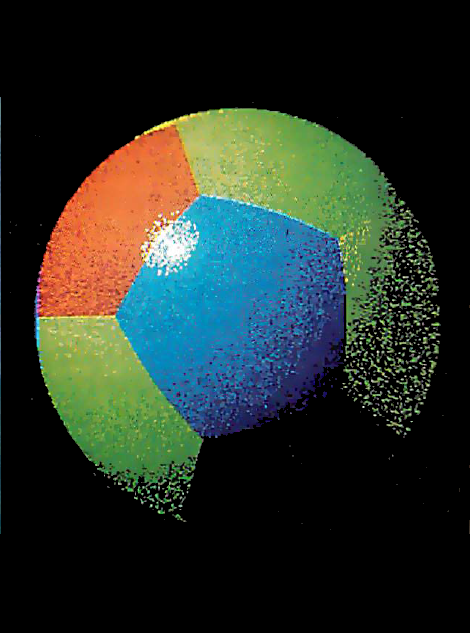
**Avoiding Coprocessor Bottlenecks, Mauro Bonomi**

**BYTE, March 1988, Vol. 13, No. 3, Pages 200, 202, & 204**

[**https://archive.org/details/byte-magazine-1988-03**](https://archive.org/details/byte-magazine-1988-03)

**Compiling: An Example**

The beach ball picture shown in photo 1 was generated on a 20-MHz Compaq Deskpro 386/20 with an EGA graphics card. The C code for this simple graphics example is presented in listing 1. The program uses the Phong shading technique to compute the intensities and colors of the surface of the ball. The idea behind this technique is to compute normal (perpendicular) vectors to each point on the surface using an interpolation scheme and then apply the shading model at each pixel displayed.



**Photo 1:** The beach ball.

The program involves a large number of floating-point computations on single-precision numbers, including the four basic instructions (add, subtract, multiply, and divide), as well as special functions (sine, cosine, and square root). The C program was compiled with a MetaWare High C compiler that supports both the Weitek 1167 and the 80387 coprocessor. The compiler generates either 80387 or Weitek 1167 object code. Users choose the desired coprocessor by setting a special switch on the command line when giving the compile command.

Running without a floating-point coprocessor, with the 80386 emulating 80387 instructions, the program takes 8 minutes to complete. Adding a 20-MHz 80387 reduces the execution time to a little more than 28 seconds. Finally, the Weitek memory-mapped coprocessor cuts execution time by about 3 times, down to only 10 seconds. The 200 percent performance improvement was achieved by simply recompiling the source code with no hand-coded optimizations.

Samples of the compiler output for both the 80387 and the Weitek 1167 are provided in listings 2 and 3. When compiled for the 80387, the code line s=r\*r-s\*s is translated into the code stream in listing 2.

The same equation is translated into 1167 memory moves by the compiler when the Weitek coprocessor is selected. The Weitek object code shown in listing 3 makes efficient use of coprocessor registers. The equation result s (s=r\*r-s\*s) , for example, is not stored back into the 80386 but is left in the coprocessor register file and is then used by the instruction (x=b+a\*sqrt(s)) , thus minimizing the data transfers between the microprocessor and the coprocessor and freeing up the system bus.

**Programming for Performance**

When writing high-level language code for a memory-mapped coprocessor, some simple techniques can lead to significant performance· improvements in the compiled code. First, variables used only in specific subroutines can be defined as local variables. The compiler then allocates such variables to coprocessor registers and will not store them back to main memory at the end of each instruction.

In the case of the 1167, the assembly language programmer has access to special instructions not otherwise available to compiler users. The multiply-accumulate (wfmac) instruction is an example. This instruction, useful in implementing matrix multiplications, specifies the multiplication of the two input operands, followed by the addition to a previously calculated partial result. A single memory move can then specify two floating-point instructions (multiply and add), further decreasing the burden on the system bus.

Additional assembly-level instructions that can improve the system performance are block moves-that is, moves of blocks of data to or from adjacent memory locations. A block move effectively encodes a stream of floating-point instructions. It is useful in loading and unloading the entire contents of the coprocessor register file, as well as in implementing vector adds and multiplies.

**Boiling Down**

Accelerating floating-point performance boils down to more than just clock rates and the size of machine words. We need the hardware assistance of dedicated ALUs and multipliers. But equally important is the need for efficient ways to move data and instructions between the main processor and the math coprocessor.

Removing the coprocessor instruction stream from the data bus-using the address bus to transmit op codes – leaves more room for passing operands. This strategy is one way to increase data bus bandwidth and improve processor/coprocessor interaction. •

**Listing 1: The C code to generate the beach ball. (Code written by Bruce Holloway of Weitek.)**

#include <math.h>

#include <stdio.h>

#include <time.h>

clock\_t start, stop;

float pi;

int colors[] = { 3, 6, 10, 13, 6, 3, 10, 13, 6, 3, 13, 10 },

d[] = { 640, 350, 1 }, i, k,

palette[] = {

000, 010, 001, 011,

020, 002, 022, 077,

040, 004, 044, 060,

006, 066, 007, 077

},

x, y, x\_min, x\_max, y\_min, y\_max;

unsigned short random;

main()

{

float a, b, c, l0, l1, l2, ln1, n0, n1, n2, p, q, r = 128, s, t, v[12][3];

int n;

/\* Put EGA in hi-res graphics mode & initialize palette. \*/

video\_int(0x10);

for (n = 0; n < 16; n++)

video\_int(0x1000, n + (palette[n] << 8));

/\* Print title & start timing. \*/

#ifdef Intel

printf("\n\n\t\t 80387 Phong Shading Demonstration\n");

#else

printf("\n\n\t\t Weitek 1167 Phong Shading Demonstration\n");

#endif

printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n \n\n\n\n\n\n\n");

start = clock();

/\* Pixel aspect ratio. \*/

a = 1.3;

/\* Screen center coordinates. \*/

b = .5 \* (d[0]-1);

c = .5 \* (d[1]-1);

/\* Unit length light source vector. \*/

l0 = -1 / sqrt(3.);

l1 = l0;

l2 = -l0;

/\* Ratio of circumference to diameter or a circle. \*/

pi = 4 \* atan(1.);

/\* A dozen vertices evenly spread over a unit sphere. \*/

v[0][0] = 0;

v[0][1] = 0;

v[0][2] = 1;

s = sqrt(5.);

for (i = 1; i < 11; i++) {

p = pi \* i / 5;

v[i][0] = 2 \* cos(p) / s;

v[i][1] = 2 \* sin(p) / s;

v[i][2] = (1. - i % 2 \* 2) / s;

}

v[11][0] = 0;

v[11][1] = 0;

v[11][2] = -1;

/\* Loop to Phong shade each pixel. \*/

y\_max = c + r;

y\_min = 2 \* c - y\_max;

for (y = y\_min; y <= y\_max; y++) {

s = y-c;

n1 = s / r;

ln1 = l1 \* n1;

s = r\*r - s\*s;

x\_max = b + a \* sqrt(s);

x\_min = 2 \* b - x\_max;

for (x = x\_min; x <= x\_max; x++) {

t = (x - b) / a;

n0 = t / r;

t = sqrt(s - t\*t);

n2 = t / r;

/\* Compute dot product and clamp to positive view. \*/

ln = l0 \* n0 + ln1 + l2 \* n2;

if (ln < 0) ln = 0;

/\* cos(e.r) \*\* 27 \*/

t = ln \* n2;

t += t - l2;

t \*= t \* t;

t \*= t \* t;

t \*= t \* t;

/\* Nearest vertex to normal yields max dot product. Get its color. \*/

for (i = 0, p = 0; i < 11; i++) {

if (p < (q = n0 \* v[i][0] + n1 \* v[i][1] + n2 \* v[i][2])) {

p = q;

k = colors[i];

}

}

/\* Aggregate ambient, diffuse, & specular intensities & do dither. \*/

i = k - 2.5 + 2.5 \* ln + t + (random = 37 \* random + 1) / 65536.;

/\* Clamp values outside range of three color levels to black or white. \*/

if (i < k - 2) i = 0; else if (i > k) i = 15;

draw\_dot();

}

}

stop = clock();

reg\_reset();

printf("\t\t\t\tTime = %3.2f seconds", ((double) (stop - start)) / CLK\_TCK);

while (clock() - stop < 5 \* CLK\_TCK);

/\* Put EGA in 80-column mode. \*/

video\_int(0x03);

}

**Listing 2: Sample of compiler output for the 80387.**

; s=r\*r-s\*s

0214 d9 46 c8 fld dword ptr -56[bp]

0217 d8 c8 fsqr

0219 d9 46 c4 fld dword ptr -60[bp]

021c d8 c8 fsqr

021e de e9 fsub

0220 d9 5e c4 fstp dword ptr -60[bp]

; x\_max=b+a\*sqrt(s)

0223 d9 46 c4 fld dword ptr -60[bp]

0226 d9 fa fsqrt

0228 d8 4e fc fmul dword ptr -4[bp]

022b d8 46 f8 fadd dword ptr -8[bp]

022e e8 ----e call \_mwtrunc

0231 a3 0054r mov @FONGBALL+84, ax

**Listing 3: Sample of compiler output for the Weitek 1 167.**

; s=r\*r-s\*s;

0388 64:a2 0719 wload .S fr2,fr25 ; mov al,fs:[]

038c 64:a2 081a wmul .S fr2,fr2 ; mov al,fs:[]

0390 64:a2 0b7a wmul .S fr26,fr26 ; mov al,fs:[]

0394 64:a2 107a wsub .S fr26,fr2 ; mov al,fs:[]

; x\_max=b+a\*sqrt(s)

0398 64:a2 471a wcvtsd .S fr2,fr26 ; mov al,fs:[]

039c 66:64:ff 36 0018 push .S fr2 ; push fs:[]

03a2 66:64:ff 36 0c1c push .S fr3 ; push fs:[]

03a8 e8 ----e call sqrt

03ab 64:a2 45a1 wcvtsd .S fr4,fr13 ; mov al,fs:[]

03af 64:a2 45a8 wcvtsd .S fr6,fr12 ; mov al,fs:[]

03b3 64:a2 889a wmul .D fr2,fr6 ; mov al,fs:[]

03b7 64:a2 8098 wadd .D fr2,fr4 ; mov al,fs:[]

03bb 64:a2 bc1a wfix .D fr2,fr2 ; mov al,fs:[]

03bf 66:64:a1 8c18 wstor .D fr2,eax ; mov eax,fs:[]

03c4 66:ef a4 c2 10 shld edx,eax,16

03c9 a3 0054r mov @FONG1167+72,ax