An Introduction To

With Joachim Bengtsson and Frank Buß

What is LuaPlayer?

LuaPlayer

+

+

```
int r[4]={0};
int g[4]={0};
int b[4]={0};
char i, pal, sub, x, y;
UWORD paletteCGB2[4];
UWORD savePalette[4];

cls();

set_bkg_palette(0, 1, hardPaletteCGB);
set_sprite_palette(0, 1, paletteCGB);

for(i = 0; i < 4; i++)
{
   set_sprite_tile(i, i + 128);
   set_sprite_prop(i, 0);

   set_sprite_tile(i+4, i + 128);
   set_sprite_prop(i+4, 0);

   r[i] = paletteCGB[i] & 0x1F;
   g[i] = (paletteCGB[i] >> 5) & 0x1F;
   b[i] = (paletteCGB[i] >> 10) & 0x1F;

   paletteCGB2[i] = paletteCGB[0];

   move_sprite(i+4, (9+i)*8, 19*8);
}
set_sprite_prop(0,1);
set_sprite_tile(0, 129);
```

```
UBYTE *gameName = (UBYTE*)0x6134;
UBYTE *ninLogo = (UBYTE*)0x6104;
UBYTE *realNinLogo = (UBYTE*)0x0104;

found = 0;
numFound = 0;

i = 1;
do
{
   SWITCH_ROM_PV(i*4);

   for(j = 0; j < 0x30; j++)
     if(realNinLogo[j] != ninLogo[j])
        break;

   if(j == 0x30)
   {
      strncpy(romName[i], gameName, 16);

      found = 1;
      numFound++;
   }
   else
      found = 0;

   i++;
```

```
.VBlank lda $2002
    bpl .VBlank
    ldx #$00
    stx $2000
    stx $2001
    ldy #$06
    sty $01
    ldy #$00
    sty $00
    lda #$40
    sta ATTSIZE
    lda #$48
    sta SPRITESIZE
    lda #$20
    sta PALETTESIZE

.ClearWRAM  sta ($00),y
    dey
    bne .ClearWRAM
    dec $01
    bpl .ClearWRAM
    ldx #$3F
    stx $2006
    ldx #$00
    stx $2006
    ldx #$27
    ldy #$20
.ClearPal stx $2007
    dey
    bne .ClearPal
.SetPallettes ldx #$3F
    stx $2006
    ldx #$00
    stx $2006
    ldx #$00
.ApplyToPal lda .Pal,x
    sta $2007
    inx
    cpx PALETTESIZE
    bne .ApplyToPal
.SetAttributeTbl ldx #$23
    stx $2006
    ldx #$C0
    stx $2006
    ldx #$00
.ApplyToAtt lda .Att,x
    sta $2007
    inx
    cpx ATTSIZE
    bne .ApplyToAtt
    ldx #$00
    ldx #$20
    stx $2006
    ldx #$00
    stx $2006
    ldx #$00
    ldy #$00
.SetMap1 lda .TitleMap1,X
    sta $2007
    inx
    cpx #$00
    bne .SetMap1
    ldy #$00
.SetMap2 lda .TitleMap2,X
    sta $2007
    inx
    cpx #$00
    bne .SetMap2
    ldy #$00
.SetMap3 lda .TitleMap2,X
    sta $2007
    inx
    cpx #$C0
    bne .SetMap4
    ldx #$00
    lda #$00
    sta $2003
.ApplyToSprite lda .Sprite,x
    sta $2004
    inx
    cpx SPRITESIZE
    bne .ApplyToSprite
    ldx #$00
    lda #%10010000
    sta $2000
    lda #%00011110
    sta $2001
.initialize
    lda #$FF
    sta BALLSPEED
    lda #$05
    sta INCSPEED
    lda #$CF
    sta MOVEX
    lda #$65
    sta MOVEY
    lda #$C7
    sta MOVE
    sta MOVE3
    lda #$CF
    sta MOVE2
    sta MOVE4
    lda #$01
    sta ANGLE
    lda #$FA
```

A quick taste of LuaPlayer...

```
green = Color.new(0, 255, 0)

screen:print(200, 100, "Hello World!", green)

for i=0,20 do
  x0 = i/20*479
  y1 = 271-i/20*271
  screen:drawLine(x0, 271, 479, y1, green)
end

screen.flip()
while not Controls.read():start() do
   screen.waitVblankStart()
end
```

```
green = Color.new(0, 255, 0)

screen:print(200, 100, "Hello World!", green)

for i=0,20 do
  x0 = i/20*479
  y1 = 271-i/20*271
  screen:drawLine(x0, 271, 479, y1, green)
end

screen.flip()
while not Controls.read():start() do
   screen.waitVblankStart()
end
```

An Introduction to LuaPlayer

With Joachim Bengtsson and Frank Buß

PSP and Its Open-Source API

With Joachim Bengtsson and Frank Buß

An Introduction to LuaPlayer

With Joachim Bengtsson and Frank Buß

```
sceUtilityGetSystemParamInt(PSP_SYSTEMPARAM_ID_INT_UNKNOWN,
            &dialog->buttonSwap); // X/O button swap

dialog->unknown[0] = 0x11;    // ???
dialog->unknown[1] = 0x13;
dialog->unknown[2] = 0x12;
dialog->unknown[3] = 0x10;
```

A simpler Hello World example

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>
#include <stdlib.h>
#include <string.h>
PSP_MODULE_INFO("CONTROLTEST", 0, 1, 1);
PSP_MAIN_THREAD_ATTR(THREAD_ATTR_USER | THREAD_ATTR_VFPU);
#define printf    pspDebugScreenPrintf
```

A simpler Hello World example

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>
#include <stdlib.h>
#include <string.h>
PSP_MODULE_INFO("CONTROLTEST", 0, 1, 1);
PSP_MAIN_THREAD_ATTR(THREAD_ATTR_USER | THREAD_ATTR_VFPU);
#define printf      pspDebugScreenPrintf
int done = 0;
int exit_callback(int arg1, int arg2, void *common)
{
    done = 1;
    return 0;
}
int CallbackThread(SceSize args, void *argp)
{
    int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
    sceKernelRegisterExitCallback(cbid);
    sceKernelSleepThreadCB();

    return 0;
}
int SetupCallbacks(void)
{
    int thid = 0;

    thid = sceKernelCreateThread("update_thread", CallbackThread,
                                 0x11, 0xFA0, 0, 0);
    if(thid >= 0)
    {
           sceKernelStartThread(thid, 0, 0);
    }

    return thid;
}
```

A simpler Hello World example

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>
#include <stdlib.h>
#include <string.h>
PSP_MODULE_INFO("CONTROLTEST", 0, 1, 1);
PSP_MAIN_THREAD_ATTR(THREAD_ATTR_USER | THREAD_ATTR_VFPU);
#define printf      pspDebugScreenPrintf


int main(void)
{
    SceCtrlData pad;

    pspDebugScreenInit();
    SetupCallbacks();

    sceCtrlSetSamplingCycle(0);
    sceCtrlSetSamplingMode(
       PSP_CTRL_MODE_ANALOG);
```

```
int done = 0;
int exit_callback(int arg1, int arg2, void *common)
{
     done = 1;
     return 0;
}
int CallbackThread(SceSize args, void *argp)
{
     int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
     sceKernelRegisterExitCallback(cbid);
     sceKernelSleepThreadCB();

     return 0;
}
int SetupCallbacks(void)
{
     int thid = 0;

     thid = sceKernelCreateThread("update_thread", CallbackThread,
                                  0x11, 0xFA0, 0, 0);
     if(thid >= 0)
    {
               sceKernelStartThread(thid, 0, 0);
    }

     return thid;
}
```

An Introduction to LuaPlayer

A simpler Hello World example

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>
#include <stdlib.h>
#include <string.h>
PSP_MODULE_INFO("CONTROLTEST", 0, 1, 1);
PSP_MAIN_THREAD_ATTR(THREAD_ATTR_USER | THREAD_ATTR_VFPU);
#define printf      pspDebugScreenPrintf

int main(void)
{
     SceCtrlData pad;

     pspDebugScreenInit();
     SetupCallbacks();

     sceCtrlSetSamplingCycle(0);
     sceCtrlSetSamplingMode(
       PSP_CTRL_MODE_ANALOG);
```

```
int done = 0;
int exit_callback(int arg1, int arg2, void *common)
{
     done = 1;
     return 0;
}
int CallbackThread(SceSize args, void *argp)
{
     int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
     sceKernelRegisterExitCallback(cbid);
     sceKernelSleepThreadCB();

     return 0;
}
int SetupCallbacks(void)
{
     int thid = 0;

     thid = sceKernelCreateThread("update_thread", CallbackThread,
                                  0x11, 0xFA0, 0, 0);
     if(thid >= 0)
    {
               sceKernelStartThread(thid, 0, 0);
    }

     return thid;
}
```

```
while(!done){
       pspDebugScreenSetXY(0, 2);

       sceCtrlReadBufferPositive(&pad, 1);
       printf("Hello, World!");
       printf("Analog X = %d ", pad.Lx);
       printf("Analog Y = %d \n", pad.Ly);

       if (pad.Buttons != 0) {
            if (pad.Buttons & PSP_CTRL_SQUARE)
                 printf("Square pressed \n");
            if (pad.Buttons & PSP_CTRL_TRIANGLE)
                 printf("Triangle pressed \n");
```

With Joachim Bengtsson and Frank Buß

An Introduction to LuaPlayer

```c
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>
#include <stdlib.h>
#include <string.h>
PSP_MODULE_INFO("CONTROLTEST", 0, 1, 1);
PSP_MAIN_THREAD_ATTR(THREAD_ATTR_USER | THREAD_ATTR_VFPU);
#define printf      pspDebugScreenPrintf

int main(void)
{
     SceCtrlData pad;

     pspDebugScreenInit();
     SetupCallbacks();

     sceCtrlSetSamplingCycle(0);
     sceCtrlSetSamplingMode(
       PSP_CTRL_MODE_ANALOG);
     while(!done){
              pspDebugScreenSetXY(0, 2);

     sceCtrlReadBufferPositive(&pad, 1);

              printf("Analog X = %d ", pad.Lx);
              printf("Analog Y = %d \n", pad.Ly);

              if (pad.Buttons != 0) {
                     if (pad.Buttons & PSP_CTRL_SQUARE)
                            printf("Square pressed \n");
                     if (pad.Buttons & PSP_CTRL_TRIANGLE)
                            printf("Triangle pressed \n");
              }
       }

     sceKernelExitGame();
     return 0;
}
```

A simpler Hello World example

```c
int done = 0;
int exit_callback(int arg1, int arg2, void *common)
{
     done = 1;
     return 0;
}
int CallbackThread(SceSize args, void *argp)
{
     int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
     sceKernelRegisterExitCallback(cbid);
     sceKernelSleepThreadCB();

     return 0;
}
int SetupCallbacks(void)
{
     int thid = 0;

     thid = sceKernelCreateThread("update_thread", CallbackThread,
                                  0x11, 0xFA0, 0, 0);
     if(thid >= 0)
    {
              sceKernelStartThread(thid, 0, 0);
    }

     return thid;
}
```

With Joachim Bengtsson and Frank Buß

```
#include <psptypes.h>
#include <stdlib.h>
#include <malloc.h>
#include <pspdisplay.h>
#include <psputils.h>
#include <png.h>
#include <pspgu.h>

#include "graphics.h"
#include "framebuffer.h"

#define IS_ALPHA(color) ((color)&0x8000?0:1)
#define FRAMEBUFFER_SIZE (PSP_LINE_SIZE*SCREEN_HEIGHT*2)
#define MAX(X, Y) ((X) > (Y) ? (X) : (Y))

u16* g_vram_base = (u16*) (0x40000000 | 0x04000000);

typedef struct
                                {
    unsigned short u, v;
    unsigned short color;
    short x, y, z;
} Vertex;

extern u8 msx[];

static unsigned int __attribute__((aligned(16))) list[256];
static int dispBufferNumber;
static int initialized = 0;
```

```
void initGraphics()
                                    {
    sceDisplaySetMode(0,SCREEN_WIDTH,SCREEN_HEIGHT);

    dispBufferNumber = 0;
    sceDisplayWaitVblankStart();
    sceDisplaySetFrameBuf((void*) g_vram_base, PSP_LINE_SIZE, 1, 1);

    sceGuInit();

    sceGuStart(GU_DIRECT, list);
    sceGuDrawBuffer(GU_PSM_5551, (void*)FRAMEBUFFER_SIZE, PSP_LINE_SIZE);
    sceGuDispBuffer(SCREEN_WIDTH, SCREEN_HEIGHT, (void*)0, PSP_LINE_SIZE);
    sceGuClear(GU_COLOR_BUFFER_BIT | GU_DEPTH_BUFFER_BIT);
    sceGuDepthBuffer((void*) 0x110000, PSP_LINE_SIZE);
    sceGuOffset(2048 - (SCREEN_WIDTH / 2), 2048 - (SCREEN_HEIGHT / 2));
    sceGuViewport(2048, 2048, SCREEN_WIDTH, SCREEN_HEIGHT);
    sceGuDepthRange(0xc350, 0x2710);
    sceGuScissor(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
    sceGuEnable(GU_SCISSOR_TEST);
    sceGuAlphaFunc(GU_GREATER, 0, 0xff);
    sceGuEnable(GU_ALPHA_TEST);
    sceGuDepthFunc(GU_GEQUAL);
    sceGuEnable(GU_DEPTH_TEST);
    sceGuFrontFace(GU_CW);
    sceGuShadeModel(GU_SMOOTH);
    sceGuEnable(GU_CULL_FACE);
    sceGuEnable(GU_TEXTURE_2D);
    sceGuTexMode(GU_PSM_5551, 0, 0, 0);
    sceGuTexFunc(GU_TFX_REPLACE, GU_TCC_RGBA);
    sceGuTexFilter(GU_NEAREST, GU_NEAREST);
    sceGuAmbientColor(0xffffffff);
    sceGuFinish();
    sceGuSync(0, 0);

    sceDisplayWaitVblankStart();
    sceGuDisplay(1);
    initialized = 1;
                                    }
```

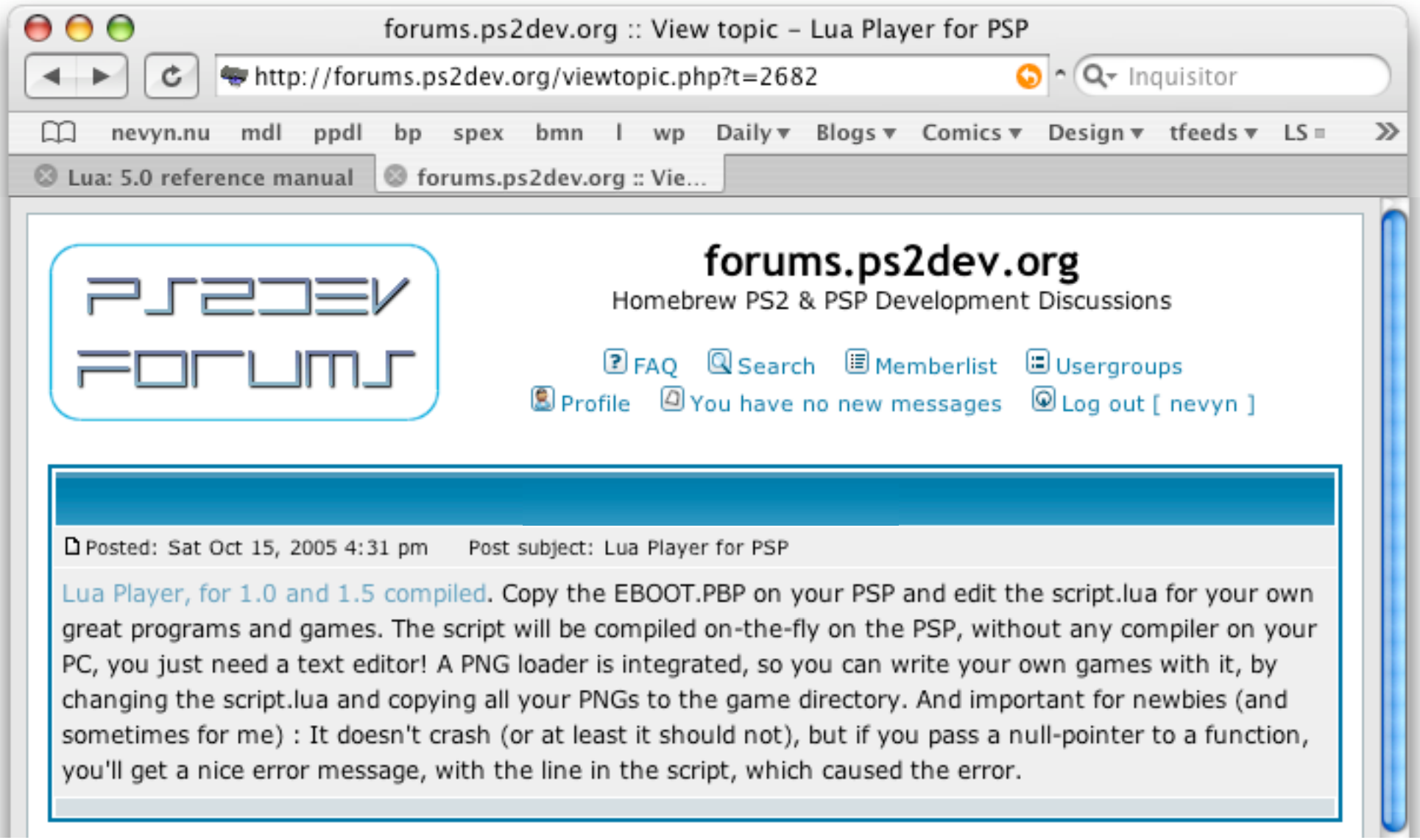
```
Color* getVramDrawBuffer()
                                                {
    Color* vram = (Color*) g_vram_base;
    if (dispBufferNumber == 0) vram += FRAMEBUFFER_SIZE / 2;
    return vram;
                                                }

Color* getVramDisplayBuffer()
                                                {
    Color* vram = (Color*) g_vram_base;
    if (dispBufferNumber == 1) vram += FRAMEBUFFER_SIZE / 2;
    return vram;
                                                }

static void drawLine(int x0, int y0, int x1, int y1, int color, Color* destination, int width)
                                                {
    int dy = y1 - y0;
    int dx = x1 - x0;
    int stepx, stepy;

    if (dy < 0) { dy = -dy;  stepy = -width; } else { stepy = width; }
    if (dx < 0) { dx = -dx;  stepx = -1; } else { stepx = 1; }
    dy <<= 1;
    dx <<= 1;

    y0 *= width;
    y1 *= width;
    destination[x0+y0] = color;
    if (dx > dy) {
            int fraction = dy - (dx >> 1);
            while (x0 != x1) {
                  if (fraction >= 0) {
                                              y0 += stepy;
                        fraction -= dx;
                  }
                  x0 += stepx;
                  fraction += dy;
                  destination[x0+y0] = color;
                                                }
```

```
    } else {
            int fraction = dx - (dy >> 1);
            while (y0 != y1) {
                  if (fraction >= 0) {
                                    x0 += stepx;
                        fraction -= dy;
                  }
                  y0 += stepy;
                  fraction += dx;
                  destination[x0+y0] = color;
            }
                                          }
}

void drawLineScreen(int x0, int y0, int x1, int y1, int color)
                                          {
    drawLine(x0, y0, x1, y1, color, getVramDrawBuffer(), PSP_LINE_SIZE);
                                          }

void printTextScreen(int x, int y, const char* text, u32 color)
                                          {
    int c, i, j, l;
    u8 *font;
    Color *vram_ptr;
    Color *vram;

    if (!initialized) return;

    for (c = 0; c < strlen(text); c++) {
            if (x < 0 || x + 8 > SCREEN_WIDTH || y < 0 || y + 8 > SCREEN_HEIGHT) break;
            char ch = text[c];
            vram = getVramDrawBuffer() + x + y * PSP_LINE_SIZE;

            font = &msx[ (int)ch * 8];
            for (i = l = 0; i < 8; i++, l += 8, font++) {
                                    vram_ptr  = vram;
                  for (j = 0; j < 8; j++) {
                        if ((*font & (128 >> j))) *vram_ptr = color;
                                    vram_ptr++;
                                     }
                  vram += PSP_LINE_SIZE;
            }
            x += 8;
                                          }
                                          }
```

Hardware specs

- TFT screen with 480x272 pixels and true color
- Two MIPS 4K CPU cores running at 220 MHz (333 MHz possible)
- 32 MB system RAM
- additional 166 MHz graphics CPU with 2 MB embedded VRAM
- hardware 3D polygons, NURBS, lighting, texture etc.
- 33 million flat-shaded polygons per second
- 664 million pixel per second fill
- UMD drive (DVD like drive for games and videos, 1.8 GB max)
- USB interface
- Wi-Fi interface (controlled by an additional ARM CPU)
- serial port interface
- memory stick slot
- analog pad, digital pad, many additonal keys

An Introduction to LuaPlayer



With Joachim Bengtsson and Frank Buß

Lua,
The Programming Language

With Joachim Bengtsson and Frank Buß

```
print("Hello world")
print("Hello world");

a = 10
b = "Hello World"

true == true
false ~= true

if a ~= b then
   while a < 12 then
         foo(a)
   end
end
```

```
chunk ::= {stat [`;´]}
block ::= chunk
stat ::= varlist1 `=´ explist1
                | functioncall
                | do block end | while exp do block end | repeat block until exp
                | if exp then block {elseif exp then block} [else block] end
                | return [explist1] | break
                | for Name `=´ exp `,´ exp [`,´ exp] do block end
                | for Name {`,´ Name} in explist1 do block end
                | function funcname funcbody | local function Name funcbody | local namelist [init]
funcname ::= Name {`.´ Name} [`:´ Name]
varlist1 ::= var {`,´ var}
var ::= Name | prefixexp `[´ exp `]´ | prefixexp `.´ Name
namelist ::= Name {`,´ Name}
init ::= `=´ explist1
explist1 ::= {exp `,´} exp
exp ::= nil | false | true
                | Number | Literal | function
                | prefixexp | tableconstructor | exp binop exp | unop exp
prefixexp ::= var | functioncall | `(´ exp `)´
functioncall ::= prefixexp args | prefixexp `:´ Name args
args ::= `(´ [explist1] `)´ | tableconstructor | Literal
function ::= function funcbody
funcbody ::= `(´ [parlist1] `)´ block end
parlist1 ::= Name {`,´ Name} [`,´ `...´] | `...´
tableconstructor ::= `{´ [fieldlist] `}´
fieldlist ::= field {fieldsep field} [fieldsep]
field ::= `[´ exp `]´ `=´ exp | name `=´ exp | exp
fieldsep ::= `,´ | `;´
binop ::= `+´ | `-´ | `*´ | `/´ | `^´ | `..´ | `<´ | `<=´ | `>´ | `>=´ | `==´ | `~=´ | and | or
unop ::= `-´ | not
```

An Introduction to LuaPlayer

With Joachim Bengtsson and Frank Buß

Key Features of Lua

- Simple syntax
- Dynamically typed
- Garbage collected
- The concept of tables
- Functions are first-class objects
- Easy integration with C and C++
- Procedural features like coroutines

Key Features of Lua

- Simple syntax
- Dynamically typed
- Garbage collected
- The concept of tables
- Functions are first-class objects
- Easy integration with C and C++
- Procedural features like coroutines

Key Features of Lua

- Simple syntax
- Dynamically typed
- Garbage collected
- The concept of tables
- Functions are first-class objects
- Easy integration with C and C++
- Procedural features like coroutines

```lua
foo = {"Hello", true, 1, 42}

foo = {}
foo[1] = "Hello"
foo[2] = true
foo[3] = 42

bar = {}
bar["x"] = 7
bar["boo"] = 8

baz = { x = 7, y = 8 }

tree = {
  root = {
    left = { 7, 8, 9 },
    middle = { true, "a" },
    more = { 3, deep = { 5 } } } }

print(tree.root.left[2])  --> 8

baz.x = 7
baz.y = 8
```

```
foo = function(x)
        print(2*x)
      end

mytable = { 7, foo }
mytable[2](mytable[1])  -->  14
```

```
function foo(x)
```

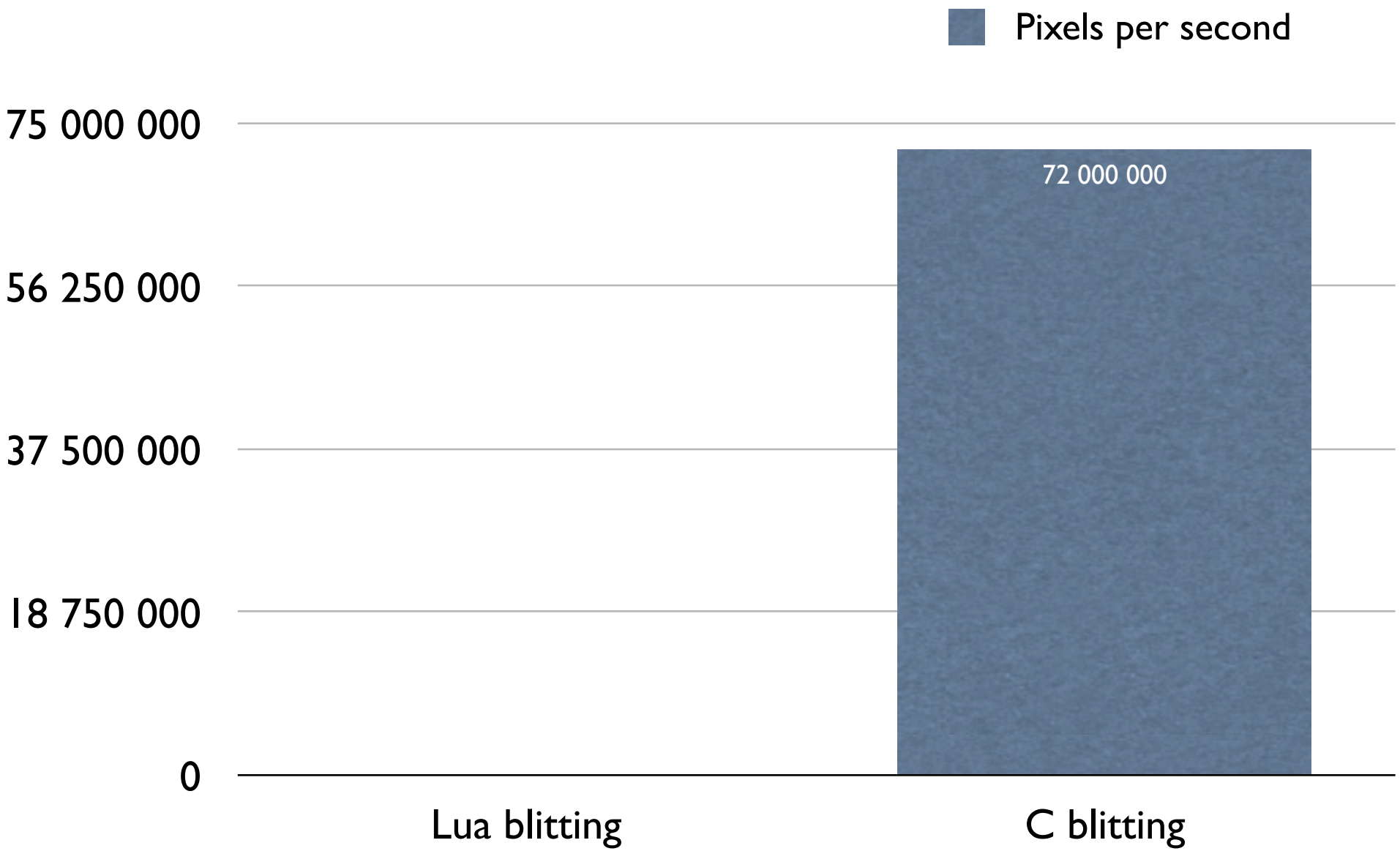
```
foo = function(x)
```

```
vectorFuncs = {}

vectorFuncs.__add = function(v1, v2)
                result = {}
  for index, value in v1 do
    result[index] = value + v2[index]
  end
  setmetatable(result, vectorFunctions)
  return result
end

vectorFuncs.__tostring = function(v)
  result = "( "
  for index, value in v do
    result = result .. value .. " "
  end
  return result .. ")"
end

v1 = { 4, 23, 9 }
v2 = { -2, 2.3, 8 }
setmetatable(v1, vectorFuncs)

print(v1 + v2)    -->    ( 2 25.3 17 )
```

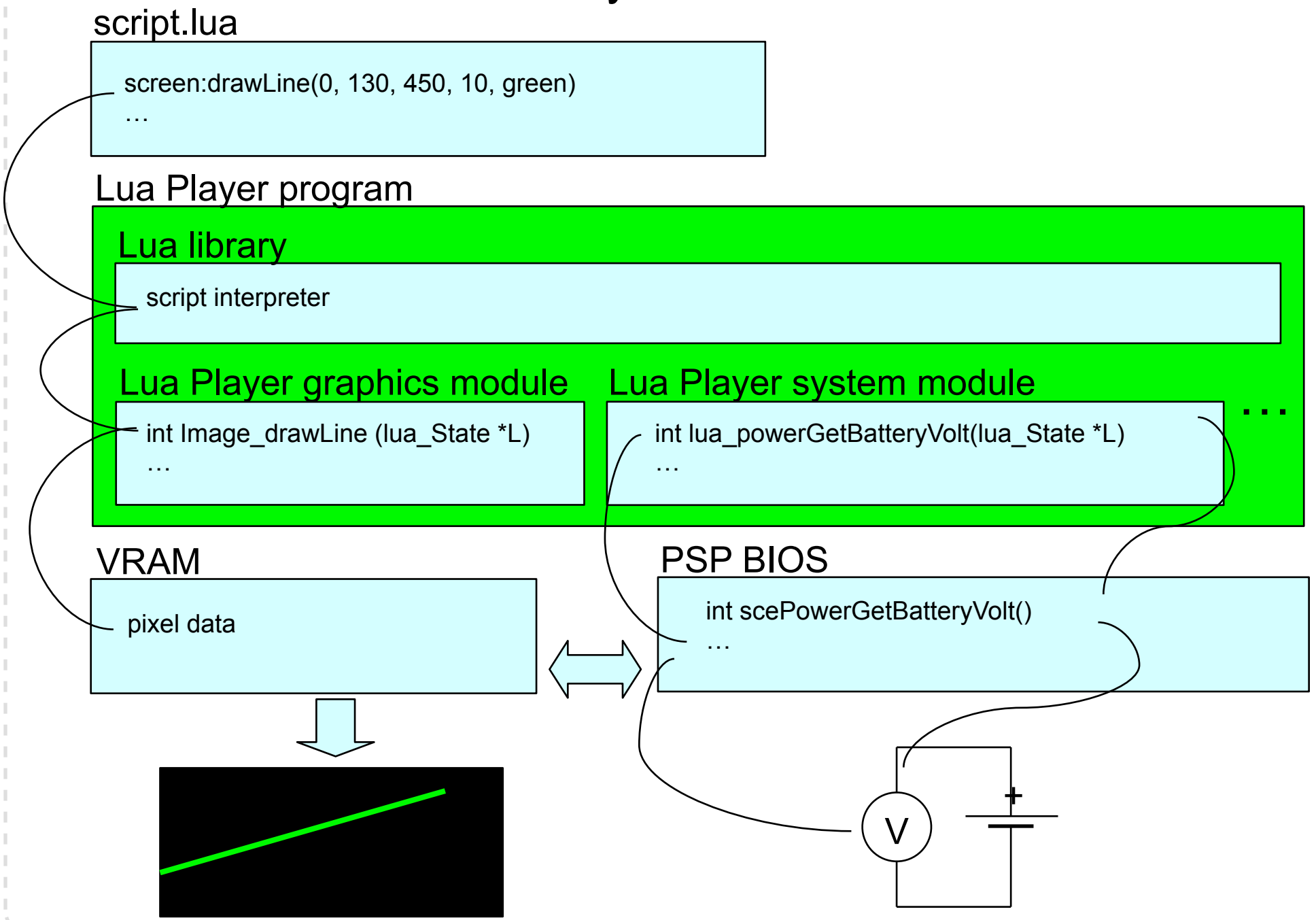
```lua
Vector = {}
Vector.__index = Vector
Vector.__add = vectorFunctions.__add
Vector.__tostring = vectorFunctions.__tostring

function Vector.create(values)
   local v = {}
   setmetatable(v, Vector)
   for index, value in values do v[index] = value end
   return v
end

function Vector:abs()
  local sum = 0
  for _, value in self do sum = sum + value * value end
  return math.sqrt(sum)
end

v = Vector.create({ 3, 4 })
print(v:abs())   -->   5
```

```
Vector = {}
Vector.__index = Vector
Vector.__add = vectorFunctions.__add
Vector.__tostring = vectorFunctions.__tostring

function Vector.create(values)
   local v = {}
   setmetatable(v, Vector)
   for index, value in values do v[index] = value end
   return v
end

function Vector:abs() ---> function Vector.abs(self, ...)
  local sum = 0
  for _, value in self do sum = sum + value * value end
  return math.sqrt(sum)
end

v = Vector.create({ 3, 4 })
print(v:abs())   -->   5
```

Combining The Two


Pixels per second
75 000 000
56 250 000
37 500 000
18 750 000
0
72 000 000
Lua blitting
C blitting

Lua Player Architecture



Programming for LuaPlayer

With Joachim Bengtsson and Frank Buß

An Introduction to LuaPlayer

- Graphics
  - creating offscreen images
  - Loading and saving PNG files
  - Blitting images to screen and to other images, with respect to the alpha channel or opaque
  - Pixel get/set, line draw, rect filling
  - Flipping the offscreen image with the visible image
  - Waiting for VSync
  - Color handling

- Controls
  - Reading current state and checking for d-pad, analog pad and all the other keys
  - Binary mask operations possible with Lua language extension

- Millisecond Timer
  - Class for measuring the time with millisecond precision

- System
  - Directory listing
  - USB connection to PC
  - Battery functions
  - Md5sum calculation (prints the same digest as "md5sum EBOOT.PBP" on Unix)
  - Serial port functions
  - Millisecond sleep function

- Sound and music
  - Music support currently with MikMod for UNI, IT, XM, S3M, MOD, MTM, STM, DSM, MED, FAR, ULT and 669
  - WAV file support for sound
  - Panning, volume, frequency etc.

An Introduction to LuaPlayer

- Graphics
  - creating offscreen images
  - Loading and saving PNG files
  - Blitting images to screen and to other images, with respect to the alpha channel or opaque
  - Pixel get/set, line draw, rect filling
  - Flipping the offscreen image with the visible image
  - Waiting for VSync
  - Color handling

- Controls
  - Reading current state and checking for d-pad, analog pad and all the other keys
  - Binary mask operations possible with Lua language extension

- Millisecond Timer
  - Class for measuring the time with millisecond precision

- System
  - Directory listing
  - USB connection to PC
  - Battery functions
  - Md5sum calculation (prints the same digest as "md5sum EBOOT.PBP" on Unix)
  - Serial port functions
  - Millisecond sleep function

- Sound and music
  - Music support currently with MikMod for UNI, IT, XM, S3M, MOD, MTM, STM, DSM, MED, FAR, ULT and 669
  - WAV file support for sound
  - Panning, volume, frequency etc.

An Introduction to LuaPlayer

- Graphics
  - creating offscreen images
  - Loading and saving PNG files
  - Blitting images to screen and to other images, with respect to the alpha channel or opaque
  - Pixel get/set, line draw, rect filling
  - Flipping the offscreen image with the visible image
  - Waiting for VSync
  - Color handling

- Controls
  - Reading current state and checking for d-pad, analog pad and all the other keys
  - Binary mask operations possible with Lua language extension

- Millisecond Timer
  - Class for measuring the time with millisecond precision

- System
  - Directory listing
  - USB connection to PC
  - Battery functions
  - Md5sum calculation (prints the same digest as "md5sum EBOOT.PBP" on Unix)
  - Serial port functions
  - Millisecond sleep function

- Sound and music
  - Music support currently with MikMod for UNI, IT, XM, S3M, MOD, MTM, STM, DSM, MED, FAR, ULT and 669
  - WAV file support for sound
  - Panning, volume, frequency etc.

An Introduction to LuaPlayer

- Graphics
  - creating offscreen images
  - Loading and saving PNG files
  - Blitting images to screen and to other images, with respect to the alpha channel or opaque
  - Pixel get/set, line draw, rect filling
  - Flipping the offscreen image with the visible image
  - Waiting for VSync
  - Color handling

- Controls
  - Reading current state and checking for d-pad, analog pad and all the other keys
  - Binary mask operations possible with Lua language extension

- Millisecond Timer
  - Class for measuring the time with millisecond precision

- System
  - Directory listing
  - USB connection to PC
  - Battery functions
  - Md5sum calculation (prints the same digest as "md5sum EBOOT.PBP" on Unix)
  - Serial port functions
  - Millisecond sleep function

- Sound and music
  - Music support currently with MikMod for UNI, IT, XM, S3M, MOD, MTM, STM, DSM, MED, FAR, ULT and 669
  - WAV file support for sound
  - Panning, volume, frequency etc.

An Introduction to LuaPlayer

- Graphics
  - creating offscreen images
  - Loading and saving PNG files
  - Blitting images to screen and to other images, with respect to the alpha channel or opaque
  - Pixel get/set, line draw, rect filling
  - Flipping the offscreen image with the visible image
  - Waiting for VSync
  - Color handling

- Controls
  - Reading current state and checking for d-pad, analog pad and all the other keys
  - Binary mask operations possible with Lua language extension

- Millisecond Timer
  - Class for measuring the time with millisecond precision

- System
  - Directory listing
  - USB connection to PC
  - Battery functions
  - Md5sum calculation (prints the same digest as "md5sum EBOOT.PBP" on Unix)
  - Serial port functions
  - Millisecond sleep function

- Sound and music
  - Music support currently with MikMod for UNI, IT, XM, S3M, MOD, MTM, STM, DSM, MED, FAR, ULT and 669
  - WAV file support for sound
  - Panning, volume, frequency etc.

With Joachim Bengtsson and Frank Buß

An Introduction to LuaPlayer

- Graphics
  - creating offscreen images
  - Loading and saving PNG files
  - Blitting images to screen and to other images, with respect to the alpha channel or opaque
  - Pixel get/set, line draw, rect filling
  - Flipping the offscreen image with the visible image
  - Waiting for VSync
  - Color handling

- Controls
  - Reading current state and checking for d-pad, analog pad and all the other keys
  - Binary mask operations possible with Lua language extension

- Millisecond Timer
  - Class for measuring the time with millisecond precision

- System
  - Directory listing
  - USB connection to PC
  - Battery functions
  - Md5sum calculation (prints the same digest as "md5sum EBOOT.PBP" on Unix)
  - Serial port functions
  - Millisecond sleep function

- Sound and music
  - Music support currently with MikMod for UNI, IT, XM, S3M, MOD, MTM, STM, DSM, MED, FAR, ULT and 669
  - WAV file support for sound
  - Panning, volume, frequency etc.

```
time = 0
pi = math.atan(1) * 4
background = Image.load("background.png")
smiley = Image.load("smiley.png")
while true do
   screen:blit(0, 0, background, 0, 0, background:width(),
                background:height(), false)

   x = math.sin(pi * 2 / 250 * time) * 200 + 220.5
   y = 172 - math.abs(math.sin(pi * 2 / 125 * time) * 150)
  screen:blit(x, y, smiley)
   time = time + 1
   if time >= 500 then
        time = 0
   end

   screen.waitVblankStart()
   screen.flip()

   pad = Controls.read()
   if pad:start() then
        break
   end
end
```

```
time = 0
pi = math.atan(1) * 4
background = Image.load("background.png")
smiley = Image.load("smiley.png")
while true do
   screen:blit(0, 0, background, 0, 0, background:width(),
                background:height(), false)

   x = math.sin(pi * 2 / 250 * time) * 200 + 220.5
   y = 172 - math.abs(math.sin(pi * 2 / 125 * time) * 150)
  screen:blit(x, y, smiley)
   time = time + 1
   if time >= 500 then
        time = 0
   end

   screen.waitVblankStart()
   screen.flip()

   pad = Controls.read()
   if pad:start() then
        break
   end
end
```

Community Response

An Introduction to LuaPlayer

With Joachim Bengtsson and Frank Buß

Live Demo