{b} = (bx, by, bz)$$

$$\vec{a} + \vec{b} = (ax + bx, ay + by, az + bz)$$

1-4-3 Vector Subtraction

To subtract vector \vec{b} from vector \vec{a} , invert vector \vec{b} 's direction first, and then add vector \vec{b} to the inverted vector \vec{b} to form the resulting vector $(\vec{a} - \vec{b})$:

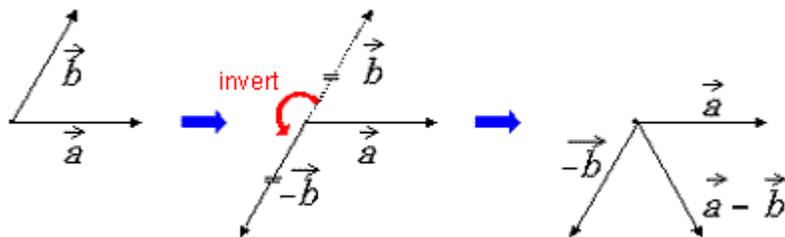


Figure 1-4-4 Vector Subtraction

The following shows how vector subtraction is done:

$$\vec{a} = (ax, ay, az) \quad \vec{b} = (bx, by, bz)$$

$$\vec{a} - \vec{b} = (ax - bx, ay - by, az - bz)$$

Scalar Multiplication of a Vector

To find the product of a vector \vec{a} and a scalar (k), multiply the length of vector \vec{a} by k, but be careful to take the sign of the scalar into account:

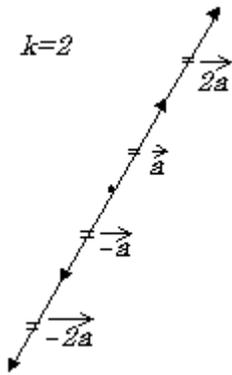


Figure 1-4-5 The product of vector \vec{a} and scalar (k) is multiplied vector \vec{a} by (k)

The following shows how scalar multiplication of a vector is done:

$$\vec{a} = (ax, ay, az)$$

$$k\vec{a} = (k \cdot ax, k \cdot ay, k \cdot az)$$

1-4-4 Normal Vector

A vector that expresses the vertical direction of a surface is called a **normal vector**:

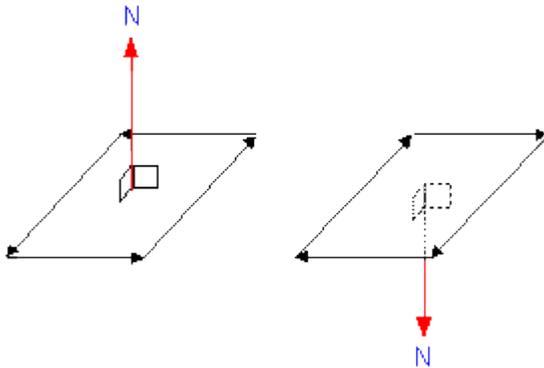


Figure 1-4-6 Normal Vector

To calculate the normal vector of a surface, you need to find what is called the cross-product of the two vectors that specify that surface. You can use the following step-by-step technique to calculate the cross-product of two vectors \vec{a} and \vec{b} :

$$\vec{a} = (ax, ay, az), \vec{b} = (bx, by, bz)$$

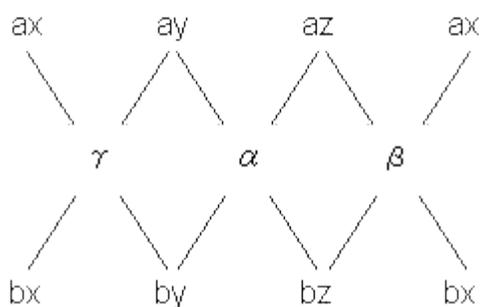
1. Arrange and write down the coordinates of each vector \vec{a} and \vec{b} as two lines like this:

$ax \quad ay \quad az$
 $bx \quad by \quad bz$

2. Repeat each X coordinate of \vec{a} and \vec{b} on the far right end of each line like this:

$ax \quad ay \quad az \quad ax$
 $bx \quad by \quad bz \quad bx$

3. Draw connecting lines like this:



4. Calculate Alpha, Beta, Gamma using these formulas:

$$\alpha = ay \times bz - az \times by$$

$$\beta = az \times bx - ax \times bz$$

$$\gamma = ax \times by - ay \times bx$$

Alpha is the coordinate of the result, Beta is the Y coordinate, and Gamma is the Z coordinate.

5. To obtain the cross-product of vectors \vec{a} and \vec{b} , use this formula:

$$(\alpha y \times bz - az \times by, az \times bx - ax \times bz, ax \times by - ay \times bx)$$

Here is a second example that shows how to find the normal vector \vec{N} of the plane defined by the three points A , B , P shown in this illustration:

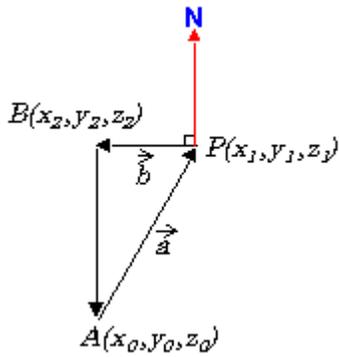


Figure 1-4-7 Example of a normal vector

When working with the two vectors \vec{a} and \vec{b} , you must be very careful to factor in each vector's direction.

$$N = \vec{a} \times \vec{b} = \begin{pmatrix} ax, ay, az \end{pmatrix} \times \begin{pmatrix} bx, by, bz \end{pmatrix}$$

To get the cross-product of the vectors, apply these formulas:

$$\begin{aligned} Nx &= ay \times bz - az \times by \\ Ny &= az \times bx - ax \times bz \\ Nz &= ax \times by - ay \times bx \end{aligned}$$

Because \vec{a} and \vec{b} are vectors, specify each coordinate component like this:

$$\begin{aligned} \vec{a} &= (ax, ay, az) = (x_1 - x_0, y_1 - y_0, z_1 - z_0) \\ \vec{b} &= (bx, by, bz) = (x_2 - x_1, y_2 - y_1, z_2 - z_1) \end{aligned}$$

Therefore, the final detailed formula for the normal vector N (Nx, Ny, Nz) is:

$$\begin{aligned} Nx &= (y_1 - y_0)(z_2 - z_1) - (z_1 - z_0)(y_2 - y_1) \\ Ny &= (z_1 - z_0)(x_2 - x_1) - (x_1 - x_0)(z_2 - z_1) \\ Nz &= (x_1 - x_0)(y_2 - y_1) - (y_1 - y_0)(x_2 - x_1) \end{aligned}$$

You need the cross-product for many geometric calculations, not just to calculate the normal vector.

1-5 Matrix Specification

A [matrix](#) is made up of rows and columns. Rows are numbered from top to bottom, and columns are numbered from left to right. A matrix that has m rows and n columns is called a matrix of m rows and n columns or an, $m \times n$, matrix. An individual cell in a matrix is called an **element** or **component** and is referenced as the element in row m and column n or as the (m, n) component.

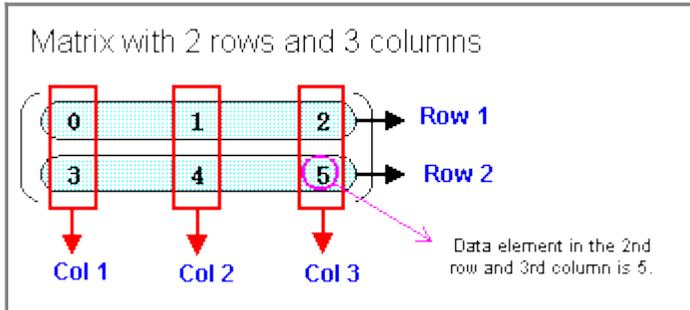


Figure 1-5-1 Matrix

You will usually use 4×4 matrices in 3D graphics and 3×3 matrices in 2D graphics. These matrices are specified like this:

$$M_a = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \text{ or } M_a = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

1-5-1 Matrix Addition and Subtraction

Adding or subtracting matrices is accomplished by using this formula:

$$M_a \pm M_b = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \pm \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & a_{13} \pm b_{13} & a_{14} \pm b_{14} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & a_{23} \pm b_{23} & a_{24} \pm b_{24} \\ a_{31} \pm b_{31} & a_{32} \pm b_{32} & a_{33} \pm b_{33} & a_{34} \pm b_{34} \\ a_{41} \pm b_{41} & a_{42} \pm b_{42} & a_{43} \pm b_{43} & a_{44} \pm b_{44} \end{bmatrix}$$

Here's an easy example showing how to add matrices together:

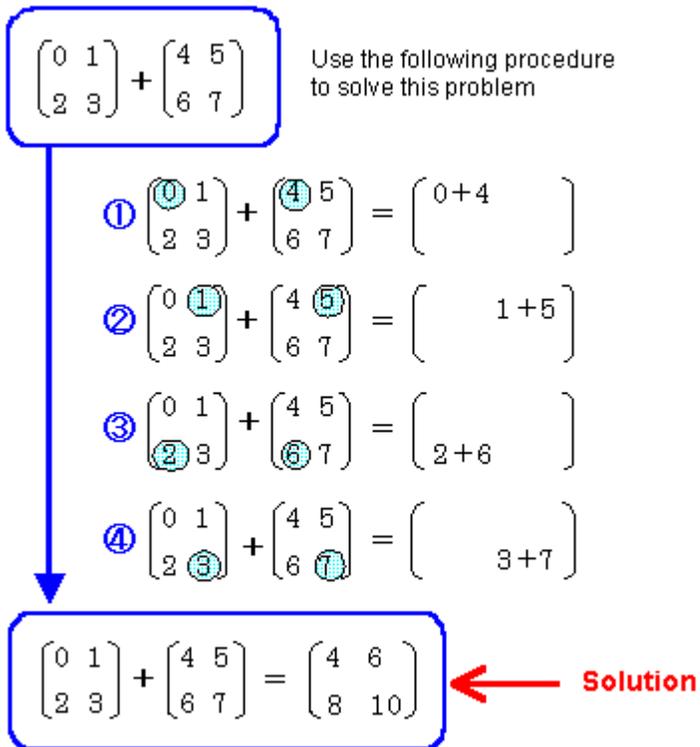


Figure 1-5-2 Matrix Addition 1

Here's another example. This one switches the order of the same matrices and shows that the result is still the same:

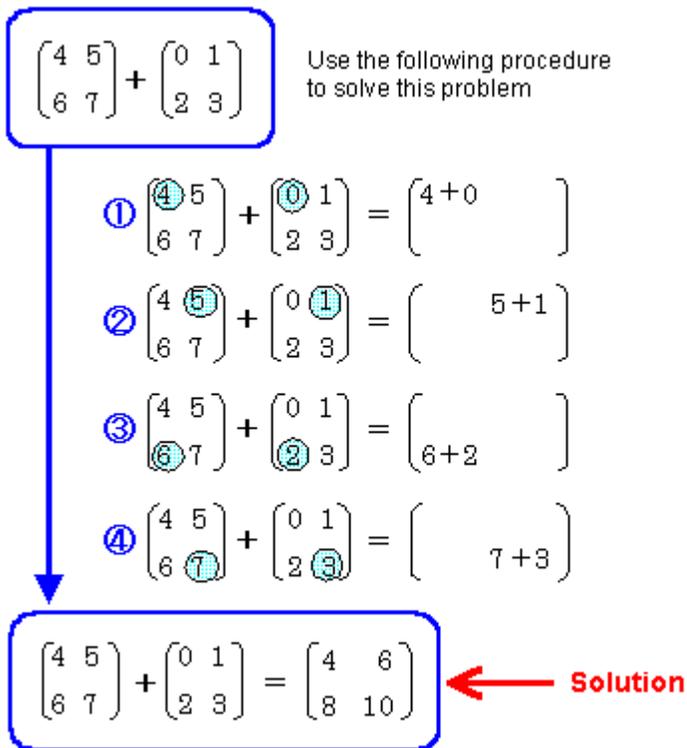


Figure 1-5-3 Matrix Addition 2

As you can see from these examples, the commutative law is valid for matrix addition and subtraction. In other words, you can switch the calculation order:

$$M_a \pm M_b = M_b \pm M_a \text{ (The commutative law is valid)}$$

$$M_a \pm M_b \pm M_c = (M_a \pm M_b) \pm M_c = M_a \pm (M_b \pm M_c) \text{ (The associative law is valid)}$$

1-5-2 Matrix Multiplication

When you multiply one matrix by another matrix, the result is also a matrix. This section explains the calculation method for Matrix Multiplication.

If you multiply matrix M_a by matrix M_b , the result is M_a, M_b . Use the following formula to obtain each element of row i and column j in the resulting matrix M_c where $M_c = M_a M_b$:

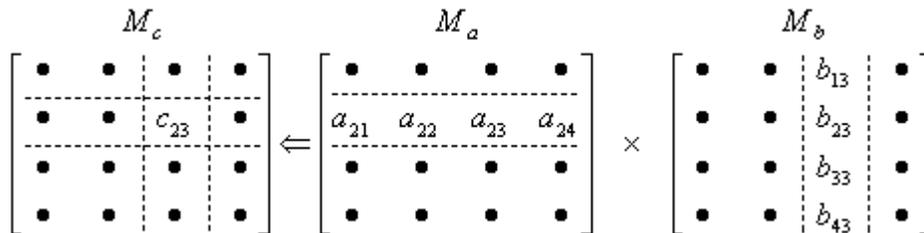
$$c_{ij} = \sum_{k=1}^4 a_{ik} b_{kj}$$

For example, to calculate the element in the second row and third column of M_c , use this formula:

$$c_{23} = \sum_{r=1}^4 a_{2r} b_{r3}$$

$$= a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} + a_{24}b_{43}$$

As you can see, the element of the second row and the third column is the product-sum of the second row of the matrix M_a and third column of the matrix M_b as illustrated here:



Note that for matrix multiplication to be possible, the number of columns in matrix A (m rows n columns) must equal the number of rows in matrix B (p rows q columns). In other words, n and p in the following illustration must be equal:

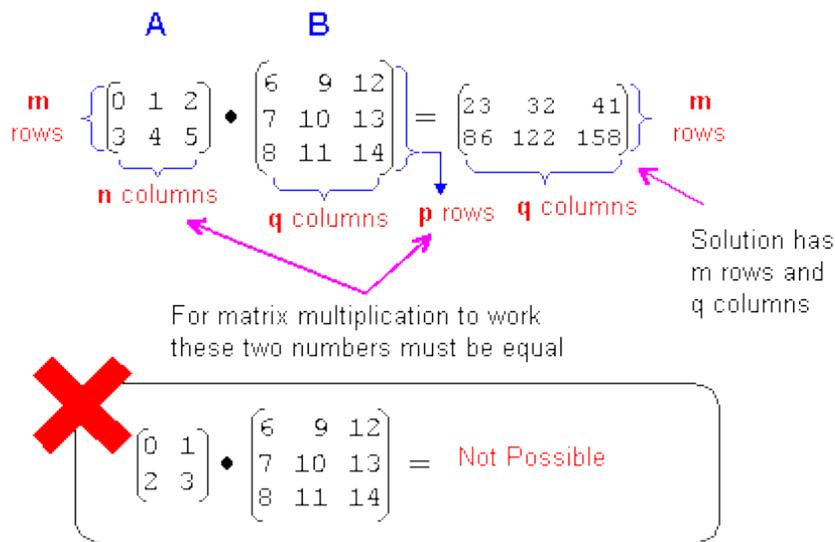


Figure 1-5-4 The criteria for matrix multiplication

Next, an additional example of multiplication will be provided using a simpler matrix.

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix}$$
 Use the following procedure to solve this problem

- ① $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} 0 \times 4 + 1 \times 6 & \\ & \end{pmatrix}$
- ② $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} & 0 \times 5 + 1 \times 7 \\ & \end{pmatrix}$
- ③ $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} & \\ 2 \times 4 + 3 \times 6 & \end{pmatrix}$
- ④ $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} & \\ & 2 \times 5 + 3 \times 7 \end{pmatrix}$

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ 26 & 31 \end{pmatrix}$$
← Solution

The following is another example which shows how the results differ when the order of the matrix multiplication is changed:

$$\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$$
 Use the following procedure to solve this problem

- ① $\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 4 \times 0 + 5 \times 2 & \\ & \end{pmatrix}$
- ② $\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} & 4 \times 1 + 5 \times 3 \\ & \end{pmatrix}$
- ③ $\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} & \\ 6 \times 0 + 7 \times 2 & \end{pmatrix}$
- ④ $\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} & \\ & 6 \times 1 + 7 \times 3 \end{pmatrix}$

$$\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 10 & 19 \\ 14 & 27 \end{pmatrix}$$
← Solution

Figure 1-5-5 Matrix Multiplication

Matrix Multiplication. However, the associative law is valid for matrix multiplication.

$$M_a M_b \neq M_b M_a \text{ (the commutative law is invalid)}$$

$$M_a M_b M_c = (M_a M_b) M_c = M_a (M_b M_c) \text{ (the associative law is valid)}$$

N64 Functions to Use for Matrix Multiplication

guMtxCatF

guMtxCatL

1-6 Geometric Transformations and Matrix

In "1-3 Coordinate System", you learned that you need to convert from one coordinate system to another in order to ultimately display a picture on the TV screen. This section shows you how to actually do the coordinate system transformations by using [matrix](#) calculations. Usually,

•**3x3 matrices are used for two-dimensional coordinate transformations**

•**4x4 matrices are used for three-dimensional coordinate transformations**

Here, we will explain about the basic two and three-dimensional coordinate transformations in that order. In addition, to display the coordinates of the vertexes, we will use the vectors below in descriptions to follow:

Two-dimensional--- $\begin{bmatrix} x & y & 1 \end{bmatrix}$

Three-dimensional--- $\begin{bmatrix} x & y & z & 1 \end{bmatrix}$

1-6-1 2D Transformation

In general, a matrix like the following is used for two-dimensional coordinate transformations:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

However, here we will classify the conversion matrices into some types and provide a basic explanation.

#Scaling matrix to specify enlargements or reductions

To apply scaling information, you multiply a scaling matrix by a vector $\begin{bmatrix} x & y & 1 \end{bmatrix}$. The scaling matrix has scaling information for the X-axis and Y-axis specified by the vector. The scaling matrix looks like this:

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To do the conversion, multiply the matrix by the vector:

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x x & s_y y & 1 \end{bmatrix}$$

The post-conversion coordinates (x', y') are as follows:

$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases}$$

In this way, the x and y coordinates are enlarged or reduced by the corresponding scaling factor (s). Note that if the scaling factor of a certain axis is negative, that axis will be reversed.

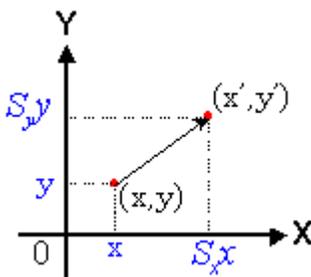


Figure 1-6-1 The two-dimensional magnified coordinate

#Translation matrix

The two-dimensional translation matrix looks like this:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

The reason that you need to specify a two-dimensional coordinate in the vector $\begin{bmatrix} x & y & 1 \end{bmatrix}$ is so you can apply the translation factors specified in the matrix to the coordinates. When you do the multiplication, you get the following result:

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} x + t_x & y + t_y & 1 \end{bmatrix}$$

The 1 added as a third element of the vector specifies that the translation factors t_x, t_y are to be added to each component.

This type of vector coordinate system that adds a new constant component 1 is called a homogeneous coordinate system.

#Rotation matrix to specify rotation around a point of origin

If you set the rotational angle to θ , the rotational matrix looks like this:

$$M = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As a standard, positive angles rotate counter-clockwise and negative angles rotate clockwise with the origin at the center as shown here:

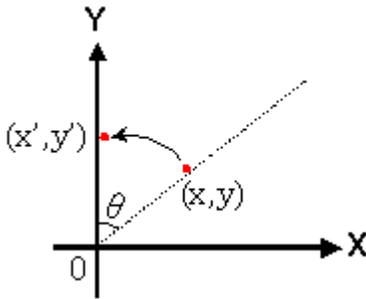


Figure 1-6-2 The two-dimensional rotational coordinate

As you can see, the three conversion matrices hold the information needed to specify **scaling, translating and rotating**.

1-6-2 3D Transformation

In general, a matrix like the following is used for three-dimensional coordinate transformations:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

However, as with the two-dimensional coordinate transformation, we will categorize it into three types of matrices for explanatory purposes. The coordinate vector that specifies the vertex to be transformed is given as:

$$A: [x \ y \ z \ 1] \text{ (before the transformation)}$$

$$A': [x' \ y' \ z' \ 1] \text{ (after the transformation)}$$

#Scaling matrix to specify enlargements or reductions

The matrix that holds the scaling information looks like this:

$$M = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The expanded matrix calculation looks like this:

$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$

As you can see, each coordinate is given scaling information s_x, s_y, s_z . Note that if a scaling factor is negative, it reverses the direction of the axis as it did with the 2D scaling transformation.

Examples of N64 functions (enlargements or reductions)

guScale
guScaleF

Obviously when you multiply a matrix where $s_x = s_y = s_z = 1$ by any vector, the vector does not change at all. This type of matrix is called an identity matrix.

An example of N64 function(identity matrix)

guMtxIdent

#Translation matrix to specify movement

The three-dimensional translation matrix looks like this:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

The translation matrix is used to translate; it translates the coordinates of an object to move it to a new location. When you multiply this matrix by a vector and expand it, you get the following result:

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

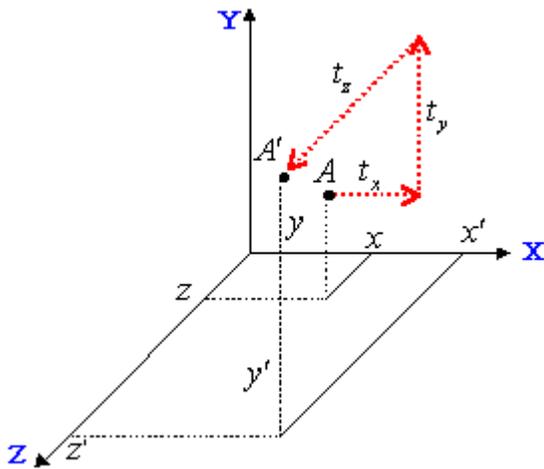


Figure 1-6-3 Translation matrix

N64 functions to translate an object's vertex to a new location

guTranslate
guTranslateF

#Rotation matrix to specify rotation around an axis of origin

Three-dimensional rotational conversion is more difficult than two-dimensional because there are an infinite number of possibilities for the axis of origin.

However, in most cases, you will specify the rotation of each coordinate axis by using the synthesis method instead of using arbitrary axis rotation (See section 1-6-4, **Coordinate System Transformation**).

N64 adapts the types of rotation direction described below.

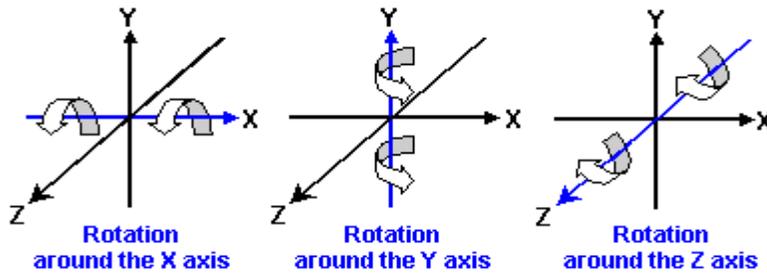


Figure 1-6-4 Rotation directions

***X-axis rotation**

x-axis rotation is specified by this matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_x & \sin \theta_x & 0 \\ 0 & -\sin \theta_x & \cos \theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When you expand the multiplication of a vector with this matrix, each coordinate component becomes:

$$x' = x, \quad y' = y \cos \theta_x - z \sin \theta_x, \quad z' = y \sin \theta_x + z \cos \theta_x$$

As you can see, the x component has not changed.

*Y-axis rotation

y-axis rotation is specified by this matrix:

$$M = \begin{bmatrix} \cos \theta_y & 0 & -\sin \theta_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta_y & 0 & \cos \theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When you expand the multiplication of a vector with this matrix, each coordinate component becomes:

$$x' = x \cos \theta_y + z \sin \theta_y, \quad y' = y, \quad z' = (-x \sin \theta_y) + z \cos \theta_y$$

As you can see, the y component has not changed.

*Z-axis rotation

z-axis rotation is specified by this matrix:

$$M = \begin{bmatrix} \cos \theta_z & \sin \theta_z & 0 & 0 \\ -\sin \theta_z & \cos \theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When you expand the multiplication of a vector with this matrix, each coordinate component becomes:

$$x' = x \cos \theta_z - y \sin \theta_z, \quad y' = x \sin \theta_z + y \cos \theta_z, \quad z' = z$$

As you can see, the z component has not changed.

Examples of N64 functions (rotation)

[guRotate](#)
[guRotateF](#)
[guRotateRPY](#)
[guRotateRPYF](#)

(Remembering the rotational direction of each-axis rotation)

As explained previously, N64 uses the right-handed coordinate system (see section 1-3, **Coordinate System**). The general method of deciding rotational direction in the right-hand coordinate system can be expressed as follows:

1. Make the right screw advance to the position direction of the axis.
2. At that time, determine the direction that the right screw rotates as the positional direction.

Following this rule, the positive direction of each coordinate axis rotation looks like this:

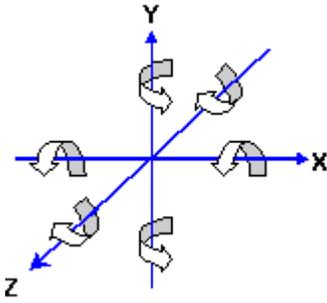


Figure 1-6-5 Rotational Direction of Each Axis Rotation

1-6-3 Projection Transformation

Projection conversion calculations determine the level of perspective that will be provided when you display the object seen from the camera on the screen. There are two methods of projection: One is Orthographic Projection. Another is Perspective Projection.

#Orthographic Projection

An orthographic projection shows the view volume (the portion of a 3D space that a camera can capture in a picture) as a rectangular parallel pipe without any perspective at all. The world appears flat but accurate measurements are possible given that you know how close or how far away the objects are in the 3D virtual world. Here is an example:

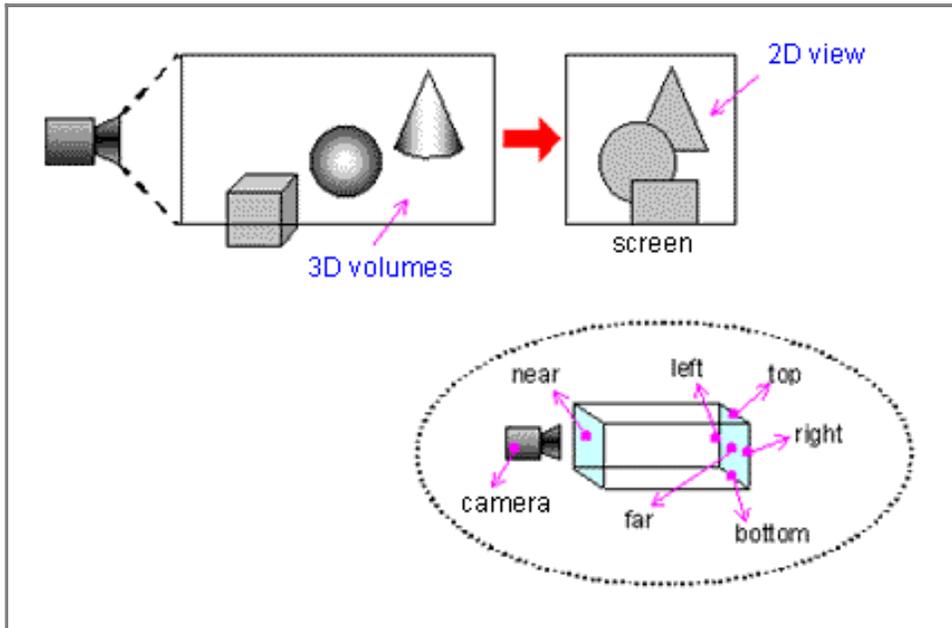


Figure 1-6-6 Orthographic Projection

"Near" is a value to display from where on the view front (the position where the camera is set) to start to draw, and "Far" is a setting value for until where on the view front to draw.

Given the following settings:
 t=top (top of the screen)
 b=bottom (bottom of the screen)

r=right (right side of the screen)
 l=left (left side of the screen)
 n=near clip (position of the near clipping plane)
 f=far clip (position of the far clipping plane)

the conversion matrix of an orthographic projection looks like this:

$$[T] = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{2}{f-n} & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & -\frac{f+n}{f-n} & 1 \end{bmatrix}$$

Although this matrix seems to be rather complicated, N64 functions calculate it for you. All you need to do is pass the original settings listed above, so an in depth understanding of the matrix is not necessary.

Examples of N64 functions (The calculation of the orthographic projection matrix)

guOrtho
guOrthoF

#Perspective Projection

A perspective projection shows the view volume (the portion of a 3D space that a camera can capture in a picture) as a quadrilateral pyramid to give the view volume perspective. Here is an example:

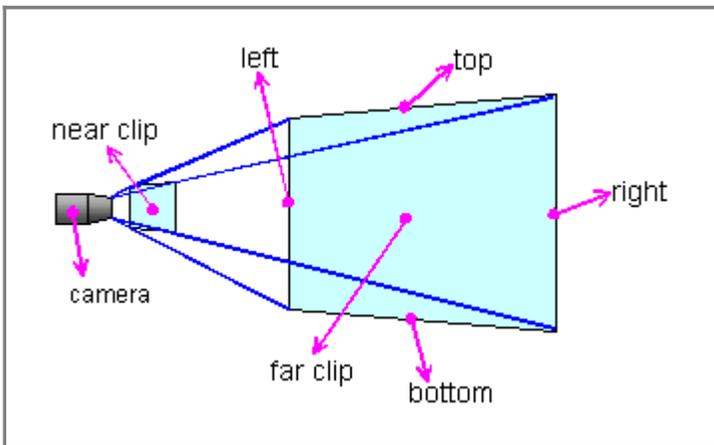


Figure 1-6-7 Perspective Projection (frustum)

Given the following settings:
 t=top (top of the screen)
 b=bottom (bottom of the screen)
 r=right (right side of the screen)
 l=left (left side of the screen)
 n=near clip(position of the near clipping plane)
 f=far clip (position of the far clipping plane)

the conversion matrix of a perspective projection looks like this:

$$[T] = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix}$$

Examples of N64 functions (The calculation of the perspective projection matrix)

guFrustum
guFrustumF

In this matrix the ratio of length to width is 1:1, sometimes making it a little hard to use for actual display on a screen. Therefore, you may need to add a different length to width ratio when preparing your projection matrix as shown here:

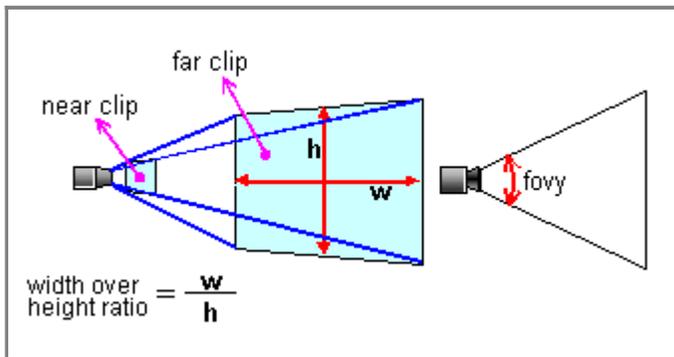


Figure 1-6-8 Perspective Projection 2 (perspective)

- Given the following settings:
 a=aspect ratio(ratio of length to breadth)
 Theta=fovy/2 (fovy)
 n=near clip(position of the nearest clipping plane)
 f=far clip (position of the furthest clipping plane)

The conversion matrix of a perspective projection that includes a length to breadth ratio looks like this:

$$[T] = \begin{bmatrix} \frac{\cos \theta}{\sin \theta} & 0 & 0 & 0 \\ a & \frac{\cos \theta}{\sin \theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix}$$

Now, you can display the object placed in the three-dimensional space.

Examples of N64 functions (Perspective Projection)

guPerspective
guPerspectiveF
gSPerspNormalize

For example, for parallel movement, the conversion expression in OpenGL is:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

It would look like this for N64:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

The conversion matrix is transverse for OpenGL and N64.

1-6-4 Coordinate System Transformation

Coordinate transformation was explained earlier by being categorized into transformations (scaling, translation, and rotation). This section explains how to transform the coordinates of one coordinate system into those of another, one after another, while moving through the coordinate transformation flow. The conversion matrix to convert from coordinate system 1 to coordinate system 2 will be depicted as M_1 , from coordinate system 2 to coordinate system 3 as M_2 , and from coordinate system 3 to that of 4 as M_3 . Coordinates in each system for a point P will be depicted as P_1, P_2, P_3, P_4 . Therefore, when you convert from one coordinate system to another in order, the following formulas apply:

$P_2 = P_1 M_1$ (equation 1: conversion equation from coordinate system 1 to coordinate system 2)

$P_3 = P_2 M_2$ (equation 2: conversion equation from coordinate system 2 to coordinate system 3)

$P_4 = P_3 M_3$ (equation 3: conversion equation from coordinate system 3 to coordinate system 4)

Next, if you substitute equation 1 for equation 2 and substitute the result for equation 3, you can go from the equation for coordinate transformation from coordinate system 1 to that of 4 in a single equation:

$P_4 = P_1 M_1 M_2 M_3$ (equation 4: conversion equation from coordinate system 1 to that of 4 in a single process)

A process that provides several conversions in a row like this is called a **conversion synthesis**. As you can see from the equations given above, a conversion synthesis involves the multiplication of conversion matrices.

Next, we will apply this to a game situation. Obviously, a single model has several vertices, and you need to provide an identical coordinate transformation for each vertex. However, to provide the same matrix multiplication repeatedly is inefficient. Therefore, do the matrix multiplication first. Then, use the matrix to convert from the first coordinate system to the last coordinate system in one process for better calculation efficiency.

For example, if:

$$M_o = M_1 M_2 M_3$$

then M_o becomes the conversion matrix from coordinate system 1 to that of 4. Thus, you can specify equation 4 using the following simple form:

$$P_4 = P_1 M_0 \quad (\text{conversion equation from coordinate system 1 to coordinate system 4})$$

Therefore, when you actually draw an image, you can provide a single coordinate transformation calculation to convert directly from the model coordinate system to the normal screen coordinate system at once. Accordingly, it is not necessary to separately perform the world and view coordinate system conversions.

Remember, however, that matrix multiplication is not commutative; that is, you can't switch the order and get the same result. Therefore, when you provide the conversion by using matrices, you need to be careful of the order in which you multiply them. If you multiply in the wrong order, a completely different result will be produced, as shown in the following illustration. Similarly, when you provide several rotational conversions, the result is very different depending on the order of rotating axes. In other words, XYZ will give a result very different from YXZ. This is very important, and you must be careful.

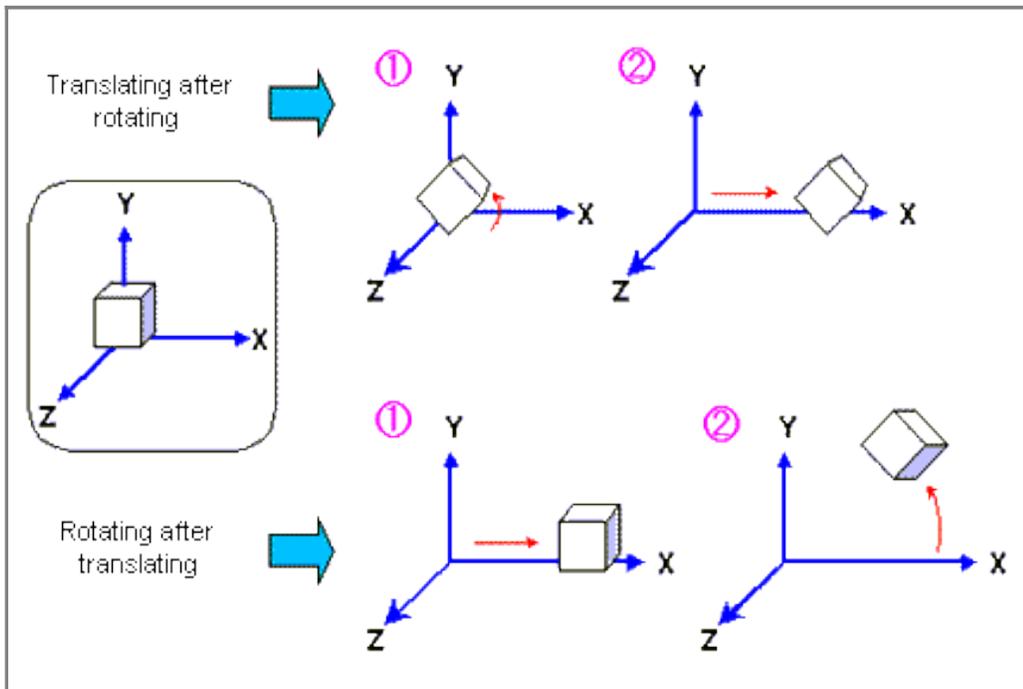


Figure 1-6-9 Differences based on the order of matrix multiplication

Note that the 3D modelers sold by various third-party companies are not standardized in terms of the directions of axes, directions, names of axes, rotational orders, or rotational directions that are used. Therefore, with the rotation of a [polygon](#) model, you need to verify the rotational procedures used by modelers before utilizing their output data.

1-7 Lighting and Colors

After drawing a model defined by the mathematical conversions explained thus far, you need to insert a light source (and shading), material properties (such as shininess), and color to make the model appear real. All objects become black in a state of no lighting. This Chapter describes how to implement color, lighting, properties, and shading in your game program.

1-7-1 Color

Colors expressed on a computer are basically composed using the **three primary colors of light** (red, green, and blue), that can be displayed on a luminous object like a TV screen. Other colors are created by mixing red, green, and blue together.

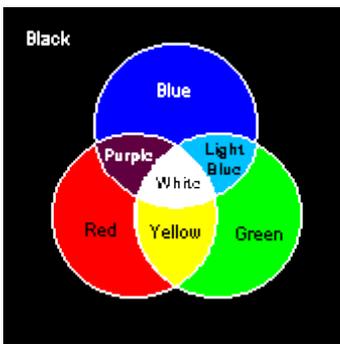


Figure 1-7-1 Three Primary Colors of Light

Although this illustration shows only eight colors, you can create many more by using different mixes and by changing their brightness. For example, if you slightly reduce the brightness of the mixed color of red, green, and blue, the color becomes light gray. Then, if you increase the brightness of the red, the color becomes pink. In this way, the computer expresses colors by combining three kinds of elements.

1-7-2 Lighting

For our eyes to be able to see an object, the object must be either emitting light or reflecting light. On a computer, it is difficult to represent in real time the situation where an object itself is emitting light, therefore this discussion will focus on showing you how to represent reflected light. To represent reflected light, you need to know the position and direction of light coming from the light source, as well as two other types of light: [ambient light](#) and [diffuse light](#). The position and direction are the setting for these, from where and to which direction the source of light emits light. Ambient light and diffused light are difficult to describe, and require a separate explanation.

Ambient Light

Ambient light is the surrounding atmospheric light. For example, the part of an object that is on the opposite side of a light source is usually not black. The reason for this is that it is lit up slightly by ambient (atmospheric) light. Ambient light occurs because surrounding objects produce irregular reflections. Ambient light has no direction, so if you give a color to ambient light, that light is reflected on everything in the scene.

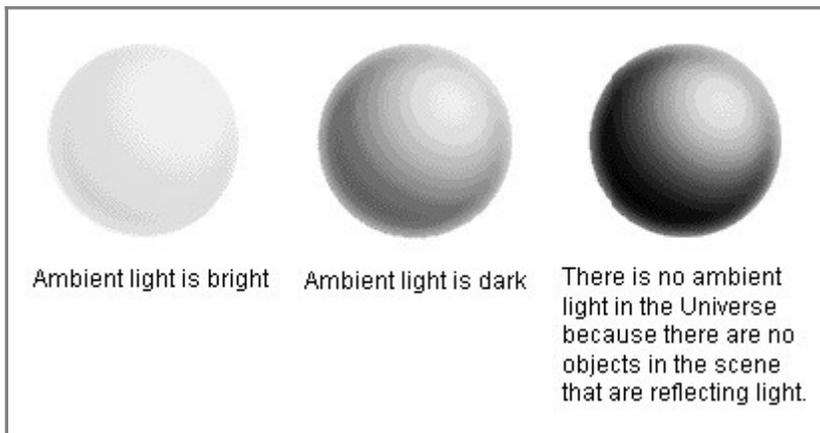


Figure 1-7-2 Ambient Light

Diffuse Light

Diffuse light is light that has been scattered through reflection or refraction. For example, transmitting a light through a translucent material diffuses the light. Diffuse lights take on the color of the translucent material that scattered the light.

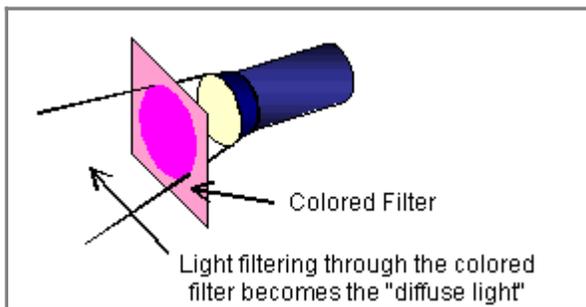


Figure 1-7-3 Diffuse Light

To implement lighting, you simply set the position and direction of the source light, and also specify appropriate ambient and diffuse lights.

1-7-3 Material Properties

There are three elements that determine the materials of an object. These are ambient light, diffused light, and reflected light.

Ambient Light

As mentioned earlier, ambient light is the color of the shadow of an object.

Diffuse Light

Unlike the diffuse light previously mentioned, the color of diffuse light comes from the color of the substance itself. The diffuse light becomes brighter as the incident angle gets closer to a right angle.

Specular Light

How a substance reflects light affects how shiny the object appears. A substance like chalk reflects light in all directions with equal intensity, so it appears dull. On the other hand, a substance like a mirror reflects light in only a

certain direction, so it appears shiny. How shiny or dull a surface appears depends on how that substance reflects light. Shiny surfaces show a shiny spot (specular highlight) that is very bright compared to the surrounded parts because most of the reflected light comes off in a single predominant direction. N64 cannot set these elements, but pseudo-expression is possible.

Example N64 Functions (Lighting)

gSPLight
gSPLightColor
gSPNumLights
gdSPDefLights

1-7-4 Shading

Shading is adding shadow to an object by calculating the materials and lighting. N64 implements [flat shading](#) and smooth shading (Gouraud shading).

Flat Shading

In flat shading each surface is shaded uniformly by using the same color as illustrated here:

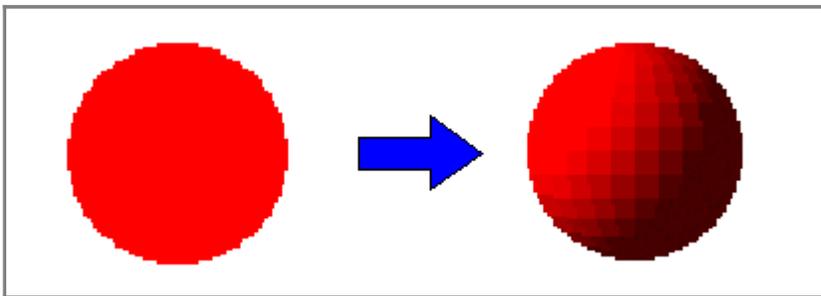


Figure 1-7-4 Flat Shading

This shading technique calculates the normal vector and the light source of a surface, and applies uniform color to the surface. As a result, objects will appear blocky (like computer graphics).

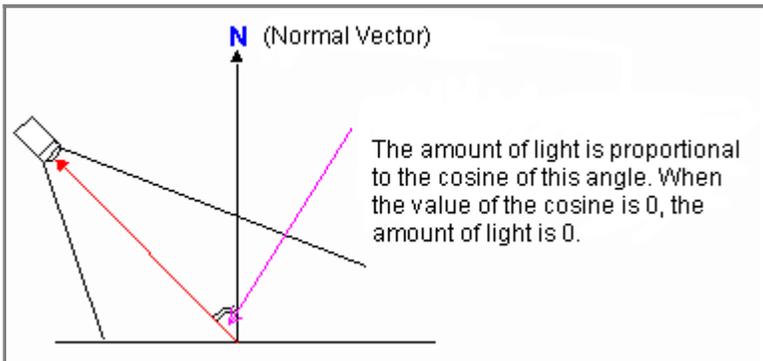


Figure 1-7-5 Normal Vector and Light Source

Example N64 Functions (Flat Shading)

Function Parameter
gSPSetGeometryMode G_SHADE

Smooth Shading

The smooth shading technique smoothes out the shading to make it appear more realistic. Note that smooth shading affects the inside shading only; it has no effect on the outline of the object, as you can see in this illustration:

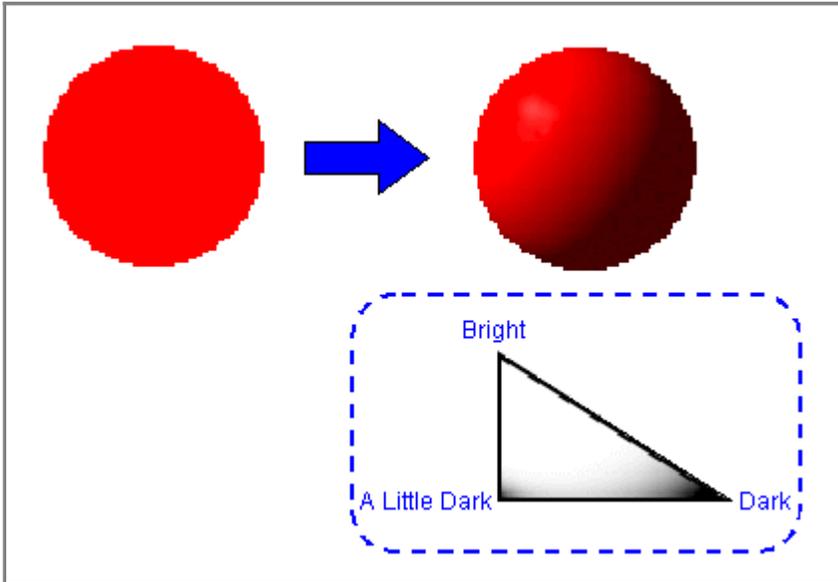


Figure 1-7-6 Smooth Shading

There are many different techniques that you could use to implement smooth shading. One of the most popular is [Gouraud shading](#). Gouraud shading, named for its developer, gives a normal vector to each vertex of the surface. In other words, it calculates both the light source and the normal vector for each vertex. In this way, it interpolates calculated light for each vertex to make the shading appear smooth as illustrated here:

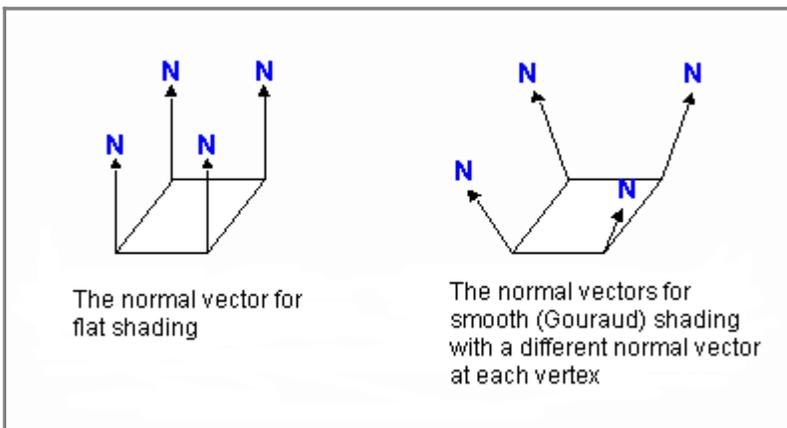


Figure 1-7-7 The difference between flat and smooth shading

By providing shading, you can put colors that are reflected from surrounding objects on your shaded object.

Example N64 Functions (Smooth Shading)

Function Parameter

```
gSPSetGeometryMode G_SHADING_SMOOTH  
gSPClearGeometryMode G_SHADING_SMOOTH
```

1-8 Texture

There is a limit to the shading method. For example, when you try to draw the Earth, you find that there are an infinite number of [surfaces](#); including sea, continents, mountains, rivers, and so on. It is impossible to draw just by shading these surfaces because there are so many of them. There are not enough resources to be able to handle all of these surfaces in a real-time game. Therefore, you need to use texture. Texture is simply a two-dimensional image that you paste on a model. In the case of the Earth, you paste a picture of the Earth on a sphere model to create a model as illustrated here:

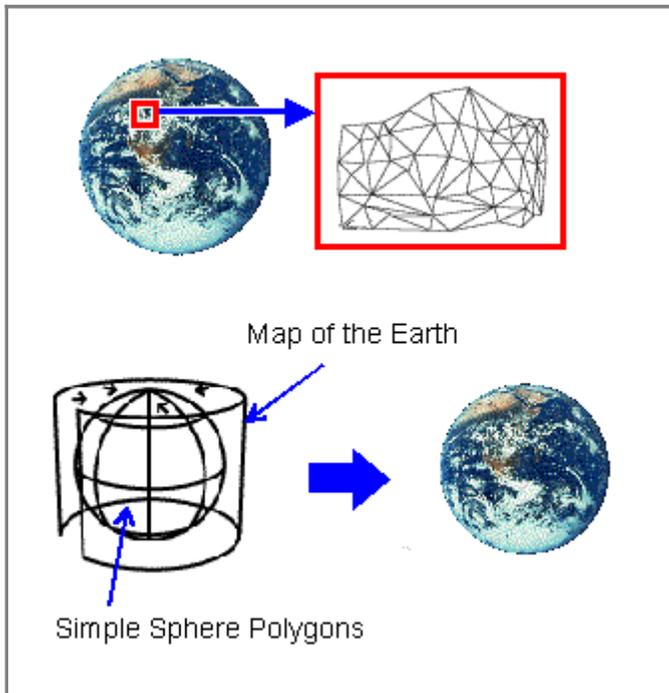


Figure 1-8-1 Shading Only and Texture

The actual technique used to paste a two-dimensional image on a three-dimensional object is called [texture mapping](#). The texture mapping process has three steps:

1. Prepare the image data for texture mapping.
2. Select a texture mapping application method for each [texel](#) (one pixel of the texture).
3. Set the ST value for each vertex and execute texture mapping.

Example N64 Functions (Texture)

```
gSPTexture  
gDPLoadTexture(3P)  
gDPLoadTextureBlock(3P)  
gDPLoadTextureBlock_4b(3P)  
gDPLoadTextureTile(3P)
```

1-8-1 Prepare 2D Image

First of all, you need to prepare the two-dimensional picture for texture mapping. The picture has various formats, but we will limit our discussion to the following main formats:

Bitmap (RGBA) Format

The bitmap format is composed of four elements of RGBA per each texel.

A is for alpha value, which indicates the level of opacity (and therefore transparency). If all the texels are at full opacity (that is, none of them are translucent), the bitmap format may use just the RGB elements.

Index Color Format

The [index color](#) format uses the [TLUT](#) (Texture Look-Up Table) and index texel data. This method loads several colors on a color pallet, and then specifies the pallet number of the texel using the index.

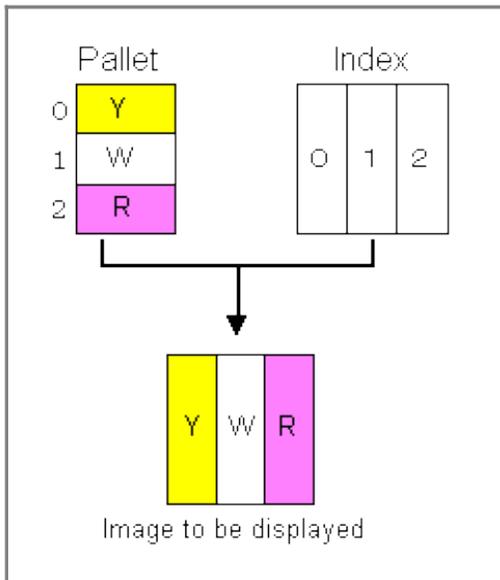


Figure 1-8-2 Pallet and Index

This format generally requires less data than does the bitmap format.

This becomes apparent if you compare the amount of data required by a bitmap picture that expresses 1 texel by using 4 bytes to hold the RGBA value (R: 8-bits, G: 8-bits, B: 8-bits, A:8-bits), with the amount of data required by an index color picture that expresses 1 texel by using 1 byte to hold the index (256 indexes), plus another 4 bytes to hold the color pallet (R: 8bit, G: 8bit, B: 8-bit, A: 8-bit). Both pictures are the same size (32x32 texels).

The **bitmap picture** needs 4 bytes for every texel, so it requires $32 \times 32 \times 4$ bytes for a total of **4096 bytes**.

The **index color** picture's index needs 1 byte for every texel, thus it needs $32 \times 32 \times 1$ bytes for a total of 1024 bytes for the index data.

In addition, the space needed to hold the index pallet, which is 256×4 bytes for a total of 1024 bytes, is required to hold the pallet data. (Add the two together and the index color picture requires only $1024 + 1024$ bytes for a total of **2048 bytes**.)

As you can see, the index color format can represent the same picture using fewer bytes.

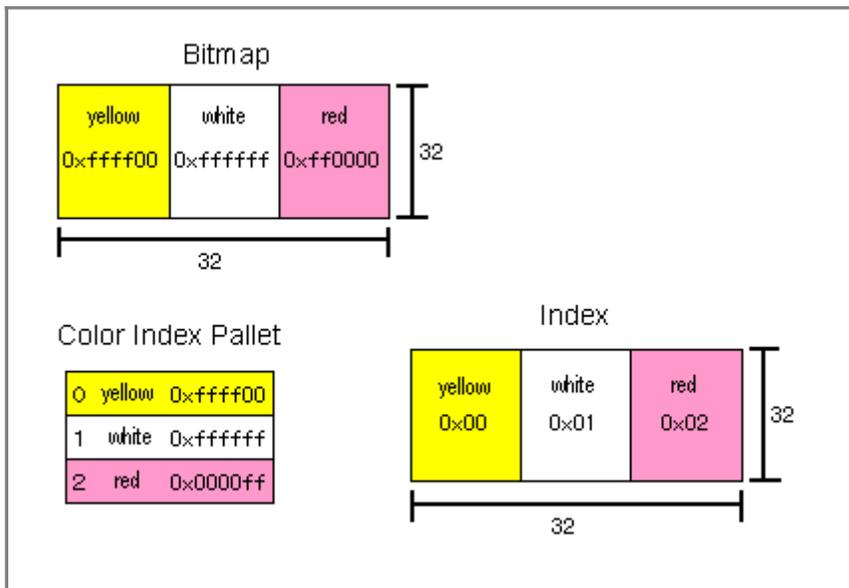


Figure 1-8-3 Comparison of the data size

However, you need to be careful because the color pallet in the index color format is limited. Because the maximum pallet number is 256, the number of colors is limited to 256. Therefore, if you have a large complex picture that requires more than 256 colors, you need to use the bitmap format if quality is important.

Example N64 Functions (Loading the TLUT)

gDPLoadTLUT_pal16
gDPLoadTLUT_pal256

Intensity Format

N64 has a format called "intensity" to express brightness. It is sometimes used as IA ([Intensity Alpha](#)) by combining intensity and alpha. Intensity holds information about brightness only; it has no color information. It can specify a color by using the color combiner, but this is like setting the color to the texture of the gray scale. As a result, it appears to be the gray scale of red or blue. In practical application, it is used to show smoke or a flame by combining [alpha values](#). The biggest distinctive feature of the Intensity format is its compact data size. It needs the minimum three elements of RGB to have colors, but the intensity contains only information for brightness. Thus, even if you compared RGBA with IA, the number of elements is only half.

1-8-2 Choose Texturing Method

You have the option of simply drawing each texel's color information on the model, or adding special effects to the picture information. The direct drawing method is called the "decal mode" or the "sticker mode." The addition of lighting and shading effects to texel colors is called the "modulation mode." Mixing texel colors with other arbitrary colors is called the "blend mixing mode." The decal mode is not complicated, but it does not show the impact of lighting effects. Therefore, if you use it in excess you may ruin the stereoscopic effect of your scene.

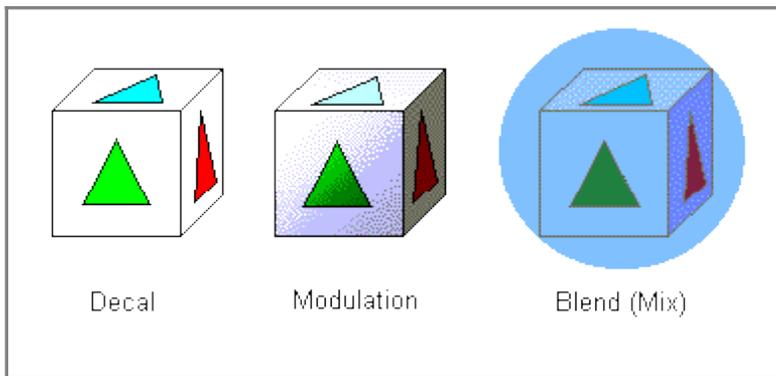


Figure 1-8-4 Choose Texturing Method

Example N64 Functions

gDPSetCombineMode (Setting for color information)
gDPSetRenderMode (Setting for **rendering information**)

1-8-3 Texture Coordinates

As a final step, you need to set the texture coordinates (S,T values) of the mapping data to each vertex and execute the texture mapping.

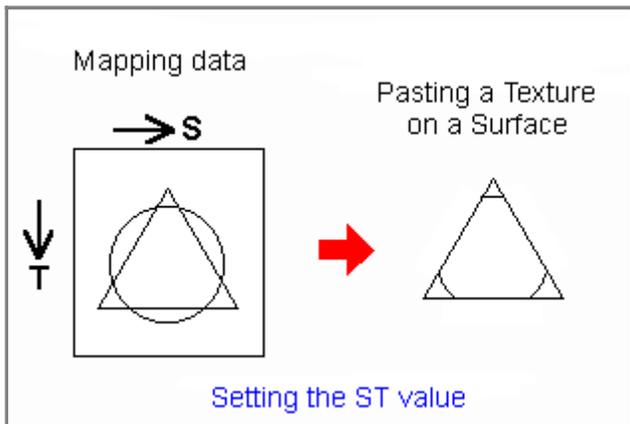


Figure 1-8-5 Pasting a Texture

Then you can map the texture on the surface.

Example N64 Functions

gSPVertex (set the vertex information)
gSPModifyVertex (modify the vertex information)

1-8-4 Mixing Process (Translucence)

In N64, it is possible to display translucence. We call the effect of seeing through to the rear of a scene, as if through jelly or cellophane, the mixing (blending or translucence) process.

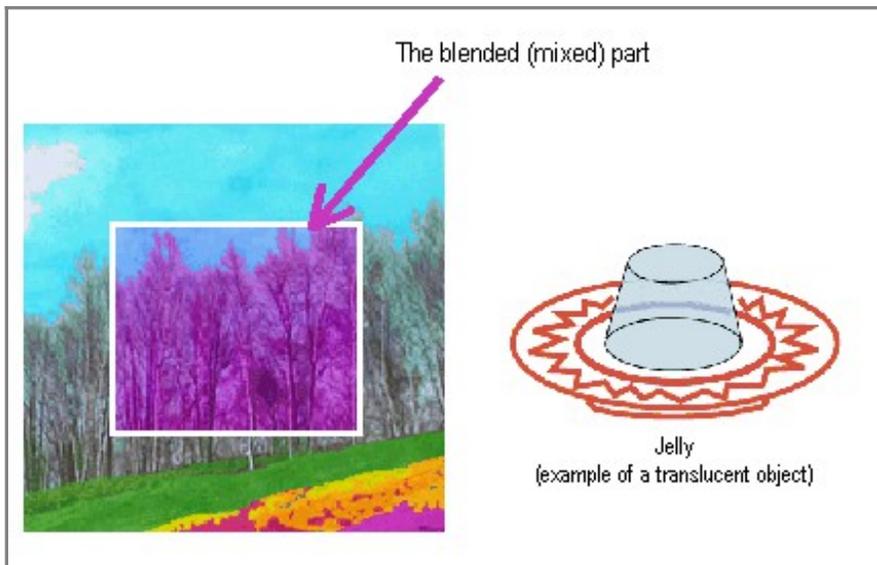


Figure 1-8-6 Example of the mixing process

There are many methods you can use to implement the mixing process. The following are two such examples:

- 1. Use the alpha value of the texture**
- 2. When drawing the surface, draw by multiplying the mixed coefficient**

Blending by Using the Alpha Value

By using this method when you create the mapping data, you can prepare the RGBA data to specify the alpha value for each texel. The alpha value shows the level of translucence (more transparent or more opaque). For example, if the alpha value is 0.3, that means blend 30% for the already drawn pixel data when you draw the mapping data. The advantage of this method is that you can implement right down to the [sprite](#) (used in previous games) because it sets the alpha value for each pixel.

Blending by Using the Mixed Coefficient

This method multiplies the mixed coefficient when you draw a surface. It blends without applying the alpha value in the mapping data, so it sets the level of transparency for the entire object. It therefore cannot be used to specify the transparency in units of texel.

When you provide the blend, you need to be careful of the drawing order of the translucent surfaces. In short, you need to follow the drawing order below.

- 1. Draw the opaque surfaces**
- 2. Draw the translucent surfaces from the rear forward**

Basically, the translucent surfaces are implemented by applying the translucent process to the picture that has already been drawn. Therefore, if you draw a translucent surface before drawing an opaque surface, the effect looks unnatural because there are no objects already drawn to be blended. Also, if you start to draw the translucent surfaces from the front, the effect looks unnatural because the rear translucent surfaces are drawn on top of the translucent surfaces in front.

- However, in reality this "unnatural look" is not very conspicuous, thus it is not absolutely necessary to draw starting from the rear.

STEP 4 About [What is 64DD?]

Up to this point, the N64 Introductory Manual has focused on explaining 3D basics and the functions that need to be understood for the creation of N64 applications. Part 4 of the manual covers the N64 expansion device known as the N64 Disk Drive (the 64DD). It is meant to provide a simple explanation of the features of the 64DD and the items which should be noted when developing 64DD applications. The 64DD enables the creation of games far larger than anything possible with ROM cartridges to date. It also makes it possible to write large volumes of data to the 64DD disk by securing a region of the disk for the storage of user data.

This part of the manual is designed to help you understand the 64DD so you can smoothly move to the development of 64DD-compatible software. Please use this manual for further application development for N64 with effective use of the 64DD.



Introduction to **NINTENDO⁶⁴**

Chapter 1 Summary of the 64DD Drive

This chapter explains about the features of the 64DD Drive.



1-1 General Architecture

The figure below shows the general configuration of the N64 Control Deck and the 64DD System.

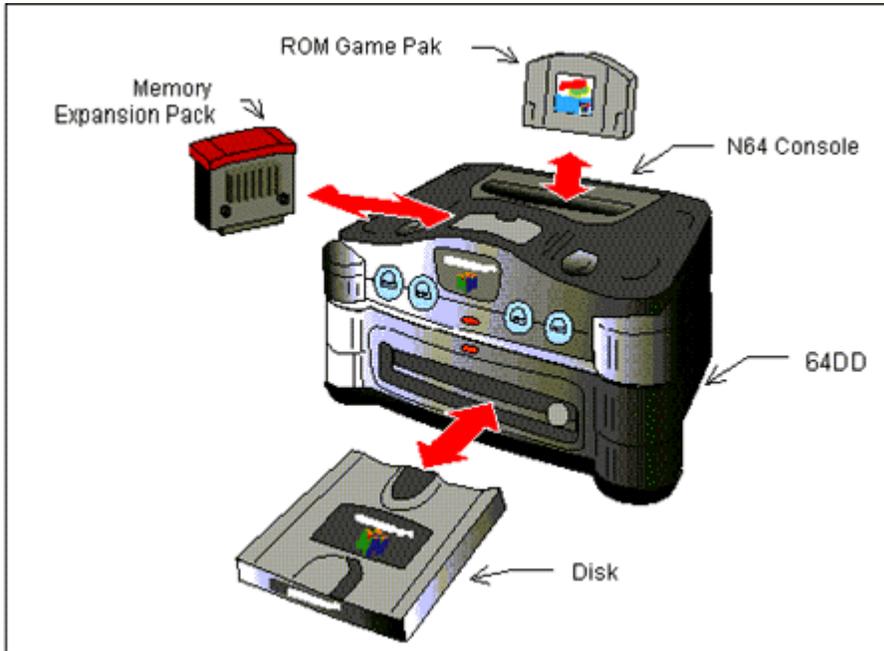


Figure 1-1-1 General Configuration of the N64 Control Deck and the 64DD System

- 64DD is a dedicated magnetic disk drive that plugs into the expansion connector on the bottom of the N64 Control Deck.
- Its total disk capacity is 64MBytes, enabling the creation of large scale games not possible with just a ROM cartridge.
- Part of the disk can be designated as a writable region from the game application, meaning the disk can also be used to back up player data.

In addition, by exchanging data stored on disks, games can be broadly expanded.

1-2 System Features

1-2-1 Hardware Features

- **Larger RAM Area**
The 64DD adds a 4M x 9bit (36Mbit) memory expansion pack to the N64 Control Deck's standard 4M x 9bit (36Mbit) of main memory. This expands N64 main memory to 8M x 9-bit (72M-bit).



Figure 1-2-1 Memory

- **High-Speed Data Transfers**

64DD boasts a maximum data transfer rate of 1MByte/sec, which is quick for disk media and about **5.4 times faster** than the CD-ROM drives used in other game machines.

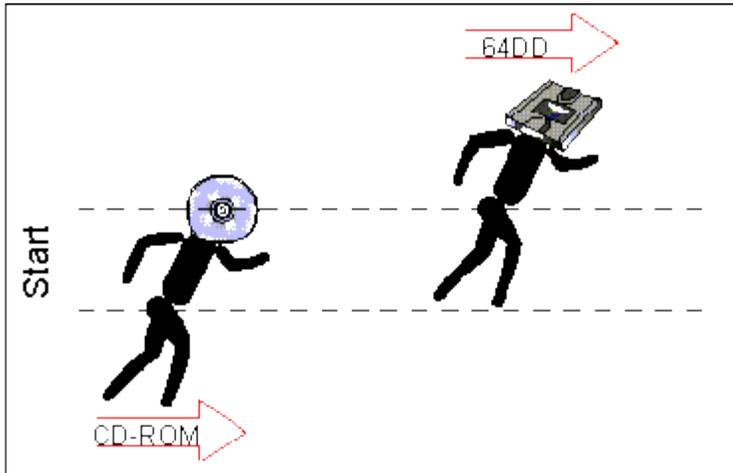


Figure 1-2-2 Comparison of transfer rate

- **Compact, Mass-Storage**

Although in outward appearance the disk is only a little larger than a floppy disk (100 x 100 x 10mm), it has a memory capacity of 64.45 MBytes, which is much larger than a floppy disk.

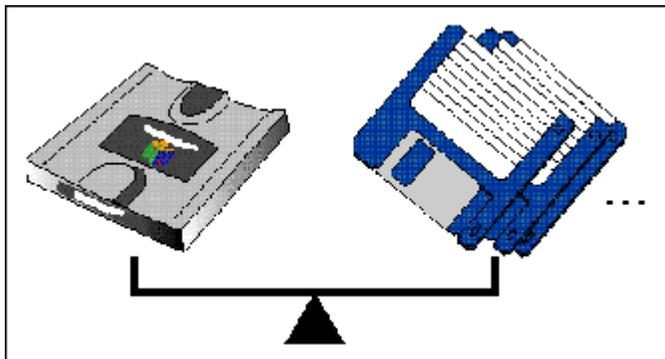


Figure 1-2-3 Disk Capacity

- **Writable**

The disk is not simply a medium for storing game programs, although it can be used in that way as a read-only medium. It is also possible to designate separate read-only (ROM) and writable (RAM) areas on the same disk. Up to 38.44 MBytes can be designated as RAM on each disk, and this writable area can be used to store vast amounts of user data.

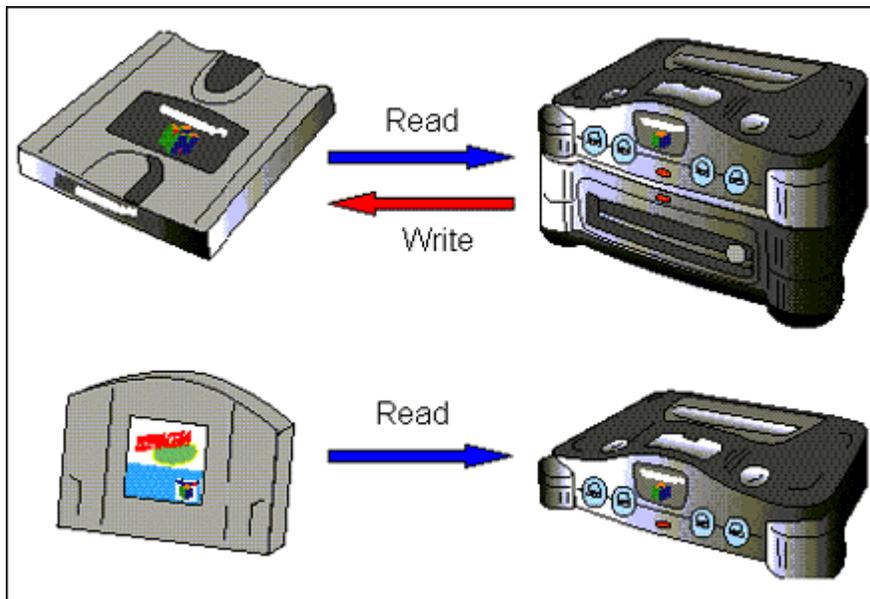


Figure 1-2-4 Writable Disk

- **Battery-Backup Real-Time Clock**

The 64DD incorporates a battery-backup real-time clock. Since this clock operates even when the power is turned off, it is possible to reflect real time in games.

Introduction to

Figure 1-2-5 Battery-Backup Real-Time Clock

1-2-2 Software Features

- **Efficiently Managed File System**

Since the work of directly programming disk-control code can be vastly reduced by efficiently managing the 64DD's expanded RAM area, the Multi File System (MFS) and an MFS library are provided as Nintendo's recommended file system. With MFS, data in the ROM area as well as the RAM area can be managed in file units.

- **Special Functions Available for Disk Access**

The special Leo Library for 64DD is available as a low-level function library that is closer to the hardware than the MFS that controls the 64DD. The Leo Library is used for disk access by the MFS Library, which is a high-level function library. Also, the Leo Library can be used directly by applications.

An overview of the location of these libraries is illustrated below.

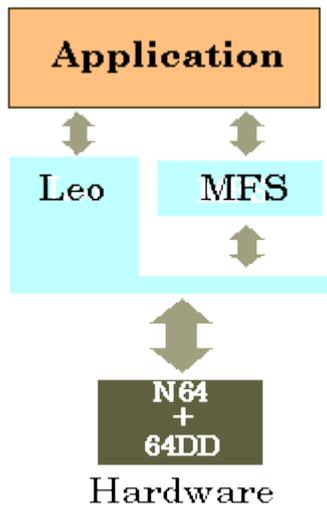


Figure 1-2-6 Position of MFS Library Functions

- See "Appendicies" for detailed information about each function.

1-3 Memory

4.1.3 Memory

When the 64DD is connected to the N64 Control Deck, the N64's built-in RDRAM is extended by the memory expansion pak that comes with the 64DD system. This doubles the conventional 4M x 9bit of main memory so that 8M x 9-bit of memory can be used as main memory. When creating 64DD applications, this expanded memory area can be allocated for frame buffers and Z buffers in the same way as the standard N64 memory.

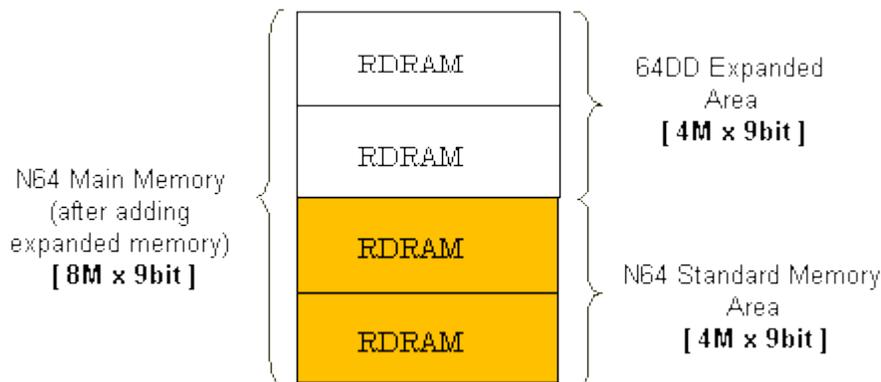


Figure 1-3-1 RDRAM Capacity

Introduction to NINTENDO⁶⁴

Chapter 2 Summary of the 64DD Disk

This chapter explains about the 64DD and the Disk.



2-1 Disk Specifications

The 64DD disk is a magnetic disk with a total capacity of 64.45 Mbytes. The game program and data are stored on the disk. Data can be written to both the front and back sides of the disk. The disk is structured such that the front side of the disk is called **Head 0** and the back side is called **Head 1**. One full circle around the disk is called a **track** and one half circle is called a **block**. On the 64DD, the smallest access unit is one block.

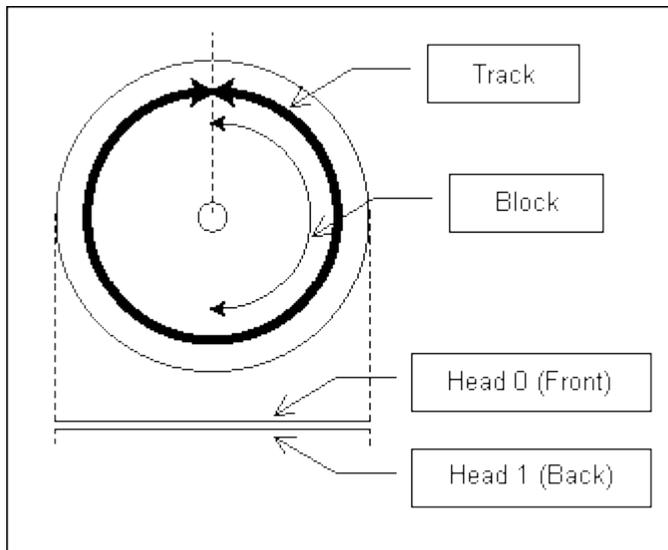


Figure 2-1-1 Structure of Disk

2-1-1 Size of Zones and Blocks

Each side of the disk is divided into eight concentric circle-shaped groups called **zones**. The size of a block varies, depending on the position of the zone it's located on. For high-density data storage, the 64DD disk is designed such that the outermost circle of the disk has the largest size of 19720 bytes, and the innermost circle has the smallest size of 9520 bytes. In addition, the disk is formatted so that each block in a given **zone** has the same size.

Since the block size differs depending on its location, it may seem difficult to handle with the program. However, the disk can be easily accessed since its file system can be manipulated using the MFS library. Because of this there is no need to know information such as block size, except when it is generated by the file system itself.

The following table gives the block size and the number of tracks in each zone:

Zone number	Head 0		Head 1	
	Block size	Number Of tracks	Block size	Number Of tracks
Zone 0	19720 bytes	134	-	
Zone 1	18360 bytes	146	18360 bytes	146
Zone 2	17680 bytes	137	17680 bytes	146
Zone 3	16320 bytes	137	16320 bytes	137
Zone 4	14960 bytes	137	14960 bytes	137
Zone 5	13600 bytes	137	13600 bytes	137
Zone 6	12240 bytes	137	12240 bytes	137
Zone 7	10880 bytes	102	10880 bytes	137
Zone 8	-		9520 bytes	102

When computing the capacity of a zone on one side of the disk, the following expressions are used:

Number of blocks in the tracks = (the number of tracks) x 2

Capacity of the zone = (the block size) x (the number of blocks in the tracks)

As explained in [2-1, Disk Specifications], the smallest unit of access on the 64DD disk is a block, and one track is composed of two blocks.

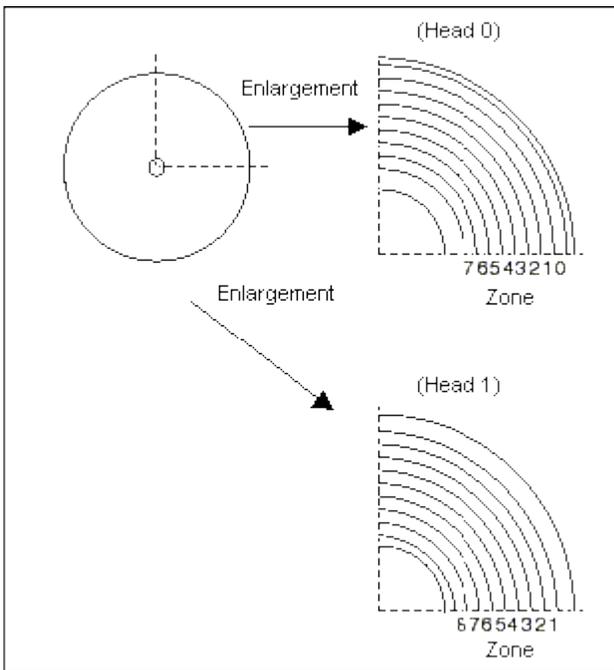


Figure 2-1-2 Head and Zone

2-1-2 Access Speed

- Rotation speed: About 25.2ms for one revolution
- Transfer rate for one block: About 12.6ms

Since the disk rotates at a constant speed, the per-byte data transfer rate is faster toward the outer perimeter of the disk. This is because the size of one block varies depending on the position of its zone, and data is formatted at a higher density toward the outer perimeter of the disk. In addition, since the storage density toward the inner part of the disk is set lower, the data transfer rate toward the center is slower. As explained in [2-1-1, Size of Zones and Blocks], since the greatest amount of data can be stored in the blocks on the outermost circles of the disk, and because data is also transferred at the highest speeds there, we recommend creating your programs so that the most important data is located in the outer parts of the disk.

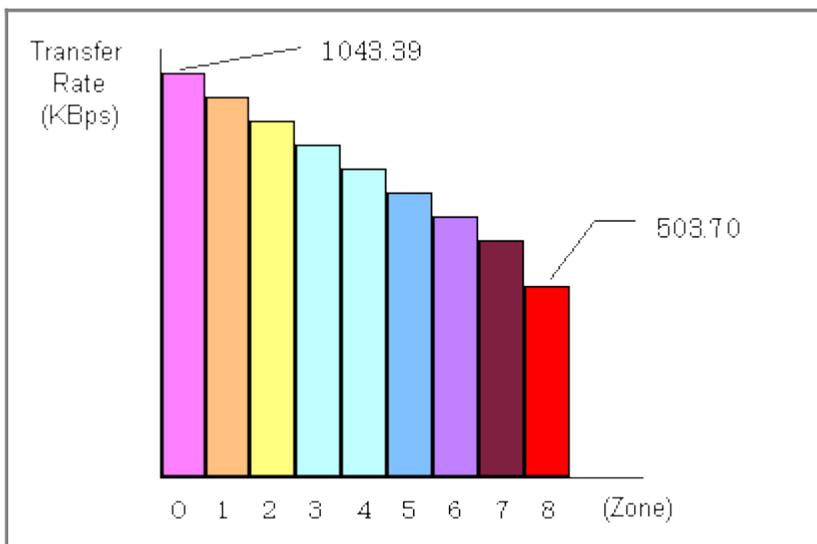


Figure 2-1-2 Zone and Transfer Rate

2-1-3 System Area

Every disk has tracks reserved for the system area. The following information is written in the system area:

Disk Type

The fact that the 64DD disk is a special writable disk and not simply a read-only medium does not mean that the entire disk can be written to. There are seven disk types, differing according to the relative sizes of the disks that are writable (RAM area), and those that are read-only (ROM area). For more details, see [2-1-4 Disk Types]

IPL Size

When power is supplied to the 64DD it displays the IPL, which is the program for booting a game from the 64DD disk. If the 64DD disk is inserted at this time, several blocks starting from the first block are read and stored in certain addresses in RDRAM. The IPL size specifies how many blocks should be read.

Load Address

The load address specifies where in RDRAM the program and data read from the disk should be stored.

Disk ID

The Disk ID identifies the number of the 64DD disk. 32 bytes of ID information are allocated to each disk. The programmer can use this information to check whether or not the MFS is used, and also to check whether an incorrect disk has been inserted during a game or to identify disks in 2-disk games.

The ID includes information submitted at production time, such as plant line number, time and date manufactured, etc. For more details, see [64DD Programming Manual]

2-1-4 Disk Types

The writable area on the 64DD disk is called the RAM area, and the read-only area where data cannot be written is called the ROM area.

Entire disk area: 64.45 Mbytes
ROM area : 26.01-64.45 Mbytes
RAM area : 0-38.44 Mbytes

There are 7 disk types, 0 - 7, corresponding to the relative sizes of the RAM area and the ROM area on the disk. The disk type of a particular disk is determined by the size of the writable area needed by the programmer. The relationship between the disk type and the ROM and RAM areas is explained in the following table:

Disk type	ROM area	RAM area
Type 0	Zone 0-2 (26,014,080 bytes)	Zone 3-8 (38,444,480 bytes)
Type 1	Zone 0-3 (34,957,440 bytes)	Zone 4-8 (29,501,120 bytes)
Type 2	Zone 0-4 (43,155,520 bytes)	Zone 5-8 (21,303,040 bytes)

		bytes)
Type 3	Zone 0-5 (50,608,320 bytes)	Zone 6-8 (13,850,240 bytes)
Type 4	Zone 0-6 (57,315,840 bytes)	Zone 7-8 (7,142,720 bytes)
Type 5	Zone 0-7 (62,516,480 bytes)	Zone 8 (1,942,080 bytes)
Type 6	All areas (64,458,560 bytes)	None

2-1-5 Logical Block Address (LBA)

The 64DD library accesses blocks on the disk using a series of consecutive numbers called the LBA (Logical Block Address). The LBAs on one disk range from 0 - 4291, and the numbers are allocated according to the following rules:

1. The ROM area is first; the RAM area is second.
2. Head 0 is first; Head 1 is second.
3. On Head 0, the outer circle (zone 0) is first. On Head 1, the inner circle (zone 8) is first.

2-2 64DD On-Board ROM

The commercial version of the 64DD (the consumer-use drive) contains built-in mask ROM. This mask ROM is called DDROM.

DDRROM contains the following data:

- IPL
- Font data
- Wave data

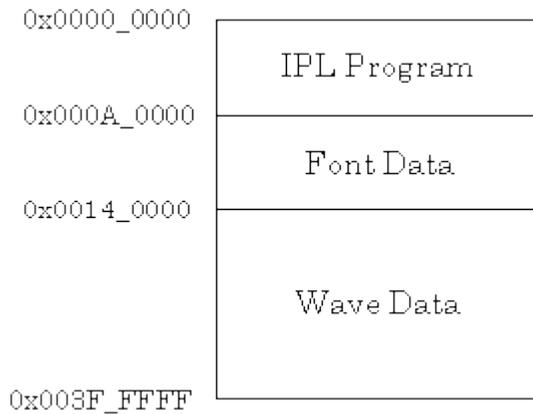


Figure 2-2-1 DDRROM Map

The development-use 64DD does not have built-in DDRROM, but font data and wave data are supplied on the IPL ROM Cartridge. The IPL program is used to transfer special development IPL to the emulation ROM and to write to the N64 Flash ROM cartridge. In addition, when emulating the 64DD on the hard disk, DDRROM can be used when there is no IPL ROM cartridge since the 64DD emulation application supports DDRROM. This font data and wave data is not compatible with the 64DD. You can use this data when the 64DD is connected to N64.

2-2-1 IPL

IPL is the program for booting a game from the 64DD disk. This program executes when power is supplied to the N64 Control Deck if a Game Pak has not been inserted in the Control Deck. The main roles of the IPL program are listed below:

- Checks for expanded memory
 - Displays a message prompting the user to insert a disk
 - Displays and sets the clock
 - Boots a game after a disk is inserted
 - Stores the ID of the booted disk to RDRAM
- All these processes must be performed on the application side when a ROM cartridge is inserted in the N64 since the N64 is started with a non-IPL ROM cartridge.

2-2-2 Font Data

Font data for both alphabetic characters and kanji characters(equivalent to JIS,First Standard) is prepared in DDRROM for displaying error messages. Three functions are available to help determine which address each font starts from (For more details, see [64DD Programming Manual]). The start address 0x000A_0000 for font data in DDRROM is defined as DDRROM_FONT_START in the header file leo.h.

2-2-3 Wave Data

Both 32KHzADPCM and 16KHzADPCM compression format data (.aifc) is available as waveform data for sounds. (For more details, see [64DD Programming Manual]). The start address 0x0014_0000 for wave data in DDROM is defined as DDROM_WAVETABLE_START in the header file leo.h.

Introduction to NINTENDO⁶⁴

Chapter 3 Developing for the 64DD

This chapter explains things to be noted when developing for the 64DD.



- #### 3-1 The Development-Use Drive vs. the Consumer-Use Drive

There are two types of 64DD drive: One for development use and the other for consumer use. Please note that these two types of drives have their own dedicated disks.

•**The Development-Use Drive : Disk Insertion Slot(Bezel) = Blue**

Disk = Blue

•**The Consumer-Use Drive : Disk Insertion Slot(Bezel) = Black**

Disk = Grey

Please be sure and use the appropriate type of disk with the proper drive. The hardware of the two systems is different, and you can damage a drive if you insert the wrong kind of disk.

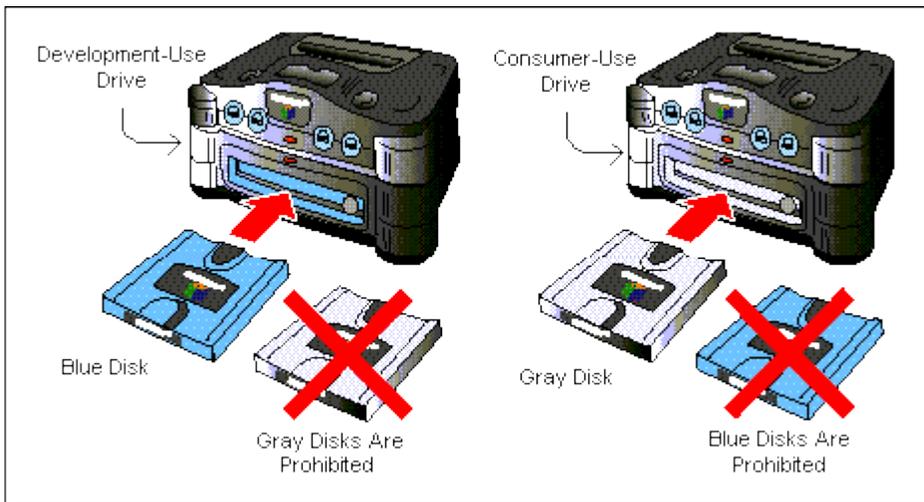


Figure 3-1-1 The Development-Use Drive vs. the Consumer-Use Drive

3-2 The Development-Use 64DD

The Development-Use 64DD does not have built-in DDRAM. To access the DDRAM in the same way as with the Consumer-Use 64DD, you need the IPLROM Cassette in the NUD Development System. Font and wave data are loaded on the board of this IPLROM Cassette. When booting The Development-Use 64DD, you need to transfer the IPL Program, etc. to the N64 Flash Cassette.

3-3 Debugging With an Emulator

The 64DD emulator is available as a development environment which does not use the 64DD. Whenever changes are made to the program during normal debugging, it is necessary to overwrite the disk, and this takes time. With the emulator, files on the hard disk can be executed and their operation checked without having to do a write operation, which significantly cuts back on debugging time. Since the emulator is only an approximate reproduction of the 64DD, a Development 64DD Drive should be used when doing debugging related to the physical operation of the 64DD.

3-4 Debugging with the Development-Use 64DD

3-4-1 Booting from the Disk

When a game is comprised only of disks, the IPL written in DDRAM is used for booting the game from the disk. This is called a disk boot.

When debugging with the N64 Control Deck, the special development-use IPL (file name `ipl4rom`) is written to the flash cassette for use. When the program is developed with Partner-N64, "resetdd" is executed in the Partner-N64 command window in order to transfer the development-use IPL into emulation ROM.

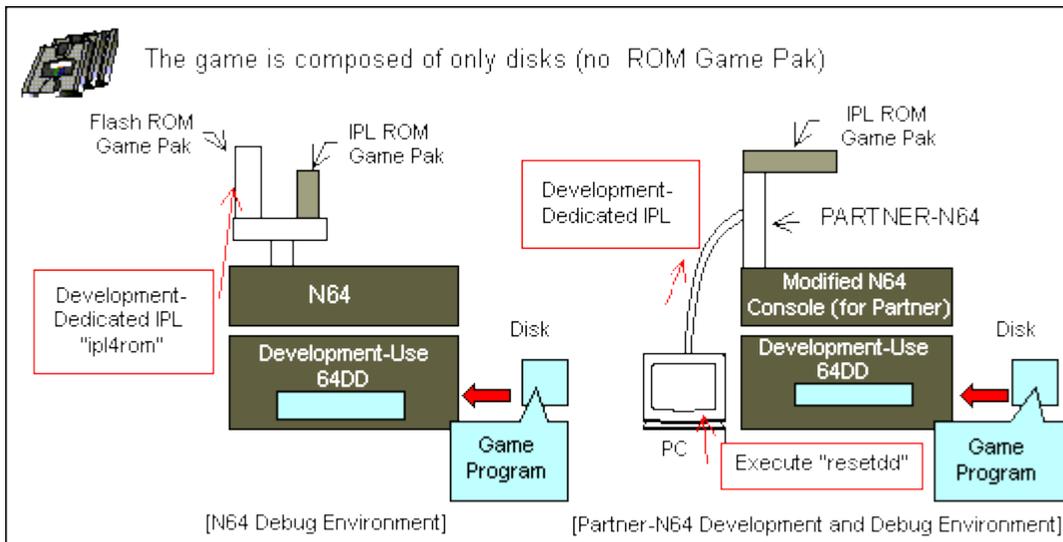


Figure 3-4-1 Debug Environment for Disk Boot

3-4-2 Booting From the Game Pak

If, on the other hand, the game is comprised of disks and a Game Pak, the ROM cartridge is booted first because it has priority. After that, data is read from the disk and possibly also written to the disk as the game progresses and the occasion demands. For a Game Pak boot, the application program is used instead of the special development-use IPL.

When debugging with the N64 Control Deck, the part of the game that is on the Game Pak is written to the flash cassette. When the program is developed with Partner-N64, the Game Pak part of the game is transferred to Partner-N64's emulation ROM and then executed.

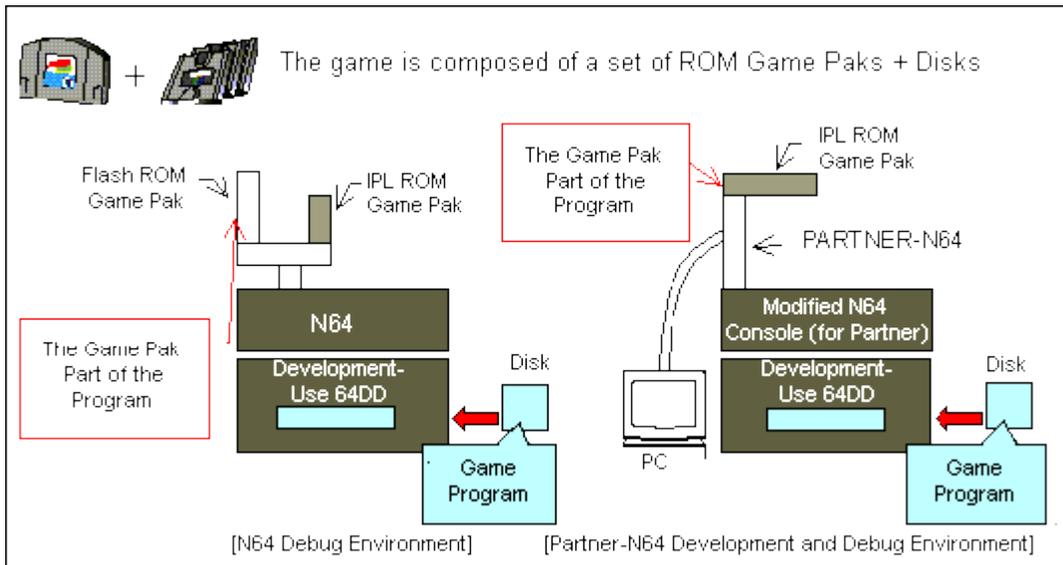


Figure 3-4-2 Debug Environment for ROM Game Paks and Disks

3-5 Accessing the Disk

64DD related instructions are designed on the assumption that another PI device is not being accessed. For this reason, a system using the 64DD should not access the disk and another PI device at the same time. The same thing applies to the use of font data and wave data. Therefore, you must come up with some work-around, such as accessing the disk after you have read the necessary data into a buffer. One way to prevent simultaneous access of the disk and another PI device is to create the program so that instructions to the disk and the ROM cartridge are managed with a single thread, and processes are performed while messages are checked.

- PI device : Mask ROM, SRAM, etc.

3-6 Booting the Game Pak when Connected to the 64DD

3-6-1 Disk Boot

- **Initialization Function**

Please note that the initialization function for a disk boot is different from that for a Game Pak boot.

- **Special Start-up Methods**

With a multiple-disk configuration, only the second and subsequent disks can be used for normal game data. Although, it is possible to restart a separate program stored on these disks by bringing up a special function. However, separate settings are required for each program and the programs cannot both be restarted.

- **Disk ID Storage Method**

Upon booting the disk, the disk id is automatically stored into RDRAM. However, if a game consists of multiple disks, the application has to store the disk id.

3-6-2 Game Pak Boot

Basically, with a game pak boot, applications have to do all the jobs done by an IPL on a booting disk.

- **Initialization Function**

Please note that the initialization function for a disk boot is different from that for a Game Pak boot.

- **System reset immediately after initialization**

Immediately after the initialization function has been executed, the system enters the reset state. In this state, the execution of any other function will generate an error. To return to the normal state, a function to cancel the reset state must be executed.

- **Check the RTC after initialization**

When using the Real-time Clock (RTC) from a Game Pak boot, the RTC must be checked after initialization.

- **Check the Memory Expansion Pak**

Use the program on the ROM cartridge to check whether the Memory Expansion Pak has been inserted. If it has not been inserted, display a message prompting the user to insert the Pak. After that, the 64DD terminates abnormally, except in special cases and cannot start another process.

- **Special Start-up Methods**

In the case of a configuration with a ROM Game Pak and a disk, the disk can be used only for normal game data. Although, it is possible to restart a separate program stored on the disk by bringing up a special

function. However, separate settings are required for each program and the programs cannot both be restarted.

- **Disk ID Storage Method**
All disk IDs must be stored on the application side.

Introduction to **NINTENDO**⁶⁴

Appendices

64DD MFS Functions
64DD Leo Functions



Appendix 64DD MFS Functions

This appendix provides descriptions of 64DD Multi File System (MFS) Library functions.

High-Level Library Functions

The ROM and RAM areas can be assigned virtual drives, and file and directory names can be specified by their full pathnames. Relative paths can also be used, and these cannot be back-tracked.

mfsHInitDiskBoot	Initialize library (for disk boot)
mfsHInitCasBoot	Initialize library (for cartridge boot)
mfsHMediaCheck	Check media status of drive
mfsHMediaFormat	Format media in MFS format
mfsHFopen	Open file
mfsHFread	Read data
mfsHFwrite	Write data
mfsHFclose	Close file
mfsHFseek	Seek head
mfsHFtell	Get file location
mfsHFstat	Get information about file
mfsHFgetAttr	Get file attributes
mfsHFsetAttr	Set file attributes
mfsHFgetCopyCount	Get file copy counter value

mfsHFsetCopyCount	Set copy counter value
mfsHFgetRC	Get file refresh counter value
mfsHFresetRC	Reset refresh counter value
mfsHRemoveFile	Delete file
mfsHRenameFile	Rename file
mfsHGetFileCopyCount	Get file copy count
mfsHSetFileCopyCount	Set file copy count
mfsHGetAttr	Get file (directory) attributes
mfsHSetAttr	Set file (directory) attributes
mfsHGetRC	Get file (directory) refresh counter value
mfsHResetRC	Reset refresh counter value
mfsHGetStat	Get file or directory information
mfsHGetCwd	Get current directory
mfsHChDir	Change current directory
mfsHRmdir	Remove directory
mfsHMkdir	Make directory
mfsHGetVolumeInfo	Get volume information
mfsHGetVolumeAttr	Get volume attributes
mfsHSetVolumeAttr	Set volume attributes
mfsHGetVolumeRC	Get volume refresh counter
mfsHResetVolumeRC	Reset volume refresh counter
mfsHFnsplit	Split structural element in path name
mfsHFnmerge	Merge elements to create path name
mfsHGetDisk	Get current drive number
mfsHSetDisk	Change current drive
mfsHFindFirst	Find directory
mfsHFindNext	Find next file

Low-Level Library Functions

There is a group of functions for the ROM area and a group of functions for the RAM area. Since writing to the ROM area is disabled, there are no write related functions in the ROM area group.

(1) Initialization Functions

mfsInit	Initialize internal variables
mfsInitDiskRom	Initialize for disk operation
mfsInitDiskRam	Initialize for disk operation
mfsInitDiskRomRam	Initialize for disk operation
mfsRamInit	Initialize RAM area library
mfsRomInit	Initialize ROM area library

(2) Disk Operation Functions

mfsDiskIdCheck	Read and check disk ID
mfsRamMediaFormat	Format media

mfsRomMediaCheck	Media check
mfsRamMediaCheck	Media check
mfsRamFlash	Write header area
mfsRamGetFreeSize	Get free space
mfsRamGetFreeDirEntryNum	Get number of files (directories) which can be registered
mfsRomGetFileNum	Get number of files associated with specified directory
mfsRamGetFileNum	Get number of files associated with specified directory
mfsRomGetVolumeAttr	Get volume attributes
mfsRamGetVolumeAttr	Get volume attributes
mfsRomGetVolumeName	Get volume name
mfsRamGetVolumeName	Get volume name
mfsRomGetVolumeDate	Get volume date
mfsRamGetVolumeDate	Get volume date
mfsRamGetVolumeRC	Get refresh counter value
mfsRomGetDiskType	Get disk type
mfsRamGetDiskType	Get disk type
mfsRomGetDestination	Get destination code
mfsRamGetDestination	Get destination code
mfsRamSetVolumeAttr	Set volume attributes
mfsRamSetVolumeName	Set volume name
mfsRamSetVolumeDate	Set volume date
mfsRamSetDestination	Set destination code
mfsRamResetVolumeRC	Reset refresh counter
mfsRamRepairHeader	Repair header area

(3) Directory Functions

mfsRamMakeDir	Make directory
mfsRomGetDirID	Get directory ID
mfsRamGetDirID	Get directory ID
mfsRomGetDirName	Get directory name
mfsRamGetDirName	Get directory name
mfsRamRemoveDir	Remove directory
mfsRamRenameDir	Rename directory

(4) File Operation Functions

mfsRamCreateFile	Create file
mfsRomGetFileDirID	Get directory ID of file
mfsRamGetFileDirID	Get directory ID of file
mfsRomReadFile	Read data
mfsRamReadFile	Read data
mfsRamWriteFile	Read data
mfsRomSeekFile	Seek head

mfsRamSeekFile	Seek head
mfsRomGetFileName	Get file name
mfsRamGetFileName	Get file name
mfsRamRemoveFile	Delete file
mfsRamRenameFile	Rename file
mfsRomGetFileAttr	Get file (directory) attributes
mfsRamGetFileAttr	Get file (directory) attributes
mfsRomGetFileDate	Get file (directory) date
mfsRamGetFileDate	Get file (directory) date
mfsRomGetFileParentDir	Get directory ID of parent directory
mfsRamGetFileParentDir	Get directory ID of parent directory
mfsRomGetFileSize	Get file size
mfsRamGetFileSize	Get file size
mfsRamGetFileCopyCount	Get file (directory) copy limit count
mfsRomGetFileGameCode	Get company code, game code of file (directory)
mfsRamGetFileGameCode	Get company code, game code of file (directory)
mfsRamGetFileRC	Get refresh counter value of file (directory)
mfsRamSetFileAttr	Set file (directory) attributes
mfsRamSetFileDate	Set file (directory) date
mfsRamSetFileCopyCount	Set file (directory) copy limit number
mfsRamSetFileGameCode	Set company code, game code of file (directory)
mfsRamResetFileRC	Reset refresh counter value of file (directory)
mfsRamSetFileParentDir	Change parent directory
mfsRamResizeFile	Resize file
mfsRomGetFileStat	Get information
mfsRamGetFileStat	Get information
mfsRomGetDirListFirst	Get directory contents (list directory)
mfsRamGetDirListFirst	Get directory contents (list directory)
mfsRomGetDirListNext	Continue to get directory contents (next page of directory list)
mfsRamGetDirListNext	Continue to get directory contents (next page of directory list)

Device Functions

mfsReadWriteLBA	Read/write from/to disk
mfsReadLBA	Read disk
mfsWriteLBA	Write to disk
mfsReadDiskID	Read disk ID
mfsSpdlMotor	Control 64DD head and motor
mfsTestUnit	Check status of 64DD
mfsReadRtc	Read date and time from internal RTC
mfsSetRtc	Set internal RTC

Other Functions

mfsGetGameCode	Get company code, game code
mfsSetGameCode	Set company code, game code
mfsStrCmp	Compare text strings
mfsStrnCmp	Compare text strings
mfsStrCpy	Copy text string
mfsStrnCpy	Copy text string
mfsStrLen	Calculate length of text string
mfsStrrChr	Find location of last appearance of character "c"
mfsStrCat	Connect text string
mfsStrWCmp	Compare text string (wild card version)
mfsSetLeoBusyFunc	Register PI control function (before using PI)
mfsSetLeoReadyFunc	Register PI control function (after using PI)
mfsSetLeoErrorFunc	Register error processing function for Leo function

Global Variables

mfsError	Stores errors when LEO function is called inside MFS library
mfsDiskInfo	Structure for storing information regarding disk inside MFS library

Appendices Leo Functions

This is the outline of functions in the Leo Library.

Initializer Functions

Functions to create a device driver called Leo Manager

LeoCreateLeoManager	Start an N64 Disk Drive game
LeoJCreateLeoManager	Start a Japanese Game Pak game
LeoCACreateLeoManager	Start an American Game Pak game

Asynchronous Functions

Actual processing is done within a different thread from the one that brought up the function. Also, with this function, processing is done asynchronous to the thread that was brought up.

LeoSpdlMotor	Control the drive's motor and the position of the head
LeoReadWrite	Perform reads and writes from the N64 DD
LeoSeek	Issue the Seek command for the 64DD
LeoReadDiskID	Get the disk ID
LeoReadRTC	Read the time from the RTC
LeoSetRTC	Set the time in RTC

Synchronous Functions

Functions to be processed within the threads brought up

LeoByteToLBA	Convert from byte to LBA
LeoLBAToByte	Convert from LBA into byte
LeoReadCapacity	Calculate the usable disk space
LeoInquiry	Check the version of the N64 Disk Drive hardware and the software.

Font-Use Functions

Functions to get the stored address for font data in the ROM built within the 64DD

LeoGetKAdr	Get the storage offset address for a kanji character specified by the shift-JIS code in the sjis argument
LeoGetAAAdr	Get the storage offset address for the ASCII character specified by the character code in the code argument
LeoGetAAAdr2	Get the storage offset address for the ASCII character specified by the character information data in the ccode argument

Other Functions

Other Important Functions

LeoReset	Interrupt the command being executed and clear the command queue.
LeoResetClear	Cancel the N64 Disk Drive reset state
LeoClearQueue	Clear the Leo Manager's command queue
LeoBootGame	Executes a reboot

Special Functions

Functions with use limitations and not for general use

LeoModeSelectAsync	Provide a way to alter the wait-time specified for changes between the active,standby, and sleep modes to meet the needs of the game
LeoRezero	Issue the recalibration command to the N64 Disk Drive
LeoTestUnitReady	Check the state of the N64 Disk Drive

