Architecture Guide

SDK Version 20-APR-2004

© 2000-2004 Nintendo

DOL-06-0298-001-A3
Released: April 29, 2004

"Confidential"

These coded instructions, statements, and computer programs contain proprietary information of Nintendo of America Inc. and/or Nintendo Company Ltd., and are protected by Federal copyright law. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

© 2000-2004 Nintendo

TM and ® are trademarks of Nintendo.

Dolby, Pro Logic and the Double-D symbol are trademarks of Dolby Laboratories.

IBM is a trademark of International Business Machines Corporation.

Roland GS Sound Set is a trademark of Roland Corporation U.S.

All other trademarks and copyrights are property of their respective owners.

Table of Contents

Hardware Overview

Revision History .... I-iii
1 Gekko CPU I-2
1.1 L1 data cache I-3
1.2 L2 caches I-4
1.3 FPU performance I-5
1.3.1 Paired singles I-5
1.3.2 Free fixed and floating point conversions I-5
1.4 Out-of-order instruction execution I-6
1.5 Branch prediction I-7
2 1TSRAM main memory I-9
2.1 DRAM bank architecture I-9
2.2 1TSRAM architecture I-9
3 Graphics Processor (GP) I-11
3.1 Functional units I-11
3.1.1 Embedded memory I-12
3.1.2 Embedded 1TSRAM memory I-13
3.2 Command Processor (CP) I-13
3.3 Transform Processor (XF) I-14
3.4 Rasterizer (RAS) I-14
3.5 Texture Environment Processor (TEV) I-16
3.5.1 Re-ordered blending I-17
3.6 Pixel Engine (PE) I-18
3.6.1 Antialiasing I-18
3.7 Internal 1TSRAM memory buffers I-19
3.7.1 Texture streaming cache I-20
3.7.2 Preloaded texture map I-20
4 The audio DSP I-21
4.1 Features and performance I-22
5 Auxiliary audio memory (ARAM) I-23
6 Optical disc drive I-25
6.1 Speculative prefetch I-25
6.2 Interleaved data access and audio streaming I-25
7 Controller (PAD) I-27
7.1 Game pad state sampling control I-27
7.2 Communication buffer I-27
8 Expansion Interface 0 (EXI0) I-29
9 Expansion Interface 1 (EXI1) I-31
10 Audio Interface (AI) I-33
11 Video Interface (VI) I-35

Software Development Kit Overview

Revision History .... II-iii
1 Goals of the Nintendo GameCube SDK II-1
2 Development hardware II-3
2.1 Programmer development system II-3
2.1.1 Goals II-3
2.1.2 Key features II-3
2.1.3 Optical disc emulation II-4
2.2 DDH/PC host communication interface II-4
3 SDK components II-5
4 Compiler and debugger suites II-7

© 2000-2004 Nintendo

5 Build environment ... II-9
6 Operating system ... II-11
6.1 Memory address map ... II-11
6.2 Execution model ... II-11
6.3 Utility functions ... II-12
6.4 Optical disc file system ... II-12
6.4.1 Random access comparison of optical disc drive to mask ROM ... II-13
7 Graphics ... II-15
7.1 Graphics library (GX) ... II-15
7.1.1 Drawing geometry ... II-15
7.1.2 Geometry processing control ... II-15
7.1.3 Texture application ... II-16
7.1.4 Other pixel operations ... II-16
7.1.5 Miscellaneous functions ... II-16
7.2 Matrix-Vector library (MTX) ... II-16
7.3 Demonstration library (DEMO) ... II-16
7.4 2D Graphics library (G2D) ... II-17
7.5 Character Pipeline (articulated animation set) ... II-17
7.5.1 Data extraction libraries and tools ... II-18
7.5.2 Runtime libraries ... II-20
8 Audio ... II-21
8.1 Audio and graphics game framework ... II-21
8.2 Factor5 MusyX sound system ... II-21
8.3 Sound sets ... II-22
8.3.1 Roland wavetable ... II-22

© 2000-2004 Nintendo

Hardware Overview

SDK Version 20-APR-2004

Contents

Revision History .... I-iii
1 Gekko CPU I-2
1.1 L1 data cache I-3
1.2 L2 caches I-4
1.3 FPU performance I-5
1.3.1 Paired singles I-5
1.3.2 Free fixed and floating point conversions I-5
1.4 Out-of-order instruction execution I-6
1.5 Branch prediction I-7
2 1TSRAM main memory I-9
2.1 DRAM bank architecture I-9
2.2 1TSRAM architecture I-9
3 Graphics Processor (GP) I-11
3.1 Functional units I-11
3.1.1 Embedded memory I-12
3.1.2 Embedded 1TSRAM memory I-13
3.2 Command Processor (CP) I-13
3.3 Transform Processor (XF) I-14
3.4 Rasterizer (RAS) I-14
3.5 Texture Environment Processor (TEV) I-16
3.5.1 Re-ordered blending I-17
3.6 Pixel Engine (PE) I-18
3.6.1 Antialiasing I-18
3.7 Internal 1TSRAM memory buffers I-19
3.7.1 Texture streaming cache I-20
3.7.2 Preloaded texture map I-20
4 The audio DSP I-21
4.1 Features and performance I-22
5 Auxiliary audio memory (ARAM) I-23
6 Optical disc drive I-25
6.1 Speculative prefetch I-25
6.2 Interleaved data access and audio streaming I-25
7 Controller (PAD) I-27
7.1 Game pad state sampling control I-27
7.2 Communication buffer I-27
8 Expansion Interface 0 (EXI0) I-29
9 Expansion Interface 1 (EXI1) I-31
10 Audio Interface (AI) I-33
11 Video Interface (VI) I-35

Code Examples

Code 1 - Out-of-order instruction handling I-6

Equations

Equation 1 - Color blending I-17

© 2000-2004 Nintendo

Figures

Figure 1 - Nintendo GameCube™ functional blocks and busses I-1
Figure 2 - CPU functional blocks I-3
Figure 3 - Data cache configuration I-4
Figure 4 - L1 and L2 caches can store a working set's code and data I-4
Figure 5 - Paired singles register format I-5
Figure 6 - Floating/fixed point conversions I-6
Figure 7 - Example of out-of-order execution I-7
Figure 8 - Branch prediction feature I-8
Figure 9 - DRAM bank architecture (16MB DRAM with two banks) I-9
Figure 10 - Graphics Processor (GP) blocks I-12
Figure 11 - Non-linear data relationship between pixel rasterization and texture memory access I-12
Figure 12 - Command Processor (CP) blocks I-13
Figure 13 - Transform Processor (XF) blocks I-14
Figure 14 - Pixel footprint analysis I-15
Figure 15 - TEV pipeline stages I-16
Figure 16 - TEV stage result reordering I-18
Figure 17 - Super-sampling antialiasing I-19
Figure 18 - Embedded frame buffer (EFB) I-19
Figure 19 - Embedded texture memory (TMEM) I-19
Figure 20 - Texture streaming cache I-20
Figure 21 - Audio DSP blocks I-21
Figure 22 - Disc drive speculative prefetch I-25
Figure 23 - Interleaved data and audio I-25
Figure 24 - Game pad interface blocks I-27
Figure 25 - Audio Interface (AI) blocks I-33

Tables

Table 1 - Polygon performance I-11
Table 2 - TEV stage fill rates I-16
Table 3 - Video formats I-35

© 2000-2004 Nintendo

Revision History

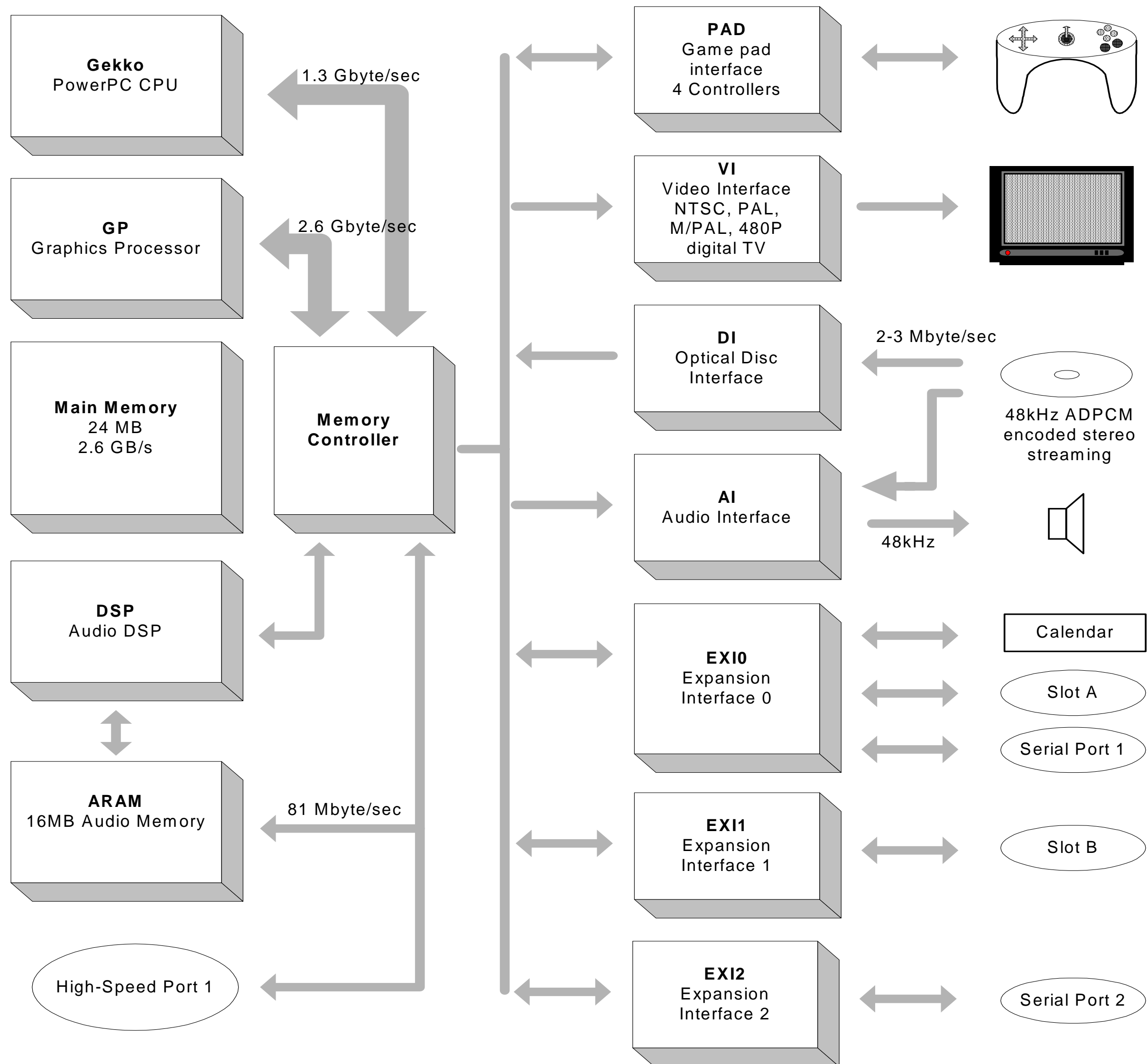
Revision No.	Date Revised	Items (Chapter)	Description	Revised By
20-APR-2004	6/30/2003	I-1	Revised Figure 1	R. Daring
		I-2	Deleted Figure 10	R. Daring
5-Sept-2002	9/5/2002	-	First release by Nintendo of America, Inc.	-

© 2000-2004 Nintendo

© 2000-2004 Nintendo

I

Figure 1 - Nintendo GameCube™ functional blocks and busses



1 Gekko CPU

The Gekko CPU includes an IBM PowerPC 750 processor core with additional functional units for performance enhancement. The Gekko CPU has the following features:

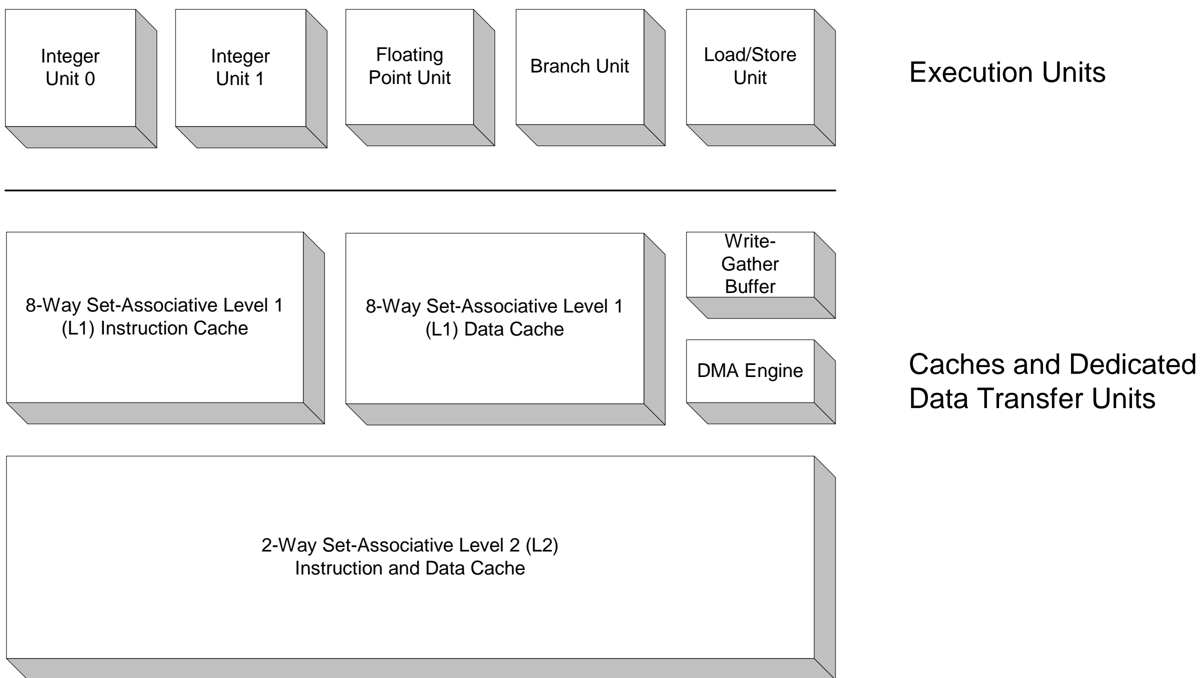
- 486Mhz internal clock operation.
- 162Mhz 64-bit bus to main memory (1GB/s peak bandwidth).
- 32KB 8-way set associative L1 Icache.
- 32KB 8-way set associative L1 Dcache (16KB data scratchpad configurable).
- Super-scalar microprocessor with five execution units: 2 integer units, 1 floating point unit, 1 load/store unit and branch unit.
- DMA unit servicing 16KB data scratchpad; 15-entry DMA request queue.
- Write-gather buffer for writing graphics command lists to the graphics chip.
- Embedded 256KB 2-way set-associative L2 unified cache.
- Two (2) 32-bit Integer Units (IU).
- 1 Floating Point Unit (FPU), 32-bit and 64-bit.
- FPU supports Floating Point Paired Singles (FP/PS).
  - FPU supports `ps_madd` (paired-single multiply-add). Most FP/PS instructions can be issued every cycle and complete in three cycles.
  - Simultaneous conversion between fixed and floating point numbers while loading/storing FPU registers without performance penalty.
- Branch Unit offers static and dynamic branch prediction.
- Out-of-order execution; i.e., when an instruction stalls on data, subsequent instructions can continue to issue and execute. All instructions complete in correct program sequence to preserve program logic.

The Gekko CPU has the following features to minimize CPU stalls on data fetching and maximize computational throughput:

- Non-blocking caches.
- Branch prediction.
- 8-way set-associative caches.
- 256KB L2 cache.
- Out-of-order execution feature.

© 2000-2004 Nintendo

Figure 2 - CPU functional blocks



1.1 L1 data cache

The 32KB L1 data cache has two modes of operation:

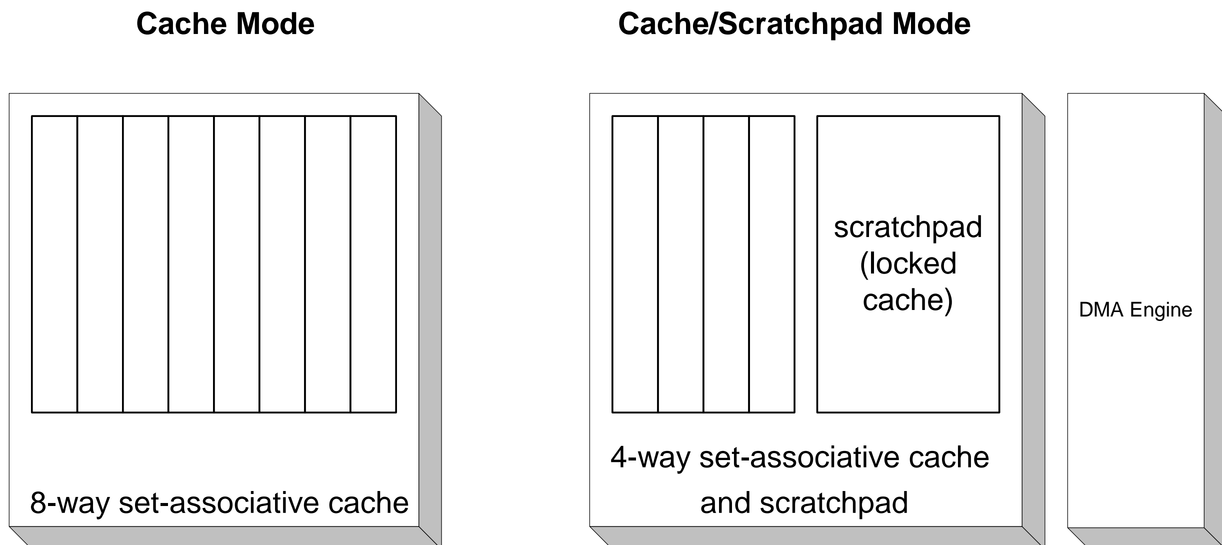
1. 32KB 8-way set-associative L1 data cache.
2. 16KB 4-way set-associative L1 data cache + 16KB data scratchpad buffer.

The DMA engine transfers data between the 16KB data scratchpad and main memory. The engine can issue up to 15 pending requests. The data scratchpad is directly programmable, so the game programmer can eliminate cache miss unpredictability by fetching the data prior to processing.

© 2000-2004 Nintendo

The Gekko CPU's 4-way or 8-way set-associative data cache significantly minimizes cache misses and cache thrashing.

Figure 3 - Data cache configuration

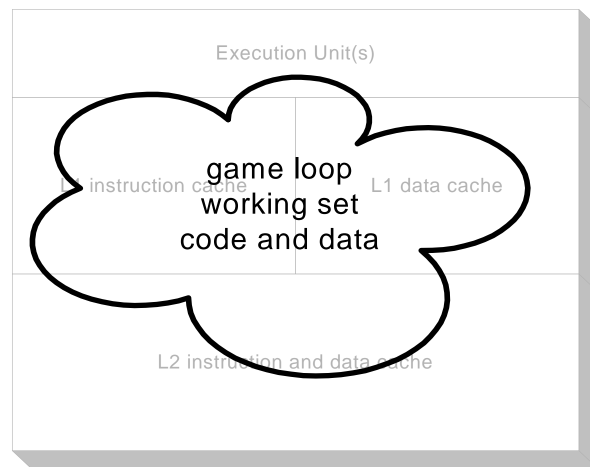


1.2 L2 caches

The amount of code and data in a modern console game program is increasing steadily, as is the complexity of its working sets (a working set is defined as the code and data used in one loop of a game event).

The Gekko CPU provides a large 256KB L2 cache to keep more code and data close to the CPU computation units. This features gives the Gekko ability to work continuously through 486 million cycles, rather than losing cycles by waiting frequently for memory.

Figure 4 - L1 and L2 caches can store a working set's code and data



© 2000-2004 Nintendo

1.3 FPU performance

Most floating point operations can be issued every cycle and can complete in three cycles. These operations include add, sub, mul, and multiply-add instructions. Reciprocal and reciprocal-square-root estimation instructions are available as well.

- One-cycle execution rate for common floating point instructions.
- Floating point instruction latency completion in three cycles.
- Two-cycle (minimum) execution for floating point reciprocal (1/x) and reciprocal square root (1/sqrt(x)) operations; completion in four cycles.

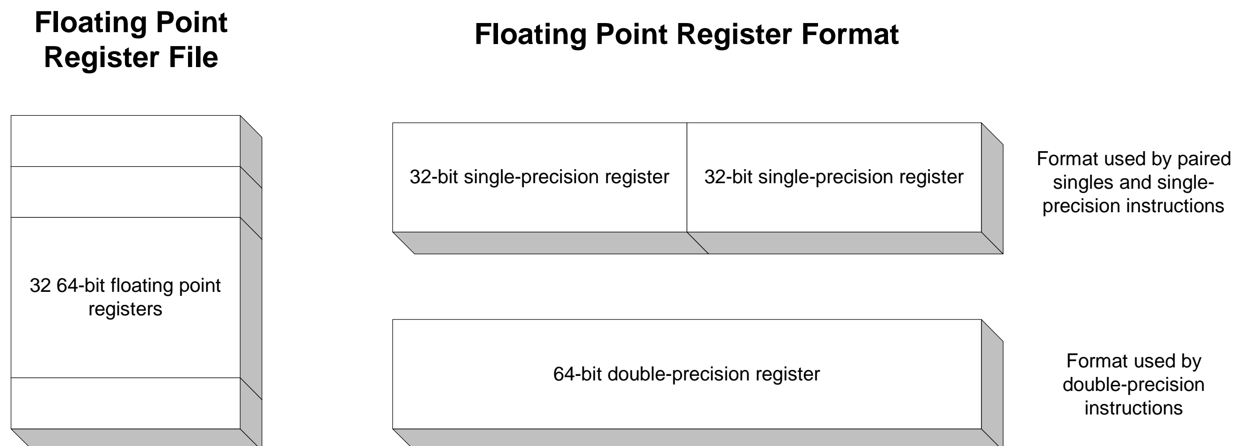
For more details, refer to the *IBM Gekko RISC Microprocessor User's Manual*, Chapter 6.7, "Instruction Latency Summary."

1.3.1 Paired singles

The FPU can also execute paired-single (PS) vector instructions. This means we can perform two floating point instructions per cycle for many instructions, including multiply-add (`ps_madd`); however, the data must be in vector format (i.e., one 64-bit word contains two single-precision floats).

- Two floating point instructions execute per cycle; latency completion in three cycles.
- Two floating point reciprocal (1/x) and/or reciprocal square root (1/sqrt(x)) instructions execute every two cycles (minimum); completion of two instructions in four cycles.
- A single-precision floating point instruction can use the lower single-precision result of a paired singles register. This feature gives the CPU the ability to interchange results between paired singles and single-precision instructions rapidly.

Figure 5 - Paired singles register format



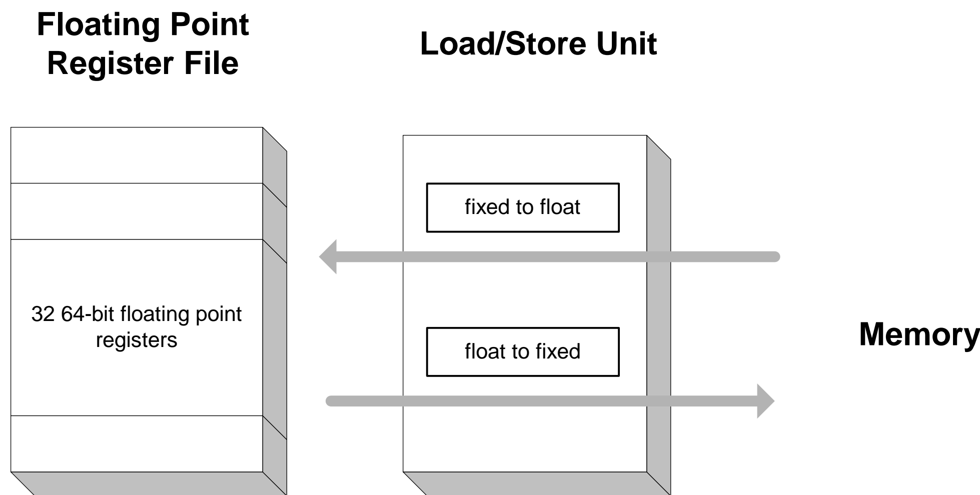
1.3.2 Free fixed and floating point conversions

The Gekko CPU includes special fixed point and floating point conversion hardware in its load/store unit. This hardware can perform the following conversions:

- Free fixed point-to-floating point conversions, if loading an FP/PS register.
- Free floating point-to-fixed point conversions, if storing an FP/PS register.
- Programmable decimal point in fixed point format.

© 2000-2004 Nintendo

Figure 6 - Floating/fixed point conversions



For more information about this Gekko CPU feature, refer to the *IBM Gekko RISC Microprocessor User's Manual*, sections 2.1.2.9, "Graphics Quantization Registers (GQRs)"; 1.2.2.4.3, "Load/Store Unit (LSU)"; and 2.3.4.3.12, "Paired Single Load and Store Instructions."

1.4 Out-of-order instruction execution

Modern microprocessors can achieve high speeds by using multiple processor cycle pipeline stages for computations. The Gekko CPU runs at 486Mhz with five different execution units, making it able to complete a lot of work in a very short time. If the CPU had to wait for all instructions to finish sequentially, many computing cycles would be lost; therefore, the Gekko CPU has an out-of-order instruction execution capability to increase the number of operations possible per cycle.

This feature enables instructions to execute while previous instructions are stalled waiting for memory. The following example instruction sequence shows how out-of-order execution can help to increase performance.

Code 1 - Out-of-order instruction handling

```
float   instruction 1 regA            // floating point instruction, output to regA
float   instruction 2 regA            // floating point instruction, input need regA
integer instruction 3 regB            // integer instruction, output is regB
branch  instruction 4 regB            // branch instruction, input is regB
```

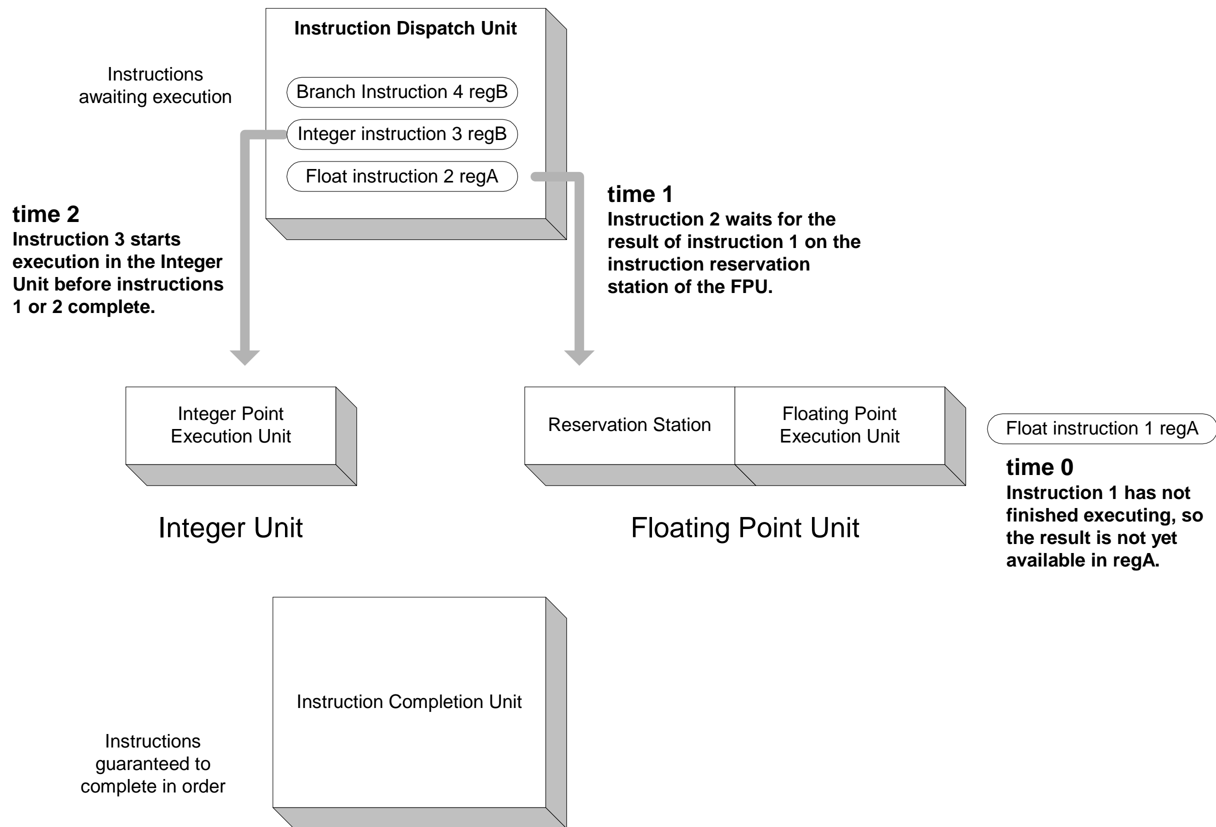
In this example, **`instruction 2`** becomes blocked while waiting for **`instruction 1`** to complete (remember that the typical floating point instruction has a three-cycle latency). However, **`instruction 3`** has no dependencies on either of the two previous instructions, so this one issues and completes in one cycle (as most integer instructions do). Branch **`instruction 4`**, which needs the result of **`instruction 3`** in order to determine branch conditions, can also start execution.

This example shows that you can complete the determination of branch while waiting for independent floating point instructions to complete. Many other code sequences can benefit from out-of-order execution.

© 2000-2004 Nintendo

An additional hardware “completion unit” guarantees that the code will complete in the correct sequence. For example, `instruction 3` is not allowed to *complete* before `instruction 2` does; however, `instruction 3` still gets all of its work done, in effect, while `instruction 2` is waiting for execution. This feature typically increases performance by 25-30% when compared to a processor without an out-of-order instruction execution feature.

Figure 7 - Example of out-of-order execution



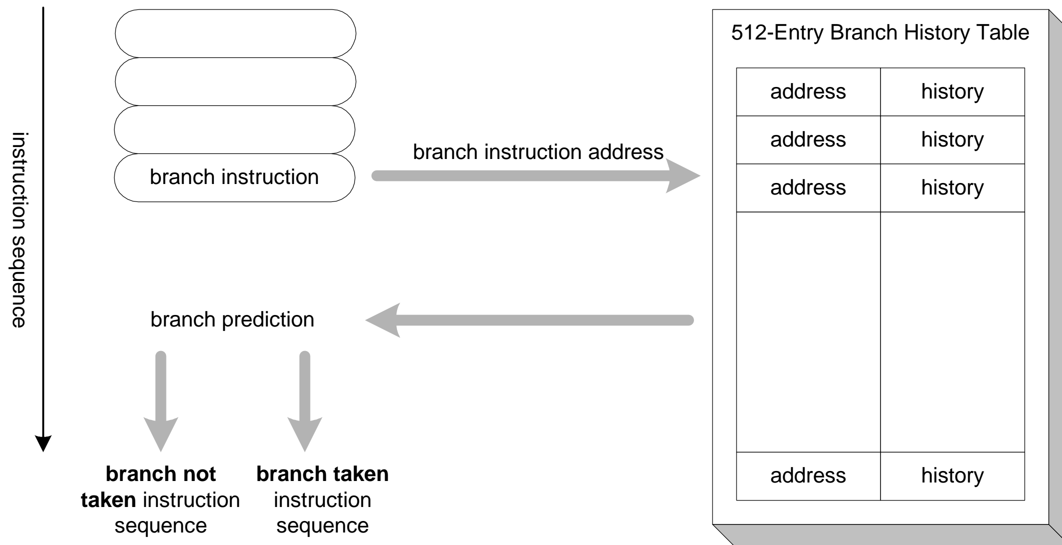
1.5 Branch prediction

Pipelining the execution stages allows for high-speed CPU performance. However, this method leaves the system open to the risk of stalling whenever the code branches because the execution pipeline must wait until the branch condition is resolved.

The Gekko CPU provides special branch prediction hardware to significantly minimize such stalling. It keeps a history of recent branches and uses this history to predict code sequence, which gives the CPU the ability to process instructions beyond the current branch before branch conditions have completed. The branch prediction hardware tracks all but the last loop branch conditions, thus it can eliminate all branch-related stalls in the pipeline except for the last branch out of the loop.

The Gekko CPU has a 512-entry table to contain branch history information, meaning that it can keep up to 512 branch program counter (PC) addresses to record prior branch results. These results act as “hints” that help the Gekko CPU predict which “direction” the next branch(es) will take. The CPU uses this information to continue executing code beyond the current branch. If the prediction turns out to be wrong when the current branch condition resolves, then the Gekko CPU knows to “rewind” all the work and return to execute the correct branch.

Figure 8 - Branch prediction feature



© 2000-2004 Nintendo

2 1TSRAM main memory

With so many high-speed processors in the Nintendo GameCube™ hardware, we need a high performance memory system that offers both high bandwidth and low latency. (Latency is often overlooked, which can result in many processors “waiting” for data from memory subsystems and thus stalling computation.) Nintendo GameCube uses high-performance 1TSRAM for main memory, which offers the following advantages:

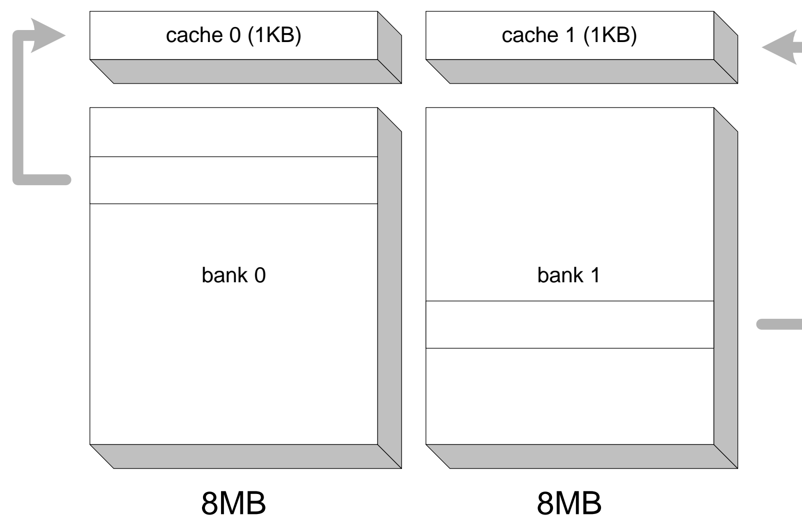
- Random access with at full peak performance.
- 2.6GB/second peak bandwidth.
- 32-byte random-access transfer packet.
- 24MB total capacity.

To understand 1TSRAM in more detail, we should briefly review DRAM architecture.

2.1 DRAM bank architecture

All modern DRAMs use bank architecture, with two to four banks typically available. Bank architecture is very easy to understand. It functions just like caches.

Figure 9 - DRAM bank architecture (16MB DRAM with two banks)



The caches for each of the DRAM banks are typically only 1KB of contiguous addressable memory. Each time a memory request falls outside a 1KB cache region, the memory set must signal the processor to wait. It is therefore difficult to minimize bank cache misses during program execution. Code instructions and data are not close to each other in memory, nor are the large amounts of texture and graphical object memory necessary for a visually-complex game environment. This means that a significant amount of processor performance is wasted waiting for memory.

2.2 1TSRAM architecture

1TSRAM has no banks. Every location in memory is randomly accessible, so a processor requesting data does not stall in memory access to any location.

I

© 2000-2004 Nintendo

© 2000-2004 Nintendo

3 Graphics Processor (GP)

The GP has the following basic performance specifications:

- 162MHz operation.
- 20M/27M/32M polygons/second peak, depending on feature selection.
- 648M pixels/second peak.

Many features may be enabled at these peak performance rates.

Table 1 - Polygon performance

Features	Performance
1 vertex color + 1 light + 1 texture	20M polygons/second
No vertex color + 1 texture	27M polygons/second
1 vertex color + no texture (Gouraud shading)	32M polygons/second

All of these polygon performance times include fogged, Z-buffered, transparency-blended pixels. For polygons with textures, the following additional operations can be performed at peak fill rate of 648M pixels/second.

- Trilinear mipmap-filtered, perspective-corrected texture.
- S3TC compressed textures.
- Single-texture map.

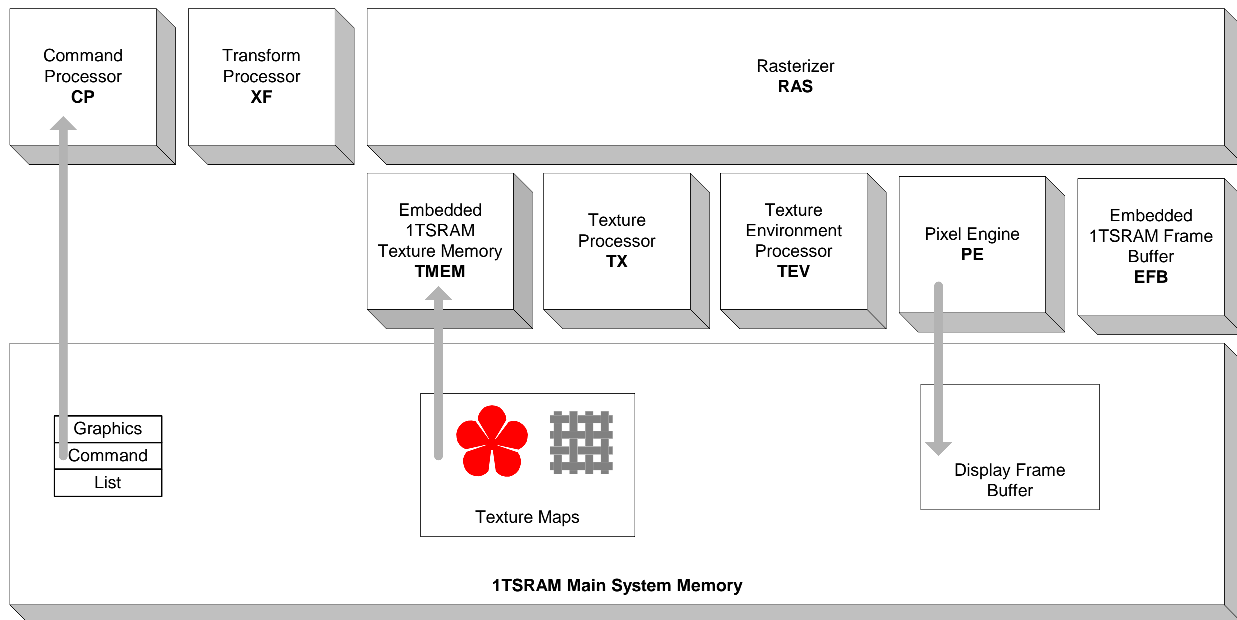
3.1 Functional units

Here is a brief description of the role played by each functional unit:

- The Command Processor (CP) interprets commands generated by the CPU. It supports vertex array indexing, making it easy to construct geometry using indices for positions, normals, colors, and texture coordinates. Duplication of graphical vertex data is not necessary.
- The Transform Processor (XF) performs Model-to-World, World-to-Viewport, texture coordinate, and normal transformations. The XF can also compute local diffuse lighting, infinite specular lighting, and planar texture projection. Clipping is also performed here.
- The Texture Processor (TX) can interpret many texture formats. It also performs texture caching and texture filtering, and it can apply multiple texture maps per polygon.
- The Texture Environment Processor (TEV) can combine the colors generated by the XF and the multiple textures generated by the TX. The TEV is a more powerful version of the Color Combiner (CC) in the Nintendo 64 (N64) system.
- The Pixel Engine (PE) performs blending, Z-buffering, and antialiasing operations. Z-buffer rejection can be performed prior to texture application, thereby maximizing performance.

© 2000-2004 Nintendo

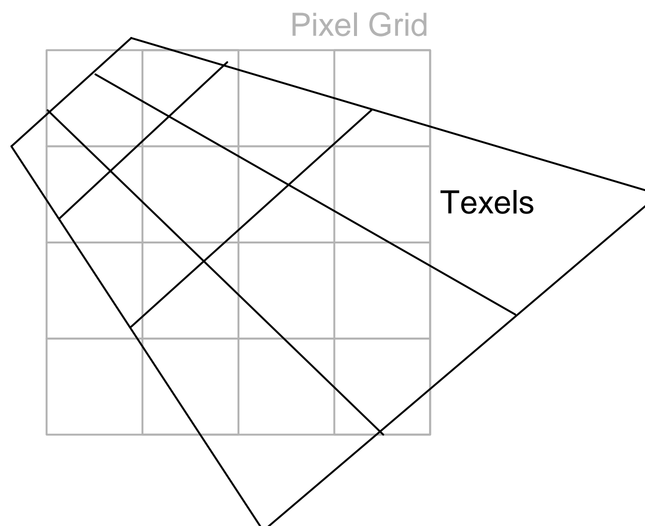
Figure 10 - Graphics Processor (GP) blocks



3.1.1 Embedded memory

Embedded memory refers to memory that is included (embedded) on the same silicon chip; in this case, the Graphics Processor. When the GP is applying textures and using bilinear or trilinear mipmap filters, it requires up to eight texels for every pixel rendered. Texture access during rasterization is a non-linear memory access operation, so TMEM uses this embedded memory technology to eliminate performance bottlenecks.

Figure 11 - Non-linear data relationship between pixel rasterization and texture memory access



When drawing many polygons, especially small ones such as particle systems, the GP will draw rapidly to different parts of the screen, which results in near random memory access patterns. The embedded frame buffer (EFB) thus uses 1TSRAM to gain high speed random access performance.

© 2000-2004 Nintendo

3.1.2 Embedded 1TSRAM memory

Traditionally, SRAM offers a 10X performance increase over DRAM; however, SRAM has always been bigger than DRAM—traditional SRAM uses six transistors (6T) per bit of storage—so silicon chips could not contain large amounts of it. Nintendo GameCube uses a revolutionary one transistor (1T)-per-bit SRAM technology. 1TSRAM gives Nintendo GameCube the ability to include SRAM performance at the same size as DRAM.

Using embedded 1TSRAM for the frame buffer means that pixel Z-buffer and color blending operations are complete free. Furthermore, the Nintendo GameCube rasterizer can render into the 1TSRAM frame buffer without stalling, whereas DRAM stalls can occur during rendering and thus lower the total performance.

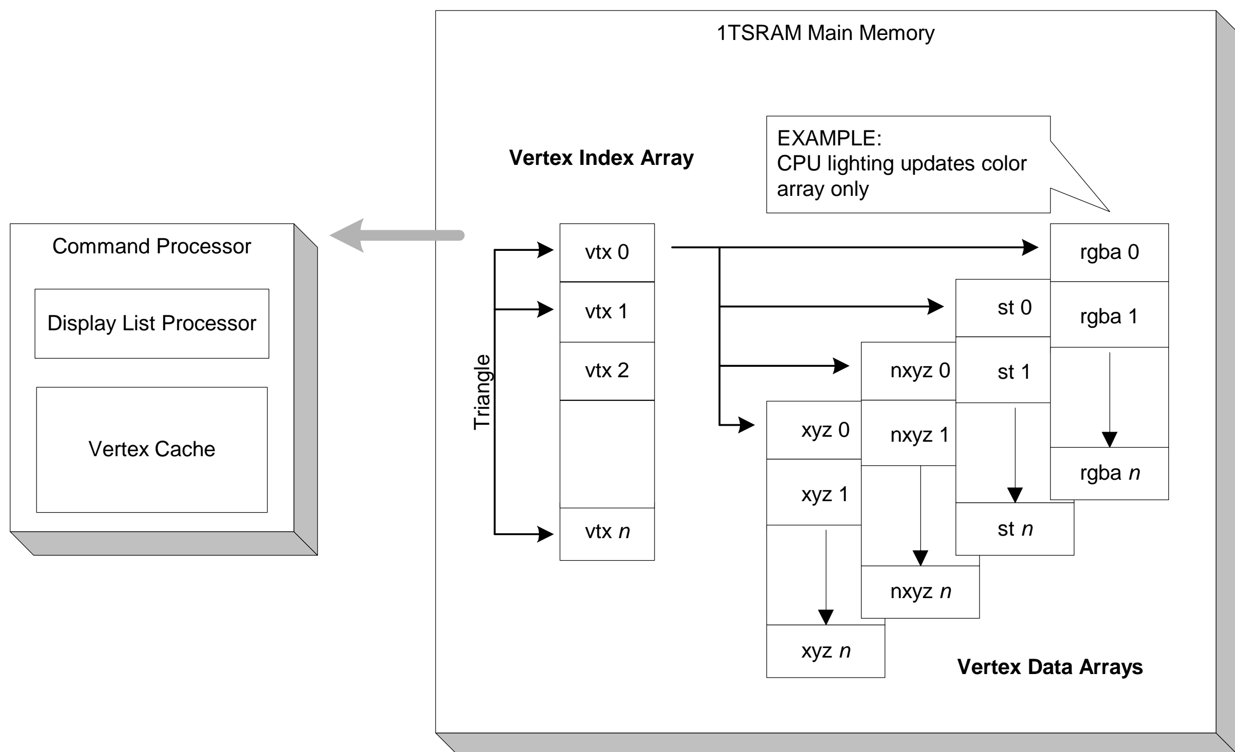
3.2 Command Processor (CP)

The CP handles a wide range of vertex and primitive data structures, from a single stream of vertex data (containing position, normal, texture coordinates, and colors) to fully-indexed arrays. Any vertex component can be index-referenced or inlined directly in the command stream, thus enabling efficient data processing by the CPU. Rather than being restricted to a rigid graphics display data structure which would result in lost performance, the CPU can perform the calculations naturally.

For example, a CPU lighting algorithm must generate only a color array from a list of normals and positions (see "Figure 12 - Command Processor (CP) blocks" on page 13). The CPU executes a simple `for()` loop to process a list of lighting parameters to generate the color array; there is no need to follow a triangle list display data structure. Likewise, there is no need to format the data for display after finishing. The data can be consumed naturally.

The CP has a one level-deep display list, meaning that the top level command stream can “call” the display list, but only one level down. This is excellent for any pre-computed commands and instancing of geometry.

Figure 12 - Command Processor (CP) blocks



© 2000-2004 Nintendo

3.3 Transform Processor (XF)

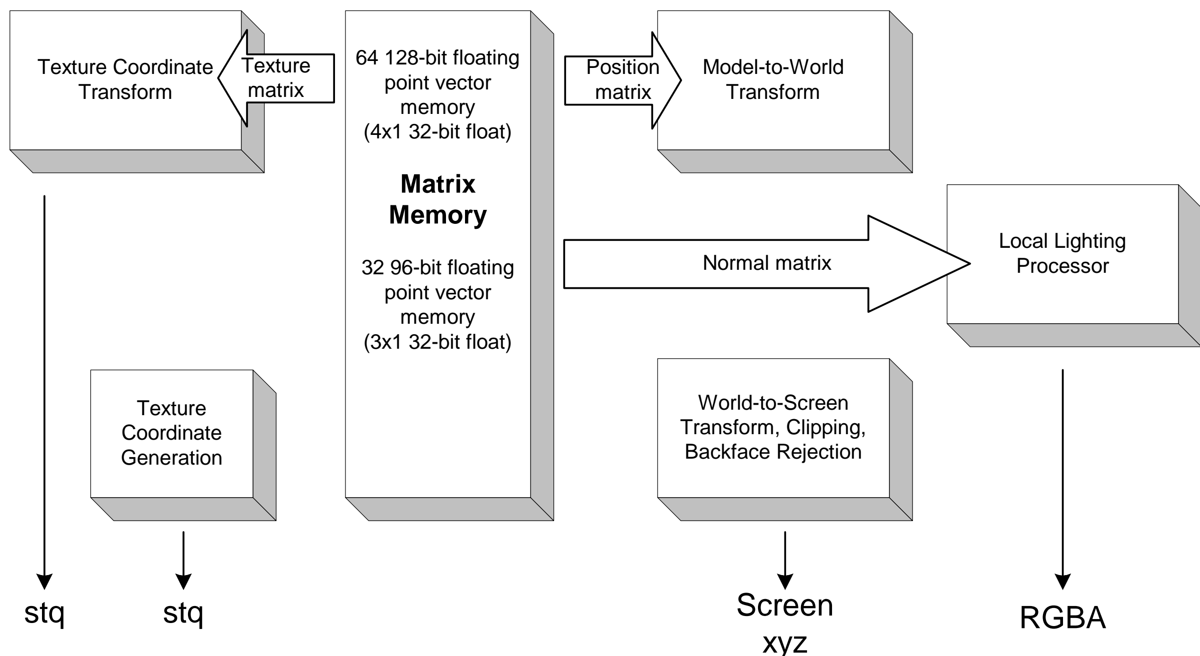
The XF is a hardware 3D geometry transformation pipeline. It performs typical 3D transform operations at a peak rate of 20M to 32M polygons/second*. Such transforms include:

- Model-to-World transform: 10+ matrices can be loaded. This supports stitching, a type of skinning operation.
- World-to-Screen transform.
- View frustum clipping.
- Backface polygon rejection.

The XF can also perform many useful lighting and texture effects, including:

- One to eight lights. One light can be computed at a peak rate of 20M vertices/second. A light can be local diffuse or infinite specular. These calculations are computed per vertex.
- 2x4 or 3x4 texture transform matrices. For multi-texturing, ten or more 3x4 matrices can be loaded. One matrix transform will be performed at the peak rate of 20M vertices/second. Projected light and shadow textures can be implemented in this way.
- Bump map texture coordinates.

Figure 13 - Transform Processor (XF) blocks



3.4 Rasterizer (RAS)

The rasterizer can perform Z tests prior to texture mapping, which can help to increase the fill rate when textured objects are not visible. Only rendered pixels will load texture into the texture cache.

The GP runs at 162Mhz, so RAS can generate four pixels per clock to reach its 648M pixels/second peak performance. The four pixels are arranged in a square pattern, commonly referred to as a “pixel quad.”

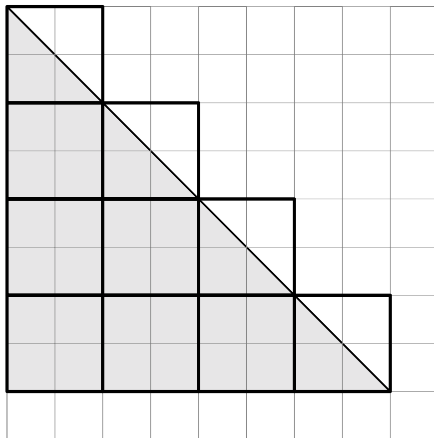
*. This peak rate assumes all polygons are triangles. It also assumes an average rate of one vertex per triangle.

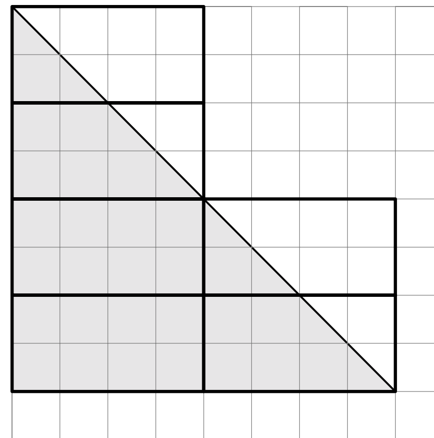
© 2000-2004 Nintendo

The performance of the pixel quad rasterization pattern is identical regardless of whether its rendering tall triangles or wide triangles. Nintendo GameCube has no preference; they are rendered at the same speed.

We are aware that, with the smaller polygons rendered in the current generation of game consoles, a bigger pixel “footprint” has nearly no performance increase over a pixel quad, and faster clock rates mean more real pixels can be drawn per second. However, we chose this pixel quad “sweet spot” for Nintendo GameCube in favor of a larger pixel “footprint” because the latter would dramatically increase the size and cost of the silicon chip.

Figure 14 - Pixel footprint analysis





10 cycles at 162Mhz = 62 ns
162Mhz = 6ns/cycle

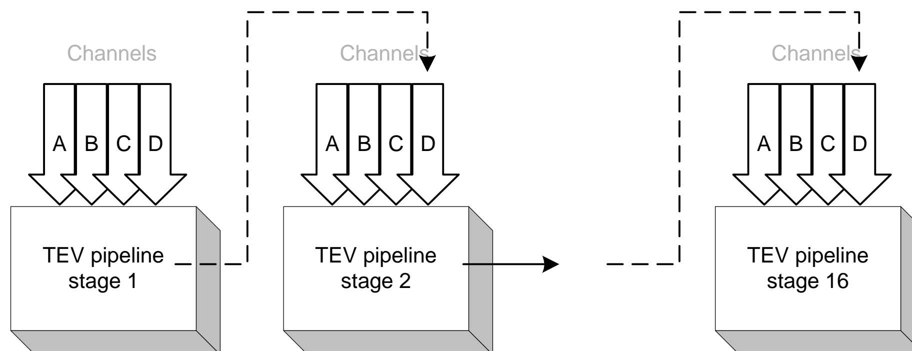
6 cycles at 150Mhz = 45ns
150Mhz = 7.5ns/cycle

The Nintendo GameCube rasterizer renders into the 1TSRAM frame buffer without risk of stalling; see "3.1.2 Embedded 1TSRAM memory" on page 13.

© 2000-2004 Nintendo

3.5 Texture Environment Processor (TEV)

Figure 15 - TEV pipeline stages



The TEV blends all of the polygon colors and texture colors together. You can enable up to 16 stages to provide blending for multiple textures. With only one blend stage, the system can perform at a peak rate of 648M pixels/second. Two blends can perform at a peak rate of 324M pixels/second, eight blends can perform at a peak rate of 81M pixels/second, and 16 blends can perform at peak rate of 50M pixels/second. Fill rates for up to eight TEV stages are shown in the table below:

Table 2 - TEV stage fill rates

Number of TEV Stages	Pixel Fill Rate
1	648M pixels/second
2	324M pixels/second
3	216M pixels/second
4	162M pixels/second
5	130M pixels/second
6	108M pixels/second
7	93M pixels/second
8	81M pixels/second

© 2000-2004 Nintendo

Each blend stage can perform the following equation and operations:

Equation 1 - Color blending

$R1 = A * (1 - C) + B * C$

$R2 = (D + sign * R1 + bias)$

$result = clamp(R2 * shiftfactor)$

- Linear interpolation (LERP). One coefficient, *C*, selects a blend between two others, *A* and *B*.
- Subtraction. *sign* can be +1 or –1. This allows equations such as (*D*-*A*).
- Signed 10-bit (-1024 to +1023) *D* input. Typically, it combines results from other blending stages.
- Additive brightening. *bias* can be 0 or ±0.5. This added offset brightens or darkens the result.
- Scaled brightening. *shiftfactor* can be 1, 2 or 4 to brighten by a multiply, or to scale *bias* up to 1.0.
- Clamping. The final result of each blend stage can be clamped to unsigned 8-bit or signed 10-bit (-1024 to +1023). No wrapping can occur.
- The TEV clamping modes allow per-pixel if/else evaluation.
- The final alpha output from all the TEV stages can be applied to a two-reference alpha compare circuit. The alpha compare pass/fail can conditionally write color and Z.

Note: *A*, *B*, *C*, and *D* can come from 14 possible sources.

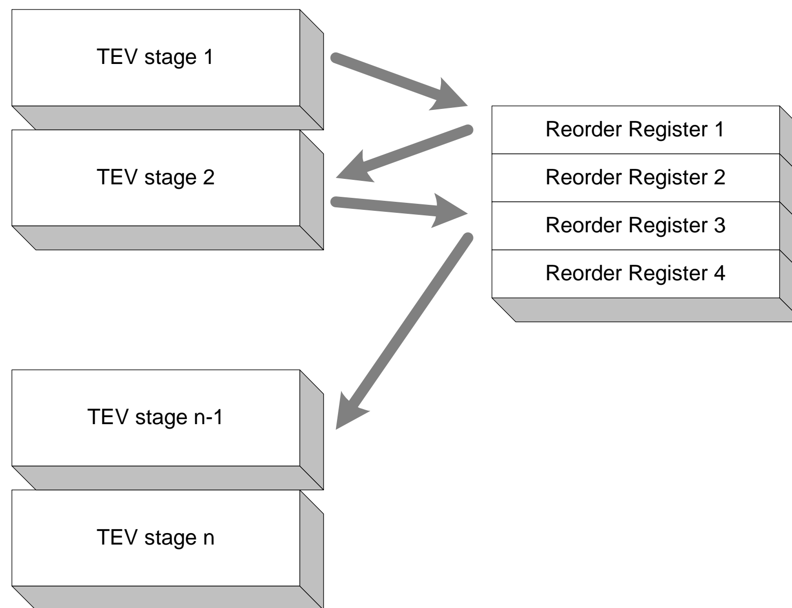
3.5.1 Re-ordered blending

The TEV can actually "reorder" blending sources to different blend stages. For example, texture 0 can be "sent" to the TEV15 stage for blending.

Note: However, this feature is not completely general and certain restrictions exist. Refer to the *Graphics Programmer's Guide* for details.

© 2000-2004 Nintendo

Figure 16 - TEV stage result reordering



After blending the stages, the TEV is responsible for calculating and blending fog. For reference, the N64 evaluated the fog equation per vertex and linearly interpolated between vertices. The result was unnatural fog on large polygons. By contrast, the Nintendo GameCube TEV computes the fog equation per quad (group of four pixels), which eliminates “bad fog” on large polygons.

3.6 Pixel Engine (PE)

The PE can perform blending and antialiasing. It can alpha-blend a pixel into the frame buffer in a variety of ways similar to those described in OpenGL and DX documents.

3.6.1 Antialiasing

The PE can also perform antialiasing by using a super-sampling technique. For each screen pixel, the PE renders color and Z data for three sub-pixels. After rendering the scene, the three sub-pixels stored in the frame buffer are averaged to compute the final image.

Super-sampling has excellent properties:

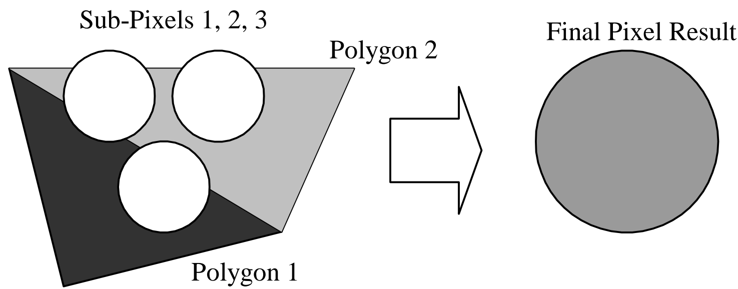
- No back-to-front polygon sorting necessary, which would be costly when handling a high number of polygons.
- No aliasing (“jaggies”) at the intersection of polygons.
- Better resolution definition than N64-style antialiasing.

However, this technique also has the following restrictions:

- Maximum fill rate is 324M pixels/second; therefore, antialiasing is free if you are using two or more TEV stages.
- Z-buffer precision is reduced from 24-bit to 16-bit. When Z resolution is 16 bits, you can use inverse floating point formats to maximize accuracy and range.
- Rendering resolutions greater than 640 x 240 will not fit into the EFB. Higher resolutions will require two passes.

© 2000-2004 Nintendo

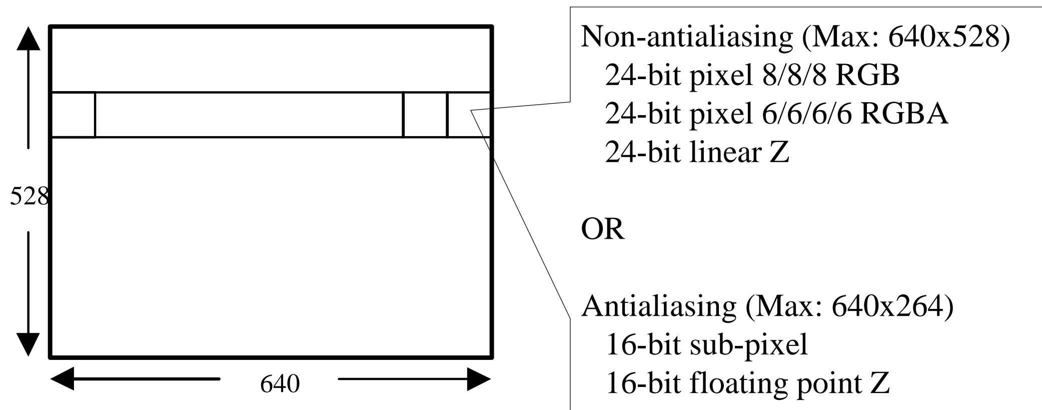
Figure 17 - Super-sampling antialiasing



3.7 Internal 1TSRAM memory buffers

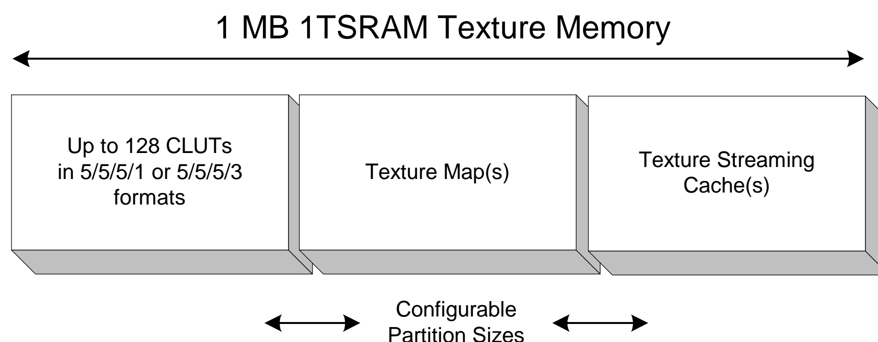
The embedded frame buffer (EFB) has enough bandwidth to blend four pixels/clock cycle at the peak fill rate of 648M pixels/second. The maximum EFB size is 640 x 528 x 24-bit color and 24-bit Z (528 lines are needed to support PAL for the European market). The EFB is singled-buffered and will transfer the finished image to the external frame buffer (XFB) for display. Any double-buffering occurs in main memory.

Figure 18 - Embedded frame buffer (EFB)



The Texture Processor contains 1MB of local texture memory (TMEM). Entire texture maps can be loaded explicitly into TMEM. Alternatively, textures can be kept in the external 1TSRAM main memory and a portion of the TMEM can be allocated for “texture caching.” Finally, up to 128 color lookup tables (CLUTs) can be loaded into the TMEM.

Figure 19 - Embedded texture memory (TMEM)



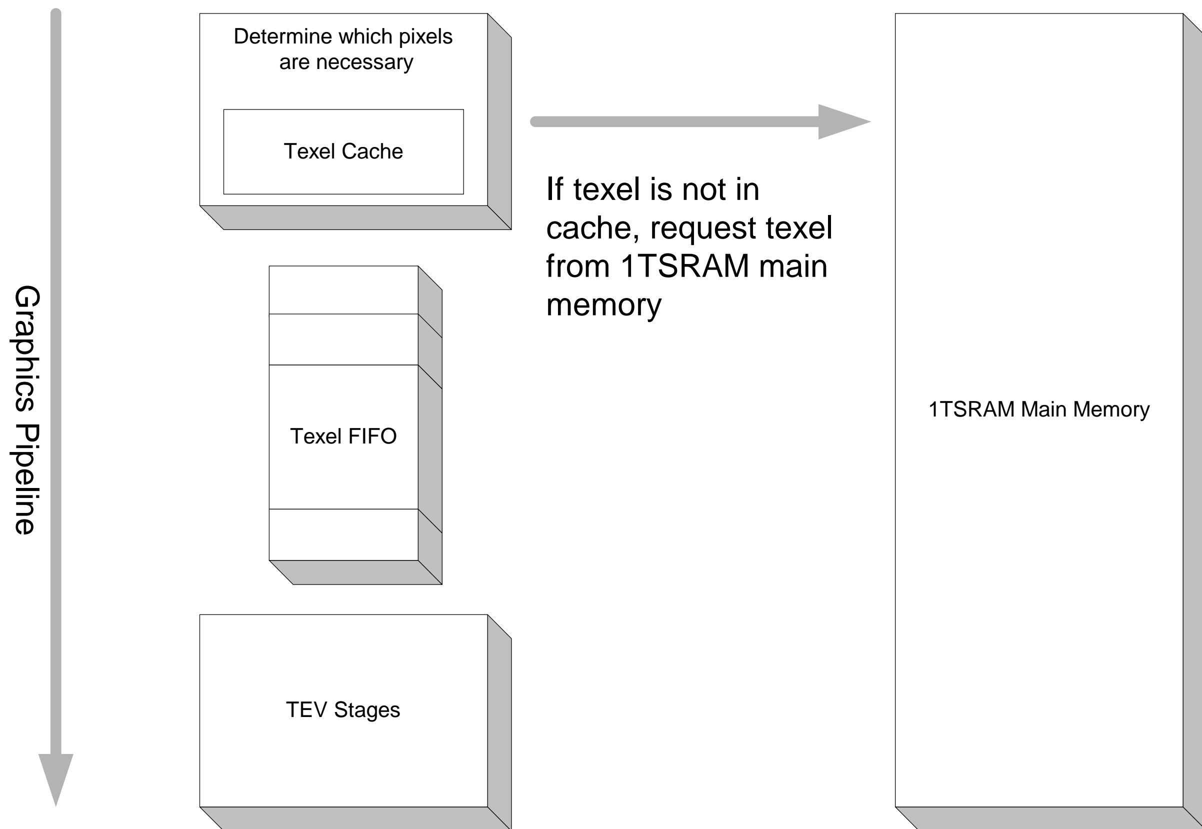
© 2000-2004 Nintendo

3.7.1 Texture streaming cache

Using a streaming cache eliminates any texture access stalls while rendering pixels.

- The most recently-used texels are cached.
- If not cached, the hardware prefetches texels much earlier than the TEV stages in which they will be used.

Figure 20 - Texture streaming cache



The texture streaming cache gives the game developer the possibility of using the entire 1TSRAM main memory as texture memory with *no* performance penalty to pixel rendering.

3.7.2 Preloaded texture map

You may want to load the texture maps of frequently-used textures into the TMEM to guarantee the best performance. Frequently-used texture maps, such as shadow and light textures, are good candidates. Texture maps with a higher texture cache miss rate, such as reflection maps, can be preloaded effectively as well.

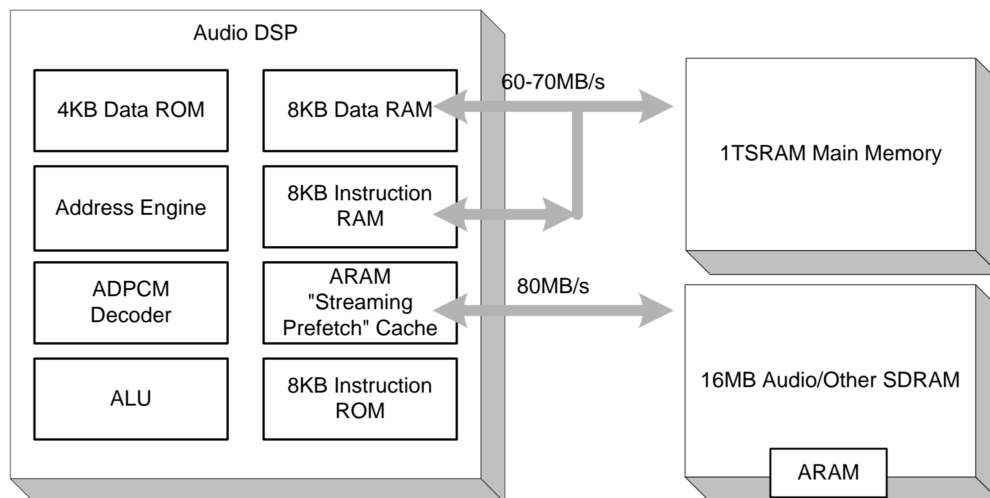
© 2000-2004 Nintendo

4 The audio DSP

The Nintendo GameCube audio hardware features a custom digital signal processor (DSP), which has the following characteristics.

- 81MHz instruction clock.
- 16-bit data words and addressing.
- 16-bit multiplier, 40-bit accumulator.
- Single cycle add, multiply, subtract, and MAC (multiply-accumulate).
- Single cycle load/store from local RAM.
- Hardware ADPCM decompression (reduces DSP workload by 30%).
- 8KB data RAM.
- 4KB data ROM.
- 8KB instruction RAM.
- 8KB instruction ROM.
- Dual-ported memory for simultaneous reads and writes.
- Parallel ALU and data load/store operations.
- DMA access to main memory.
- Cached interface to ARAM.
- “Mailbox” register interface with CPU.
- Hardware addressing engine for automatic data and instruction loops.

Figure 21 - Audio DSP blocks



© 2000-2004 Nintendo

The DSP relieves the Gekko CPU of the more onerous audio processing tasks. Specifically, the DSP is responsible for:

- ADPCM decompression (hardware accelerated).
- Sample rate conversion.
- Volume envelope articulation.
- Mixing (voices and effects).
- Per-voice filtering.
- Dolby Surround encoding.

4.1 Features and performance

The DSP can generate audio samples with the following specifications:

- Up to 64 stereo Surround voices.
- 32KHz mixing rate, 48KHz output.
- Two dedicated stereo Surround effects busses.

© 2000-2004 Nintendo

5 Auxiliary audio memory (ARAM)

The DSP's on-chip memory is supplemented by auxiliary RAM (ARAM) with the following characteristics:

- 16MB of internal DRAM.
- 8-bit data bus.
- DMA interface to main memory (60-70MB/second peak rate).
- Streaming cache interface to DSP (80MB/second peak rate).

Although ARAM is intended primarily for the storage of audio samples, developers may also place graphics and animation data in it. Such data can be “paged” into main memory with a latency of approximately one video frame, making it ideal for buffering transactions between the very slow optical disc drive and very fast main memory. Note, however, that DSP access to ARAM has priority over the main memory DMA.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

6 Optical disc drive

Nintendo GameCube uses optical read-only disc technology, which includes the following features:

- Large capacity (1,459,978,240 bytes).
- Constant Angular Velocity (CAV); i.e., the drive motor spins at the same speed at all times, regardless of optical disc laser position on an inner or outer track.
- ~2-3MB/second data transfer rate.

  Note: The transfer rate is different between inner and outer tracks because of CAV (inner tracks have lower transfer rates).
- Data cache in the disc drive. Speculative sequential prefetch algorithm minimizes seek time.
- Separate audio streaming port.

6.1 Speculative prefetch

After the optical disc drive has completed a requested transfer, the disc drive controller will speculatively retrieve data positioned sequentially after the requested transfer. Since a game will often transfer sequential groups of data, the speculative prefetch mechanism reduces the access latency for subsequent data transfer requests.

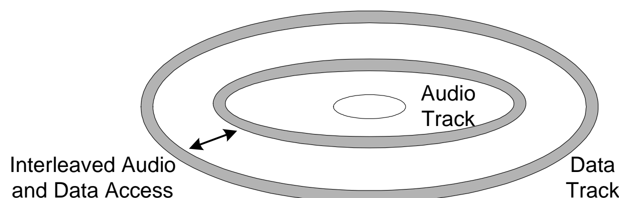
Figure 22 - Disc drive speculative prefetch

6.2 Interleaved data access and audio streaming

The optical disc drive can also stream 48KHz ADPCM audio data from the disc. This data is transferred directly into the audio interface (AI).

If simultaneous data transfer requests exist, the disc drive controller moves the optical disc laser reading mechanism intelligently between the data and audio data regions. Audio streaming is guaranteed to never skip; however, the data transfer rate will suffer depending on the distance between the two data regions.

Figure 23 - Interleaved data and audio



© 2000-2004 Nintendo

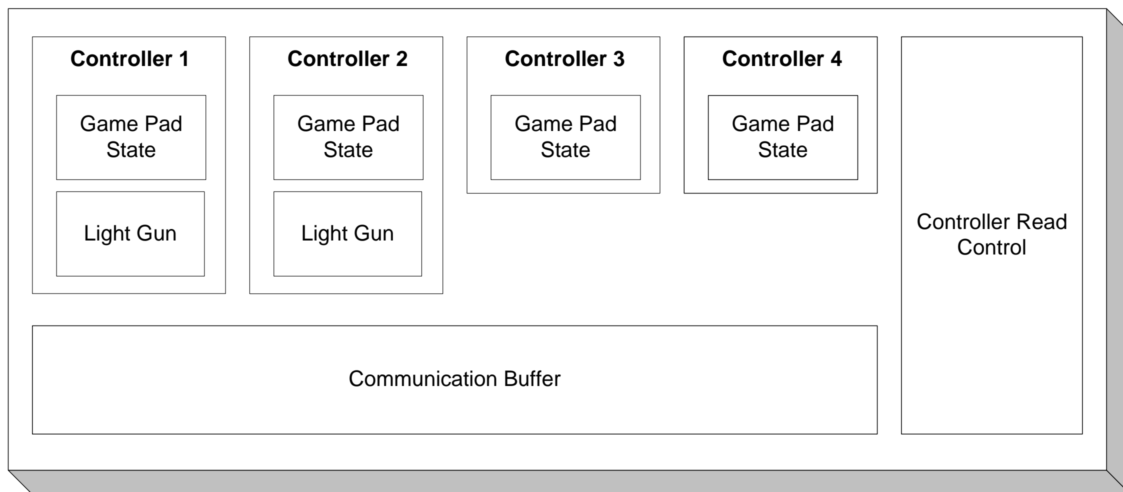
© 2000-2004 Nintendo

7 Controller (PAD)

The Controller interface (PAD) has the following features:

- Capacity to handle up to four Controllers.
- Data transfers to and from the Controller. Because the Controller wire is long, the CPU may need to perform a checksum and retry if data is corrupt.
- Programmable intervals to retrieve Controller state automatically; no CPU servicing necessary.
- Support for two light guns (“Zappers”).

Figure 24 - Game pad interface blocks



7.1 Game pad state sampling control

The hardware reads the Controller state automatically at fixed intervals. Game developers can program the interval in order to retrieve the current Controller state prior to computing game logic.

7.2 Communication buffer

This buffer transfers a block of data between main memory and memory-mapped devices, such as a memory pack, in the Controller. This transfer occurs over long Controller wires; therefore, the CPU has to perform a checksum for data integrity and may request a retry.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

8 Expansion Interface 0 (EXI0)

This is a serial interface port and has the following features:

- 1MHz, 2MHz, 4MHz, 8MHz, 16MHz selectable clock per device. Maximum transfer rate is 2 MB per second.
- Supports real time clock/calendar device.
- External expansion port.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

9 Expansion Interface 1 (EXI1)

This is a serial interface port and has the following features:

- 1 MHz, 2 MHz, 4 MHz, 8 MHz, 16 MHz selectable clock per device. Maximum transfer rate is 2 MB per second.
- External expansion port.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

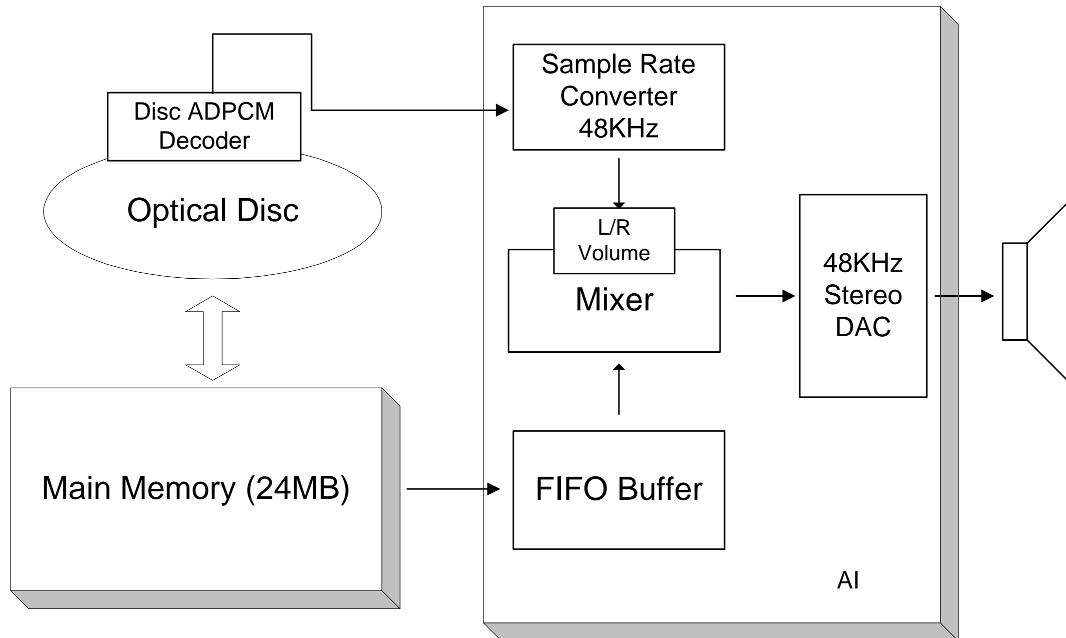
10 Audio Interface (AI)

The Audio Interface (AI) manages the transfer of audio data into the output digital-to-analog converter (DAC). The AI has the following inputs:

- Audio data (48KHz) from the optical disc drive.
- Audio data from a buffer in main memory (containing DSP output).

The AI mixes these two inputs together and generates 48KHz stereo samples for the output DAC.

Figure 25 - Audio Interface (AI) blocks



The AI provides the following features:

- Automatic conversion of 32KHz audio data from the optical disc drive into 48KHz samples.
- Left and right volume controls for audio data from the optical disc drive.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

11 Video Interface (VI)

The Video Interface (VI) has the following features:

- Full support for NTSC, M/PAL and PAL interlaced and non-interlaced modes.
- 480-line progressive digital television output (480P).

Table 3 - Video formats

Signal Type	Frame Buffer Size	Mode
NTSC, M/PAL	640x480	• Non-interlaced rendering (<= 30Hz). • Interlaced display.
NTSC, M/PAL	640x240	• Interlaced rendering (60Hz field rendering). • Interlaced display.
NTSC, M/PAL	640x240	• Non-interlaced rendering (60Hz). • Double-strike display (SNES style).
PAL	640x528	• Non-interlaced rendering (<= 25Hz). • Interlaced display.
PAL	640x264	• Interlaced rendering (50Hz field rendering). • Interlaced display.
PAL	640x264	• Non-interlaced rendering (50Hz). • Double-strike display (SNES style).
480P Digital Television	640x480	• 480-line progressive digital television. • Connect to component input and D Terminal of digital television.

The frame buffer displays from main memory using a YUV format to minimize main memory usage. The YUV color resolution is 16 bits/pixel (YUYV8888); for example, a single 640x480 frame buffer will consume 600KB.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

Software Development Kit Overview

SDK Version 20-APR-2004

Contents

Revision History .... II-iii
1 Goals of the Nintendo GameCube SDK II-1
2 Development hardware II-3
2.1 Programmer development system II-3
2.1.1 Goals II-3
2.1.2 Key features II-3
2.1.3 Optical disc emulation II-4
2.2 DDH/PC host communication interface II-4
3 SDK components II-5
4 Compiler and debugger suites II-7
5 Build environment II-9
6 Operating system II-11
6.1 Memory address map II-11
6.2 Execution model II-11
6.3 Utility functions II-12
6.4 Optical disc file system II-12
6.4.1 Random access comparison of optical disc drive to mask ROM II-13
7 Graphics II-15
7.1 Graphics library (GX) II-15
7.1.1 Drawing geometry II-15
7.1.2 Geometry processing control II-15
7.1.3 Texture application II-16
7.1.4 Other pixel operations II-16
7.1.5 Miscellaneous functions II-16
7.2 Matrix-Vector library (MTX) II-16
7.3 Demonstration library (DEMO) II-16
7.4 2D Graphics library (G2D) II-17
7.5 Character Pipeline (articulated animation set) II-17
7.5.1 Data extraction libraries and tools II-18
7.5.2 Runtime libraries II-20
8 Audio II-21
8.1 Audio and graphics game framework II-21
8.2 Factor5 MusyX sound system II-21
8.3 Sound sets II-22
8.3.1 Roland wavetable II-22

Code Examples

Code 1 - GX library functions II-15
Code 2 - 3D geometry conversion API II-18

Figures

Figure 1 - Dolphin Development Hardware (DDH) system blocks II-3
Figure 2 - DDH/PC host communication interface II-4
Figure 3 - Cached and uncached memory address map II-11
Figure 4 - Thread communication II-11
Figure 5 - Interrupt event callback handler II-12
Figure 6 - Character Pipeline data extraction path II-18

© 2000-2004 Nintendo

Figure 7 - Character Pipeline runtime libraries II-20
Figure 8 - Independent scheduling for CPU processing of audio and graphics II-21

Tables

Table 1 - SDK components II-5
Table 2 - Table 2 Utility libraries II-12
Table 3 - Optical disc and mask ROM comparison II-13

© 2000-2004 Nintendo

Revision History

Revision No.	Date Revised	Items (Chapter)	Description	Revised By
20-APR-2004	6/30/2003	8	Revised Figure 8 units	R. Daring
5-Sept-2002	9/5/2002	-	First release by Nintendo of America, Inc.	-

© 2000-2004 Nintendo

© 2000-2004 Nintendo

1 Goals of the Nintendo GameCube SDK

During development of the Nintendo GameCube console, we noted that game developers had many difficult tasks to accomplish in a short period of time in order to develop and release a game. Therefore, we designed the Nintendo GameCube SDK to provide useful, flexible software components and numerous examples, our goal being to help game developers familiarize themselves with the machine as quickly as possible.

We know that game developers must code and select all of the software necessary to make their games, so the Nintendo GameCube SDK is really just a list from which developers can choose the components and features that they find useful. Simple software components, designed specifically for video games, give game developers flexibility in deciding how to use various system resources.

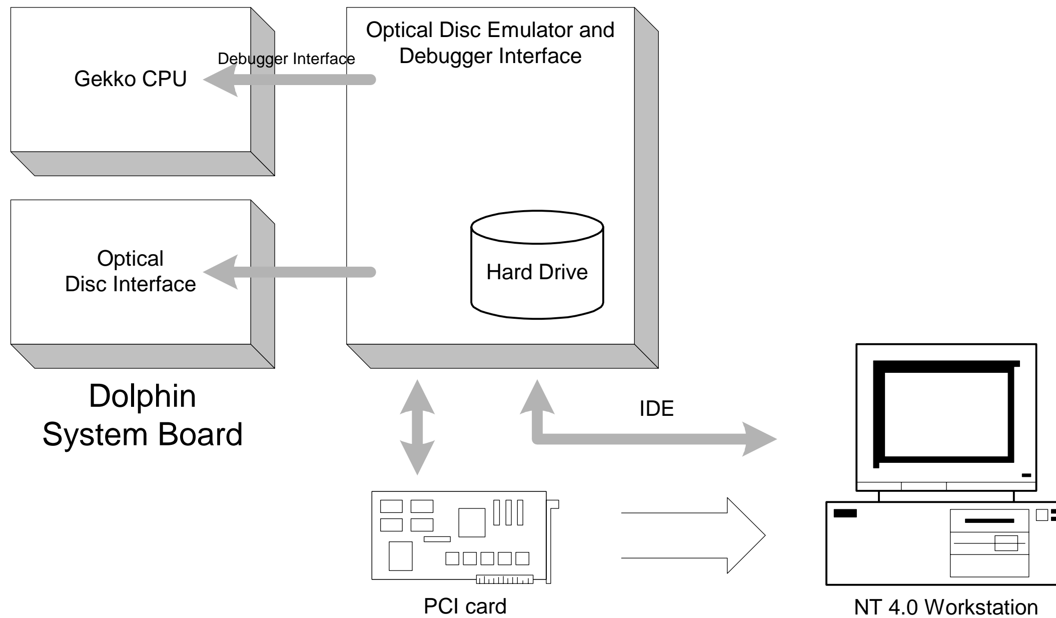
© 2000-2004 Nintendo

© 2000-2004 Nintendo

2 Development hardware

2.1 Programmer development system

Figure 1 - Dolphin Development Hardware (DDH) system blocks



2.1.1 Goals

The goals of the Dolphin Development Hardware (DDH) are:

- Accurate emulation of console hardware.
- Rapid program change/debug sessions.
- Extra memory to enhance capabilities of development tools.
- Reasonable cost so that each programmer can have a dedicated station.

2.1.2 Key features

Key features of the DDH include:

- Parity of all functional blocks between the development system and the final console.
- Optical disc drive performance emulation and file system emulation.
- Double main memory size (48MB).

Note: Some of these features may not be available on initial versions of the DDH.

© 2000-2004 Nintendo

2.1.3 Optical disc emulation

The optical disc emulation system provides the following features:

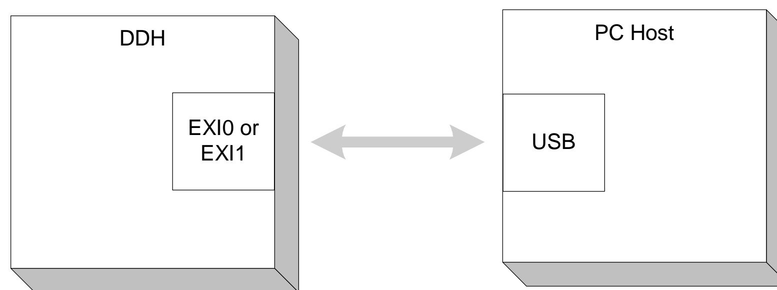
- The hard drive connects directly to the PC, which enables high speed updates of data files and program files to the game's data optical disc data sets.
- Emulation of correct seek and transfer timing, using disc geography information supplied by the developer to define file location on the optical disc.
- Translation of Windows FAT symbolic file system to Nintendo GameCube optical disc file system by generating a symbol table describing the directory and file hierarchy.
- Error emulation (e.g., "scratched disc" messages).

Note: Some of these features may not be available on the initial versions of the DDH.

2.2 DDH/PC host communication interface

The DDH can facilitate game development through its ability to communicate directly with the PC. We provide a software library to access the hardware interface linking the DDH's EXI0 or EXI1 interface with the PC's USB interface.

Figure 2 - DDH/PC host communication interface



© 2000-2004 Nintendo

3 SDK components

Table 1 - SDK components

Component Type	Components
Compiler Suite	Metrowerks CodeWarrior compiler/debugger suite SN System compiler/debugger suite
Operating System	Nintendo GameCube operating system Optical disc file system Controller (PAD) library
Graphics	3D graphics library Matrix library 2D graphics library Video display library Demonstration library Texture conversion tools
Audio	Wavetable construction tool sound effects design tool Wavetable and sound effects synthesis library Optical disc streaming library

© 2000-2004 Nintendo

© 2000-2004 Nintendo

4 Compiler and debugger suites

Nintendo GameCube has two compiler/debugger suites from which to choose: Metrowerks CodeWarrior and SN Systems ProDG. Both suites include the following components:

- C/C++ language compiler.
- PPC assembler, including the new Gekko CPU instruction set.
- Standard C libraries.
- Debugger.
- Command line compiler for the Makefile build environment.
- IDE build environment.

© 2000-2004 Nintendo

© 2000-2004 Nintendo

5 Build environment

The Nintendo GameCube SDK ships with many demonstration programs to show how to use different library components. These demos all use the Makefile build environment from the Cygnus Cygwin UNIX-style command shell and make environment. We supply the freeware Cygwin package on the SDK CD-ROM. You can also download the latest Cygwin distribution from the following URL:

http://sources.redhat.com/cygwin/

For more details, see “Build System” in the *Nintendo GameCube Programmer’s Guide*.

© 2000-2004 Nintendo

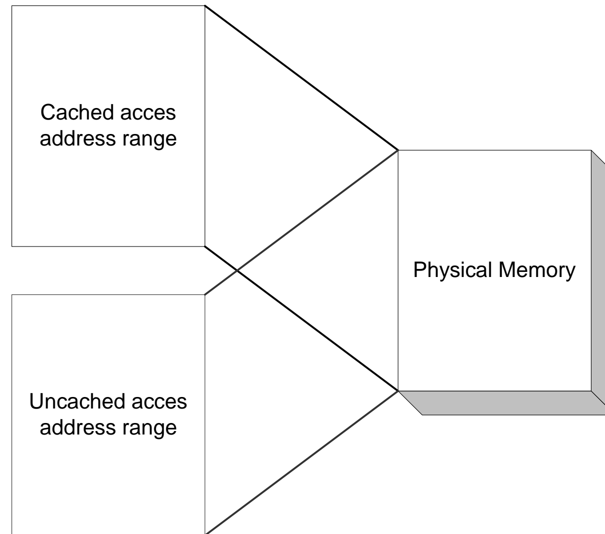
© 2000-2004 Nintendo

6 Operating system

6.1 Memory address map

During the execution of a game, we need to switch quickly between cached and uncached access. The operating system accesses most code and data through cached addresses for speed, but it must access hardware registers as well. For this reason, we use a memory address map similar to the Nintendo 64 (N64) MIPS memory map. There are separate memory address ranges for cached access and uncached access.

Figure 3 - Cached and uncached memory address map

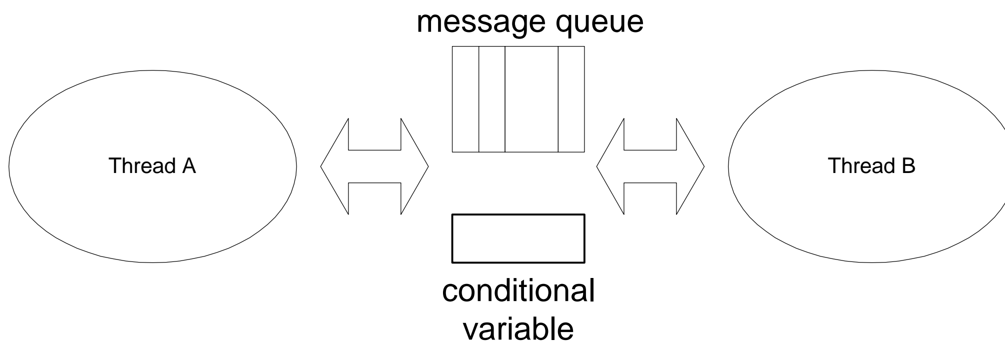


6.2 Execution model

Nintendo GameCube supports threads and interrupt event callback handlers. Game developers can choose the most suitable and familiar execution models for specific games.

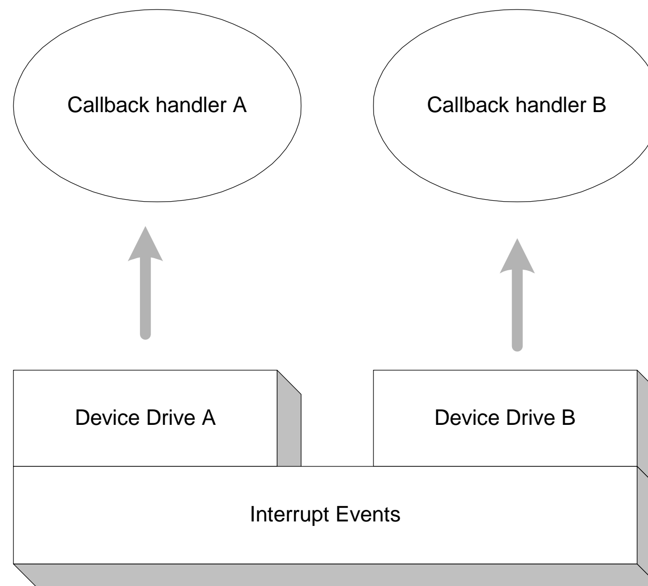
Threads have message queues and conditional variable synchronization functions. Using threads can cause critical sections in reentrant code. The Nintendo GameCube OS provides mutual execution functions to protect these critical code sections.

Figure 4 - Thread communication



© 2000-2004 Nintendo

Figure 5 - Interrupt event callback handler



6.3 Utility functions

We provide the following utility functions for game development:

Table 2 - Table 2 Utility libraries

Utility Library	Purpose
Memory allocation	Basic multiple heap memory allocation library.
Standard C libraries	Provided by compiler suite. Handle math, string, buffer file IO, …
Stopwatch, alarms	Stopwatch handles elapsed-time capture, accurate to 50Mhz resolution. Alarms provide countdown timer functionality.
Dynamic linking	Supports dynamic relocatable modules.

6.4 Optical disc file system

The SDK provides file system access for the large-capacity optical disc drive. This file system includes the following feature set:

- Symbolic character-string names for files, directories, and paths.
- Variably-sized files.
- Non-blocking and blocking access methods (i.e., asynchronous and synchronous access, respectively).

© 2000-2004 Nintendo

6.4.1 Random access comparison of optical disc drive to mask ROM

On the N64, many game developers randomly-accessed the mask ROM to load sound and animation data into the main memory. The latency in this case was negligible, making it possible on the N64 to load tens of thousands of files per second. The Nintendo GameCube optical disc drive has a random-access latency of over 100 milliseconds (i.e., over six 60Hz frames). This means it will be impossible to load more than 10 non-sequential files per second on Nintendo GameCube.

Therefore, we suggest that developers used to N64 programming take time to consider the critical issue of disc access time. In order to minimize random-access disc seeks, it is very important that you plan out how your game will access data. In particular, merging data into large blocks is essential.

The Nintendo GameCube optical disc emulation system provides tools to specify disc file placement; however, these tools cannot solve every problem. If a large number of files need to be loaded—and the files are not contiguous on the disc—the system will take an unacceptably long time to load the files. Therefore, the developer should plan early in the game design process to utilize the massive capacity of the optical disc while minimizing the effect of the longer seek time.

Table 3 - Optical disc and mask ROM comparison

Feature	Optical Disc	Mask ROM
Random access	100 milliseconds 1/10th of a second	Few microseconds
Capacity	Very Large	8MB

© 2000-2004 Nintendo

© 2000-2004 Nintendo

7 Graphics

7.1 Graphics library (GX)

The Nintendo GameCube Graphics library (GX) has functions to render geometry with many attributes. The GX library's main purpose is to provide the logical API that game engines need to perform rendering, and it is implemented as a thin layer of code above the hardware to ensure highest performance. If you are familiar with OpenGL, you will find that the GX library is similar to it in many respects.

7.1.1 Drawing geometry

The GX library has two main methods for drawing geometry. The first method is very much like OpenGL immediate mode, with functions that look like this:

Code 1 - GX library functions

```
GXBeginTriangle();
GXPosition(x, y, z);
GXColor(r, g, b, a);
…
```

This immediate mode API is ideal when the CPU must synthesize geometry data from a higher-level description (e.g., a height field or Bezier patch). These calls are inline functions and the compiler performs excellent optimization.

The second method sends geometry directly to the Graphics Processor (GP) by way of a memory-resident display list format. This method offers superior performance for non-animated data.

The immediate mode API and the Display List (DL) format both support configurable vertex representations, which in turn support:

- Direct or indexed vertex components. Vertex components (position, normal, color, and each texture coordinate) may all be indexed from arrays independently, or placed in the memory stream directly.
- Each vertex component can have a different-sized representation and precision. The available direct types are: **8-bit signed and unsigned integer**, **16-bit signed and unsigned integer**, and **32-bit floating point**. A scale is available to position the decimal point for the integer types. The indirect types **8-bit index** or **16-bit index** can be used to index into an array of any of the direct types.

This flexible representation allows the game developer to organize vertex data in a way that is appropriate for specific games. The ability to index each component separately eliminates a great deal of data duplication.

7.1.2 Geometry processing control

For geometry processing, the GX library contains functions to define and/or set the following:

- Local lighting.
- Modelview matrices and the projection matrix.
- Backface and view-frustum clipping.
- Viewport.
- Texture coordinate projection mappings.
- Reflection mapping.
- Bump maps.

© 2000-2004 Nintendo

7.1.3 Texture application

For the application of textures, the GX library contains functions for these operations:

- Define texture objects (bitmap location, wrap and mirror parameters, filter type).
- Load a Color Look-Up Table (CLUT).
- Configure Graphics Processor (GP) TMEM texture caches.
- Set multiple texture combine operators (TEV).

7.1.4 Other pixel operations

The GX library supports many pixel operations, including:

- Antialiasing.
- Z buffer control.
- Blending.
- Fog.

7.1.5 Miscellaneous functions

The GX library also contains functions to provide:

- CPU-to-GP FIFO control.
- Performance counters.

7.2 Matrix-Vector library (MTX)

The Nintendo GameCube SDK provides a Matrix-Vector library (MTX) to perform common matrix operations. It can:

- Construct matrices by common parameters (translation, rotation, scale, quaternion).
- Perform typical 3D vector operations (transformation, dot product, cross product, normalization).
- Build forward and inverse matrix stacks.

The Nintendo GameCube SDK includes the source code to this library.

7.3 Demonstration library (DEMO)

The Demonstration library (DEMO) performs simple configuration of system resources, which sometimes requires the use of functions from several different libraries. For example, one of the main functions of the DEMO library is to set video display resolution. To set a simple video resolution configuration, we must:

- Allocate the correct size of frame buffer to match the video resolution.
- Set graphics viewports to the correct size.
- Set video mode correctly.

The Nintendo GameCube SDK includes the source code to this library.

© 2000-2004 Nintendo

7.4 2D Graphics library (G2D)

The 2D Graphics library (G2D) supports:

- Multiple image layers.
- Image formatting with tiles (like the SNES-type of 2D consoles).
- 2D rotation of images.
- Bitmap tile sorting for fast display (i.e., efficient use of the texture cache).
- Viewport clipping for fast display.

The Nintendo GameCube SDK includes the source code to this library.

7.5 Character Pipeline (articulated animation set)

Effective with the 4/3/2001 release of the Nintendo GameCube SDK, the Nintendo GameCube Character Pipeline (CP) was separated into a distinct SDK with its own release schedule. The CP SDK provides a set of example libraries and tools to perform articulated animation. These libraries and tools are intended as a guide for developers who wish to learn the basic feature set of the hardware. By no means do they represent a complete solution for game development. These tools and libraries perform the following functions:

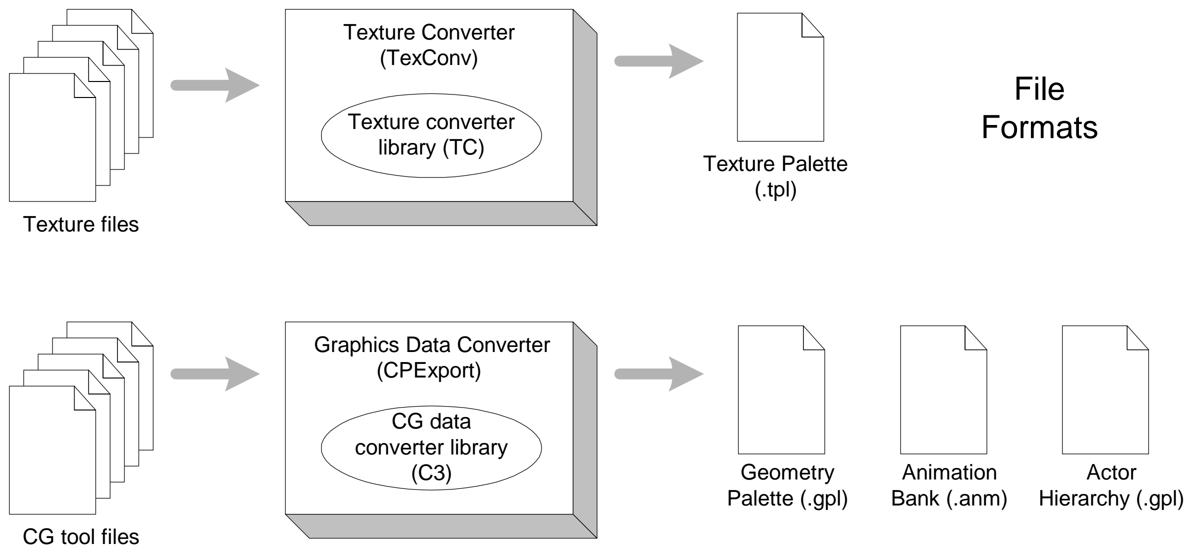
- Extract texture for the graphics hardware (using `TexConv`/TC).
- Extract geometry for the graphics hardware (using `CPExport`/C3).
- Extract hierarchy and animation tracks for character animation (using `CPExport`/C3).
- Display geometry and texture using the GX library.
- Animate characters.

For easy reference, we call this suite of tools and libraries the “Character Pipeline.”

© 2000-2004 Nintendo

7.5.1 Data extraction libraries and tools

Figure 6 - Character Pipeline data extraction path



During development of the Nintendo GameCube system, we observed that game developers use many different CG tools to make games. Because computer graphics tool vendors release new suites fairly frequently, we were faced with the daunting proposition of trying to provide a ready-made converter for every CG tool available. Of course this was impracticable, so we resolved instead to provide game developers with an easy method for making their own converters. As a result, the Nintendo GameCube SDK provides both converter tools *and* a converter library:

- **TC** – Texture extraction and conversion library.
- **TexConv –** Texture converter, which uses the TC library for texture conversion.
- **C3** – 3D geometry and animation data extraction and conversion library.
- **CPExport –** Converter for 3D Studio MAX and Maya, which uses the C3 library for conversion.

Both of the libraries are very easy to use and have simple APIs to load data. Developers can use a CG tool's data extraction API and then insert this data into these libraries to generate a Nintendo GameCube output format (GPL, TPL, ACT, ANM). Here is an example of the C3 API:

Code 2 - 3D geometry conversion API

```
C3BeginPolyPrimitive();
C3BeginVertex();
C3SetPosition(x, y, z);
C3SetColor(r, g, b, a);
```

7.5.1.1 Texture conversion tool (TexConv/TC)

The Graphics Processor can support:

- Trilinearly-filtered mipmaps.
- S3TC-compressed textures.

© 2000-2004 Nintendo

The GP requires that the texture image be organized in a tiled format with correct alignment and padding of image data. The CP SDK provides a texture converter tool and library to convert textures to Graphics Processor formats. The texture converter tool is called `TexConv`, the texture converter library is called TC. `TexConv` and TC both use the TGA (`.tga`) file format. The TGA format can support RGBA, monochrome intensity, color-indexed, and CLUT data, so it is very useful for games.

The CP SDK includes the source code to `TexConv` and the TC library.

(1) TexConv and TC library data optimization

`TexConv` and TC performs the following operations on textures:

- Tiling and padding required to prepare data for graphics hardware.
- S3TC texture format encoding.
- Mipmap generation.

7.5.1.2 Geometry conversion tool (CPExport/C3)

The CP SDK provides two versions of `CPExport`, one for 3D Studio MAX and the other for Maya. The `CPExport` plug-ins convert 3D Studio MAX and Maya data sets to the Nintendo GameCube geometry format. The SDK also provides the C3 converter library, which can be used to build other CG tool converters.

The Nintendo GameCube SDK includes the source code to both the `CPExport` plug-in and the C3 library.

(1) CPExport and C3 library data optimization

The C3 library performs many data optimization features. Here is a short list of speed and memory optimizations:

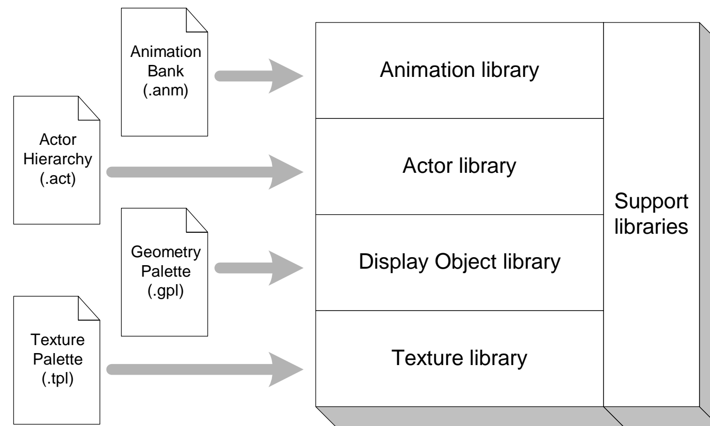
- Triangle-strip creation.
- Elimination of common vertex component data using vertex component indexing.
- Quantization of floating point to fixed-point numbers to reduce storage requirements; e.g., quantization on position, color, normal, and texture coordinate data.
- Minimized matrix loads for stitched character skin animation.

© 2000-2004 Nintendo

7.5.2 Runtime libraries

The Character Pipeline includes many runtime libraries to support animated character display. These libraries load the character animation file formats and handle the hierarchical animation of a game character.

Figure 7 - Character Pipeline runtime libraries



7.5.2.1 Display Object library (GPL)

The GEOPalette library handles the loading, 3D-space manipulation, and display of a static graphical object. The goal of this library is to enable artists to construct any object they would like to display. The GEOPalette library supports:

- Different textures for different groups of faces.
- Different shading (flat, Gouraud) for different groups of faces.
- Different polygon display optimizations (triangle strips, quads).

7.5.2.2 Actor library (ACT)

The ACT library handles the loading and display of a hierarchical 3D actor. (3D hierarchy is sometimes referred to as the skeleton.)

7.5.2.3 Animation library (ANM)

The Animation library handles the loading and sequencing of animation for an actor. The ANM library will support:

- Multiple sequences (e.g., “walk,” “run,” “jump”).
- Keyframing and a variety of interpolation methods, including: Tension/Continuity/Bias (TCB), Bezier (smooth), Linear, and Quaternion SLERP.

© 2000-2004 Nintendo

8 Audio

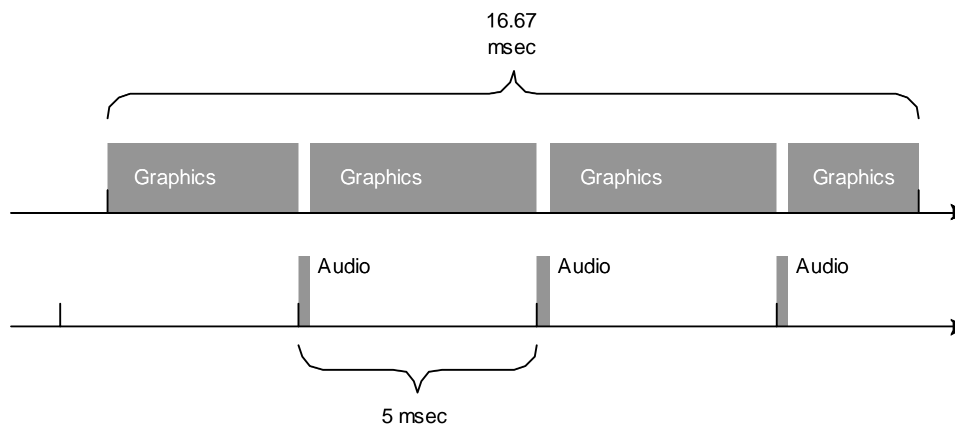
8.1 Audio and graphics game framework

Nintendo GameCube has a dedicated DSP for audio synthesis. Developers used to N64 programming may be glad to learn that they no longer have to write the DSP scheduler to manage time-sharing of the DSP processor between graphics and audio.

Note, however, that the Nintendo GameCube CPU is shared between graphics and audio. The CPU is interrupted every five milliseconds so that the audio library can generate commands for the DSP.

Audio processing is not synchronized to graphics processing, so there should be no difference in sound between NTSC and PAL formats.

Figure 8 - Independent scheduling for CPU processing of audio and graphics



8.2 Factor5 MusyX sound system

Nintendo GameCube uses Factor 5's MusyX sound system with specific enhancements for this console. You can learn more about MusyX on Factor 5's web site.

The MusyX sound system has these features:

- Graphical user interface tool to construct wavetables and preview MIDI scores.
- Runtime API library for interactive playback.
- Wavetable synthesis with ADPCM-compressed wavetables (i.e., no ADPCM-frame restrictions on loops.)
- 3D audio.
- Dolby Surround Sound.
- Reverberation.
- Macro language enabling the musician to generate sounds procedurally.
- DLS wavetable input format which allows use of the DLS standard to provide data exchange between many wavetable construction/editing tools.

© 2000-2004 Nintendo

The synthesizer has the following quality and performance characteristics:

- Wavetable synthesis parameters that update every 1 millisecond.
- 64 channels.
- Multiple auxiliary busses (effects channels).
- Output sample rate at 32kHz Dolby surround encoded.

8.3 Sound sets

MusyX will ship two sounds sets:

- General MIDI wavetable from Roland's Sound Canvas product.
- Factor5's wavetable.

8.3.1 Roland wavetable

The Roland General MIDI wavetable in DLS format can be imported into MusyX tools. This wavetable set features samples at 22.050kHz, 226 instruments, and eight drum sets. Both 8-bit and 16-bit wavetables are supplied with MusyX.

© 2000-2004 Nintendo