H O M E - b r e w e d Issue One:June 7, 1993
the Network Twenty-One Wireheads Information Zine
"Freebirthing the 90's into deep space"
Chicago, IL
Public Access Line at (312) 784-2852

C O N - t e n t s:

B E - g i n

S E G A - t h r e e - D I M E N S I O N S

First things first, the Sega 3d glasses DO NOT ACCEPT A VIDEO SIGNAL.
You do not "feed" them NTSC or VGA, they are "shutter glasses".
One goes on as the other goes off, and vice versa. Your system will
simulate two different views of the scene, thus, when the glasses
perform their "shutter" action, you are actually watching two different
images, one out of each eye and one at a time, very quickly.
This creates the S T E R E O S C O P I C effect.

--
The sega shutter glasses as explained to me by Ross Leonard:

Sega shutter glasses do not display an image. Instead, each LCD panel
(1 per eye) alternates on or off, therefore blocking light that can pass
through the respective lens. The computer (or Sega system) alternately
displays left and right side images on the screen that are synchronized
to the glasses, creating a rough stereoscopic view. When the computer
displays a right-side image, the left "shutter" is closed, so you can only
see it with the right eye, etc. I'm not sure of the exact display rate, but
since there is a slight flicker, I would assume it would be in the 15 to 20

per second range.

Ross Leonard

__

The gods then said "You will need parts to complete your circuit."

P A R T - l i s t i n g

RSPTN= Radio Shack part number
NM = Part name
QTY = Quantity in package
PCN = Packages needed
$$$ = Subtract this figure from net worth after purchase

(Network 21 not responsible for errors in figures, and is not affiliated
with the Radio Shack corporation)

RSPTN	NM	QTY	PCN	$$$
276-1617	2N2222 transistor	15	1	1.98
271-1335	10K 1/4 watt resistor	5	1	.39
271-1339	22K 1/4 watt resistor	5	1	.39
272-131	.01uF capacitor	2	1	.49
272-1026	22uF capacitor,	?	1	.69
unknown	Rectifier diode	2	3 needed	.49-.79
276-175	Breadboard	1	1	7.49

(RCA CD4030) Quad XOR gate (not available at most Chicago area RS)
__ Quad XOR gate will most likley have to be special ordered.

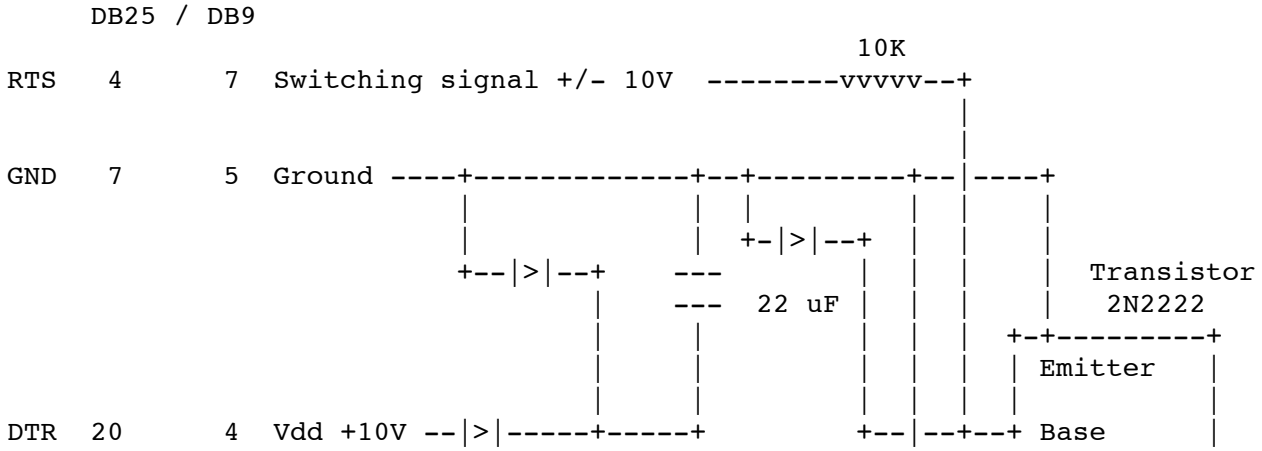
Soldering iron & solder
Cable for appropriate RS-232 port (to be cannibalized)
Stereo jack (to be cannibalized)

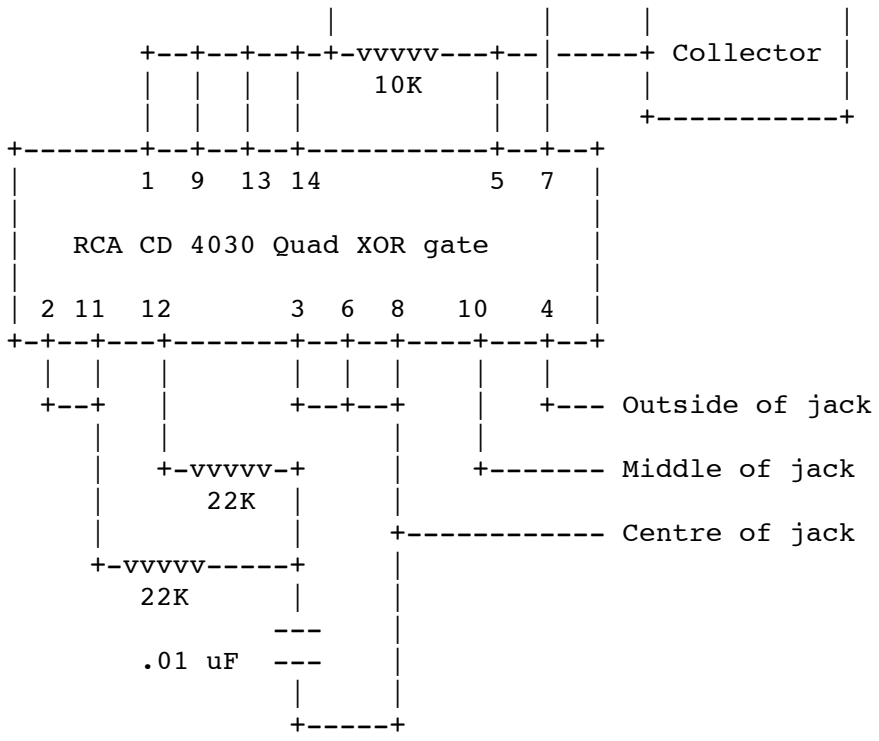
Well, there you have it, teenage america.
(Part no. lookups courtesy of gsumners@ntwrk21.chi.il.us.)

N O W !! t h e C I R C U I T !

Sender: hlab@milton.u.washington.edu (Human Int. Technology Lab)

Circuit to connect SEGA 3D glasses to RS-232 port.





Diode: --|>|--

Resistor: --vvvvv--

Capacitor:
```
  |
 ---
 ---
  |
```

——

W H A T a b o u t C O D E ?

The following code was written by F van der Hulst at a space-time fix of Feb.2, Aut 91.
It is source for viewing three-dimensional images with the Sega glasses and your constructed circuit.
Relevant info within comment text.

——

```
/*

VGA 320 * 400 * 256 * 2 frames routines.

Written by: F van der Hulst, 20/2/91

These routines display pixels in 320*400 mode by modifying the VGA
registers, as outlined in Programmer's Journal V7.1 (Jan/Feb '89)
article, pages 18-30, by Michael Abrash.

The advantage of 320 * 400, is that it gives two separate video pages,
which can be displayed on the screen independently. These can contain
two views of a scene, taken from slightly different viewpoints. These
are displayed alternately on the screen, in sync with a pair of
"chopper glasses", to give a 3D effect.
*/
```

```
#include <conio.h>

typedef unsigned char DacPalette[256][3];

/* Setvgapalette sets the entire 256 color palette     */
/* PalBuf contains RGB values for all 256 colors       */
/* R,G,B values range from 0 to 63                     */
/* Taken from SVGA256.H, by Jordan Hargraphix Software */

void setvgapalette(DacPalette *PalBuf)
{
  struct REGPACK reg;

  reg.r_ax = 0x1012;
  reg.r_bx = 0;
  reg.r_cx = 256;
  reg.r_es = FP_SEG(PalBuf);
  reg.r_dx = FP_OFF(PalBuf);
  intr(0x10,&reg);
}


unsigned int act_page = 0; /* Current page being written to */


#define VGA_SEGMENT        0xa000
#define SC_INDEX           0x3c4
#define GC_INDEX           0x3ce
#define CRTC_INDEX         0x3d4
#define DISPIO             0x3DA

#define MAP_MASK           2
#define MEMORY_MODE        4
#define GRAPHICS_MODE      5
#define MISCELLANEOUS      6
#define VRT_bit            8
#define MAX_SCAN_LINE      9
#define START_ADDRESS_HIGH 0x0c
#define UNDERLINE          0x14
#define MODE_CONTROL       0x17

void writepixel(int x, int y, unsigned char colour)
{
long addr;

        addr = ((x >> 2) + 320/4 * y + act_page);
        addr = ((addr & 0xffff0000l) << 4) + (addr & 0xffffL) + ((long) VGA_SEGM
ENT << 16);
        outport(SC_INDEX, (0x100 << (x & 3)) | MAP_MASK);
        *(char far*)addr = colour;
}

void set320x400mode(void)
{
struct REGPACK regs;
unsigned char x;

        regs.r_ax = 0x13;                                  /* Set 320*200*256 graph
ics mode via BIOS */
        intr(0x10, &regs);
```

```
/* Change CPU addressing of video memory to linear (not odd/even, chain, or
        chain 4), to allow access to all 256K of display memory. Each byte will
now
        control one pixel, with 4 adjacent pixels at any given address, one pixe
l
        per plane. */

        outportb(SC_INDEX, MEMORY_MODE);
        x = inportb(SC_INDEX+1);
        x &= 0xf7;
        /* Turn off chain 4  */
        x |= 4;
        /* Turn off odd/even */
        outportb(SC_INDEX+1, x);
        outportb(GC_INDEX, GRAPHICS_MODE);
        x = inportb(GC_INDEX+1);
        x &= 0xef;
        /* Turn off odd/even */
        outportb(GC_INDEX+1, x);
        outportb(GC_INDEX, MISCELLANEOUS);
        x = inportb(GC_INDEX+1);
        x &= 0xfd;
        /* Turn off chain */
        outportb(GC_INDEX+1, x);

/* Now clear the whole screen, since the mode 13h set only clears 64K. Do this
        before switching CRTC out of mode 13h, so that we don't see grabage on t
he
        screen. */

        outport(SC_INDEX, 0x0f00 | MAP_MASK);                   /* Write to 4 pl
anes at once */
        setmem(MK_FP(VGA_SEGMENT, 0), 0xffff, 0);

/* Change mode to 320*400 by not scanning each line twice. */
        outportb(CRTC_INDEX, MAX_SCAN_LINE);
        x = inportb(CRTC_INDEX+1);
        x &= 0xe0;
/* Set maximum scan line to 0 */
        outportb(CRTC_INDEX+1, x);

/* Change CRTC scanning from doubleword to byte mode, allowing the CRTC to
        scan more than 64K */
        outportb(CRTC_INDEX, UNDERLINE);
        x = inportb(CRTC_INDEX+1);
        x &= 0xbf;                                      /* Turn off doubleword *
/
        outportb(CRTC_INDEX+1, x);
        outportb(CRTC_INDEX, MODE_CONTROL);
        x = inportb(CRTC_INDEX+1);
        x |= 0x40;                                      /* Turn on the byte mode
 bit, so memory is linear */
        outportb(CRTC_INDEX+1, x);
}

void end320x400mode(void)
{
struct REGPACK regs;

        regs.r_ax = 3;                          /* Return to text mode */
        intr(0x10, &regs);
```

```
}

/* Set visible page */

void setvispage(int page)
{
        outport(CRTC_INDEX, (page << 15) | START_ADDRESS_HIGH);
}

/* Set active page (page being written to */

void setactpage(int page)
{
        act_page = page ? 0x8000 : 0;
}


void WaitForVerticalRetrace(void)
{
static char chopper = 1;

        while (inportb(DISPIO) & VRT_bit) /* wait */ ;
        while ((inportb(DISPIO) & VRT_bit) == 0) /* wait */ ;
        if ((chopper++ & 1)== 0)                outportb(0x3fc, 1);
        else
outportb(0x3fc, 3);
}

void main(int argc, char *argv[])
{
   set320x400mode();

/* Now fill the rgb_palette structure in memory with colour info */

   setvgapalette(&rgb_palette);

   setactpage(0);
/* Now call writepixel to put stuff on page 0 */
   setactpage(1);
/* Now call writepixel to put stuff on page 1 */

   while (!kbhit()) {
           WaitForVerticalRetrace();
           setvispage(0);
           WaitForVerticalRetrace();
           setvispage(1);
   }
   getch();
   end320x400mode();
}

--

W H A T  a b o u t  S O F T W A R E ?

- rend 386

Quoted from documentation:

"This version now support stereoscopic viewing; the assumption is that you
 have the Sega 3D glasses ....
```

```
   -x    ensable stereo (use if you don't have sega glasses)
   -m    use mirror stereo
   -r    reverse eyes (left-for-right); useful if your wiring is wrong
   -1    use COM1 for Sega glasses
   -2    use COM2 for Sega glasses"
--
Rend 386 by Dave Stampe and Bernie Roehl is available via anonymous ftp
from sunee.uwaterloo.ca in the directory "/pub/rend386"
--

"There is also a mailing list, rend386@sunee.uwaterloo.ca (to be added
 to the list, send mail to rend386-request@sunee.uwaterloo.ca).  Traffic
 should be reasonbly low; if it gets too high, we're willing to go to a digest
 format."
                                    Bernie Roehl


__
The demonstration and libraries are also available on Network 21.
It is worth checking out.
It REQUIRES at least a 386 or 486, and standard VGA. It -will not- run on
an 80286.
Rend 386 also supports the Mattel Powerglove as a 3d input / object
manipulation device (it is neat, and -fast-!)
Very "C like". Very powerful.
Two nodes up.
__



                  t h e  P O W E R  g l o v e


Included:
Connections to the IBM PC, Atari ST, NeXt, Amiga, info on the Power Glove
mailing list, source code for the IBM PC, Amiga, ST, and sources for Mac
Power-Glove / Sega information.


W H A T - i s ?

 The Mattel Powerglove was originally intended as a device for making
 your video-game life more enjoyable. Manufactured for the Nintendo
 Entertainment system, it originally carried a price tag of $99.00+.
 Now it has been adopted by the street as a 3d input device for a
 wide-range of personal and super-computers. I personally think that
 you get better control with the mouse, but there are disadvantages to
 the mouse: two dimensions, you can't wear it, etc.
 Code is now widley available for use with the power-glove, as well as
 third party software such as the forementioned rend 386.

Tme following are instructon texts gathered from various places on "hooking
in" your perhaps lawnmower-man inspired input device:

P O W E R  g l o v e  -  t o  I B M  P C

Be sure and check out the BYTE magazine article in the subject line,
(issue: July, 1990;page 288). This article provided widespread inspiration
for the PG's current "home brewed" vr use. Illustration included.


__
```

Submitted to the net by:
wgerlt@atl.ge.com
on:
December 20th, 1991
(This is probably the most widley distributed and popular version of the
 power-glove hack)
—

Included:

1. How to hook them up using an external power source
    A. parts and aprox cost
    B. circuit diagram
2. How to hook them up using power from the PC
    A. parts and aprox cost
    B. circuit diagram

Not Included:

1. Required software information
    maybe a second document when I get that far, unless someone else wants to write it up before I start asking a bunch of questions again - any volunteers? :-) Wouldn't it be nice if we clearly documented our work so that newcomers could quickly and easily be brought up to speed without having to take up a lot of everyone's time and net bandwidth?

Obligatory Disclaimer:

I am not an electronics expert of any kind. I don't understand most of what is included in this summary. This is only a summary of replies I got to several questions I asked on this particular topic as well as some information obtained from an article in Byte (July, 1990). I assume no responsibility for anything that happens if you attempt to use any of the information in this post. I don't care if you distribute, alter, copy, etc anything in this post. It would be nice though if you made your additions, alterations, comments, etc available to the glove mailing list at glove-list@karazm.math.uh.edu. Also, the contents of this post do not reflect any opinions other than my own, especially GE's.

Introduction

There may be many ways to hook a PowerGlove to a PC/clone. The following discussion describes two possibilities. I used the first method and it seems to work. A number of others have reported using the second method which is simpler and apparently works just as well. There may be errors in the methods, but I've tried to faithfully reproduce what I've done and what has been reported by others.

The second method is the easier and cheaper of the two. However, it requires you to use another port on your PC (in addition to the printer port that both methods require) or tapping unused power and ground lines somewhere in your PC. That's no big deal if you know what you're doing and have an extra port or power source to tap.

The first method requires no PC modifications. It only uses the printer port. However, it requires an external 5v power supply. The discussion in that method assumes you have a power supply, but that it isn't 5v.

The aproximate prices shown are about what I paid at my local Radio Shack, execept for the glove and extension cable. You might be able to beat these prices by shopping around or rummaging through your basement.

1. Hooking them together using an external power supply

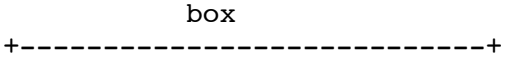
Needed:

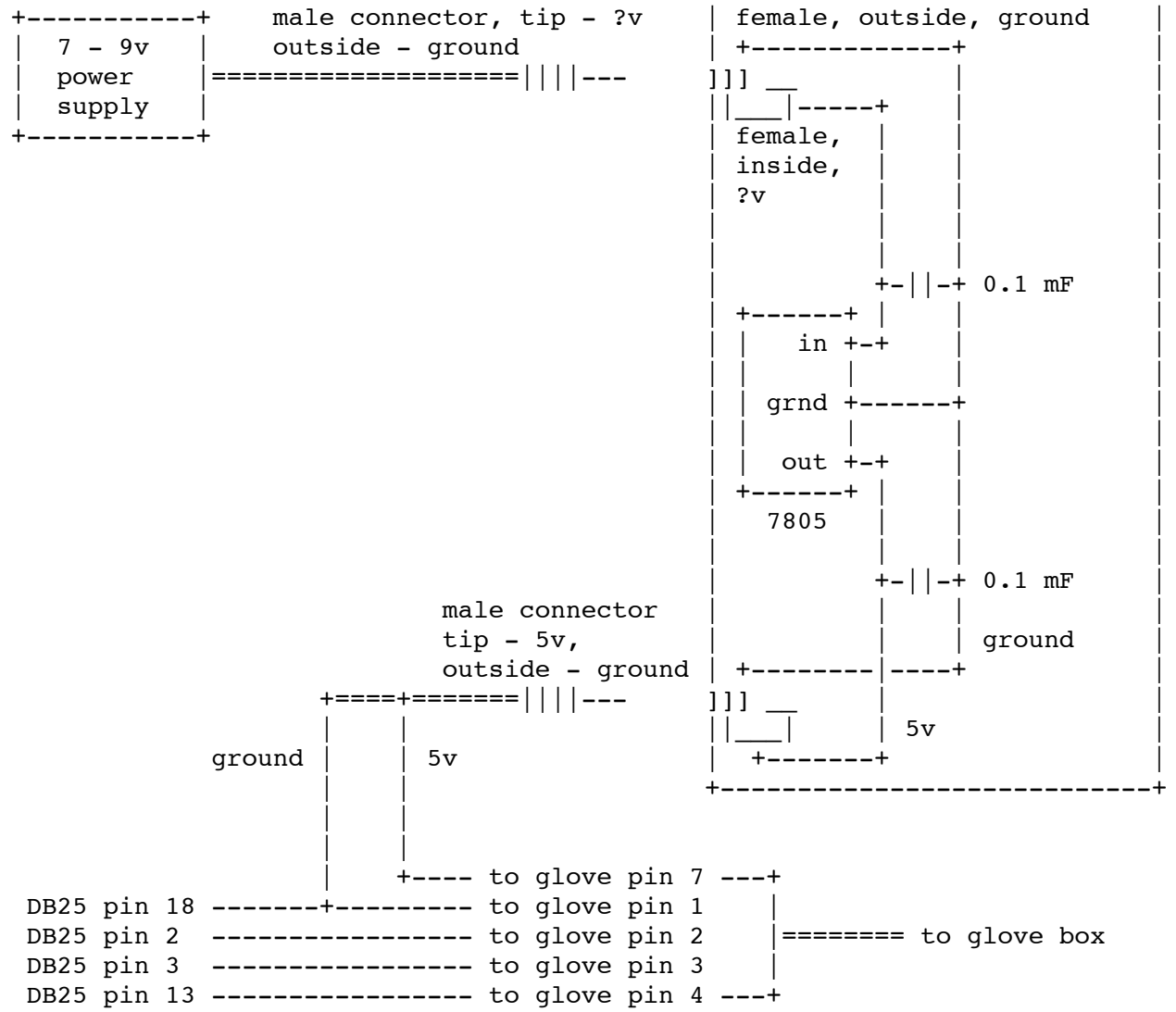
Nintendo PowerGlove
US$20-50
power supply (ac/dc, battery pack, etc) that produces in the range of 7 - 9 volts at around 300mA
I bought one with adjustable settings at Radio Shack for about $13. I use the 7.5v setting. You might be able to get by with a 6v source, but the 7805 regulator might not produce a steady 5v unless you give it at least 7v. You might be able to get by OK with 9v or 12v, but you may need a heat sink at higher voltages.
(2) 0.1 mF capacitors
less than US$1 for ceramic
7805 +5vdc voltage regulator
about US$1.25
DB25 connector and cover
about US$3.50
heat shrink tubing and/or electrical tape
soldering iron, heat sink, and solder
wire

Optional

extension cable for the PowerGlove
A package of two Curtis Super Extendo cables costs just under US$10. The only reason to get these is so that you can modify an extension cable instead of the one attached directly to the glove hardware. They also give you some extra length, if that matters. I think Nintendo makes extension cables, too, but I don't know what they cost.
2 female and 1 male connector
My power supply had 1/8" (3.5mm) male phone jack connector on it. I mounted a female connector on a small box to plug the power supply into. I also used a pair of the connectors at the output side of the box so I could disconnect the box easily from the modified glove cable. I didn't want to chop up the power supply's cable, so the first connector seems like a good idea. The others are completely unnecessary. I only used this type of connector because my power supply came with one. The connectors cost me about US$3.00.
box to house completed circuit
Mine cost about US$1.75.

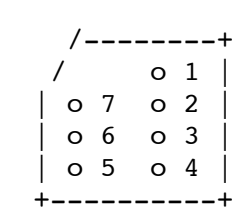
The circuit:





NOTE: Connect the capacitors within about 1/4" of the 7805 for best results

glove connector pin numbers



glove pin number	nintendo wire color	curtis wire color
1 ground	black	orange
2 data clock	orange	red
3 latch	yellow	brown
4 data out	green	black
5 unused		
6 unused		
7 5v	red	white

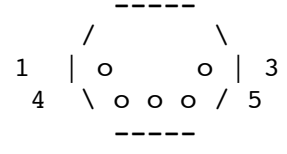
DB25 pins
2 data clock
3 latch

```
 13 data in
 18 ground
```

2. Hooking them together using power from the PC

    Ignore the circuit that provides the 5v and ground lines above. Instead,
    get the 5v and ground from either:
         1. an unused disk drive cable in your PC or
         2. 5v from pin 5 on a keyboard port and ground from pin 4 of the
             same keyboard port
    All the other connections from the glove cable to the DB25 are the same,
    including the ground connection.

    keyboard connector



Hope this helps. Feel free to add to and/or correct this information.

Bill   wgerlt@atl.ge.com   12/20/91

__
The "circuit method" seems like a tedious way of hooking up the glove,
I use a spare drive cable from my PC for power, and it works fine.
(Usually RED and BLACK, +5v, GND, respectivley)
__

W H A T a b o u t C O D E ?

__
This was transmitted via e-mail from the power-glove mailing list
(more info on glove-list later in document)
__
"
Okay, here's (I hope) the C code for IBMPC 386.
The Byte Box I referred to was merely a box containing the proper connections
ala Byte magazine to interface the PG to a PC printer port.
This was written by Bandit and Michael Hevern, I merely poured the beer
and kept the programming on track ...

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

#define                DELAY                delay_val
#define                PORT_CTL        0x0378
#define                PORT_DATA        0x0379
#define                LATCH_LO        0
#define                LATCH_HI        0x02
#define                CLOCK_LO        0
#define                CLOCK_HI        0x01

unsigned char        get_glove(void);
void display_str( unsigned char );
```

```
int               delay_val=100;

main()
{
        char        ch[20];
        unsigned char
                x,
                last_x;

        x = 0;

        printf("Enter the delay[100] -> ");
        delay_val = atoi(gets(ch));
        while (1) {
                if( x != last_x )
                {
                     /*        printf("X = %02x  ", x); */
                        display_str( x );
/*                        printf("Press key (q)-> ");
                        ch = getche();
                        if (ch == 'q')
                                break;
                        if (ch == '+')
                          {delay_val++;
                           printf("Delay Value = %d \n", delay_val);
                        if (ch == '-')
                          {delay_val = delay_val - 1;
                           printf("Delay Value = %d \n", delay_val);
                          }
*/                }

                last_x = x;
                x = get_glove();
                }
}

#define IN_BYTE( xx )        { xx = inportb( PORT_DATA ); }
#define OUT_BYTE( xx )        { outportb( PORT_CTL, xx ); \
                                  for (delay=0; delay < DELAY; delay++); }

unsigned char        get_glove()
{
        unsigned char
                far        *ctl,
                far        *data,
                value;
        unsigned char
                results;
        int        x,
                delay;

        ctl = MK_FP(0, PORT_CTL);
        data = MK_FP(0, PORT_DATA);

        /*        *ctl = LATCH_LO + CLOCK_HI;        */
        OUT_BYTE( (LATCH_LO + CLOCK_HI) );
        OUT_BYTE( (LATCH_HI + CLOCK_HI) );
        OUT_BYTE( (LATCH_LO + CLOCK_HI) );

        for (results=0, x=0; x < 8; x++) {
                results <<= 1;
```

```
                IN_BYTE(value);
                if (value & 0x10)
                        results |= 1;
                OUT_BYTE( (LATCH_LO + CLOCK_LO) );
                OUT_BYTE( (LATCH_LO + CLOCK_HI) );
                }
        return( ~results );
}

typedef struct
{
        unsigned char mask;
        char *off_str;
        char *on_str;
} DISP_STRUCT;

DISP_STRUCT disp[] =
{
        {        0x80,        "  A_UP",        "A_DOWN"        },
        {        0x40,        "  B_UP",        "B_DOWN"        },
        {        0x20,        "NO_SEL",        "SELECT"        },
        {        0x10,        "NO_SRT",        " START"        },
        {        0x08,        "      ",        "    UP"        },
        {        0x04,        "      ",        "  DOWN"        },
        {        0x02,        "      ",        "  LEFT"        },
        {        0x01,        "      ",        " RIGHT"        },

};

void display_str( unsigned char value )
{
        int i;
        char *str;

        printf( " %02x:", value );

        for( i = 0; i < 8; i++ )
        {
                if( value & disp[ i ].mask )        str = disp[i].on_str;
                else                                str = disp[i].off_str;

                printf( " %s", str );
        }
        printf( "\n" );
}
```

********!!!!!!!
Okay,
that's it. Good luck with it, and my best wishes to those of you working
to decode High Resolution mode. We're rooting for you here at UW!
"

__

T H E b r a i n B O X

This exerpt from the PGSI.FAQ available on Network 21, and via
anonymous FTP (site below), gives a good description of the
"Brain Box", or "PGSI".
-Sega 3d shutter glasses supported-

__
"Maintained by: Ben Gross (pgsi@uiuc.edu)

```
 Stored on: FTP.cso.uiuc.edu (128.174.5.59) in /ACM/PGSI as pgsi-faq
...
The PGSI is a small device measuring a scant 3.35" by 1" by 1.5" that plugs
right unto any DB-25 style RS-232 connector.  The device is guaranteed to work
on any IBM, IBM compatible, Amiga, Macintosh, Workstation, or other type of
computer that has such a port.  The IBM, Amiga, and workstations typically
have such a port, but Mac users will either need to purchase the MAC MiniDin 8
to DB-25 adapter, or purchase one from ACM at UIUC.  The device integrates the
PowerGlove and SEGA style shutter glasses into the computer as input/output
devices, and communicates with them using standard RS-232 data values.  The
interface emulates the AGE command set and the Menelli command set, has
extensions to allow the user to read in values from 8 Analog-Digital Ports, 1
digital port consisting of 8 pins, and can be reprogrammed with code updates
or extensions by the end user.
...
The system comes with a user's manual, a diskette featuring a library of
routines which will allow users to port code between machines with no
recoding, and the PGSI.  The disk also contains demonstration programs, and is
available in Mac, IBM, and Amiga formats.  Other formats may be requested.
Also, all code and updates will be put on an anonymous FTP site."
```

——

```
The last time I heard, the PGSI was running for $90.00 assembled, $85.00
for a kit. For more information on ordering, etc, I would reccomend posting
to sci.virtual-worlds (moderated), or subscribing to the glove-list.
Order form is available on Network 21 (second run), however, administration
takes no responsibility for it's use.
Information resources will be listed in the end of this document.
```

——

```
                t h e  A T A R I  s t  H A C K

by J. David Beutel (jdb9608@cs.rit.edu)
(Relevant inforamation stored within the code's comment text)
This is in Low-Res mode, for more information, see the Power-Glove FAQ.
```

——

```
#include <stdio.h>

extern unsigned char    lpgpol();       /* powerglove poll routine */
extern unsigned char    Glove;          /* results also returned here */

main()
{
    while( 1) {
        printf( "Glove = %x\n", (int) lpgpol());
        printf( "Glove = %x", (int) Glove);
        printf( "\t up = %c\n", (Glove & 0x08 ? 'y' : 'n'));
    }
}
-------------------------cut-----------------------------------------
; lo-res PowerGlove interface poll routine (lpgpol)
;
; (tested on the Atari 1040ST^F with Sozobon C 'jas')
;
; declaration and usage in C:
;
; extern unsigned char  lpgpol();       /* Read glove and return state. */
```

```
; extern unsigned char  Glove;          /* Return state available here too. */
;
; printf( "%x\n", (int) lpgpol());      /* Read and print the glove's state, */
; going_up = Glove & 0x08;              /* or test bits from the last poll. */
;
; See the "glove return values" below for interpreting the glove's state.
;
; I used the _Atari_ST_Internals_ (by Abacus Software),
; _The_C_Programming_Language_ (by K&R), and of course
; _M68000_Programmer's_Reference_Manual_ (by Motorola)
; (not to mention the July 1990 _Byte_ article) as references.
;
; J. David Beutel  11011011  -1/9107.26

; Wiring info:
; function  Atari ST                PowerGlove
; +5v       pin 7 joystick port     pin 7
; ground pin 8 joystick port     pin 1
; data      pin 1 parallel port     pin 4
; latch     pin 2 parallel port     pin 3
; clock     pin 3 parallel port     pin 2
;
; pin-outs looking into the connectors on the machines
; joystick port         parallel port               PowerGlove
;   9 8 7 6         13 12 11... 3 2 1                 1
;  5 4 3 2 1         25 24 23...  14                 7 2
;                                                    6 3
;                                                    5 4

; PSG (programable sound generator) is the Yamaha YM-2149,
; which controls most of the parallel port on the ST.
; It's accessed thru memory-mapped I/O.  These location
; can be accessed only in supervisor mode.
;
psg     .equ    $ff8800             ; register select / read data
psgw    .equ    $ff8802             ; write data

; registers in the PSG
;
conreg  .equ    7                   ; read/write control register in PSG
areg    .equ    14                  ; port A register in PSG
breg    .equ    15                  ; port B register in PSG

; read/write bits in PSG register 7
;
; (note: do not use the bset instruction with Sozobon's jas,
; because apparently it doesn't assemble it correctly when
; using immediate mode to a data register (because of byte v. long
; confusion, I think).)
;
aread   .equ    $bf                 ; A read bit in register 7 (and)
bread   .equ    $7f                 ; B read bit in register 7 (and)
awrite  .equ    $40                 ; A write bit in register 7 (or)
bwrite  .equ    $80                 ; B write bit in register 7 (or)

; bits from the parallel port
;
gdata   .equ    $20                 ; Centronics strobe bit in port A
                                    ; is pin 1 on the parallel interface
                                    ; and the glove data (in) line.
glatch  .equ    $01                 ; Bit 1 port B is pin 2 on the parallel
```

```
                                        ; interface and the glove reset (out) line.
gclock  .equ    $02                     ; Bit 2 port B is pin 3 on the parallel
                                        ; interface and the glove clock (out) line.

; glove return values ('and' these values with the result in _Glove
; to get a boolean in a C program).  These are here just for
; documentation.  They should be #define'd in a C program.
;
PG_rt       .equ    $01
PG_left     .equ    $02
PG_dn       .equ    $04
PG_up       .equ    $08
PG_start    .equ    $10
PG_select   .equ    $20
PG_B        .equ    $40
PG_A        .equ    $80

; timing should be tricky for the hardware
;
delayval    .equ    0               ; 7 microsecond delay

; The result is stored in this global variable (BYTE Glove).
; I return this result in d0 for conventional C function return
; value, but initially I passed the results by global variable,
; so I'm leaving it in here.  Use whichever access method is
; most convenient.
;
        .bss
        .globl  _Glove
        .comm   _Glove,2            ; a byte, word-aligned

; This is the function callable from a C program.
; It returns the byte in _Glove.  It takes no arguments.
;
        .text
        .globl  _lpgpol             ; extern BYTE   lpgpol();
_lpgpol:
        link    a6, #-0             ; frame for local variables (C format)
        jsr     lpgpol
        unlk    a6
        rts                         ; returning value in d0

; This is the xbios call to Supexec, to put us in supervisor mode
; so we can write to the PSG I/O memory.  You can call this from
; an assembly language program without the C style linking.
;
        .globl  lpgpol              ; for assembly language calls
lpgpol:
        move.l  #guts, -(sp)        ; routine to execute
        move.w  #38, -(sp)          ; xbios 38 = Supexec
        trap    #14                 ; call the xbios to run guts
        addq.l  #6, sp              ; pop arguments
        rts

; Here's the real logic.
;
guts:
        movem.l d1-d3/a0-a3,-(a7)   ; save registers
        clr.w   _Glove              ; clear the result (word for accum shift)
        moveq.l #7, d2              ; bit counter (read 8 bits from glove)
        jsr     preset              ; pulse the reset line
```

```
bit_loop:       ; the bit loop reads the bits in one by one
        jsr     readbit             ; read in a bit from the glove
        jsr     pclock              ; pulse the clock line
        dbne    d2, bit_loop        ; loop to read all 8 bits

        not.w   _Glove              ; invert the results by convention
        movem.l (a7)+,d1-d3/a0-a3   ; restore registers
        move.b  _Glove, d0          ; return the byte in d0
        rts

;;;;;;;;;;;;;;;;; subroutines ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; delay depends on the clock speed and other finicky hardware stuff.
; cycles = 20 (jsr) + 8 (move.w) + 16 (rts) + 12 (dbne exit) (= 7us)
;           + 10 * delayval (dbne iteration)    (= 1.25 * delayval us)
;
delay:  move.w  #delayval, d1       ; reset counter
L1:     dbne    d1, L1              ; loop
        rts

; prepare port B for output
; (note: the PSG has input and output on different bytes,
; so I can't simply 'or' its control register directly.)
;
bout:   move.b  #conreg, psg        ; access the control register
        move.b  psg, d1             ; read the control register
         ori.b   #bwrite, d1         ; set the write B bit
        move.b  d1, psgw            ; write the control register
        move.b  #breg, psg          ; access port B
        rts

; the RESET pulse.  An L_H_L pulse, a minimum of 4 microseconds long.
preset: jsr     bout
        move.b  #gclock, psgw               ; latch low + clock hi
        jsr     delay
        move.b  #(glatch + gclock), psgw    ; latch hi + clock hi
        jsr     delay
        move.b  #gclock, psgw               ; latch low + clock hi
        rts

; the CLOCK pulse.  An H_L_H pulse, about 3 microseconds long.
pclock: jsr     bout
        move.b  #0, psgw                    ; latch low + clock low
        jsr     delay
        move.b  #gclock, psgw               ; latch low + clock hi
        jsr     delay
        rts

; prepare port A for input
; (note: the PSG has input and output on different bytes,
; so I can't simply 'or' its control register directly.)
;
ain:    move.b  #conreg, psg        ; access the control register
        move.b  psg, d1             ; read the control register
        andi.b  #aread, d1          ; reset the read A bit
        move.b  d1, psgw            ; write the control register
        move.b  #areg, psg          ; access port A
        rts

; Read a bit from the data in line from the powerglove,
```

```
; shift the contents of _Glove, and store the bit from the glove
; as bit 0 in _Glove so that in the end all 8 bits will be in _Glove.
;
readbit:
        jsr     ain                 ; set port A for input
        move.b  psg, d1             ; read port A
        andi.b  #gdata, d1          ; mask the input bit
        lsr.b   #5, d1              ; shift the input bit over to bit 0
        lsl.w   _Glove              ; shift the result byte left 1 bit
        add.b   d1, _Glove          ; accumulate the input bit in _Glove
        rts

        .end
```

__

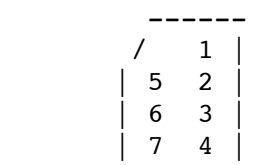
t o > A M I G A
by Alan Bland

(Based on the ATARI 1040ST hack by manfredo@opal.cs.tu-berlin.de.)

*/ "This Amiga hack was done by Alan Bland. It is not directly compatible with the AC Tech hack, the BYTE hack, nor the ATARI 1040ST hack, but you should be able to modify the code to work with any of those hacks.

The Amiga code is ugly, as was the original ST code. The parallel port hardware is accessed directly without knowledge of the operating system."

P I N - o u t s:



Mattel PowerGlove	Amiga
GND pin1	pin18 parallel port
clock pin2	pin2 parallel port
latch pin3	pin3 parallel port
data pin4	pin4 parallel port
+5V pin7	pin14 power +5V

Datapacket: (12 bytes)

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
A0	X	Y	Z	rot	finger	keys	00	00	3F	FF	FF

```
**********************************************************************/

#include <stdio.h>
#include <hardware/cia.h>

extern struct CIA far ciaa;

/* bits from parallel port -- Alan's hack */
#define         GDATA           0x04    /* glove data in */
#define         GLATCH          0x02    /* glove latch out */
#define         GCLOCK          0x01    /* glove clock out */
#define         GCLOLAT         (GLATCH|GCLOCK) /* latch and clock */

#define getbit()        (ciaa.ciaprb & GDATA) >> 2
#define initport()      ciaa.ciaddrb = GCLOLAT

/* delays in microseconds */
#define     D2BYTES      96
#define     D2BITS       22
#define     D2SLOW       14720

#define     C0L0()       ciaa.ciaprb = 0        /* clock 0 latch 0 */
#define     C0L1()       ciaa.ciaprb = GLATCH   /* clock 0 latch 1 */
#define     C1L0()       ciaa.ciaprb = GCLOCK   /* clock 1 latch 0 */
#define     C1L1()       ciaa.ciaprb = GCLOLAT  /* clock 1 latch 1 */

#define     setporta()  delay(3)
#define     setportb()  delay(3)

void Hires (void);
unsigned char getbyte (void);

/* convert microseconds to cia ticks */
#define delay(usec) timersleep((usec*1397)/1000)

int control_c()
{
        closetimer();
        printf("<<goodbye>>\n");
        return 1; /* causes exit */
}

void main ()
{
        unsigned char buf[12];
        register unsigned char *bp;

        opentimer();
        onbreak(control_c);

        Hires (); /* set PG into 'hires' mode */

        printf("Press ^C to stop\n\nx   y   z       rot   finger button\n");
        for ( ; ; ) /* read 12 byte packets */
        {
                bp = buf;
                *bp++ = getbyte ();
                delay (D2BYTES);
                *bp++ = getbyte ();
                delay (D2BYTES);
                *bp++ = getbyte ();
```

```
                delay (D2BYTES);
                *bp++ = getbyte ();
                delay (D2BYTES);
                *bp++ = getbyte ();
                delay (D2BYTES);
                *bp++ = getbyte ();
                delay (D2BYTES);
                *bp++ = getbyte ();
                delay (D2BYTES);
                *bp++ = getbyte ();
                delay (D2SLOW);
                *bp++ = getbyte ();
                delay (D2SLOW);
                *bp++ = getbyte ();
                delay (D2SLOW);
                *bp++ = getbyte ();
                delay (D2SLOW);
                *bp++ = getbyte ();
                delay (D2SLOW);

                /* Glove packet isn't quite as described above */
                /* Let's see if we can figure it out */
                {
                int i,n;

                /* look for FF FF A0 */
                n = -1;
                for (i=0; i<12; ++i) {
                        if (buf[i] == (unsigned char)0xff &&
                            buf[(i+1)%12] == (unsigned char)0xff &&
                            buf[(i+2)%12] == (unsigned char)0xa0) {

                                /* yah! */
                                n = (i+3)%12;

                                printf("%-3d %-3d %-3d     %-3d     %-2x     %-2x\n",
                                        buf[n], buf[(n+1)%12], buf[(n+2)%12],
                                        buf[(n+3)%12],
                                        buf[(n+4)%12],
                                        buf[(n+5)%12]);
                                break;
                        }
                }
                if (n < 0) printf("\033[K\n");
                printf ("Glove %-2x %-2x %-2x %-2x %-2x %-2x %-2x %-2x %-2x %-2x %-2x
%-2x\r\033[A",
                                buf[0], buf[1], buf[2], buf[3], buf[4], buf[5],
                                buf[6], buf[7], buf[8], buf[9], buf[10], buf[11]);
                }
        }
}

unsigned char getbyte ()
{
        register unsigned Glov = 0;

        /* prepare port b as output port */
        setportb ();

        /* generate a reset (latch) pulse */
        C1L0 ();
```

```
C1L1 ();
delay(5); /* 5 us delay */
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
```

```
        setporta ();

        /* read a bit */
        Glov <<= 1;
        Glov |= getbit();

        /* prepare port b as output port */
        setportb ();

        /* generate a clock pulse */
        C0L0 ();
        C1L0 ();

        /* configure port a as input */
        setporta ();

        /* read a bit */
        Glov <<= 1;
        Glov |= getbit();

        /* prepare port b as output port */
        setportb ();

        /* generate a clock pulse */
        C0L0 ();
        C1L0 ();

        /* configure port a as input */
        setporta ();

        /* read a bit */
        Glov <<= 1;
        Glov |= getbit();

        /* prepare port b as output port */
        setportb ();

        /* generate a clock pulse */
        C0L0 ();
        C1L0 ();

        /* configure port a as input */
        setporta ();

        /* read a bit */
        Glov <<= 1;
        Glov |= getbit();

        /* prepare port b as output port */
        setportb ();

        /* generate a clock pulse */
        C0L0 ();
        C1L0 ();

        return (unsigned char) Glov;  /* return the byte */
}


void Hires ()
{
```

```
register unsigned char Glov = 0;

/* initialize hardware interface */
initport();

/* read 4 bits from glove */
setportb ();

/* generate a reset (latch) pulse */
C1L0 ();
C1L1 ();
delay(5);  /* 5 us delay */
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
Glov |= getbit();

/* prepare port b as output port */
setportb ();

/* generate a clock pulse */
C0L0 ();
C1L0 ();

/* configure port a as input */
setporta ();

/* read a bit */
Glov <<= 1;
```

```
        Glov |= getbit();

        /* prepare port b as output port */
        setportb ();

        /* generate a clock pulse */
        C0L0 ();
        C1L0 ();

        /* end of read 4 bits */

        /* prepare port b as output port */
        setportb ();

        C1L0 ();
        delay(7212);
/*      delay (16950);  /* 7212 us delay */

        setportb ();

        C1L1 ();
        delay(2260);
/*      delay (4750);   /* 2260 us delay */

        /* prepare port b as output port */
        setportb ();

        C1L0 ();        /* Start of 1. Byte */
        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);
        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();
```

```
C1L1 ();
C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C1L0 ();
C0L0 ();
C1L0 ();
delay (D2BYTES);

/* prepare port b as output port */
setportb ();

C1L1 ();        /* Start of 2. Byte */
C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C1L0 ();
C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();
```

```
C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C1L1 ();
C0L1 ();
C1L1 ();
delay (D2BYTES);

/* prepare port b as output port */
setportb ();

C1L0 ();        /* Start of 3. Byte */
C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C1L1 ();
C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C1L0 ();
C0L0 ();
```

```
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BYTES);

/* prepare port b as output port */
setportb ();

C0L0 ();        /* start of 4. byte */
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();
```

```
C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BYTES);

/* prepare port b as output port */
setportb ();

C0L0 ();        /* start of 5. byte */
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L0 ();
C1L0 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C1L1 ();
C0L1 ();
C1L1 ();
delay (D2BITS);
```

```
/* prepare port b as output port */
setportb ();

C1L0 ();
C0L0 ();
C1L0 ();
delay (D2BYTES);

/* prepare port b as output port */
setportb ();

C1L1 ();        /* start of 6. byte */
C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();

C0L1 ();
C1L1 ();
delay (D2BITS);

/* prepare port b as output port */
setportb ();
```

```
        C0L1 ();
        C1L1 ();
        delay (D2BYTES);

        /* prepare port b as output port */
        setportb ();

        C1L0 ();        /* start of 7. byte */
        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C0L0 ();
        C1L0 ();
        delay (D2BITS);

        /* prepare port b as output port */
        setportb ();

        C1L1 ();
        C0L1 ();
        C1L1 ();
```

```
        delay(892);
/*      delay (1090); /* 892 us delay (end of 7. byte) */

        /* prepare port b as output port */
        setportb ();

        C1L0 ();
        delay(50000);
/*      delay (60000);   /* some time for the glove controller
                                              to relax */
}
__
```

N e X t !

Jiro Nakamura

The DSP can do only digital-in. It has no analog-to-digital functionality itself. Digging up my NeXT Reference Manual moreover gives the following clues:

Pin 4 of the DSP ("SCLK") -- to Nintendo Pin 2 ("clock")
Pin 5 of the DSP ("RXD") -- to Nintendo Pin 4 ("data")
Pin 6 of the DSP ("TXD") -- to Nintendo Pin 5 ("latch")
Pin 11 of the DSP ("GND") -- to Nintendo Pin 1 ("Gnd")
+5v to Nintendo Pin 7
Gnd to Nintendo Pin 1

For the folks who don't know the ordering of Nintendo pins, it is given in jfreem's library as:

```
   1
7  2
6  3
5  4    (this is looking at the socket head on)
```

Note that although jfreem says that Pin 5 is "latch", it is in fact connected to the TXD (transmit data) port of the DSP -- which seems to indicate that jfreem is in actuality programming the DSP.

Two problems with reverse engineering his design. One is that he has the DSP instructions in a ".snd" (sound) file. This means that I can't use the Ariel 56K debugger provided with the NeXT since it only take .LOD files (the manual says "DSP absolute load image" which is in ASCII format (seems like assembler output to me)). Ariel doesn't provide a binary debugger.

The other is that I'm not sure if jfreem is using the DSP merely as an input device (then cracking the raw data with the 040) or using it also as an input and cracking device.

More news at 11 (hopefully).

- jiro nakamura

jiro@shaman.com

ps. If anyone needs my computer resources on this, they
are more than welcome to ask me. The library code itself is
in .a format (which may or may not be the same as Sun's 68020
.a format, but should still be reverse assembled):

```
shaman> ar t libPowerGlove.a

__.SYMDEF SORTED
PowerGlove.o

shaman> ar xv libPowerGlove.a
x - __.SYMDEF SORTED
x - PowerGlove.o

shaman> more "__.SYMDEF SORTED"
p.objc_class_name_PowerGlove
```

GDB disassembly
==========

```
0x0:    linkw fp,#-8
0x4:    movel a2,-(sp)
0x6:    moveal 8(fp),a2
0xa:    movel @#0xac4,-(sp)
0x10:   movel a2,-8(fp)
0x14:   movel @#0x91c,-4(fp)
0x1c:   pea -8(fp)
0x20:   bsr 0x0
0x26:   movel a2,-(sp)
0x28:   movel @#0xac8,-(sp)
0x2e:   movel a2,-(sp)
0x30:   bsr 0x0
0x36:   moveb #1,37(a2)
0x3c:   moveb #1,36(a2)
0x42:   moveb #1,35(a2)0x48:   movel a2,d0
0x4a:   moveal -12(fp),a2
0x4e:   unlk fp0x50:   rts
0x52 <-[PowerGlove setCmdPort:]>:       linkw fp,#0
0x56 <-[PowerGlove setCmdPort:]+4>:     moveal 8(fp),a0
0x5a <-[PowerGlove setCmdPort:]+8>:     movel 16(fp),8(a0)
0x60 <-[PowerGlove setCmdPort:]+14>:    movel a0,d0
0x62 <-[PowerGlove setCmdPort:]+16>:    unlk fp
0x64 <-[PowerGlove setCmdPort:]+18>:    rts
0x66 <for_loop>:        linkw fp,#-16
0x6a <for_loop+4>:      moveml #14368,-(sp)
0x6e <for_loop+8>:      movel 8(fp),d4
0x72 <for_loop+12>:     clrl d3
0x74 <for_loop+14>:     clrl -4(fp)
0x78 <for_loop+18>:     clrl -8(fp)
0x7c <for_loop+22>:     pea -8(fp)
0x80 <for_loop+26>:     pea -4(fp)
0x84 <for_loop+30>:     clrl -(sp)
0x86 <for_loop+32>:     clrl -(sp)
0x88 <for_loop+34>:     clrl -(sp)
0x8a <for_loop+36>:     clrl -(sp)
0x8c <for_loop+38>:     clrl -(sp)
0x8e <for_loop+40>:     pea @#0x2
0x92 <for_loop+44>:     bsr 0x0
```

```
0x98 <for_loop+50>:     movel d0,d2
0x9a <for_loop+52>:     addaw #32,sp
0x9e <for_loop+56>:     beq 0xc6 <for_loop+96>
0xa0 <for_loop+58>:     movel d2,-(sp)
0xa2 <for_loop+60>:     bsr 0x0
0xa8 <for_loop+66>:     movel d0,-(sp)
0xaa <for_loop+68>:     pea @#0x7b0
0xb0 <for_loop+74>:     pea @#0x28
0xb6 <for_loop+80>:     bsr 0x0
0xbc <for_loop+86>:     pea @#0x1
0xc0 <for_loop+90>:     bsr 0x0
0xc6 <for_loop+96>:     pea -12(fp)
0xca <for_loop+100>:    movel -8(fp),-(sp)
0xce <for_loop+104>:    movel -4(fp),-(sp)
0xd2 <for_loop+108>:    bsr 0x0
0xd8 <for_loop+114>:    addaw #12,sp
0xdc <for_loop+118>:    tstl d0
0xde <for_loop+120>:    beq 0xf8 <for_loop+146>
0xe0 <for_loop+122>:    movel d0,-(sp)
0xe2 <for_loop+124>:    pea @#0x7da
0xe8 <for_loop+130>:    bsr 0x0
0xee <for_loop+136>:    pea @#0x1
0xf2 <for_loop+140>:    bsr 0x0
0xf8 <for_loop+146>:    movel -12(fp),-(sp)
0xfc <for_loop+150>:    movel @#0xacc,-(sp)
0x102 <for_loop+156>:   movel d4,-(sp)
0x104 <for_loop+158>:   lea @#0x0,a2
0x10a <for_loop+164>:   jsr (a2)
0x10c <for_loop+166>:   clrl -(sp)
0x10e <for_loop+168>:   pea -16(fp)
0x112 <for_loop+172>:   pea @#0x7f7
0x118 <for_loop+178>:   bsr 0x0
0x11e <for_loop+184>:   movel d0,d2
0x120 <for_loop+186>:   addaw #24,sp
0x124 <for_loop+190>:   beq 0x182 <for_loop+284>
0x126 <for_loop+192>:   pea @#0x806
0x12c <for_loop+198>:   movel @#0xad0,-(sp)
0x132 <for_loop+204>:   pea @#0x815
0x138 <for_loop+210>:   bsr 0x0
0x13e <for_loop+216>:   addqw #4,sp
0x140 <for_loop+218>:   movel d0,-(sp)
0x142 <for_loop+220>:   jsr (a2)
0x144 <for_loop+222>:   movel d0,d3
0x146 <for_loop+224>:   movel @#0xad4,-(sp)
0x14c <for_loop+230>:   movel d3,-(sp)
0x14e <for_loop+232>:   jsr (a2)
0x150 <for_loop+234>:   movel d0,-16(fp)
0x154 <for_loop+238>:   addaw #20,sp
0x158 <for_loop+242>:   tstl d3
0x15a <for_loop+244>:   bne 0x182 <for_loop+284>
0x15c <for_loop+246>:   movel d2,-(sp)
0x15e <for_loop+248>:   bsr 0x0
0x164 <for_loop+254>:   movel d0,-(sp)
0x166 <for_loop+256>:   pea @#0x81b
0x16c <for_loop+262>:   pea @#0x28
0x172 <for_loop+268>:   bsr 0x0
0x178 <for_loop+274>:   pea @#0x1
0x17c <for_loop+278>:   bsr 0x0
0x182 <for_loop+284>:   movel -16(fp),-(sp)
0x186 <for_loop+288>:   movel -8(fp),-(sp)
0x18a <for_loop+292>:   movel -4(fp),-(sp)
```

```
0x18e <for_loop+296>:   bsr 0x0
0x194 <for_loop+302>:   movel d0,d2
0x196 <for_loop+304>:   addaw #12,sp
0x19a <for_loop+308>:   beq 0x1c2 <for_loop+348>
0x19c <for_loop+310>:   movel d2,-(sp)
0x19e <for_loop+312>:   bsr 0x0
0x1a4 <for_loop+318>:   movel d0,-(sp)
0x1a6 <for_loop+320>:   pea @#0x83d
0x1ac <for_loop+326>:   pea @#0x28
0x1b2 <for_loop+332>:   bsr 0x0
0x1b8 <for_loop+338>:   pea @#0x1
0x1bc <for_loop+342>:   bsr 0x0
0x1c2 <for_loop+348>:   tstl d3
0x1c4 <for_loop+350>:   beq 0x1d8 <for_loop+370>
0x1c6 <for_loop+352>:   movel @#0xad8,-(sp)
0x1cc <for_loop+358>:   movel d3,-(sp)
0x1ce <for_loop+360>:   bsr 0x0
0x1d4 <for_loop+366>:   addqw #8,sp
0x1d6 <for_loop+368>:   bra 0x1e4 <for_loop+382>
0x1d8 <for_loop+370>:   movel -16(fp),-(sp)
0x1dc <for_loop+374>:   bsr 0x0
0x1e2 <for_loop+380>:   addqw #4,sp
0x1e4 <for_loop+382>:   pea @#0x1
0x1e8 <for_loop+386>:   movel -8(fp),-(sp)
0x1ec <for_loop+390>:   movel -4(fp),-(sp)
0x1f0 <for_loop+394>:   bsr 0x0
0x1f6 <for_loop+400>:   addaw #12,sp
0x1fa <for_loop+404>:   tstl d0
0x1fc <for_loop+406>:   beq 0x22c <for_loop+454>
0x1fe <for_loop+408>:   movel d0,-(sp)
0x200 <for_loop+410>:   pea @#0x853
0x206 <for_loop+416>:   bsr 0x0
0x20c <for_loop+422>:   pea @#0x1
0x210 <for_loop+426>:   bsr 0x0
0x216 <for_loop+432>:   bsr 0x0
0x21c <for_loop+438>:   movel @#0xae0,-(sp)
0x222 <for_loop+444>:   movel d4,-(sp)
0x224 <for_loop+446>:   bsr 0x0
0x22a <for_loop+452>:   addqw #8,sp
0x22c <for_loop+454>:   movel @#0xadc,-(sp)
0x232 <for_loop+460>:   movel d4,-(sp)
0x234 <for_loop+462>:   bsr 0x0
0x23a <for_loop+468>:   addqw #8,sp
0x23c <for_loop+470>:   tstb d0
0x23e <for_loop+472>:   bne 0x216 <for_loop+432>
0x240 <for_loop+474>:   pea @#0x1
0x244 <for_loop+478>:   movel #134217728,-(sp)
0x24a <for_loop+484>:   movel #134217728,-(sp)
0x250 <for_loop+490>:   movel -12(fp),-(sp)
0x254 <for_loop+494>:   bsr 0x0
0x25a <for_loop+500>:   addaw #16,sp
0x25e <for_loop+504>:   tstl d0
0x260 <for_loop+506>:   beq 0x272 <for_loop+524>
0x262 <for_loop+508>:   movel d0,-(sp)
0x264 <for_loop+510>:   pea @#0x868
0x26a <for_loop+516>:   bsr 0x0
0x270 <for_loop+522>:   addqw #8,sp
0x272 <for_loop+524>:   movel -12(fp),-(sp)
0x276 <for_loop+528>:   movel @#0x0,-(sp)
0x27c <for_loop+534>:   bsr 0x0
0x282 <for_loop+540>:   addqw #8,sp
```

```
0x284 <for_loop+542>:   tstl d0
0x286 <for_loop+544>:   beq 0x298 <for_loop+562>
0x288 <for_loop+546>:   movel d0,-(sp)
0x28a <for_loop+548>:   pea @#0x87d
0x290 <for_loop+554>:   bsr 0x0
0x296 <for_loop+560>:   addqw #8,sp
0x298 <for_loop+562>:   movel -8(fp),-(sp)
0x29c <for_loop+566>:   movel @#0x0,-(sp)
0x2a2 <for_loop+572>:   bsr 0x0
0x2a8 <for_loop+578>:   addqw #8,sp
0x2aa <for_loop+580>:   tstl d0
0x2ac <for_loop+582>:   beq 0x2be <for_loop+600>
0x2ae <for_loop+584>:   movel d0,-(sp)
0x2b0 <for_loop+586>:   pea @#0x89b
0x2b6 <for_loop+592>:   bsr 0x0
0x2bc <for_loop+598>:   addqw #8,sp
0x2be <for_loop+600>:   movel -4(fp),-(sp)
0x2c2 <for_loop+604>:   movel @#0x0,-(sp)
0x2c8 <for_loop+610>:   bsr 0x0
0x2ce <for_loop+616>:   addqw #8,sp
0x2d0 <for_loop+618>:   tstl d0
0x2d2 <for_loop+620>:   beq 0x2e4 <for_loop+638>
0x2d4 <for_loop+622>:   movel d0,-(sp)
0x2d6 <for_loop+624>:   pea @#0x8bb
0x2dc <for_loop+630>:   bsr 0x0
0x2e2 <for_loop+636>:   addqw #8,sp
0x2e4 <for_loop+638>:   clrl -(sp)
0x2e6 <for_loop+640>:   bsr 0x0
0x2ec <for_loop+646>:   moveml -32(fp),#1052
0x2f2 <for_loop+652>:   unlk fp
0x2f4 <for_loop+654>:   rts
0x2f6 <-[PowerGlove startDSP:]>:        linkw fp,#0
0x2fa <-[PowerGlove startDSP:]+4>:      moveml #8224,-(sp)
0x2fe <-[PowerGlove startDSP:]+8>:      moveal 8(fp),a2
0x302 <-[PowerGlove startDSP:]+12>:     addql #1,20(a2)
0x306 <-[PowerGlove startDSP:]+16>:     movel #1,d1
0x308 <-[PowerGlove startDSP:]+18>:     cmpl 20(a2),d1
0x30c <-[PowerGlove startDSP:]+22>:     bge 0x312 <-[PowerGlove startDSP:]+28>
0x30e <-[PowerGlove startDSP:]+24>:     clrl d0
0x310 <-[PowerGlove startDSP:]+26>:     bra 0x33a <-[PowerGlove startDSP:]+68>
0x312 <-[PowerGlove startDSP:]+28>:     movel a2,-(sp)
0x314 <-[PowerGlove startDSP:]+30>:     pea @#0x66 <for_loop>
0x31a <-[PowerGlove startDSP:]+36>:     bsr 0x0
0x320 <-[PowerGlove startDSP:]+42>:     movel d0,d2
0x322 <-[PowerGlove startDSP:]+44>:     pea @#0x8d9
0x328 <-[PowerGlove startDSP:]+50>:     movel d2,-(sp)
0x32a <-[PowerGlove startDSP:]+52>:     bsr 0x0
0x330 <-[PowerGlove startDSP:]+58>:     movel d2,-(sp)
0x332 <-[PowerGlove startDSP:]+60>:     bsr 0x0
0x338 <-[PowerGlove startDSP:]+66>:     movel a2,d0
0x33a <-[PowerGlove startDSP:]+68>:     moveml -8(fp),#1028
0x340 <-[PowerGlove startDSP:]+74>:     unlk fp
0x342 <-[PowerGlove startDSP:]+76>:     rts
0x344 <-[PowerGlove stopDSP:]>: linkw fp,#0
0x348 <-[PowerGlove stopDSP:]+4>:       moveal 8(fp),a0
0x34c <-[PowerGlove stopDSP:]+8>:       tstl 20(a0)
0x350 <-[PowerGlove stopDSP:]+12>:      beq 0x35c <-[PowerGlove stopDSP:]+24>
0x352 <-[PowerGlove stopDSP:]+14>:      subl #1,20(a0)
0x356 <-[PowerGlove stopDSP:]+18>:      tstl 20(a0)
0x35a <-[PowerGlove stopDSP:]+22>:      ble 0x360 <-[PowerGlove stopDSP:]+28>
0x35c <-[PowerGlove stopDSP:]+24>:      clrl d0
```

```
0x35e <-[PowerGlove stopDSP:]+26>:      bra 0x362 <-[PowerGlove stopDSP:]+30>
0x360 <-[PowerGlove stopDSP:]+28>:      movel a0,d0
0x362 <-[PowerGlove stopDSP:]+30>:      unlk fp
0x364 <-[PowerGlove stopDSP:]+32>:      rts
0x366 <-[PowerGlove feedGloveToMouse:]>:        linkw fp,#0
0x36a <-[PowerGlove feedGloveToMouse:]+4>:      movel a2,-(sp)
0x36c <-[PowerGlove feedGloveToMouse:]+6>:      moveal 8(fp),a2
0x370 <-[PowerGlove feedGloveToMouse:]+10>:     moveb 19(fp),d0
0x374 <-[PowerGlove feedGloveToMouse:]+14>:     moveb d0,d1
0x376 <-[PowerGlove feedGloveToMouse:]+16>:     extbl d1
0x378 <-[PowerGlove feedGloveToMouse:]+18>:     movel d1,12(a2)
0x37c <-[PowerGlove feedGloveToMouse:]+22>:     tstb d0
0x37e <-[PowerGlove feedGloveToMouse:]+24>:     beq 0x3b2 <-[PowerGlove
feedGloveToMouse:]+76>
0x380 <-[PowerGlove feedGloveToMouse:]+26>:     tstl 16(a2)
0x384 <-[PowerGlove feedGloveToMouse:]+30>:     bgt 0x3b2 <-[PowerGlove
feedGloveToMouse:]+76>
0x386 <-[PowerGlove feedGloveToMouse:]+32>:     pea @#0x2
0x38a <-[PowerGlove feedGloveToMouse:]+36>:     pea @#0x8e2
0x390 <-[PowerGlove feedGloveToMouse:]+42>:     bsr 0x0
0x396 <-[PowerGlove feedGloveToMouse:]+48>:     movel d0,16(a2)
0x39a <-[PowerGlove feedGloveToMouse:]+52>:     addqw #8,sp
0x39c <-[PowerGlove feedGloveToMouse:]+54>:     bge 0x3c6 <-[PowerGlove
feedGloveToMouse:]+96>
0x39e <-[PowerGlove feedGloveToMouse:]+56>:     pea @#0x8ec
0x3a4 <-[PowerGlove feedGloveToMouse:]+62>:     pea @#0x28
0x3aa <-[PowerGlove feedGloveToMouse:]+68>:     bsr 0x0
0x3b0 <-[PowerGlove feedGloveToMouse:]+74>:     bra 0x3c6 <-[PowerGlove
feedGloveToMouse:]+96>
0x3b2 <-[PowerGlove feedGloveToMouse:]+76>:     tstl 16(a2)
0x3b6 <-[PowerGlove feedGloveToMouse:]+80>:     blt 0x3c2 <-[PowerGlove
feedGloveToMouse:]+92>
0x3b8 <-[PowerGlove feedGloveToMouse:]+82>:     movel 16(a2),-(sp)
0x3bc <-[PowerGlove feedGloveToMouse:]+86>:     bsr 0x0
0x3c2 <-[PowerGlove feedGloveToMouse:]+92>:     clrl 16(a2)
0x3c6 <-[PowerGlove feedGloveToMouse:]+96>:     movel a2,d0
0x3c8 <-[PowerGlove feedGloveToMouse:]+98>:     moveal -4(fp),a2
0x3cc <-[PowerGlove feedGloveToMouse:]+102>:    unlk fp
0x3ce <-[PowerGlove feedGloveToMouse:]+104>:    rts
0x3d0 <-[PowerGlove free]>:     linkw fp,#-8
0x3d4 <-[PowerGlove free]+4>:   moveml #8224,-(sp)
0x3d8 <-[PowerGlove free]+8>:   movel 8(fp),d2
0x3dc <-[PowerGlove free]+12>:  clrl -(sp)
0x3de <-[PowerGlove free]+14>:  movel @#0xae4,-(sp)
0x3e4 <-[PowerGlove free]+20>:  movel d2,-(sp)
0x3e6 <-[PowerGlove free]+22>:  lea @#0x0,a2
0x3ec <-[PowerGlove free]+28>:  jsr (a2)
0x3ee <-[PowerGlove free]+30>:  movel d2,-(sp)
0x3f0 <-[PowerGlove free]+32>:  movel @#0xae8,-(sp)
0x3f6 <-[PowerGlove free]+38>:  movel d2,-(sp)
0x3f8 <-[PowerGlove free]+40>:  jsr (a2)
0x3fa <-[PowerGlove free]+42>:  movel @#0xad8,-(sp)
0x400 <-[PowerGlove free]+48>:  movel d2,-8(fp)
0x404 <-[PowerGlove free]+52>:  movel @#0x91c,-4(fp)
0x40c <-[PowerGlove free]+60>:  pea -8(fp)
0x410 <-[PowerGlove free]+64>:  bsr 0x0
0x416 <-[PowerGlove free]+70>:  movel d2,d0
0x418 <-[PowerGlove free]+72>:  moveml -16(fp),#1028
0x41e <-[PowerGlove free]+78>:  unlk fp
0x420 <-[PowerGlove free]+80>:  rts
0x422 <-[PowerGlove feedingMouse]>:     linkw fp,#0
```

```
0x426 <-[PowerGlove feedingMouse]+4>:   moveal 8(fp),a0
0x42a <-[PowerGlove feedingMouse]+8>:   moveb 15(a0),d0
0x42e <-[PowerGlove feedingMouse]+12>:  extbl d0
0x430 <-[PowerGlove feedingMouse]+14>:  unlk fp
0x432 <-[PowerGlove feedingMouse]+16>:  rts
0x434 <-[PowerGlove dspIsRunning]>:     linkw fp,#0
0x438 <-[PowerGlove dspIsRunning]+4>:   moveal 8(fp),a0
0x43c <-[PowerGlove dspIsRunning]+8>:   tstl 20(a0)
0x440 <-[PowerGlove dspIsRunning]+12>:  ble 0x446 <-[PowerGlove
dspIsRunning]+18>
0x442 <-[PowerGlove dspIsRunning]+14>:  movel #1,d0
0x444 <-[PowerGlove dspIsRunning]+16>:  bra 0x448 <-[PowerGlove
dspIsRunning]+20>
0x446 <-[PowerGlove dspIsRunning]+18>:  clrl d0
0x448 <-[PowerGlove dspIsRunning]+20>:  unlk fp
0x44a <-[PowerGlove dspIsRunning]+22>:  rts
0x44c <-[PowerGlove getDataByte:]>:     linkw fp,#0
0x450 <-[PowerGlove getDataByte:]+4>:   movel 16(fp),d0
0x454 <-[PowerGlove getDataByte:]+8>:   movel #7,d1
0x456 <-[PowerGlove getDataByte:]+10>:  cmpl d0,d1
0x458 <-[PowerGlove getDataByte:]+12>:  blt 0x45e <-[PowerGlove
getDataByte:]+18>
0x45a <-[PowerGlove getDataByte:]+14>:  tstl d0
0x45c <-[PowerGlove getDataByte:]+16>:  bge 0x462 <-[PowerGlove
getDataByte:]+22>
0x45e <-[PowerGlove getDataByte:]+18>:  clrl d0
0x460 <-[PowerGlove getDataByte:]+20>:  bra 0x46c <-[PowerGlove
getDataByte:]+32>
0x462 <-[PowerGlove getDataByte:]+22>:  moveal 8(fp),a0
0x466 <-[PowerGlove getDataByte:]+26>:  moveb 26(a0)[d0.l],d0
0x46a <-[PowerGlove getDataByte:]+30>:  extbl d0
0x46c <-[PowerGlove getDataByte:]+32>:  unlk fp
0x46e <-[PowerGlove getDataByte:]+34>:  rts
0x470 <-[PowerGlove gloveMoved]>:       linkw fp,#0
0x474 <-[PowerGlove gloveMoved]+4>:     bsr 0x0
0x47a <-[PowerGlove gloveMoved]+10>:    moveal 8(fp),a0
0x47e <-[PowerGlove gloveMoved]+14>:    moveb 24(a0),d0
0x482 <-[PowerGlove gloveMoved]+18>:    extbl d0
0x484 <-[PowerGlove gloveMoved]+20>:    unlk fp
0x486 <-[PowerGlove gloveMoved]+22>:    rts
0x488 <-[PowerGlove setxHyst:yHyst:zHyst:]>:    linkw fp,#0
0x48c <-[PowerGlove setxHyst:yHyst:zHyst:]+4>:  moveal 8(fp),a0
0x490 <-[PowerGlove setxHyst:yHyst:zHyst:]+8>:  moveb 23(fp),d0
0x494 <-[PowerGlove setxHyst:yHyst:zHyst:]+12>: moveb 27(fp),d1
0x498 <-[PowerGlove setxHyst:yHyst:zHyst:]+16>: moveb 19(fp),35(a0)
0x49e <-[PowerGlove setxHyst:yHyst:zHyst:]+22>: moveb d0,36(a0)
0x4a2 <-[PowerGlove setxHyst:yHyst:zHyst:]+26>: moveb d1,37(a0)
0x4a6 <-[PowerGlove setxHyst:yHyst:zHyst:]+30>: movel a0,d0
0x4a8 <-[PowerGlove setxHyst:yHyst:zHyst:]+32>: unlk fp
0x4aa <-[PowerGlove setxHyst:yHyst:zHyst:]+34>: rts
0x4ac <-[PowerGlove fingerMoved]>:      linkw fp,#0
0x4b0 <-[PowerGlove fingerMoved]+4>:    moveal 8(fp),a0
0x4b4 <-[PowerGlove fingerMoved]+8>:    moveb 25(a0),d0
0x4b8 <-[PowerGlove fingerMoved]+12>:   extbl d0
0x4ba <-[PowerGlove fingerMoved]+14>:   unlk fp
0x4bc <-[PowerGlove fingerMoved]+16>:   rts
0x4be <-[PowerGlove gloveX]>:   linkw fp,#0
0x4c2 <-[PowerGlove gloveX]+4>: moveal 8(fp),a0
0x4c6 <-[PowerGlove gloveX]+8>: clrb 24(a0)
0x4ca <-[PowerGlove gloveX]+12>:        moveb 27(a0),d0
0x4ce <-[PowerGlove gloveX]+16>:        extbl d0
```

```
0x4d0 <-[PowerGlove gloveX]+18>:        unlk fp
0x4d2 <-[PowerGlove gloveX]+20>:        rts
0x4d4 <-[PowerGlove gloveY]>:   linkw fp,#0
0x4d8 <-[PowerGlove gloveY]+4>: moveal 8(fp),a0
0x4dc <-[PowerGlove gloveY]+8>: clrb 24(a0)
0x4e0 <-[PowerGlove gloveY]+12>:        moveb 28(a0),d0
0x4e4 <-[PowerGlove gloveY]+16>:        extbl d0
0x4e6 <-[PowerGlove gloveY]+18>:        unlk fp
0x4e8 <-[PowerGlove gloveY]+20>:        rts
0x4ea <-[PowerGlove gloveZ]>:   linkw fp,#0
0x4ee <-[PowerGlove gloveZ]+4>: moveal 8(fp),a0
0x4f2 <-[PowerGlove gloveZ]+8>: clrb 24(a0)
0x4f6 <-[PowerGlove gloveZ]+12>:        moveb 29(a0),d0
0x4fa <-[PowerGlove gloveZ]+16>:        extbl d0
0x4fc <-[PowerGlove gloveZ]+18>:        unlk fp
0x4fe <-[PowerGlove gloveZ]+20>:        rts
0x500 <-[PowerGlove gloveRot]>: linkw fp,#0
0x504 <-[PowerGlove gloveRot]+4>:       moveal 8(fp),a0
0x508 <-[PowerGlove gloveRot]+8>:       moveb 30(a0),d0
0x50c <-[PowerGlove gloveRot]+12>:      extbl d0
0x50e <-[PowerGlove gloveRot]+14>:      unlk fp
0x510 <-[PowerGlove gloveRot]+16>:      rts
0x512 <-[PowerGlove thumb]>:    linkw fp,#0
0x516 <-[PowerGlove thumb]+4>:  moveal 8(fp),a0
0x51a <-[PowerGlove thumb]+8>:  clrb 25(a0)
0x51e <-[PowerGlove thumb]+12>: moveb 31(a0),d0
0x522 <-[PowerGlove thumb]+16>: asrb #6,d0
0x524 <-[PowerGlove thumb]+18>: movel #3,d1
0x526 <-[PowerGlove thumb]+20>: andl d1,d0
0x528 <-[PowerGlove thumb]+22>: unlk fp
0x52a <-[PowerGlove thumb]+24>: rts
0x52c <-[PowerGlove index]>:    linkw fp,#0
0x530 <-[PowerGlove index]+4>:  moveal 8(fp),a0
0x534 <-[PowerGlove index]+8>:  clrb 25(a0)
0x538 <-[PowerGlove index]+12>: moveb 31(a0),d0
0x53c <-[PowerGlove index]+16>: asrb #4,d0
0x53e <-[PowerGlove index]+18>: movel #3,d1
0x540 <-[PowerGlove index]+20>: andl d1,d0
0x542 <-[PowerGlove index]+22>: unlk fp
0x544 <-[PowerGlove index]+24>: rts
0x546 <-[PowerGlove middle]>:   linkw fp,#0
0x54a <-[PowerGlove middle]+4>: moveal 8(fp),a0
0x54e <-[PowerGlove middle]+8>: clrb 25(a0)
0x552 <-[PowerGlove middle]+12>:        moveb 31(a0),d0
0x556 <-[PowerGlove middle]+16>:        asrb #2,d0
0x558 <-[PowerGlove middle]+18>:        movel #3,d1
0x55a <-[PowerGlove middle]+20>:        andl d1,d0
0x55c <-[PowerGlove middle]+22>:        unlk fp
0x55e <-[PowerGlove middle]+24>:        rts
0x560 <-[PowerGlove ring]>:     linkw fp,#0
0x564 <-[PowerGlove ring]+4>:   moveal 8(fp),a0
0x568 <-[PowerGlove ring]+8>:   clrb 25(a0)
0x56c <-[PowerGlove ring]+12>:  moveb 31(a0),d0
0x570 <-[PowerGlove ring]+16>:  movel #3,d1
0x572 <-[PowerGlove ring]+18>:  andl d1,d0
0x574 <-[PowerGlove ring]+20>:  unlk fp
0x576 <-[PowerGlove ring]+22>:  rts
0x578 <-[PowerGlove padUp]>:    linkw fp,#0
0x57c <-[PowerGlove padUp]+4>:  moveal 8(fp),a0
0x580 <-[PowerGlove padUp]+8>:  cmpib #13,32(a0)
0x586 <-[PowerGlove padUp]+14>: bne 0x58c <-[PowerGlove padUp]+20>
```

```
0x588 <-[PowerGlove padUp]+16>: movel #1,d0
0x58a <-[PowerGlove padUp]+18>: bra 0x58e <-[PowerGlove padUp]+22>
0x58c <-[PowerGlove padUp]+20>: clrl d0
0x58e <-[PowerGlove padUp]+22>: unlk fp
0x590 <-[PowerGlove padUp]+24>: rts
0x592 <-[PowerGlove padDown]>:  linkw fp,#0
0x596 <-[PowerGlove padDown]+4>:        moveal 8(fp),a0
0x59a <-[PowerGlove padDown]+8>:        cmpib #14,32(a0)
0x5a0 <-[PowerGlove padDown]+14>:       bne 0x5a6 <-[PowerGlove padDown]+20>
0x5a2 <-[PowerGlove padDown]+16>:       movel #1,d0
0x5a4 <-[PowerGlove padDown]+18>:       bra 0x5a8 <-[PowerGlove padDown]+22>
0x5a6 <-[PowerGlove padDown]+20>:       clrl d0
0x5a8 <-[PowerGlove padDown]+22>:       unlk fp
0x5aa <-[PowerGlove padDown]+24>:       rts
0x5ac <-[PowerGlove padLeft]>:  linkw fp,#0
0x5b0 <-[PowerGlove padLeft]+4>:        moveal 8(fp),a0
0x5b4 <-[PowerGlove padLeft]+8>:        cmpib #12,32(a0)
0x5ba <-[PowerGlove padLeft]+14>:       bne 0x5c0 <-[PowerGlove padLeft]+20>
0x5bc <-[PowerGlove padLeft]+16>:       movel #1,d0
0x5be <-[PowerGlove padLeft]+18>:       bra 0x5c2 <-[PowerGlove padLeft]+22>
0x5c0 <-[PowerGlove padLeft]+20>:       clrl d0
0x5c2 <-[PowerGlove padLeft]+22>:       unlk fp
0x5c4 <-[PowerGlove padLeft]+24>:       rts
0x5c6 <-[PowerGlove padRight]>: linkw fp,#0
0x5ca <-[PowerGlove padRight]+4>:       moveal 8(fp),a0
0x5ce <-[PowerGlove padRight]+8>:       cmpib #15,32(a0)
0x5d4 <-[PowerGlove padRight]+14>:      bne 0x5da <-[PowerGlove padRight]+20>
0x5d6 <-[PowerGlove padRight]+16>:      movel #1,d0
0x5d8 <-[PowerGlove padRight]+18>:      bra 0x5dc <-[PowerGlove padRight]+22>
0x5da <-[PowerGlove padRight]+20>:      clrl d0
0x5dc <-[PowerGlove padRight]+22>:      unlk fp
0x5de <-[PowerGlove padRight]+24>:      rts
0x5e0 <-[PowerGlove aPressed]>: linkw fp,#0
0x5e4 <-[PowerGlove aPressed]+4>:       moveal 8(fp),a0
0x5e8 <-[PowerGlove aPressed]+8>:       cmpib #10,32(a0)
0x5ee <-[PowerGlove aPressed]+14>:      bne 0x5f4 <-[PowerGlove aPressed]+20>
0x5f0 <-[PowerGlove aPressed]+16>:      movel #1,d0
0x5f2 <-[PowerGlove aPressed]+18>:      bra 0x5f6 <-[PowerGlove aPressed]+22>
0x5f4 <-[PowerGlove aPressed]+20>:      clrl d0
0x5f6 <-[PowerGlove aPressed]+22>:      unlk fp
0x5f8 <-[PowerGlove aPressed]+24>:      rts
0x5fa <-[PowerGlove bPressed]>: linkw fp,#0
0x5fe <-[PowerGlove bPressed]+4>:       moveal 8(fp),a0
0x602 <-[PowerGlove bPressed]+8>:       cmpib #11,32(a0)
0x608 <-[PowerGlove bPressed]+14>:      bne 0x60e <-[PowerGlove bPressed]+20>
0x60a <-[PowerGlove bPressed]+16>:      movel #1,d0
0x60c <-[PowerGlove bPressed]+18>:      bra 0x610 <-[PowerGlove bPressed]+22>
0x60e <-[PowerGlove bPressed]+20>:      clrl d0
0x610 <-[PowerGlove bPressed]+22>:      unlk fp
0x612 <-[PowerGlove bPressed]+24>:      rts
0x614 <-[PowerGlove startPressed]>:     linkw fp,#0
0x618 <-[PowerGlove startPressed]+4>:   moveal 8(fp),a0
0x61c <-[PowerGlove startPressed]+8>:   cmpib #-126,32(a0)
0x622 <-[PowerGlove startPressed]+14>:  bne 0x628 <-[PowerGlove
startPressed]+20>
0x624 <-[PowerGlove startPressed]+16>:  movel #1,d0
0x626 <-[PowerGlove startPressed]+18>:  bra 0x62a <-[PowerGlove
startPressed]+22>
0x628 <-[PowerGlove startPressed]+20>:  clrl d0
0x62a <-[PowerGlove startPressed]+22>:  unlk fp
0x62c <-[PowerGlove startPressed]+24>:  rts
```

```
0x62e <-[PowerGlove selectPressed]>:    linkw fp,#0
0x632 <-[PowerGlove selectPressed]+4>:  moveal 8(fp),a0
0x636 <-[PowerGlove selectPressed]+8>:  cmpib #-125,32(a0)
0x63c <-[PowerGlove selectPressed]+14>: bne 0x642 <-[PowerGlove
selectPressed]+20>
0x63e <-[PowerGlove selectPressed]+16>: movel #1,d0
0x640 <-[PowerGlove selectPressed]+18>: bra 0x644 <-[PowerGlove
selectPressed]+22>
0x642 <-[PowerGlove selectPressed]+20>: clrl d0
0x644 <-[PowerGlove selectPressed]+22>: unlk fp
0x646 <-[PowerGlove selectPressed]+24>: rts
0x648 <-[PowerGlove selectPressed]+26>: linkw fp,#-36
0x64c:  moveml #8224,-(sp)
0x650:  moveal 8(fp),a2
0x654:  tstl 20(a2)
0x658:  bgt 0x660
0x65a:  clrl d0
0x65c:  bra 0x7a6
0x660:  movel #8,d2
0x662:  movel d2,-36(fp)
0x666:  pea @#0x1
0x66a:  pea @#0x4
0x66e:  pea -36(fp)
0x672:  pea -32(fp)
0x676:  movel 8(a2),-(sp)
0x67a:  bsr 0x0
0x680:  addaw #20,sp
0x684:  tstl d0
0x686:  bne 0x68e
0x688:  cmpl -36(fp),d2
0x68c:  beq 0x6a6
0x68e:  movel d0,-(sp)
0x690:  pea @#0x901
0x696:  bsr 0x0
0x69c:  pea @#0x1
0x6a0:  bsr 0x0
0x6a6:  moveb -29(fp),26(a2)
0x6ac:  moveb 27(a2),d0
0x6b0:  extbl d0
0x6b2:  moveb -25(fp),d1
0x6b6:  extbl d1
0x6b8:  subl d1,d0
0x6ba:  movel d0,-(sp)
0x6bc:  bsr 0x0
0x6c2:  moveb 35(a2),d1
0x6c6:  extbl d1
0x6c8:  addqw #4,sp
0x6ca:  cmpl d0,d1
0x6cc:  bge 0x6f2
0x6ce:  moveb #1,24(a2)
0x6d4:  moveb 27(a2),d0
0x6d8:  extbl d0
0x6da:  moveb -25(fp),d1
0x6de:  extbl d1
0x6e0:  moveal d1,a0
0x6e2:  lea 0(a0)[d0.l*2],a0
0x6e6:  movel a0,d0
0x6e8:  movel #3,d2
0x6ea:  divsll d2,d0,d0
0x6ee:  moveb d0,27(a2)
0x6f2:  moveb 28(a2),d0
```

```
0x6f6:  extbl d0
0x6f8:  subl -24(fp),d0
0x6fc:  movel d0,-(sp)
0x6fe:  bsr 0x0
0x704:  moveb 36(a2),d1
0x708:  extbl d1
0x70a:  addqw #4,sp
0x70c:  cmpl d0,d1
0x70e:  bge 0x734
0x710:  moveb #1,24(a2)
0x716:  moveb 28(a2),d0
0x71a:  extbl d0
0x71c:  moveb -21(fp),d1
0x720:  extbl d1
0x722:  moveal d1,a0
0x724:  lea 0(a0)[d0.l*2],a0
0x728:  movel a0,d0
0x72a:  movel #3,d2
0x72c:  divsll d2,d0,d0
0x730:  moveb d0,28(a2)
0x734:  moveb 29(a2),d0
0x738:  extbl d0
0x73a:  moveb -17(fp),d1
0x73e:  extbl d1
0x740:  subl d1,d0
0x742:  movel d0,-(sp)
0x744:  bsr 0x0
0x74a:  moveb 37(a2),d1
0x74e:  extbl d1
0x750:  cmpl d0,d1
0x752:  bge 0x778
0x754:  moveb #1,24(a2)
0x75a:  moveb 29(a2),d0
0x75e:  extbl d0
0x760:  moveb -17(fp),d1
0x764:  extbl d1
0x766:  moveal d1,a0
0x768:  lea 0(a0)[d0.l*2],a0
0x76c:  movel a0,d0
0x76e:  movel #3,d2
0x770:  divsll d2,d0,d0
0x774:  moveb d0,29(a2)
0x778:  moveb -13(fp),30(a2)
0x77e:  moveb 31(a2),d2
0x782:  cmpb -9(fp),d2
0x786:  beq 0x794
0x788:  moveb #1,25(a2)
0x78e:  moveb -9(fp),31(a2)
0x794:  movel #6,d0
0x796:  moveb 65507(fp)[d0.l*4],26(a2)[d0.l]
0x79c:  addql #1,d0
0x79e:  movel #7,d2
0x7a0:  cmpl d0,d2
0x7a2:  bge 0x796
0x7a4:  movel a2,d0
0x7a6:  moveml -44(fp),#1028
0x7ac:  unlk fp
0x7ae:  rts
0x7b0:  movel 10784(a2),d5
0x7b4:  041557
0x7b6:  072554
```

```
0x7b8:  bcc 0x7da
0x7ba:  bgt 0x82b
0x7bc:  movel #32,d2
0x7be:  bsr 0x823
0x7c0:  070565
0x7c2:  bvs 0x836
0x7c4:  bcs 0x7e6
0x7c6:  negw (a3)
0x7c8:  addqb #8,a2
0x7ca:  subqw #1,fp
0x7cc:  negw d1
0x7ce:  bls 0x841
0x7d0:  072551
0x7d2:  movel #101,d1
0x7d4:  movew -(a0),d5
0x7d6:  movel 0(a3)[d0.l*2],17249(a2)
0x7dc:  bgt 0x84c
0x7de:  ble 0x854
0x7e0:  moveal -(a1),a0
0x7e2:  bls 0x855
0x7e4:  072551
0x7e6:  movel #101,d1
0x7e8:  moveal -(a3),a0
0x7ea:  ble 0x859
0x7ec:  blt 0x84f
0x7ee:  bgt 0x854
0x7f0:  moveal 28535(1948254320(a0)),a0
0x7fa:  bcs 0x86e
0x7fc:  beq 0x86a
0x7fe:  ble 0x876
0x800:  bcs 0x830
0x802:  bge 0x873
0x804:  bcc 0x7875
0x808:  073545
0x80a:  movel #103,d1
0x80c:  bge 0x87d
0x80e:  movel #101,d3
0x810:  moveal 100(a3)[d6.l*8],sp
0x814:  oriw #28533,(a3)
0x818:  bgt 0x87e
0x81a:  oriw #24942,d3
0x81e:  bgt 0x88f
0x820:  movel #32,d2
0x822:  movel #97,d0
0x824:  movel #115,d1
0x826:  bcs 0x848
0x828:  negw (a3)
0x82a:  addqb #8,-(a0)
0x82c:  bge 0x89d
0x82e:  bsr 0x894
0x830:  moveal 28001(a1),a0
0x834:  beq 0x89b
0x836:  movel 0x285d,d0
0x83a:  071412
0x83c:  oriw #24942,d3
0x840:  bgt 0x8b1
0x842:  movel #32,d2
0x844:  bhi 0x8b5
0x846:  ble 0x8bc
0x848:  moveal -(a4),a0
0x84a:  071560
```

```
0x84c:  movel 0x2873,d0
0x850:  071412
0x852:  oriw #24942,d3
0x856:  bgt 0x8c7
0x858:  movel #32,d2
0x85a:  071545
0x85c:  movel #32,d2
0x85e:  movel #114,d0
0x860:  ble 0x8d6
0x862:  ble 0x8c7
0x864:  ble 0x8d2
0x866:  movel d0,d0
0x868:  041541
0x86a:  bgt 0x8da
0x86c:  ble 0x8e2
0x86e:  moveal 111(a3)[d7.w*4],a0
0x872:  movel #32,d0
0x874:  movel #104,d2
0x876:  bcs 0x898
0x878:  bcc 0x8ed
0x87a:  movel #32,d0
0x87c:  oriw #28533,d3
0x880:  bge 0x8e6
0x882:  bgt 0x8ab
0x884:  movel #32,d2
0x886:  bcc 0x8ed
0x888:  bsr 0x8f6
0x88a:  bge 0x8fb
0x88c:  bls 0x8ef
0x88e:  movel #101,d2
0x890:  moveal -(a3),a0
0x892:  blt 0x8f8
0x894:  subqw #7,28533(1948254275(a0))
0x89e:  bge 0x904
0x8a0:  bgt 0x8c9
0x8a2:  movel #32,d2
0x8a4:  bcc 0x90b
0x8a6:  bsr 0x914
0x8a8:  bge 0x919
0x8aa:  bls 0x90d
0x8ac:  movel #101,d2
0x8ae:  moveal 30574(sp),a0
0x8b2:  bcs 0x926
0x8b4:  subqw #7,28533(1948254275(a0))
0x8be:  bge 0x924
0x8c0:  bgt 0x8e9
0x8c2:  movel #32,d2
0x8c4:  bcc 0x92b
0x8c6:  bsr 0x934
0x8c8:  bge 0x939
0x8ca:  bls 0x92d
0x8cc:  movel #101,d2
0x8ce:  moveal -(a4),a0
0x8d0:  bcs 0x948
0x8d2:  subqw #7,28535(1948254288(a0))
0x8dc:  bcs 0x950
0x8de:  negw (a3)
0x8e0:  addqb #8,d0
0x8e2:  movel -(a4),25974(sp)
0x8e6:  movel -(a5),30323(sp)
0x8ea:  movew d0,d0
```

```
0x8ec:  bls 0x94f
0x8ee:  bgt 0x964
0x8f0:  moveal 28773(sp),a0
0x8f4:  bgt 0x916
0x8f6:  bcs 0x96e
0x8f8:  071440
0x8fa:  bcc 0x96e
0x8fc:  bvs 0x974
0x8fe:  bcs 0x972
0x900:  oriw #24942,d3
0x904:  bgt 0x975
0x906:  movel #32,d2
0x908:  movel #101,d1
0x90a:  bsr 0x970
0x90c:  moveal -(fp),a0
0x90e:  movel #111,d1
0x910:  blt 0x932
0x912:  negw (a3)
0x914:  addqb #8,-(a0)
0x916:  orb #0,d0
0x91a:  btst d4,Invalid arg format in opcode table: "@s".
```

——

t h e P O W E R - g l o v e M A I L I N G l i s t

The power-glove mailing list is probably the best way to keep on top
of new glove developments, voice your questions about the powerglove,
pgsi, sega glasses and other third party hardware, and contact software
gurus and wireheads who are knowledgable on the subject.

To subscribe, send a message to listserv@boxer.nas.nasa.gov with body
of "subscribe glove-list your full name"
Instructions for receiving archives via the listserv process are
available by sending e-mail to the above address with body of
"get glove README"
or
"get glove INFO"

Help files for listserv and the "glove README" and "glove INFO" files
are also available on Network 21.

——

W H A T a b o u t T H E m a c - I N T O S H ?

Someone sent me the following via e-mail. It includes some resource
locations and other mac-relevant information.
Compilers names located at end of segment.

——

==========

Macintosh PowerGlove Stuff Self Extracting Archive of a whole bunch of
neat stuff on hooking up a Powerglove to the Mac, including the Ron
Minelli 68HC11 programs, plus a Mac based compiler/comm stuff for the
68HC11.Found this on ftp.apple.com
==========

% PPP File ROTATE.SEA, binary, 114432 bytes, 0 downloads Uploaded by
      72330,770 on 4/19/93 Title: Macintosh Rotation Controllers

MAC ROTATION CONTROLLERS DEMO

MAC SelfExtracting Arvhive. Contains 2 programs that demonstrate 5 different methods of rotating objects. One program tests user speed. Created for Siggraph '88 by M.Chen, SJMountford & A Sellen. Uploaded by Jerry Isdale (Compuserve)
==========

% PPP File MAC3D.SEA, binary, 105344 bytes, 1 download Uploaded by 72330,770 on 4/19/93 Title: Sega3D circuit, Program & source for Macintosh

MAC SEGA 3D SOURCE C LCD GLASSES

Macintosh Self Extracting Archive. Contains description of circuit to connect LCD shutter glasses (Sega) to Mac, also program to test w/think C source code. NOTE: sega glasses are NOT available from sega as files in this archive state. Uploaded by Jerry Isdale (Compuserve)
==========

From: daver@sunspot.ssl.berkeley.edu (David Ray) Subject: Re: SOFTWARE: Max? Date: 31 Mar 1993 20:33:59 GMT Organization: /etc/organization

I use Max with the Power Glove for interactive music performance. A couple of other people use it for the same kind of thing, though most users don't read sci-vw. There is an internet mailing list for Max users, and a topic on the WELL. It is sometimes discussed on rec.music.makers.synth.

Max runs on the Macintosh or Next. The most recent release includes drivers to interface the Gold Brick, which is a hardware interface going between the glove and the CPU. The glove data comes into the program as a stream of numbers, basically a set of 8 numbers corresponding to the 8 degrees of freedom of the glove. The keypad on the glove's wrist sends keyboard data to the CPU, just the same as typing the keys from the CPU keyboard.

Max is a development environment. You have to write your own program to map the stream of numbers in the way that you want to produce the desired output. Max will output MIDI, serial data, will play sounds from the hard drive (through the internal speaker) and play quicktime movies. The number of mathematical and algorithmic controls available for Max (to process the input data) is mind- boggling. Basically, you can program the glove to do just about anything that Max can do.

I have written a program that allows me to combine many of the output features of Max with a single glove. In short, I can simultaneously play virtual keyboards, manipulate rhythm tracks (over which I improvise keyboards), play sounds from the hard drive, all with glove gestures.

I recently did a little survey of Max users and prepared a list of some people on the internet using Max for various things. I have appended this users list below, so you can get an idea of the kinds of things peope do with Max.

Max is marketed by a company called Opcode. For more info about Max, send mail to opcode@well.sf.ca.us.

-Dave

MAX - related resources on the internet:

Max-edinburgh mailing list To subscribe, send an internet e-mail message to MajorDomo@dcs.ed.ac.uk with a null (empty) subject line and a message body that reads: subscribe max-edinburgh YourName

where "YourName" is your normal name, like Jane Doe (not your e-mail name)

The WELL, Sausalito, CA (415) 332-4335 Max has a discussion topic #252 in the MIDI conference.

FTP site: Some Max patches have been made available by anonymous ftp at the following locations:

IRCAM, Paris, France ftp.ircam.fr

Netjam Group at Berkeley, CA xcf.berkeley.edu (128.32.138.1) cd misc/netjam/submissions/max

==========

Date: Sat, 17 Apr 1993 14:40:07 -0400 (EDT) From: Kris Nybakken <nybakken@world.std.com> Subject: Re: Mac VR SW To: Caio Barra Costa <cabcosta@cat.cce.usp.br> In-Reply-To: <9304170555.AA13828@cat.cce.usp.br> Message-Id: <Pine.3.07.9304171405.A3585-b100000@world.std.com> Mime-Version: 1.0 Content-Type: TEXT/PLAIN; charset=US-ASCII

I can tell you about VOGL. This is, aparently, a PDish port of the standard 3D graphics library available for Silicon Graphics machines. I pulled down a alpha version for the mac (try an archie search on '-s vogl', or I can dig up the address), and the demo programs ran fine on my Mac II. The problem that I have had is finding docs and/or people that know how to code to the library. If you find any more info, let me know.

Here's one site that has vogl stuff, but it might not be the closest/fastest for you...

Host wuarchive.wustl.edu

Location: /graphics/graphics/mirrors/avalon.chinalake.navy.mil/utils/display FILE -r--r--r-- 519271 Nov 30 21:13 vogle.tar.Z Location: /graphics/graphics/mirrors/echidna FILE -r--r--r-- 500013 Oct 7 17:34 vogl.tar.Z FILE -r--r--r-- 519271 Oct 7 17:46 vogle.tar.Z Location: /graphics/graphics/misc/echidna FILE -r--r--r-- 500071 Dec 11 07:18 vogl.tar.Z FILE -r--r--r-- 519271 Oct 8 03:46 vogle.tar.Z Location: /mirrors3/archive.umich.edu/mac/development/libraries FILE -rw-rw-r-- 250927 Oct 17 1991 macvogl1.0a.cpt.hqx

==========

Date: Fri, 23 Apr 93 08:43:36 edt From: Rick Duffy <rick@maxai.den.mmc.com>

Message-Id: <9304231243.AA02339@maxai> To: cabcosta@cat.cce.usp.br
Subject: re: Software Status: O

Hello Caio Costa. I am in a similar situation as yourself - also a C programmer, with a GB and powerglove. One package of which I am aware that might be fun for you to fiddle with is RTRACE, which can create 3-d renderings of a geometrically-modeledd scene. I is available on ftp.u.washington.edu. (It crashes on my FX for some reason, but nobody else seems to have this problem). Also, I just received my first copy of MAX (from Sweetwater Sound in the eastern United States), so I will be learning here first hand (so to speak) how I can use the glove on the mac.

Any info you come across I would be greatful to receive. I'll let you know how things go.

... Rick Duffy

==========
Originator: glove-list@boxer Errors-To: jet@nas.nasa.gov Reply-To: glove-list@nas.nasa.gov Sender: glove-list@nas.nasa.gov Version: 5.5 -- Copyright (c) 1991/92, Anastasios Kotsikonas From: lindahl@cse.uta.edu (Charlie Lindahl) To: cabcosta@cat.cce.usp.br Subject: Mac VR SW

This being the second message asking for code/help on the GoldBrick, I hearby offer a THINKC demo (after extensive study of the MPW C source code supplied with the glove).

My demo moves an icon around the screen (2d). I have a very simple filter to help with noisy data (not an oustanding example of Macintosh interface programming, but it suffices).

TO which archiv(s) should I post the source? I probably won't get around to it util later in the week.

Charlie
==========
"Cruelty is the soul of humor." | Charlie S. Lindahl (Anon) | lindahl@cse.uta.edu | Electrical Engineering Dept | University of Texas at Arlington
---
Disclaimer: #include <std/disclaimer.h>
==========

Message: #19746, S/11 VR Tech Date: Sat, Apr 17, 1993 6:52:28 PM
Subject: #19710-Mac VR From: Jerry Isdale 72330,770 To: Jerry Isdale 72330,770

Luiz, and others Today I talked to Michael Starks of 3DTV. He is actively looking for a programmer to work on some Mac VR projects. While he cant afford to pay really well, he can offer hardware, etc. Contact Michael directly at (415) 479-3516

Thanks

Andrew Dent <dent@DIALix.oz.au>
Charlie S. Lindahl <lindahl@cse.uta.edu>
David Ray <daver@sunspot.ssl.berkeley.edu>

Jerry Isdale <72330,770> John S.
Conrader <CONRADERJ@h8700a.boeing.com>
Keith Rettig <KMR100G@oduvm.cc.odu.edu>
Kris Nybakken <nybakken@world.std.com>
Michael Rose <mrose@sfwmd.gov>
Rick Duffy <rick@maxai.den.mmc.com>
Timothy M. Kunau <timothy.m.kunau@pace.medtronic.com>

——

g l o v e E P I L O U G E

"At my right is Chris Gentile, one of the creators of the
Mattel Power Glove for the Nintendo entertainment system. "Slip this on,"
says Gentile, sliding a thick glove made of gray plastic over my hand and up
my forearm. Immediately, a cartoon-colored rendering of a handball court
appears on the monitor. There, above a light-brown floor, floats a dark-blue
ball waiting to be whacked. To the left and below, a sky-blue hand gently
rises and falls in time with my breathing.

"The glove on the screen will move the same way you move your hand," Gentile
says. "Swing as though you were hitting the ball."

I bring back my arm and whip my hand through an imaginary ball floating in
front of me. The glove on the screen suddenly looms larger, as if it were
approaching me, and then shrinks as if receding. It hits the ball on the
screen, sending it caroming off the walls. As the rebounding ball approaches
me from the left side of the screen, I move back a few steps, line up the
next shot and swing again. On screen, the disembodied glove merely freezes as
the ball bounces over it, then rebounds back under it.

"Take a couple steps toward the screen," says Gentile, pulling my elbow.
"You're too far away." "

- from "Adventues in Cyberspace" by Walter Lowe, Jr.

——

V R - s i t e l i s t i n g s
by bill@apple.com
(posted to the glove-list)

The items in this list are probably your best-bet for finding the
most recent information / code / support for your home-brewed items.
IMHO, sunsite.unc.edu and ftp.u.washington.edu are good places to start.
Sega / pg items on wuarchive.wustl.edu.

Here are a list of sites, discussions, systems, and bulletin
boards related to the discussions or the archiving of virtual worlds
technology.

If you know of any other please send info to me at billc@apple.com.

FTP sites:

sunee.uwaterloo.ca (129.97.50.50)
- home of REND386 (freeware VR library/package)
/pub (misc directories on pglove, raytracing, et al.)
/pub/vr

stein.u.washington.edu (140.142.56.1)
- home of sci.virtual-worlds, huge faq w/ great info!

```
        - if unable to use try ftp.u.washington.edu
        /public/virtual-worlds

karazm.math.uh.edu (129.7.128.1)
        - home of the glove list, nice code
        /pub/VR

ftp.apple.com (130.43.2.3)
        - sites list, Macintosh vr, CAD projects info
        /pub/VR

avalon.chinalake.navy.mil (129.131.31.11)
                - huge repository of 3D objects in all types of formats
                /pub

sunsite.unc.edu (152.2.22.81)
        - virtual reality demos, iris info, glasses, mirrors some of
                ftp.u.washington.edu, uforce info
        /pub/academic/computer-science/virtual-reality

src.doc.ic.ac.uk (146.169.2.1)
        - great usenet archives including stuff from sci.v-w, also
                has info on ISIS and a VR app for ISIS
        /usenet/comp.archives/auto/comp.sys.isis
        /usenet/comp.archives/auto/sci.virtual-worlds

wuarchive.wustl.edu (128.252.135.4)
        - complete mirror of ftp.u.washington.edu VR archive
        - docs for sega glasses --> RS232 iface, nintendo glove stuff
        - wuarchive is also a graphics archive with over 500 megs of
          graphics related info and source (/graphics)

cogsci.uwo.ca (129.100.6.10)
        - mirrors some of sugrfx.syr.edu and karazm.math.uh.edu and has some
                local stuff
        /pub/vr

sunsite.unc.edu (152.2.22.81)
        - cool 3D stuff to play with. take a look!
        /pub/academic/computer-science/virtual-reality/3d

TELNET sites:

phantom.com (38.145.218.228)
        - home of the MindVox system. telnet to and log in as guest.
        (you will be charged an access fee if you decide to become a user)



*Discussions*

Internet
- sci.virtual-worlds and sci.virtual-worlds-apps, moderated by Bob Jacobson
        and Mark A. DeLoura, are the main online VR discussions.

- alt.cyberpunk and alt.cyberpunk.tech often concerns topics related to VR,
        though that is not the focus of the discussions themselves.

- alt.cyberspace has pretty light-hearted discussions on the present and
        future implementation of cyberspace in the world.
```

America Online:
- The VR discussion on AOL is sponsored by Virtus Corporation and can be found in their directory. To get to the discussion, use the keyword VIRTUS,and look in the "Let's Discuss" folder.

The Well
- Telnet 192.132.30.2 or use dial up (who will send me a list?) and type 'go vr'

BIX
- The BIX conference is moderated by Dan Duncan and can be reached by typing "j virtual.world" There are a number of papers by such as Bob Jacobson, Brenda Laurel, William Bricken, Brad Smith, Randy Walser and others available on line, as well as some lively discussion about both the technical, philosophical, and political impact of VR.
BIX can be called on your modem at 1-800-695-4882 for details and rates.

CompuServe
- Once you are on CompuServe, to get to vr type "go graphdev". Look in the VR TECH areas for mail and libs.

__

Check out the Future Culture FAQ for more sitelistings, resources, dial-up bbs systems (for those without access), mailing lists, etc.

Future culture FAQ info:
(excerpted from the README file)

Requests to FutureCulture must be sent to: future-request@nyx.cs.du.edu

The subject of future-request mail must have one of the following:

subscribe realtime	-subscribe in realtime (reflector) format
subscribe digest	-subscribe in daily-digest (1 msg / day format)
subscribe faq	-subscribe to faq only (1 msg every few months)
unsubscribe realtime	
unsubscribe digest	
unsubscribe faq	
help	-receive info on the list and subscribing
send info	-receive info on the FutureCulture list
send faq	-receive a recent copy of the faq if you need one (list subscribers *automatically* receive this)

list administrator: andy (hawkeye)(dali)freshjive)
ahawks@nyx.cs.du.edu
ahawks@mindvox.phantom.com

__

S O F T - w a r e !

1 M U L T I V E R S E
2 G O S S A M E R
3 F L Y !
4 R E N D - t h r e e - E I G H T Y - s i x
5 O T H E R - p r o g r a m s

There are many experimental "vr" research projects available on the net that are in the public domain. The following are data-sheets on the above listed software entities.

__

V I R T U A L r e a l i t y O N t h e X
or M U L T I - v e r s e V 1 . 0 . 2
submitted to the net by:robert@acsc.com

*******Announcing the release of Multiverse-1.0.2*******

Multiverse is a multi-user, non-immersive, X-Windows based Virtual Reality system, primarily focused on entertainment/research.

Features:

Client-Server based model, using Berkeley Sockets.
No limit to the number of users (apart from performance).
Generic clients.
Customizable servers.
Hierachical Objects (allowing attachment of cameras and light sources).
Multiple light sources (ambient, point and spot).
Objects can have extension code, to handle unique functionality, easily attached.

Functionality:

Client:
The client is built around a 'fast' render loop. Basically it changes things when told to by the server and then renders an image from the user's viewpoint. It also provides the server with information about the user's actions - which can then be communicated to other clients and therefore to other users.

The client is designed to be generic - in other words you don't need to develop a new client when you want to enter a new world. This means that resources can be spent on enhancing the client software rather than adapting it. The adaptations, as will be explained in a moment, occur in the servers.

This release of the client software supports the following functionality:

o Hierarchical Objects (with associated addressing)

o Multiple Light Sources and Types (Ambient, Point and Spot)

o User Interface Panels

o Colour Polygonal Rendering with Phong Shading (optional wireframe for faster frame rates)

o Mouse and Keyboard Input

(Some people may be disappointed that this software doesn't support the PowerGlove as an input device - this is not because it can't, but because I don't have one! This will, however, be one of the first enhancements!)

Server(s):
This is where customization can take place. The following basic support is provided in this release for potential world server developers:

o Transparent Client Management

o Client Message Handling

This may not sound like much, but it takes away the headache of

```
accepting and
   terminating clients and receiving messages from them - the
application writer
   can work with the assumption that things are happening locally.

   Things get more interesting in the object extension functionality. This is
   what is provided to allow you to animate your objects:

    o Server Selectable Extension Installation:
        What this means is that you can decide which objects have extended
        functionality in your world. Basically you call the extension
        initialisers you want.

    o Event Handler Registration:
        When you develop extensions for an object you basically write callback
        functions for the events that you want the object to respond to.
        (Current events supported: INIT, MOVE, CHANGE, COLLIDE & TERMINATE)

    o Collision Detection Registration:
        If you want your object to respond to collision events just provide
        some basic information to the collision detection management software.
        Your callback will be activated when a collision occurs.

    This software is kept separate from the worldServer applications because
    the application developer wants to build a library of extended objects
    from which to choose.

    The following is all you need to make a World Server application:

    o Provide an initWorld function:
        This is where you choose what object extensions will be supported, plus
        any initialization you want to do.

    o Provide a positionObject function:
        This is where you determine where to place a new client.

    o Provide an installWorldObjects function:
        This is where you load the world (.wld) file for a new client.

    o Provide a getWorldType function:
        This is where you tell a new client what persona they should have.

    o Provide an animateWorld function:
        This is where you can go wild! At a minimum you should let the objects
        move (by calling a move function) and let the server sleep for a bit
        (to avoid outrunning the clients).

    That's all there is to it! And to prove it here are the line counts for the
    three world servers I've provided:

        generic - 81 lines
        dactyl - 270 lines (more complicated collision detection due to the
                            stairs! Will probably be improved with future
                            versions)
        dogfight - 72 lines

Location:

   This software is located at the following site:
   ftp.u.washington.edu
```

Directory:
pub/virtual-worlds

File:
multiverse-1.0.2.tar.Z

Futures:

Client:

- Texture mapping.
- More realistic rendering: i.e. Z-Buffering (or similar), Gouraud shading
- HMD support.
- Etc, etc....

Server:

- Physical Modelling (gravity, friction etc).
- Enhanced Object Management/Interaction
- Etc, etc....

Both:

- Improved Comms!!!

I hope this provides people with a good understanding of the Multiverse software,
unfortunately it comes with practically zero documentation, and I'm not sure whether that will ever be able to be rectified! :-(

I hope people enjoy this software and that it is useful in our explorations of the Virtual Universe - I've certainly found fascinating developing it, and I would *LOVE* to add support for the PowerGlove...and an HMD :-)!!

Finally one major disclaimer:

This is totally amateur code. By that I mean there is no support for this code other than what I, out the kindness of my heart, or you, out of pure desperation, provide. I cannot be held responsible for anything good or bad that may happen through the use of this code - USE IT AT YOUR OWN RISK!

Disclaimer over!

Of course if you love it, I would like to here from you. And anyone with POSITIVE contributions/criticisms is also encouraged to contact me. Anyone who hates it: > /dev/null!

robert@acsc.com

——

G O S S A M E R v 1 . z e r o
(c) 1993 Jon Blossom (johnbl@microsoft.com)

Annotators note: GOSSAMER is NOT a Microsoft product!

I'm happy to announce the imminent release of Gossamer 1.0, a public domain polygon-based 3D rendering engine for Macintosh. If you're interested, please read on! You can send questions, comments, etc. to me: jonbl@microsoft.com.

Gossamer 1.0 is a general-purpose package for managing, manipulating, and rendering three-dimensional scenes on the Macintosh. It has been designed and with the intention to release it to the public domain to encourage the production
of homebrew VR systems, animation systems, and simple related applications.

Before I go into details, let me just say that Gossamer will not be available for
a few more weeks - I'm predicting some time in early June since I still have some cleaning up to do.

Gossamer 1.0 will be provided as a library in Symantec THINK C 5.0 format with header files, fairly extensive documentation, and a demonstration program with source code that displays some of Gossamer's capabilities.

The package itself is a first-generation version that includes many but not all of
the features that I have planned. I will be extending the feature set as soon as
possible and releasing version 2.0, but I wanted to get *something* out to interested parties so that the state of homebrew Mac VR work would not remain as stagnant as it is now. Please forgive me the shortcomings and consider this a step towards a more complete implementation. My soapbox: "It'll only get faster and more robust."

The first generation of this library has been compiled for 68020 machines with 68881 coprocessors. Apologies to those excluded by these restrictions-- I'll be switching to integer-based fixed-point math in the next version, but the
'020 restriction will remain. These are the only hardware requirements: Gossamer is designed to be independent of monitor size or pixel depth and requires only a CGrafPort to make it go. The included demonstration will handle shading and pseudo-transparency on monitors from 1 to 8-bits per pixel.

Although Gossamer is NOT a strict port of Rend386, the nature of the project and the example of Rend386 make for many similarities between the two systems and should enable porting Rend386 programs. The demo files will contain procedures for reading Rend386 PLG files and, possibly, FIG and WLD files as well.

Support for the PowerGlove and Sega glasses will have to come in future revisions or from the application author. I haven't connected a pair of glasses (and doubt that Gossamer could handle stereo rendering just now), and my "ancient" 68HC11 EVBU pseudo-Menelli board is far from a standard in Power Glove connectivity.

Also, a standard disclaimer: I have written Gossamer to work on my personal development machine, a Mac IIci with a standard 13" Apple monitor. I can not guarantee that the code will work without hitches on any other machines. Actually, that's part of my reasoning for releasing this library in its current state: I can't test this thing all by myself. I NEED BUG REPORTS!

software for commercial ventures, including shareware, must be licensed from the author.

If you have any questions, problems, ideas, or code relating to Gossamer, please
contact me at jonbl@microsoft.com. Gossamer is NOT a Microsoft product, nor is it even remotely endorsed by my employers. It is strictly a labor of love and the
exclusive property and responsibility of the author. Sorry, I had to throw that in just in case.

—

F L Y !

This is the "FLY" software "FAQ" sheet, which is available from ftp.u.washington.edu

FLY!-3D Software is available at: ftp.u.washington.edu (/oldpublic/fly)

1. What are the differences between the FLY! Demo and the real product?

The FLY! demonstration is a fully functioning copy of the FLY! program except that it will ONLY work on the supplied demonstration data.

2. Is there a more extensive demonstration available above and beyond that on the Internet?

Yes. PCI has a demonstration that consists of the following:

a) FLY! exectuables for about 10 different workstations
b) a 512 x 512 data set (tiny.pix) (1.5 Mbytes)
c) a 1024 x 1024 data set (small.pix) (6 Mbytes)
d) a 1536 x 1536 data set (big.pix) (15 Mbytes)
e) a hardcopy manual
f) a pair of (cheap) anaglyph (red/blue lensed) glasses for 3-D viewing

Upon special request we can include a 2048 x 2048 (24 Mbytes) data set and a manual describing the PCIDSK database file format the data sets are distrbuted in.
* Note: 2048 x 2048 dataset is only useful on machines that have 48+ Mbytes of RAM.

This demonstration is basically identical to that on the InterNet except that you get some larger data sets and the extra documentation.
To cover the media, shipping and handling costs we ask for $99. >:-(
You are encouraged to share the demonstration with others or use it on as many machines as you like (or can get access to).

3. Who are PCI Inc. anyway?

PCI is a company which develops imaging software for Satellite and Aircraft Remote Sensing which, we sell all over the world.
We are located in Richmond Hill (Toronto really...), Ontario, Canada.
FLY! is only one of many products, though probably the most fun...

4. Why is FLY! special?

The concept of draping image data over elevation is not new, its been
around for 10+ years. However it has always been slow, on a VAX 780
it would take 15 minutes or more to get a single rendering.
Other approaches in the past solved this problem with special hardware,
obtaining much faster (occasionally realtime) speeds, but at enormous cost.
FLY! combines an innovative algorithm coupled with large amount of RAM
and the fast processing speed of on todays workstations to bring this
time down to the 1/10th to 1 second range. This means anyone can now have
'near' realtime performance on a general purpose workstation. No special
purpose hardware add ons or graphics cards are required.

The FLY! algorithm can also be parallelized, more on this in further
question answers.
FLY! considers every pixel as a unique polygon. This gives maximum
realism in rendering in 3-D.

5. How fast is the FLY! algorithm?

The speed, in frames per second, depends on a lot of factors:

- speed of your CPU
- how big the input database is
- how big an output rendering frame you want
- various perspective parameters, such as cone of view...

As a base reference we will consider the following:

- a 1024 pixel by 1024 line database (1048576 polygon equivalent...)
- a 320 x 240 output rendered scene
- a 60 degree view cone

Experimentation across a number of machines has shown that it takes about
22 MIPS of power to generate a rendered scene in 1 (one) second. For example,

- a 17 MIPS machine (eg Sparc 1+)   gets about 17/22 = .8 frames/sec
- a 33 MIPS machine (eg, SGI 4D-35) gets about 33/22 = 1.5 frames/sec
- a 130 MIPS machine (eg, DEC Alpha) gets about 130/22 = 6.0 frames/sec

Of course not all MIPS are equal across all machines, and I am not to sure
about the relationship to SPECmarks or SPECints.

We have further noted the following rules:

- Doubling the database size tends to decrease performance by 25%.
- Computational power increases with the square of the X size of the
  output rendering frame (Y size is just extra sky and is not important)
  (eg. 640 x 480 output rendered scene requires 22 x 4 = 88 MIPS)

>From these we have developed the following chart which gives the MIPS
required for various configurations to generate a single frame in one second.

Rendering sizes are 160x120, 320x240, 450x170, 640x480.

Database size	1024x1024				2048x2048				4096x4096			
Render size	160	320	450	640	160	320	450	640	160	320	450	640
MIPS	6	22	44	88	7	28	56	112	8	34	68	140

Thus given a 2048x2048 database, a 160 by 120 rendering frame, and a DEC alpha
workstation (130 MIPS) we would get 130/7 = 19 frames/second.

Given a 1024x1024 database, a 640 by 480 rendering frame, and a SparcStation 2 (27 MIPS) we would get 27/88 = .31 frames second, or about 3 seconds per frame.

The FLY! algorithm can be run on parallel processors. We have done this on the Silicon Graphics Power Series (up to 8 processors). In general FLY! scales well. We have found that the MIPS rating of a parallel machine (for the purposes of FLY!) can be computed as:

MIPS = #processors * MIPS/processor * 0.8

The 0.8 is due to the parts of the algorithm which cannot be parallelized and general overhead. In the case of an 8 processor SGI we got an effective MIPS rating of: 8 * 33 * 0.8 = 210 MIPS

Please consider the above figures as rough. The following factors can change the above in some way:

- Whats a MIP anyway? How do you compare across machines?
- Large Caches help.
- You had better have enough RAM. If ANY disk swapping/thrashing occurs the frame rate plummets. See question on memory below.
- The quoted MIPS/frame/sec assume a worst case. If you are near an edge looking at the edge frame rates increase...

6. How much memory is required?

FLY! requires about 8Mbytes of RAM + 6 bytes per database pixel. Thus a if FLY! is run with a 1024x1024 database it needs about 8 + 6 = 14Mbytes.
A 2048x2048 database would require about 8 + 24 = 32 Mbyes.
A 4096x4096 database would require about 8 + 96 = 104 Mbytes.

On top of this is the normal OS and X11/Motif requirements, say another 8 to 10 Mbytes. Thus FLY! using a 1024 x 1024 database should be run on a system with about 24Mbytes of RAM.

If you don't have enough RAM then the OS will start swapping parts of the FLY! data to disk. Since FLY! typically has to look through most of the data for each frame this creates a LOT of paging and frame rates drop to almost nothing.

7. Can you tell me more about the FLY! algorithm?

The FLY! algorithm is proprietary, source code is not available. Like most interesting things it was developed after hours when we could grab a few minutes when the pressing work of meetings, documentation, phone tag and tech support diminished somewhat :-). No papers have been published.

PCI won't divulge much but we will point out:

- it uses hierarchies of databases so that as data gets further away it uses progressively less computational power.
- each frame is done 'brute force'. No information is saved across scenes. This means turning, repositioning has no performance penalty.
- The inner loops do not use any floating point, everything is fixed point.
- Lots of stuff is table driven.

8. Why doesn't FLY! make use of the special graphics hardware on my machine? Wouldn't this make FLY! even faster? or improve the visual quality?

FLY! was designed to run on the widest variety of hardware possible with the minimum of programming. Using special graphics operations would limit the FLY! to a small subset of machines. The FLY! algorithm is designed in such a way that using special graphics shading/polygon rendering would be very difficult.

Remember that a 1024 x 1024 data base has 1,000,000+ potentially unique (square) polygons, a 2048x2048 4,000,000+ polygons. If we divide the squares to triangle the numbers double. It is a pretty unique board that can handle 8,000,000 triangles and render the appropriate ones 10 to 20 times per second.

Of course it is possible to reduce the number of polygons/triangles by approximating large area's and use texture mapping. This has three drawbacks:

- the realism and data accuracy is compromised for visual appearance (FLY! is primarily a visual analysis tool where accuracy is vital)
- in 3-D modes (left and right eye) FLY! gives excellent results because the input data often has natural shading and lots of detail since every pixel is rendered separately. The brain is uses this to give a better 3-D interpretation.
- Lots of packages already exist that to this type of thing very well.

This is not to say that careful consideration of a particular graphics board, with substantial changes to the FLY! algorithm, might improve speed/quality, however PCI has no plans in this direction in the near future.

8. What about parallelism?

The FLY! algorithm is designed to take advantage of parallel processors. PCI has tested this on parallel processor Silicon Graphics machines already and improvements in speed are nearly linear with the number of processors.

FLY! requires the following characteristics for using parallel processors:

- a Multiple Instruction, Multiple Data (MIMD) architecture
- every processor must have access to the full shared memory, not just a small local amount.
- there needs to be a large bandwidth to the host computer which shows the rendered images. For example: for a 512x512 24bit image at 10 frames/sec, we need at least 10Mbytes/second to the host computer or video board that shows the frames, probably a lot more.
- Floating point performance is irrelevant, the FLY! algorithm is integer.

We have had a number of inquires about using special parallel hardware boards. Unfortunately, so far, these have not had the characteristics we require. Also PCI does not have the resources to reprogram FLY! on the off chance of selling a limited number of special boards.

More promising is the use of parallel workstations, typically these are already integrated nicely into the memory/graphics subsystem and simple to program. The drawback of course, is limited numbers of processors and a higher cost, but this is changing slowly.

PCI has already experimented with the SGI Power Series (up to 8 processors). In the future we hope to try the SGI Onyx series (up to 36), the SUN Sparc (up to 4) and have heard rumours of parallel HP 700's, DEC Alpha's and IBM Risc System 6000's (numbers of CPU's unknown).

9. Is there any other development in progress on FLY!?

Currently development is limited while we evaluate the market potential of FLY! (and do more boring work on other projects). However by late summer 1993 you can expect the following:

- options to improve rendering quality using fitted polygons and interpolated colors in the foreground. Makes much better looking renderings at low altitude, but there is a performance hit.
- DEC Alpha support (Open VMS and Unix)

In the longer term (say end of 1993) you might see:

- Flight Planning
- Windows 3.1, Windows NT, MAC/OS and OS/2 support
- Parallel versions for other workstations (like SUN)

10. What about Virtual Reality?

PCI, (well us programmers actually, and happily the company President, though not the sales people who can't see any possible sales for years ;-) are really, really interested in doing VR development.

Now, by VR, we mean a head mounted display with at least 10 frames per second frame rate, at some reasonable resolution (say a minimum of 442 x 350). FLY! already has a 3-D option using anaglyph (red/blue) glasses, so we know it can do a good job of generating left/right images. We also anticipate interfacing a cyberglove, for virtual gesture recognition, along with auxilary devices such as the spaceball, LCD glasses, etc.

The real problem has to do with compute power. Refering to question 5, a 2048x2048 database with 450 resolution needs 56 MIPS per frame per second. Thus 10 frames/sec = 560 MIPS PER eye = 1120 MIPS, minimum.

Now there would be some non parallel overhead and we would probably need to do some extra work in the foreground to get less blocky images. Let's say 1400 MIPS all told. Not many machines out there with this performance, but still possible. For example: an SGI Onyx with 12 processors, perhaps by the end of '93 an 8 processor Alpha, etc...

While R and D at PCI is gung ho to get such a machine there is the small problem of cost... PCI is too small to carry the full burden so we are currently examining the possibility of sharing costs with a larger organization. Some ideas in progress, stay tuned.

Our VR work would concentrate on using true remote sensing (satellite) data to allow the user to manipulate real world data. Allowing the user to fly through 100's of square miles, to manipulate entire mountain ranges, etc... Applications would be in the area of Geology, Forestry, GIS visualization, etc...

Of course we have some concern's: how many frames/second do we need as a minimum to prevent nausea?; there is a tendency for the FLY! algorithm to generate 'shimmer' or speckles in the distance as small objects show up and disappear with small changes in position/angles, how do we solve this, or minimize it?; much of our data is very coarse (e.g., 30 meter resolution) does this matter, or is there some threshold people require?; what is the best way to allow the user to interact with the program, a glove?

It might take us a while to start work in this area, but as the cost of

MIPS and VR drop, it is going to happen.

For further technical questions, feel free to contact the
designer of Fly! himself! David Stanley (stanley@pci.on.ca)

We hope you enjoy the demo, and feel free to give us
feedback, positive or negative (flaming excluded :-)
or any ideas, comments, suggestions, etc..

Karim Ismail | Richmond Hill, Ontario
ismail@pci.on.ca | Canada
PCI Inc | +1 416 764-0614
Support Engineer | +1 416 764-9604 (fax)

——

R E N D - t h r e e - E I G H T Y - s i x

Here is a quick overview of features from the demonstration documentation.
For more information on Rend 386, you may communicate with the authors
via the Internet, or post your questions to the glove mailing list.

REND386 -- A 3-D Polygon Rendering Package for the 386 and 486
Written by Dave Stampe and Bernie Roehl

...
The libraries are available for free; the only reason for making the demo
a separate set of files is to give people who aren't interested in writing
software a chance to see just what can be done on widely-available hardware.
...
The system is fast. How fast, you ask? Well, speed is not a straightforward
thing to measure. There is a relationship between the speed of the processor,
the complexity of the scene, and the number of frames per second.
...
With this software, a 512-polygon scene can be rendered at speeds up to
15 frames/second on a 486/25; this corresponds to a speed of over 7000
polys/second. If you have a 486/33, it'll go slightly faster; if you have
a 386/33, it'll go slightly slower. You get the idea. If you want more
frames/second, use a simpler scene (i.e. fewer polygons).
...
This version now support stereoscopic viewing; the assumption is that you
have the Sega 3D glasses and the interface described in sega.txt installed.
...
The Nintendo Powerglove is also supported: see the July 1990 Byte page 288
for a schematic showing how to wire up an adapter. Version 4 of the demo
uses the glove to manipulate objects as well.

-- Bernie Roehl, August 1992

——

R A C Q U E T - b a l l

PCVR Presents:

Virtual Racquetball for the PC

This program allows a person using an IBM personal computer to play
racquetball in a virtual environment. The system uses the VR rendering
package REND386 and a Mattel Powerglove.

When the game begins you face the inside of a racquetball court. To your right is a racquet and right in front of you is your hand. As you look around, a purple ball bounces off of the walls. You grab your racquet and begin trying to hit the ball.

Requirements: 80386 IBM personal computer or better Mattel Powerglove attached to Parallel Port

RACBALL.EXE (self-extracting) is available via Network 21 or can be transferred from listserv@boxer.nas.nasa.gov.
listserv@boxer message body: "get glove RACBALL.EXE"
from Network 21: /net21/third/racball.exe

——

t h e - H A N D
(gradeki@rodeo.uwyo.edu)

The program requires a fast! machine and a VGA video card.

To execute the program, simply run it. You will be prompted to exercise your glove and press a key to start. It is very important that you exercise your glove in order to refresh the value going to the PC.

Once a key is pressed, the hand on the screen will react to the Power Glove. If you make a fist, the hand will make a fist. If you roll your hand from horizontal to vertical, the hand will respond. If the indicator lights are not flashing, exercise the glove a little and press center. The virtual hand will not respond until the glove is sending data.

The Roll of the Power Glove is not as good as hoped. If you keep your hand in a 45 degree position between horizontal and vertical, the virtual hand may! switch back and forth between the two states. Therefore, try to keep your hand in either horizontal or vertical position. Notice that in the vertical position, we are all set for a pong game or handball!

If you have any! problems or questions, I can be contacted at
gradecki@rodeo.uwyo.edu, or post to the glove list.

NOTES:

The faster the machine the better. The program tries to achieve real-time representation of the motions that you make. There are limits. If you go outside the reception area of the glove, or if there is excessive noise, the hand can become lost or the center will move.

You can break the video screen if there is a numerical overflow. You will get an overflow, simply run the program again.

The program is not perfect. I have only spent a week (during slow times during the day) working on this program. I am releasing it now because there is NOTHING else out there for the Power Glove and those people who have one.

Being that one of my areas is networking, I will have a graphical virtual handshake in a week or two. After which is virtual world manipulation. You will be able to pick objects up and move them using the Power Glove.

THEHAND.ZIP is available on Network 21 and can also be transferred from

listserv@boxer.nas.nasa.gov.
listserv@boxer message body: "get glove THEHAND.ZIP"
from Network 21: /net21/third/thehand.zip

—

This concludes issue one of "Home Brewed". I hope this gives you a basic idea of what resources are currently available.
I am not responsible for any of the information presented within this document. I did nothing more than compile existing information into a formatted text. This does not make the authors of this text responsible for the content, either.
If you buy a product such as a Mattel Powerglove, and use public domain and third party schematics to alter it, then end up frying it out, the only persona responsible for that pcb smell are you and your Corona.
Use all information at own risk, and buy your gloves and glasses from the street. Turn off your television sets. Don't wash your hair. Hail max headroom.

Submissions are welcome.
Annotations are welcome.
Food stamps are welcome.

Good trashing spots desired.
Intelligent female without attachments desired.

Send all relevant information and phone numbers via:
Uucp:system@ntwrk21.chi.il.us
or
dial-in:
+1 312.784-2852
(Chicago, IL, login as NEW)

N E T W O R K - t w e n t y - O N E
"Freebirthing the 90's into deep space!"
24 hour space-time fix.