Reverse engineering HoD PSX bitmaps

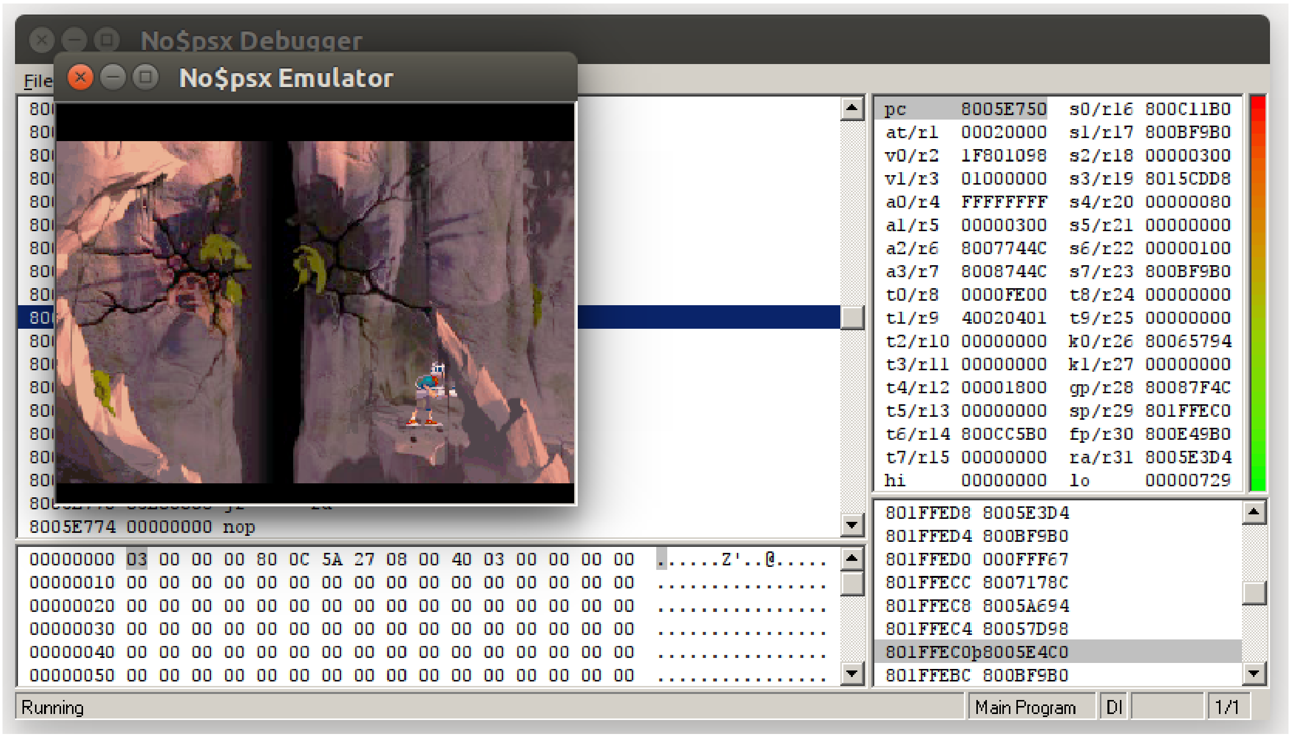
Heart of Darkness from Amazing Studio is known for its long development time. When released in 1998, the game was available on both Windows and PlayStation platforms.

The Windows version uses a 256 colors palette for its graphics. This is likely inherited from its early development years when DOS and VGA cards were the principal gaming configuration on PC.

The PlayStation version was released a few months after the Windows version. Let's figure out the native graphics file format used.

Taking a first guess

Loading the game in the emulator and playing the first few screens, some JPEG artifacts can be observed in some areas of the background bitmaps.



```
8005E774 00000000 nop
```

```
pc      8005E750  s0/r16 800C11B0
at/r1   00020000  s1/r17 800BF9B0
v0/r2   1F801098  s2/r18 00000300
v1/r3   01000000  s3/r19 8015CDD8
a0/r4   FFFFFFFF  s4/r20 00000080
a1/r5   00000300  s5/r21 00000000
a2/r6   8007744C  s6/r22 00000100
a3/r7   8008744C  s7/r23 800BF9B0
t0/r8   0000FE00  t8/r24 00000000
t1/r9   40020401  t9/r25 00000000
t2/r10  00000000  k0/r26 80065794
t3/r11  00000000  k1/r27 00000000
t4/r12  00001800  gp/r28 80087F4C
t5/r13  00000000  sp/r29 801FFEC0
t6/r14  800CC5B0  fp/r30 800E49B0
t7/r15  00000000  ra/r31 8005E3D4
hi      00000000  lo     00000729
```

```
00000000 03 00 00 00 80 0C 5A 27 08 00 40 03 00 00 00 00  ......Z'..@.....
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
801FFED8 8005E3D4
801FFED4 800BF9B0
801FFED0 000FFF67
801FFECC 8007178C
801FFEC8 8005A694
801FFEC4 80057D98
801FFEC0þ8005E4C0
801FFEBC 800BF9B0
```

Figure 1: PSX Emulator Screenshot 1

The PlayStation has a hardware block (MDEC) specialized in decoding JPEG like encoded frames. This is usually used by games for playing FMV sequences.

Checking disassembly

Running strings and grep over the game executable show some left-over references to MDEC.

```
$ strings slus_006.96 | grep -i mdec
MDEC_rest:bad option(%d)
MDEC_in_sync
MDEC_out_sync
```

Going through the disassembly and comparing the assembly code with known libraries (such as libbs) allows to rename and map a few functions related to the MDEC unit.

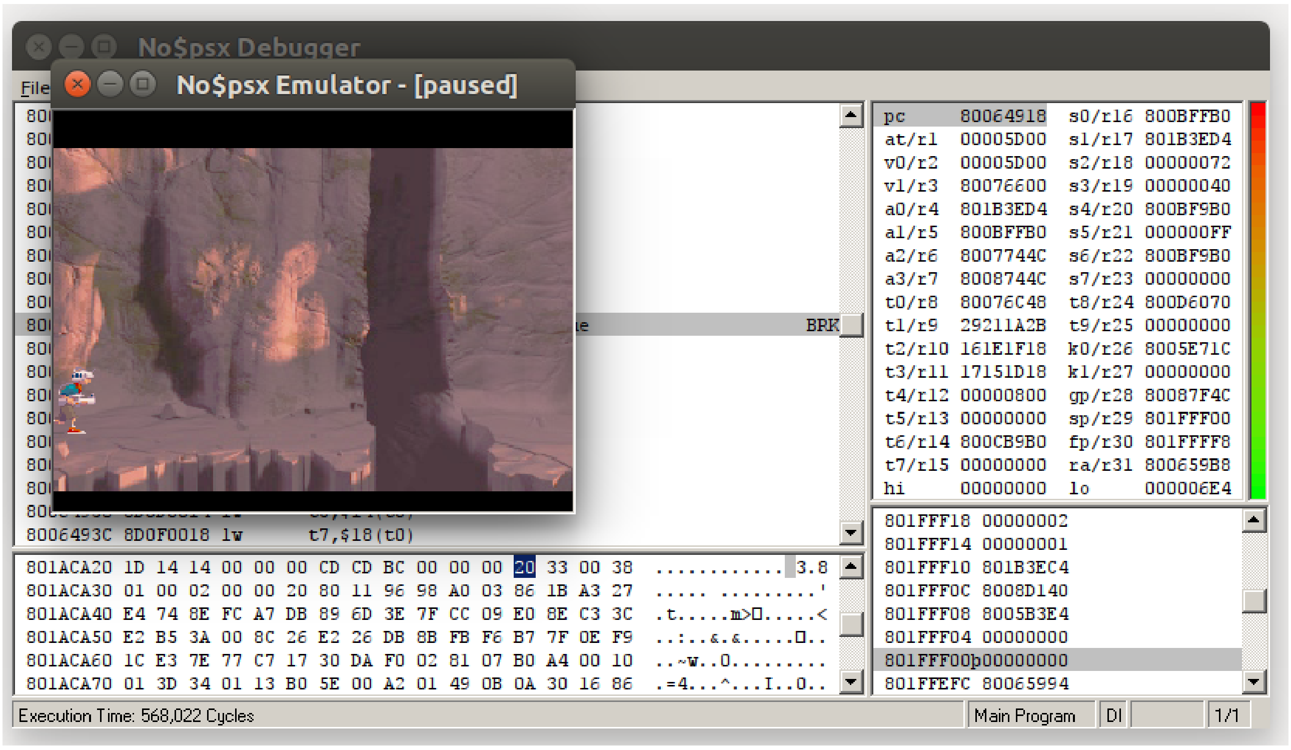
```
.text:8005E184 DecDCTReset
.text:8005E420 DecDCTout
.text:800648D0 DecDCToutCallback
.text:80064900 DecDCTvlc
```

The DecDCTvlcfunction is especially interesting since this is where the MDEC compressed bitstream is fed to the hardware block.

```
.text:80064900 DecDCTvlc:
.text:80064918      bnez $a0, loc_80064954
```

Breakpoint and memory dump

To confirm the MDEC is used for background bitmap, a breakpoint is set at 0x80064918 before moving Andy to the second screen.



```
pc     80064918  s0/r16 800BFFB0
at/r1  00005D00  s1/r17 801B3ED4
v0/r2  00005D00  s2/r18 00000072
v1/r3  80076600  s3/r19 00000040
a0/r4  801B3ED4  s4/r20 800BF9B0
a1/r5  800BFFB0  s5/r21 000000FF
a2/r6  8007744C  s6/r22 800BF9B0
a3/r7  8008744C  s7/r23 00000000
t0/r8  80076C48  t8/r24 800D6070
t1/r9  29211A2B  t9/r25 00000000
t2/r10 161E1F18  k0/r26 8005E71C
t3/r11 17151D18  k1/r27 00000000
t4/r12 00000800  gp/r28 80087F4C
t5/r13 00000000  sp/r29 801FFF00
t6/r14 800CB9B0  fp/r30 801FFFF8
t7/r15 00000000  ra/r31 800659B8
hi     00000000  lo     000006E4

801FFF18 00000002
801FFF14 00000001
801FFF10 801B3EC4
801FFF0C 8008D140
801FFF08 8005B3E4
801FFF04 00000000
801FFF00þ00000000
801FFEFC 80065994

8006493C 8D0F0018 lw      t7,$18(t0)

801ACA20 1D 14 14 00 00 00 CD CD BC 00 00 00 20 33 00 38  ............ 3.8
801ACA30 01 00 02 00 00 20 80 11 96 98 A0 03 86 1B A3 27  ..... .........'
801ACA40 E4 74 8E FC A7 DB 89 6D 3E 7F CC 09 E0 8E C3 3C  .t.....m>□.....<
801ACA50 E2 B5 3A 00 8C 26 E2 26 DB 8B FB F6 B7 7F 0E F9  ..:..&.&.....□..
801ACA60 1C E3 7E 77 C7 17 30 DA F0 02 81 07 B0 A4 00 10  ..~w..0.........
801ACA70 01 3D 34 01 13 B0 5E 00 A2 01 49 0B 0A 30 16 86  .=4...^...I..0..

Execution Time: 568,022 Cycles    Main Program    DI    1/1
```

Figure 2: PSX Emulator Screenshot 2

The program pauses when Andy moves to the next screen. The contents of memory pointed at by the register A0 are

```
CD CD BC 00 00 00 20 33 00 38 01 00 02 00
```

Ignoring the first 6 bytes, the sequence appears to match the documented format a MDEC frame.

```
 0 . . . 2 . . little . Unknown
                         Number of run length codes in the frame?
                         Size of data following this header?
 2 . . . 2 . . little . Always 0x3800
 4 . . . 2 . . little . Frame quantization scale
 6 . . . 2 . . little . Version of the frame
 8 . . . . . . . . . .  Compressed macro blocks
                         Stream of 2 byte little-endian values
```

The contents of the memory are dumped to a file (100 KB) for conversion.

Data dump decode

To confirm the data is a MDEC frame, we can feed it to a decoder.

ffmpeg has a MDEC decoder in libavcodec. ffmpeg actually supports playing back .STR files but this is not relevant here as only the codec code is necessary.

With a few lines of C, the data dump is correctly decoded.

```
const AVCodec *codec = avcodec_find_decoder(AV_CODEC_ID_MDEC);
AVCodecContext *ctx = avcodec_alloc_context3(codec);
ctx->width = 256;
ctx->height = 192;
avcodec_open2(ctx, codec, 0);

AVPacket pkt;
av_new_packet(&pkt, len);

AVFrame *frame = av_frame_alloc();
avcodec_decode_video2(ctx, frame, &hasFrame, &pkt);
```

After converting the decoding frame, the second screen of the first level can be displayed, without glitches.

This confirms the use of a MDEC compressed frame as the native format of the background bitmaps. Next step is to extract all pictures from the game data files and convert them.

Data files

The files found on the PlayStation CDs are similar to the ones from the PC Windows version. For each level, there are two files :

- .LVL : contains the palettes, bitmaps, sprites and pre-calculated tables for shadows
- .MST : contains the bytecode and triggers for the monster logic

Scanning for MDEC frames in the .lvl files and feeding the data to the ffmpeg decoder results in some ac-texerrors, indicating damaged or wrong data.

```
Read 1048576 bytes at 0x1a330c
MDEC len 13088, VLC_ID 0x3800
qscale 1 version 2
avcodec_decode_video2 ret -1094995529
[mdec @ 0x55a04ab001a0] ac-tex damaged at 2 0
```

Forcing the output to a file and continuing on errors, some 8x8 blocks are correctly decoded while others are off (wrong position and/or colors).

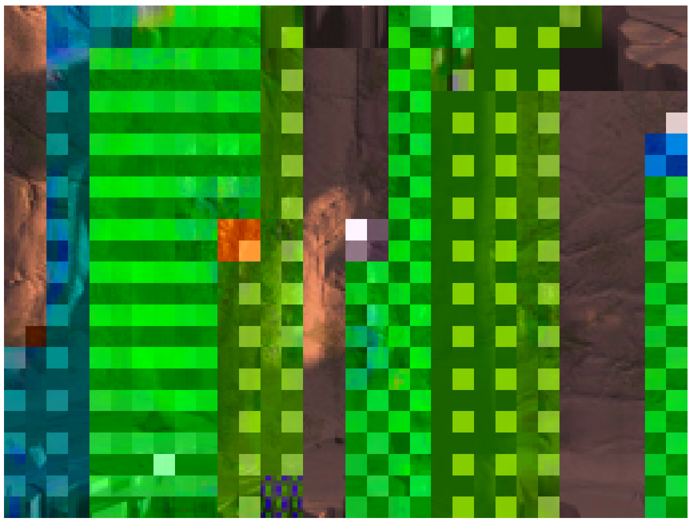


Figure 3: Decode Glitch

Comparing the data found in the files with the previous memory dump shows no difference at the beginning of the buffer.

```
$ cmp 001a330c.bss a0.bss
001a330c.bss a0.bss differ: byte 1265, line 8
```

There are four extra bytes in 001a330c.bss at 0x4F0 (1265-1) in the data files when comparing with the memory dump.

The original PC demo data files had a similar pattern : every 2048 bytes, the last 4 bytes corresponds to a checksum of the previous 2044 bytes.

```
uint32_t fioUpdateCRC(uint32_t sum, uint8_t *buf, uint32_t size) {
  assert((size & 3) == 0);
  size >>= 2;
  while (size--) {
    sum ^= READ_LE_UINT32(buf); buf += 4;
  }
  return sum;
}
void SectorFile::refillBuffer() {
  int size = fread(_buf, 1, 2048, _fp);
  if (size == 2048) {
    uint32_t crc = fioUpdateCRC(0, _buf, 2048);
    assert(crc == 0);
```

Calculating the address confirms these four extra bytes are aligned to 0x800.

```
$ printf %x $(( 0x001a330c + 0x4f0 ))
1a37fc
```

The extraction code is updated to skip four bytes every 2048 bytes read. After feeding the data to the ffmpeg decoder, the second level screen can be decoded without any errors.

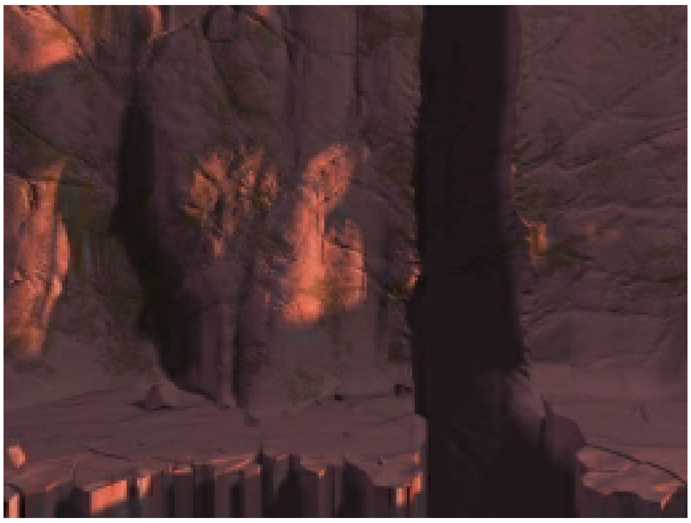


Figure 4: Level1 Screen2 PSX

It is interesting to note the PSX data file use the same sector based file format for its data files.

Batch conversion

With the file format understood, all of the level screens bitmaps can be extracted and compared with the PC version.

Below is the comparaison of the first three screens, on the left the Windows PC paletted 256 colors and on the right the PlayStation YUV bitmaps.

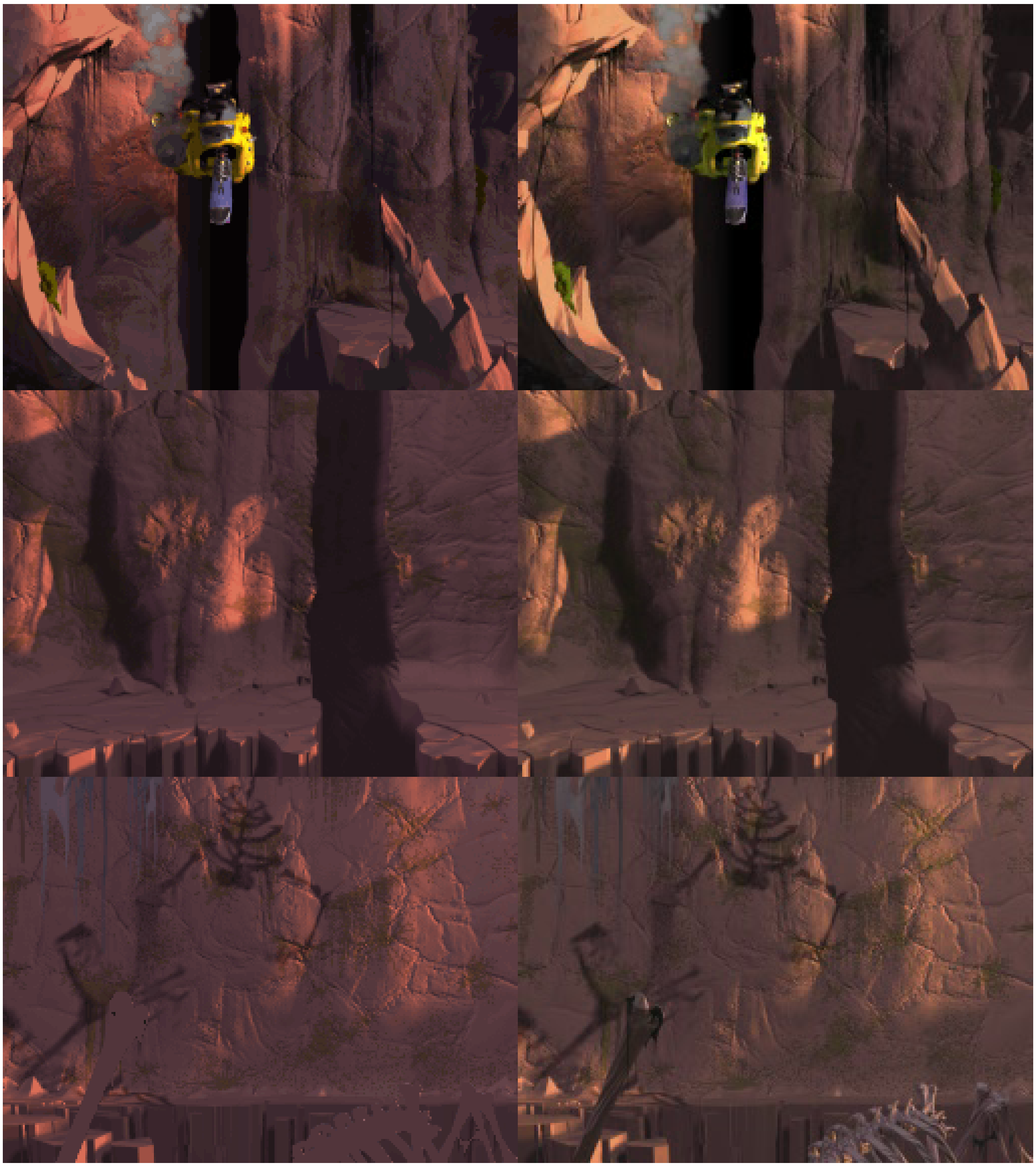


Figure 5: Level1 PC PSX