
DESIGNING AND PROGRAMMING THE EMOTION ENGINE

EMOTION SYNTHESIS REQUIRES A HUGE AMOUNT OF CALCULATION FOR THE PHYSICAL AND THOUGHT SIMULATION NEEDED TO GENERATE HUMAN EMOTIONAL REPRESENTATION WITH REALISTIC COMPUTER GRAPHICS OUTPUT. TO ACHIEVE THIS, THE EMOTION ENGINE MAIN PROCESSOR CONTAINS THREE INDEPENDENT FLOATING-POINT PROCESSORS.

Masaaki Oka
Masakazu Suzuki
Sony Computer
Entertainment

..... A home entertainment computer, the so-called consumer video game, requires perfect compatibility in its parts. The console cannot change its functions or performance during its lifetime. As all CD disks can be played on all CD players, all application titles must execute on all versions of the console. For this tight compatibility, no specification change, including frequency improvement, is allowed. Considering this, we have developed a new entertainment computer system designed to provide the highest performance. We adopted the latest technology and the most advanced manufacturing technology in the early design stages to secure a long product life with the same high performance.

High-quality computer graphics and good-quality entertainment software require a huge amount of calculation to provide attractive graphics and to simulate thinking inference and physical phenomenon. To produce these features, the new product needs ample computing resources.

One of the most advanced manufacturing technologies necessary for improving performance in computer graphics is the embedded DRAM, which is equipped with both an operation circuit and memory. EDRAM tech-

nology enables a very wide memory bandwidth such as 2,048 bits in one die. By inserting the EDRAM into the rendering engine, we greatly reduced the demand for a high bandwidth between memory and processor. This eliminates a bottleneck with the pixel fill rate, which has been a problem with the present rendering engine, and improves the drawing performance dynamically.

Then the geometry process performance became relatively insufficient because of the improved drawing performance. To reinforce geometry processing and distribute the load, we designed the architecture differently from the previous version. Our new architecture runs geometry engines in parallel and allows two or more processors to share the same rendering engine through timesharing. This approach sets the rendering engines in parallel. Here, we describe the architecture of our system, the programming paradigm, and a programming example on this processor.

Architecture overview

Our system consists of a graphics synthesizer, the Emotion Engine, an I/O processor (IOP), and a sound-processing unit (SPU). Figure 1 shows the system's block diagram.

The IOP controls peripheral devices such as a controller and a disk drive and detects controller input, which is sent to the Emotion Engine. According to this signal, the Emotion Engine updates the internal virtual world of the game program within the video frame rate. Many physical equations need to be solved to determine the behavior of the characters in the game world. After this is determined, the calculated object position is transformed according to the viewpoint, and a drawing command sequence (display list) is generated. When the graphics synthesizer receives the display list, it draws the primitive shape based on connected triangles on the frame buffer. The contents of the frame buffer are then con-

verted from digital to analog, and the final video image appears on a TV screen.

Figure 2 shows the block diagram of the

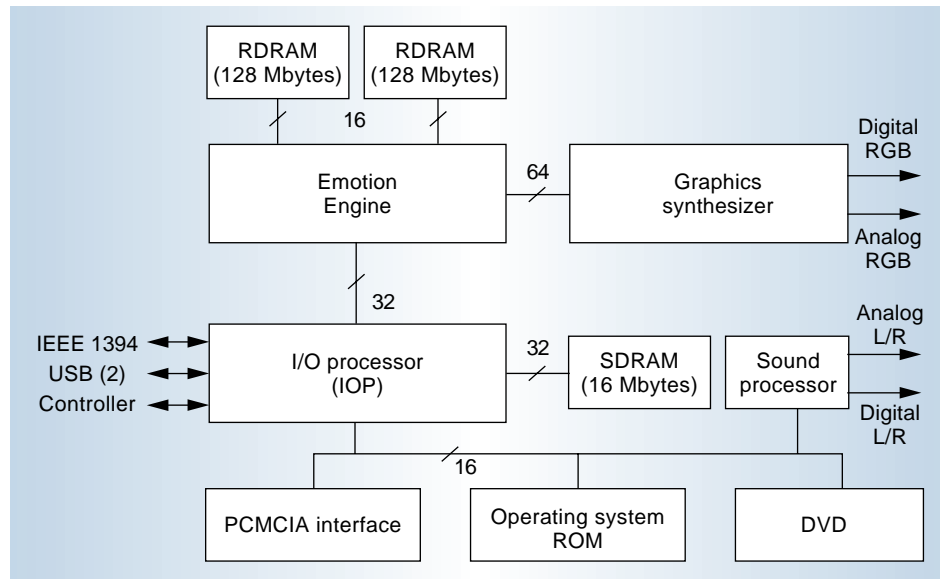


Figure 1. System block diagram.

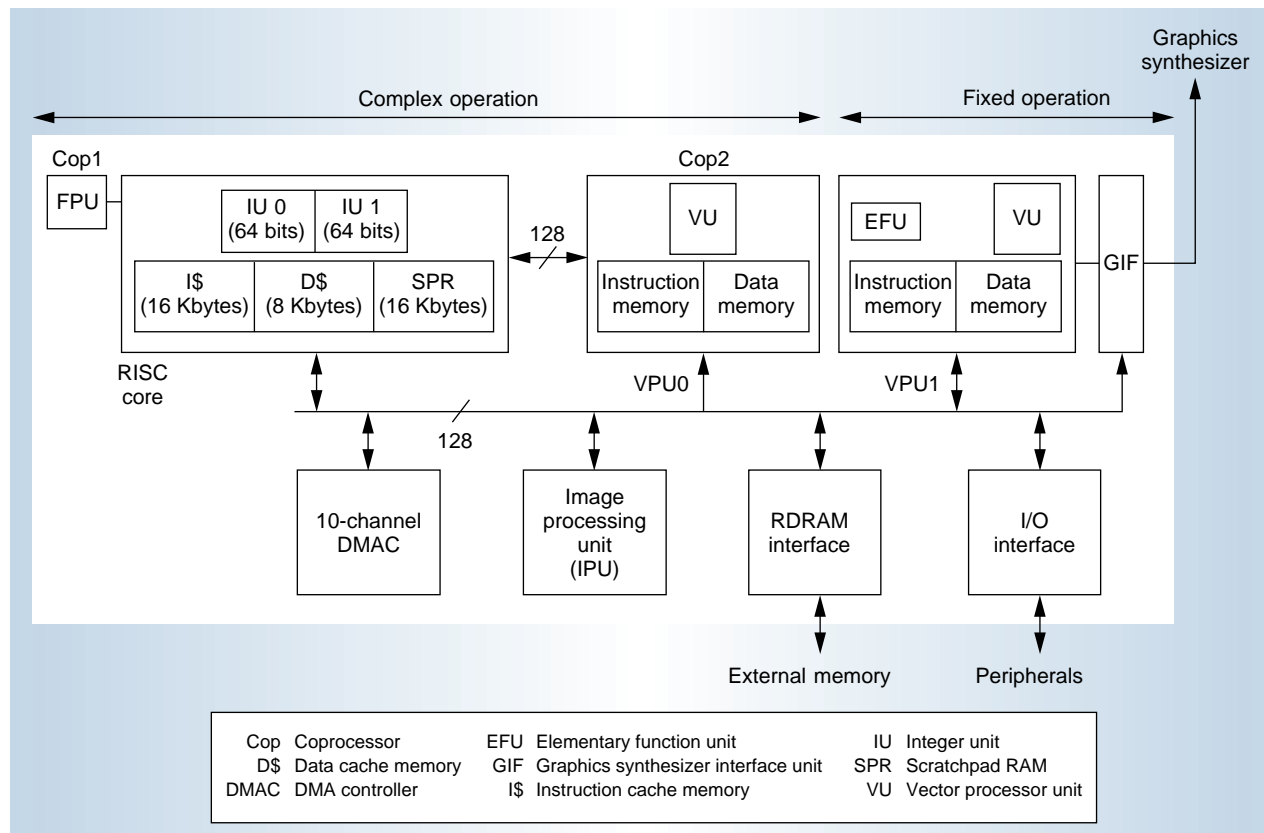


Figure 2. Processor organization.

Emotion Engine, the main processor. It consists of three independent processors. One is a traditional MIPS-based processor with a

Table 1. Main features of the Emotion Engine.
FMAC: floating-point multiply-adder-calculator;
FDIV: floating-point divider.

Feature	Description
CPU	MIPS III two-issue superscalar with 128-bit multimedia extension
Instruction cache	16 Kbytes, two-way
Data cache	8 Kbytes, two-way
Scratchpad RAM (SPR)	16 Kbytes
Data bus	128 bits
FPU	1 FMAC + 1 FDIV
VPU	Five-way, 64-bit VLIW
VPU0	4 FMACs + 1 FDIV
VPU1	4 FMACs + 1 FDIV
Elementary function unit	1 FMAC + 1 FDIV
IPU	MPEG-2 decode accelerator
DMA controller (DMAC)	10 channels
System clock	250 MHz

floating-point coprocessor, which is the overall controller. The others are floating-point vector processors: VPU0 and VPU1. VPU0 processes thought simulation and physical simulation while VPU1 supports fixed geometry operations such as simple 4×4 -matrix operation and perspective correction. For local parallelism, each VPU has a SIMD-VLIW architecture. In addition, the Emotion Engine includes an image processing unit (IPU) for real-time image data decompression to save the main memory. Table 1 lists the basic features of the Emotion Engine.^{1,2}

Figure 3 shows the VPU1 block diagram. VPU0 has the same microarchitecture as the VPU1 except for the coprocessor path and the memory sizes. The data memory and the floating-point registers have a 128-bit width interface with the 128-bit registers divided into four 32-bit fields (x,y,z,w). Four floating-point multiply-adder-calculators (FMACs) are applied to each field respectively. The throughput of an FMAC is one clock cycle. This means the VPU can execute

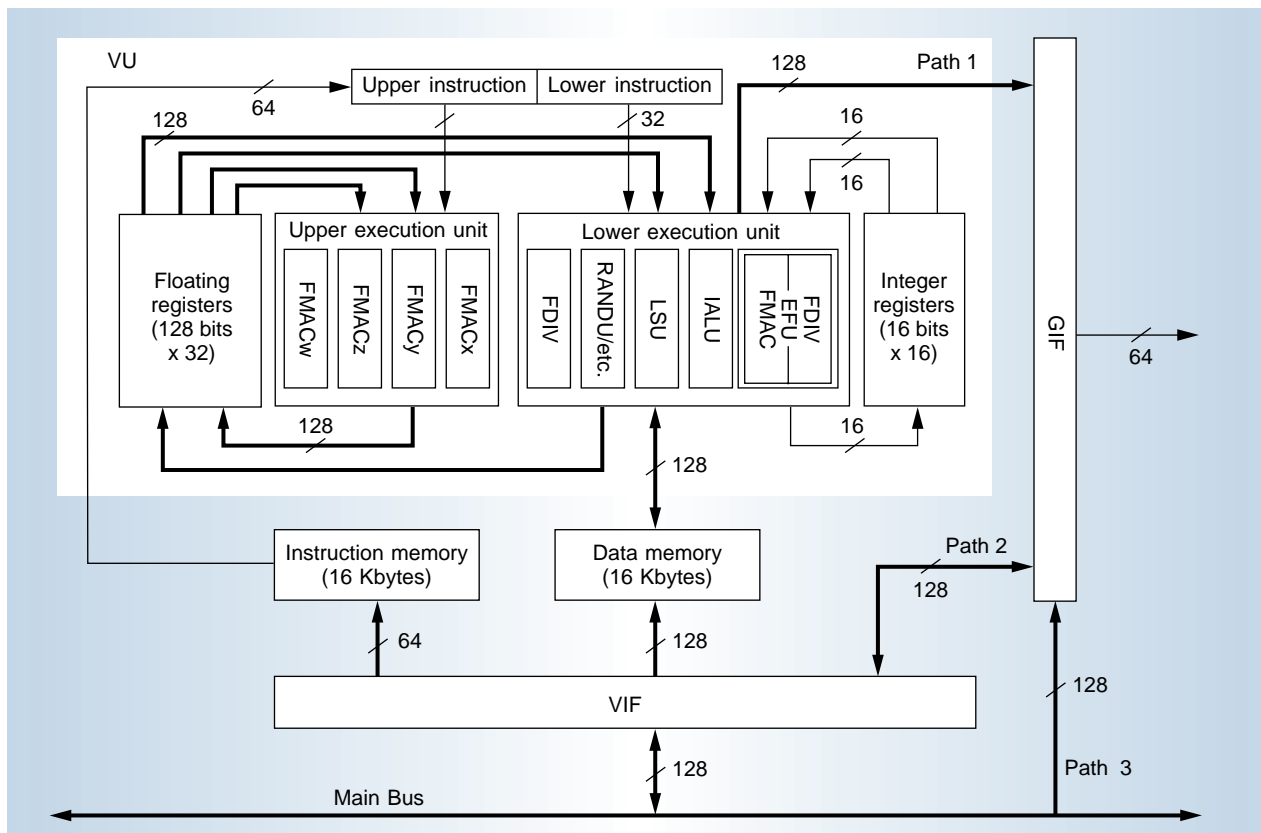


Figure 3. VPU1 block diagram.

the following vector calculation in one cycle:

$$\begin{aligned}x_3 &= x_0 \times x_1 + x_2 \\y_3 &= y_0 \times y_1 + y_2 \\z_3 &= z_0 \times z_1 + z_2 \\w_3 &= w_0 \times w_1 + w_2\end{aligned}$$

This type of equation frequently appears in computer graphics, physical simulation, and other vector calculations.

The instruction memory is 64 bits wide; one instruction consists of 32-bit upper and lower fields that execute on the same clock cycle. The upper field controls the FMACs, and the lower controls the load-store functions as well as the divider. Figure 4 shows the instruction format of the VPU, and Table 2 lists basic computer graphics performance using VPU0 and VPU1.

VPU programming paradigm

Figure 5 (next page) shows the VPU1 macro data flow. Memory access cycles are hidden under the floating-point operation cycles to achieve effective performance. The basic concept of the data flow consists of

- 1. loading source data from the large main memory to local data memory,
- 2. reading from the local data memory,
- 3. calculating the data,
- 4. storing the result back to the local data memory, and
- 5. sending the result to the graphics synthesizer.

The local memory is quad-buffered, pipelining the execution of each step. For this requirement the local memory has a logical triple port for the processor, direct memory access between main memory and the local memory, and direct memory access between the local memory and the graphics synthesizer interface unit (GIF).

PlayStation2

On 13 September in Japan, Sony Computer Entertainment (SCE) disclosed the next-generation game console called PlayStation2. PS2 is the successor to the original PlayStation, 50 million sets of which have been sold worldwide.

PS2 replaces the CD-ROM drive in PlayStation1 with a DVD-ROM drive that can play DVD videodiscs. PS2 also contains the whole circuit of the PS1 micro-processor core as an I/O controller that enables PS1 compatibility. Furthermore, PS2 supports some general I/O formats such as the IEEE 1394 (iLink) and Universal Serial Bus for future extension, especially for the future network extension planned for 2001 availability.

The Emotion Engine described in this article is the central processor in the PS2 system and is designed for real-time entertainment. The PlayStation's target application is entertainment, not office applications.



Figure A. PS2 outlook.

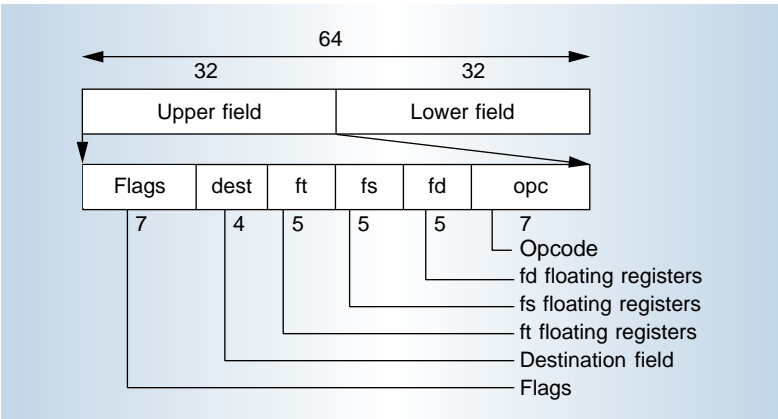


Figure 4. VPU instruction format.

Table 2. VPU features.

Feature	Description	VPU0	VPU1
Floating-point registers	128 bits × 32		
Integer registers	16 bits × 16		
FMAC	Single-precision FP × 4		
Multiply/add	1 cycle		
Min/max	1 cycle		
FDIV	Single-precision FP × 1		
Divide	7 cycles		
Square root (SQRT)	7 cycles		
Inverse SQRT	13 cycles		
Instruction bus	64 bits		
Coprocessor bus		128 bits	—
Instruction memory		4 Kbytes	16 Kbytes
Data memory		4 Kbytes	16 Kbytes

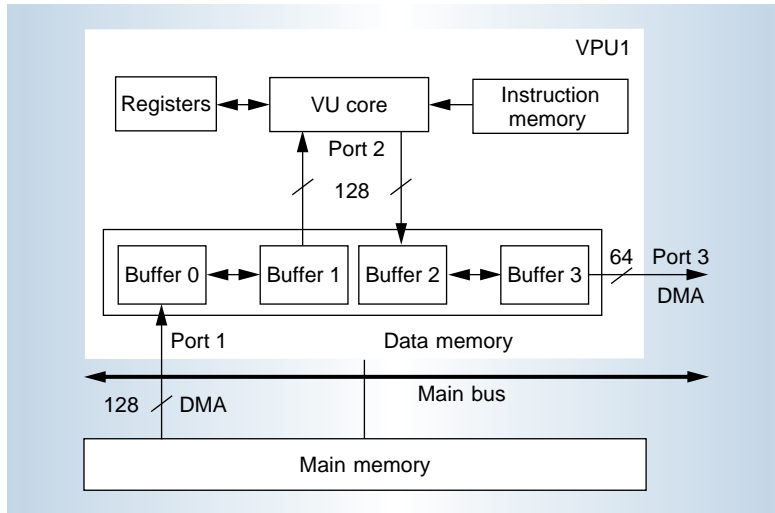


Figure 5. VPU macro data flow.

play list format. After the DMAC sends the list from the main memory to the VIF, the VIF decodes the header of the list (VIFcode). The VIFcode contains the size and location information of the following list. According to the VIFcode instruction, the VIF delivers the contents of the list to the proper address of the VPU instruction or data memory. The VIFcode also has special assignments that indicate when to start the VPU program. This special code is followed after all the VPU memory setup is completed. Figure 7 shows the VIFcode's instruction format.

After the VIF activates the VPU core (VU), the core executes the program in instruction memory. After a floating-point calculation is completed, the VU sends the results to the graphics synthesizer through the GIF.

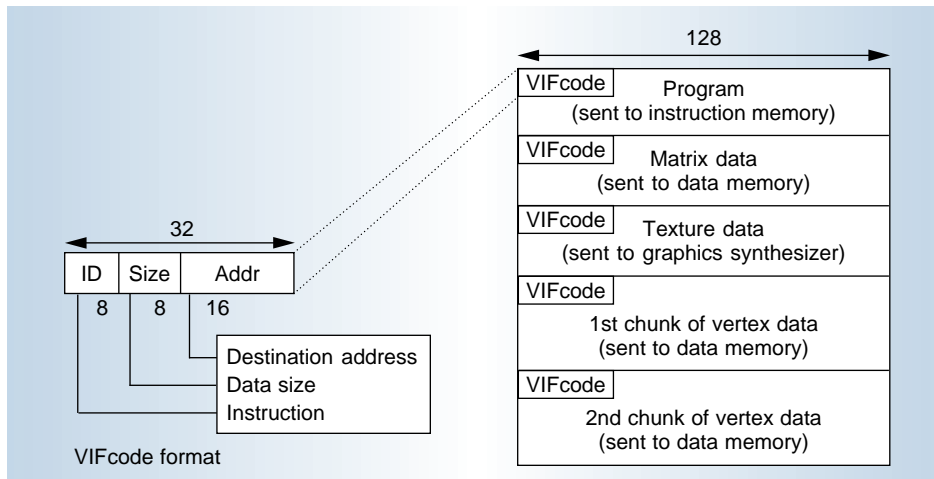


Figure 6. Typical VPU display list format.

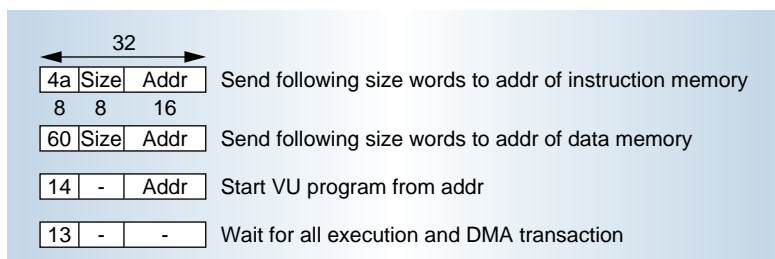


Figure 7. VIF instruction example.

The VPU program and data are provided from the main memory through the DMA controller (DMAC) and the VPU interface unit (VIF). This input data is called the 3D display list. Figure 6 shows a typical 3D display

VPU0/VPU1 collaboration

The processor connection topology is important for parallel processing performance. The complete symmetrical duplication of the processors does not effectively increase the total system performance, which includes software productivity. Our approach is to fix the role of VPU0 and VPU1 in advance.

VPU1 works independently for fixed, simple geometry operations that generate display lists to be sent to the graphics synthesizer. For example, geometry transformation of background buildings in a racing game is very simple because the coordinate of vertices comprising a building can be calculated with one 4×4 matrix and one division operation. The main CPU, with its complex functions, is unnecessary in such a simple operation. However, output data may be very large because simple geometry operations do not require the operation cycles per geometry transformation of the vertex. So VPU1's output should not be transferred through the main bus. To avoid disturbing other operations on the main data bus, we gave VPU1 an exclusive path to the graphics synthesizer.

On the other hand, VPU0 works as a coprocessor to the CPU core for complex pro-

cedures such as physical simulations. In the racing game just mentioned, VPU0 may calculate the behavior of the suspension and friction to the ground. The result may be a display list or front-end data for VPU1. In any case, the size of the output data produced per unit time may not be as large because the VPU0 task to produce a similar amount of output is more complex and time consuming than that of VPU1. Therefore the VPU0 result is buffered temporarily in the scratchpad RAM. Using the scratchpad RAM as a hub, the data is transferred to the graphics synthesizer or to VPU1 under software control.

Figure 8 shows a serial and parallel connection example. In the serial connection application, the VPU0 output data is stored in the scratchpad memory temporarily and also is sent to VPU1. In this case, VPU0 works as VPU1's preprocessor. In the parallel connection application, it is possible to send two independent display lists asynchronously from VPU0 and VPU1. For this requirement, the graphics synthesizer contains two sets of internal contexts (the drawing environment). Most rendering commands have a context ID field. It enables the graphics synthesizer to distinguish which context each command uses.

Parallel connection has another merit. After simulating an application, we found that VPU1 did not continuously generate the resulting display list. This happened because VPU1 must suspend sending the output display list until the entire input data chunk is prepared. However, the microrendering pipeline requires a constant input of data. As a result, the GIF monitors the status of the graphics synthesizer. If there is no data from VPU1 and the graphics synthesizer is idle, the GIF reads from the scratchpad RAM or main memory, where VPU0 may have prepared its own display list in parallel. Figure 9a shows the image generated by VPU1. The additional display list generated by VPU0 is merged with this. Figure 9b shows the final image.

Programming sample

Here, we show an example of the collaboration of VPU1 and VPU0 serial connection. Image 1 in Figure 10 (next page) shows one scene of a demonstration simulating the wave motion along the terrain and the refraction and reflection of the water surface. The shape

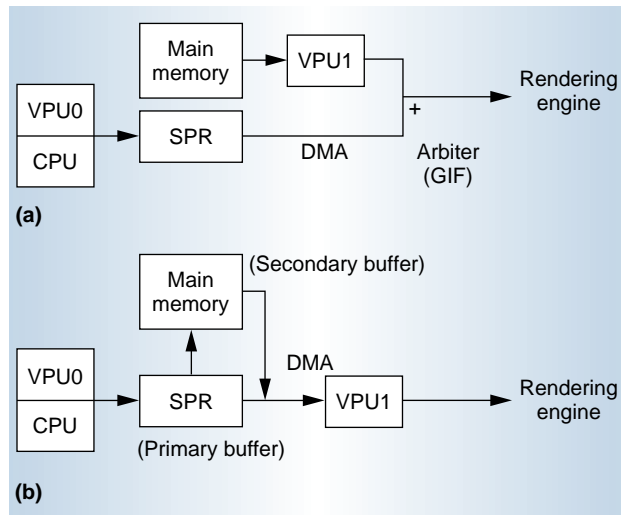


Figure 8. Multiple display list data flow: parallel connection (a) and serial connection (b).

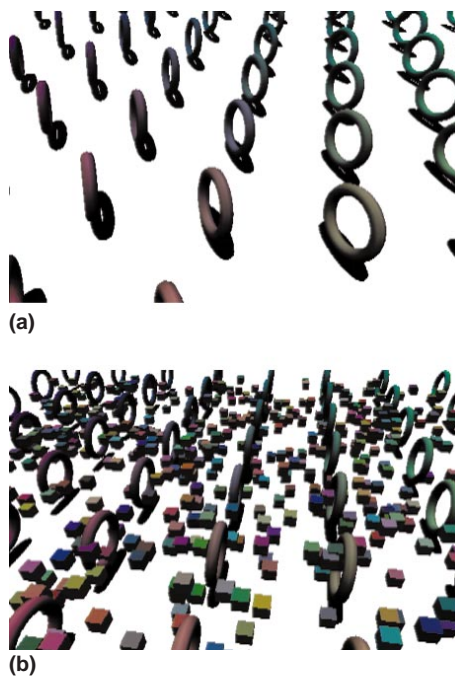


Figure 9. Multiple display list example: VPU1-generated image (a) and final VPU0-merged image (b).

of the terrain changes the speed of the wave motion. Since the shallow bed decreases the speed of the wave, we can simulate the wave motion lapping along the terrain using the integral of depth from a wave source to an arbitrary point as the phase angle of the wave in a simple physical model.

To simulate refraction and reflection, we piled up an opaque surface and two semitransparent surfaces, as shown in Figure 11. The bottom layer is drawn as terrain, and the second and the third layers as the water's surface. The second surface uses the modulated texture of the terrain to render the refraction; the third surface uses the sky and cloud textures to render the reflection. The flow of the calculation and rendering occurs in the following six steps:

1. *Generate the terrain.* Generate the terrain's shape by the fractal method.

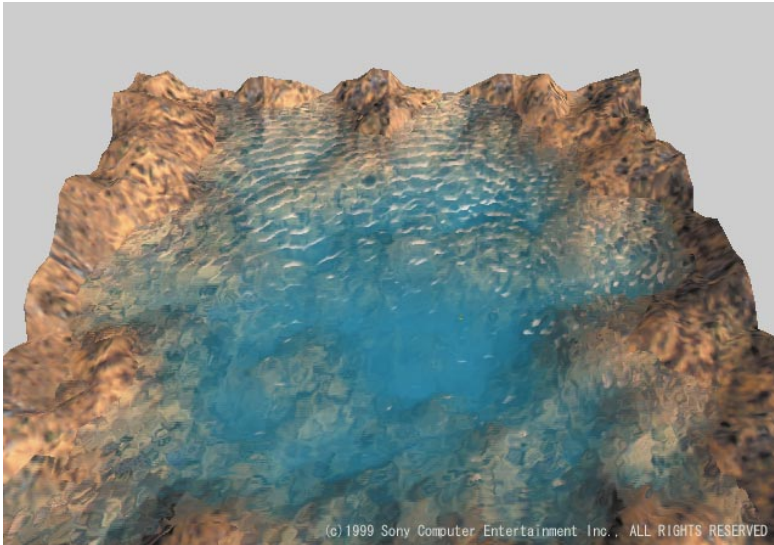


Figure 10. Wave motion image.

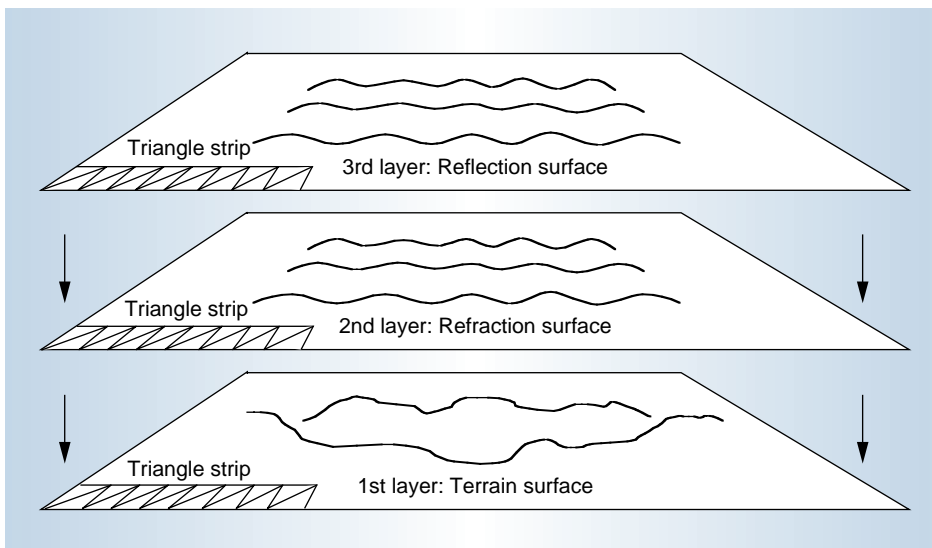


Figure 11. Two semitransparent surfaces on top of one opaque surface.

2. *Calculate the effect of the terrain's shape for wave motion.* Integrate the water depth from a wave source to each point to obtain/calculate the wave's phase angle.
3. *Render the terrain.* Calculate the geometry operation and render the terrain.
4. *Generate the wave's shape.* Generate the wave's shape from the phase angle using a trochoid curve.
5. *Render the wave's refraction.* Calculate view vectors to each point and the normal vectors on it. Calculate refraction vectors from the view vectors and normal vectors. Modulate the underwater texture according to the refraction vectors. Render the semitransparent wave surface using the modulated underwater texture.
6. *Render the wave's reflection.* Calculate reflection vectors from view vectors and normal vectors. Modulate the sky and cloud textures according to the reflection vectors. Render the semitransparent wave surface using the modulated sky and cloud textures.

Hardware resource allocation

Considering the characteristics of the six process steps, we allocated the hardware resources (processor and memory) to each process in advance.

Steps 1 and 2 need to read and generate large-scale data and store it. The calculation includes many conditional branches and variable-length loops. These processes are suitable for the CPU core and the main memory.

VPU1 and its local memory process step 3 most effectively because the calculation algorithm is a combination of simple geometry operations and the result is output to the graphics synthesizer. We divide the mesh data representing the terrain shape into many triangle strips, the data length of which is small enough to keep it in VPU1's local memory.

Steps 4, 5, and 6 need vector normalization or vector

products, and they are more complicated than the simple geometry operations. These calculations are suitable for VPU0 using the CPU core's conditional branch commands. We can also divide the wave surface mesh data into small triangle strips to keep it in the VPU0 local memory with steps 4 to 6. After the process of each strip in VPU0 completes, VPU1 processes the output data for geometry operations. VPU0 passes the preoperated data to VPU1 through the scratchpad memory.

Figure 12 shows the data flow of this example according to our hardware allocation. In this example, the load of calculation is largest in steps 4 to 6, but we can almost fully process them in VPU0, VPU1, and scratchpad memory. The CPU core and main memory are used only for conditional branch and data input operations. This means the CPU core and the main memory can support other operations at the same time as steps 4 to 6.³⁻⁶

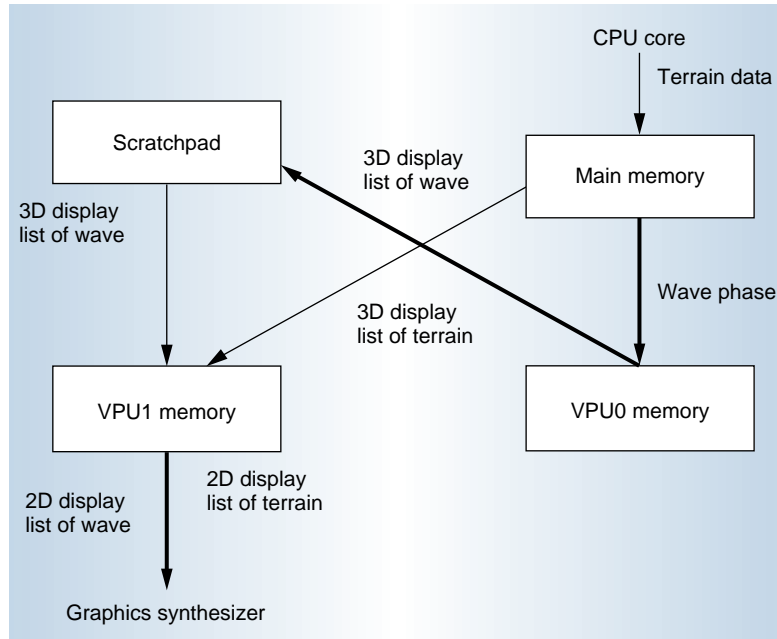


Figure 12. VPU0 calculates the three-dimensional wave shape using the local memory then stores the result in the Scratchpad.

The Emotion Engine can optimize the allocation of hardware resources according to the necessary amount of memory and the complexity of calculation. This ability gives the following advantages to programmers:

- They can easily obtain local high performance because each local processor is designed to the specific application style.
- They can easily obtain the highest performance because the connection of each local processor is selectable by software through fast local scratchpad memory.

In addition, although the macro dataflow path is different, the microarchitecture of VPU0 and VPU1 is identical. It provides another merit:

- It is easy to obtain high reliability because the prototyping on VPU0's macroarchitecture before production on VPU1's microarchitecture is possible.

These advantages allow programmers to suggest many new ideas. We believe these challenges produce many good-quality entertainment software packages.

Acknowledgments

We thank all other members of the next-generation PlayStation team, especially Hidetaka Magoshi and Mena Sato for their careful review of this issue and helpful advice.

References

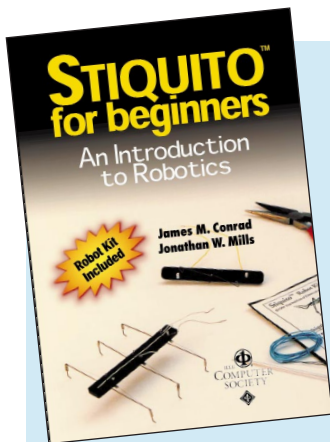
1. K. Kutaragi et al., "A Micro Processor with a 128b CPU, 10 Floating-Point MACs, 4 Floating-Point Dividers, and an MPEG2 Decoder," *ISSCC (Int'l Solid-State Circuits Conf.) Digest of Tech. Papers*, Feb. 1999, pp. 256-257.
2. F.M. Raam et al., "A High Bandwidth Superscalar Microprocessor for Multimedia Applications," *ISSCC Digest of Technical Papers*, Feb. 1999, pp. 258-259.
3. B.B. Mandelbrot et al., "Fractional Brownian Motions, Fractional Noises and Applications," *SIAM Review*, Vol. 10, No. 4, Oct. 1968, pp. 422-437.
4. B.B. Mandelbrot, *The Fractal Geometry of Nature*, Freeman & Co., New York, 1982.
5. D.R. Peachey, "Modeling Waves and Surf" *Proc. Siggraph 86*, ACM, New York, Aug. 1986, pp. 65-74.
6. A. Fournier, "A Simple Model of Ocean Waves," *Proc. Siggraph 86*, ACM, Aug. 1986, pp. 75-84.

Masaaki Oka is vice president of Sony Computer Entertainment, Tokyo, where he is responsible for LSI architecture. His technical interests include computer generation of graphics images. Oka received the BS degree in science from Kyoto University, Kyoto, Japan.

Masakazu Suzuoki is a seminar director in the Software Development Department in Sony Computer Entertainment. Previously, he was assigned to Sony's Information Processing

Laboratory, where he developed real-time computer graphics systems. His technical interests include image processing, computer graphics, digital signal processing, and compiler optimization. Suzuoki received the MS degree in electronic engineering from Tokyo University, Tokyo, Japan.

Direct questions to Masakazu Suzuoki at Sony Computer Entertainment, 7-1-1 Akasaka, Minato-ku, Tokyo, 107-0052 Japan, suzu@rd.scei.sony.co.jp.



STIQUITO™ for beginners An Introduction to Robotics

James M. Conrad and Jonathan W. Mills

This new book on Stiquito™ presents the a unique opportunity to learn about the field of engineering, electronics, and robotics in an original way. Materials and instructions to build the Stiquito robot and its electronic controls are included in the text.

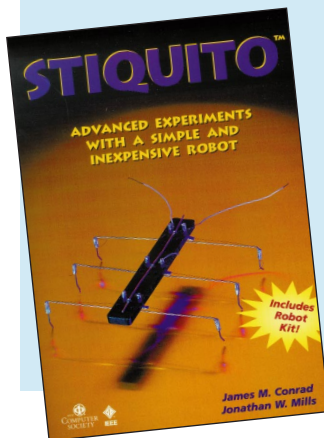
168 pages. Dec. 1999. ISBN 0-8186-7514-4.
Catalog # BP07514 — 24.00 Members / \$30.00 List
Price includes the robot kit

STIQUITO™ Advanced Experiments with a Simple and Inexpensive Robot

James M. Conrad and Jonathan W. Mills

Experiments in this advanced text lead the reader on a tour of the current state of robotics research. The authors describes the building process, modifications, and increased load capacity. Other chapters examine designs for simple controllers to enhance functionality and hardware upgrades that allow the robot to perform independent, intelligent operations.

328 pages. Dec. 1997. ISBN 0-8186-7408-3.
Catalog # BP07408 — \$36.00 Members / \$48.00 List
Price includes the robot kit



Order Today!

Online Catalog
<http://computer.org>

In the U.S. & Canada call
+1 800 CS BOOKS

