VM LABS

NUON Sprite Library

Programmer's Guide

Version 0.6

VM Labs, Inc.
520 San Antonio Road
Mountain View, CA 94040
Tel: (650) 917 8050
Fax: (650) 917 8052

Copyright notice

Contents

1. Introduction

1.1 Overview

The **`libsprite.a`** library is a powerful 2D sprite engine. It was originally written by Jeff Minter (also known as YaK) as part of his object rendering engine used in Tempest 3000 and the Virtual Light Machine. This library is the attempt to make Jeff's work available to C programmers.
The sprite engine features:

- 32bpp sprites
- no sprite size limitation
- free sprite scaling
- free sprite rotation
- various sprite types (incl. one with bi-linear interpolation, blending and transparency)
- easy to use C-API

The basic concept is that sprites are defined as rectangular regions in a source image. So created sprites can than be aded to a display list. the display list can be drawn by the use of one or more render MPEs.

1.2 Future versions

This version is missing a few features which we will add in future revisions of the library:

- Support for 16bpp sprites
- collision detection
- more sprite types
- Z-Buffer support

2. API Description

2.1 Initialization

2.1.1 SPRInit

`int SPRInit(int startmpe, int endmpe, int sliceHeight)`

This call initializes the sprite library and is starting the renderers on the selected MPEs.

Parameter	**Values**	**Comments**
startmpe	0-3	First MPE used for rendering
endmpe	0-3	Last MPE used for rendering
sliceHeight		Height of slices (when rendering with more than MPE). 16 is a good value!
Return value	int	0 = No error -1 = Error

2.1.2 SPRInstallTGAImage

`int SPRInstallTGAImage(char *tga, int mode, SPR_IMAGE_INFO *info, int trans-ColorRGB)`

Installs a given TGA image in SysRAM as a sprite source.
Since the sprite source has to be in the internal frame buffer format, it is necessary to install a regular TGA image. During this process the image is copied to SDRAM (which get allocated with the _MemAlloc BIOS call).
Currently the TGA image has to be in 8-bit indexed, RLE format. The internal storage format in this release is 32bpp.
The optional alpha channel in TGA file is supported and can be used for transparent sprites. For TGA images without alpha channel, it's possible to define one RGB value which is then used as the transparent color. **`SPRInstallTGAImage`** will compare all pixel values during the conversion with this RGB value and create an alpha mask on the fly.
The function fills out a given info structure. This structure is then used in subsequent **`SPRCreateSprite`** calls. Of course it is possible to have more than one source image (as SDRAM permits).

Parameter	**Values**	**Comments**
tga		TGA image in memory (8-bit, RLE format!)
mode	0,1	Internal storage format 0 = 16 bpp (currently not supported!) 1 = 32 bpp
info		Pointer to a SPR_IMAGE_INFO structure
transColorRGB		RGB color in TGA image used as transparent
Return value	int	0 = No error -1 = Error

2.1.3 SPRSetSourceImage

`int SPRSetSourceImage(char *scr, int dmafl, int width, int height, SPR_IMAGE_INFO *info)`

Installs a given frame buffer in SDRAM as a sprite source. It has to be in 32bpp, NUON-native DMA format! The function fills out a given info structure. This structure is then used in subsequent **`SPRCreateSprite`** calls. Of course it is possible to have more than one source image (as SDRAM permits).

2.2 Sprite API

2.2.1 SPRCreateSprite

`SPRITE *SPRCreateSprite(SPR_IMAGE_INFO *info,int x, int y, int w, int h)`

Creates a new sprite. The source image is given by the $info$ parameter. A sprite is defined by the given coordinates within the source image.
A newly created sprite is not immediately visible! In order to make it visible on the screen, it has to be added to the display list with **`SPRAddSprite`**.

Parameter	**Values**	**Comments**
info		source image
x,y,w,h		coordinates within source image
Return value		Sprite handle 0 = Error

2.2.2 SPRCloneSprite

`SPRITE *SPRCloneSprite(SPRITE *s)`

Clones an existing sprite.

2.2.3 SPRAddSprite

`int SPRAddSprite(SPRITE * sprite, int x, int y, int angle, int xscale, int yscale, int type, int tcol, int depth)`

Add sprite to display list. The sprite has to be created with **`SPRCreateSprite`**. If the given sprite is already on the display list, the function returns an error.

Parameter	**Values**	**Comments**
sprite		Sprite handle
x,y		destination coordinates
angle		rotation angle (16.16), 1.0 = 360 degrees
xscale,yscale		scale factors (16.16) Max: 0x30000
type	 kSpriteSimple kSpriteSimpleTrans kSpriteInterpolatedTrans kSpriteInterpolated	Bit 0-3: Sprite Type Non-Interpolated, no Transparency Non-Interpolated, single transparency value Interpolated, Blended, Transparency Interpolated, no Transparency Bit 4-31: Blending value (2.26) only for kSpriteInterpolatedTrans
tcol		Transparent color for kSpriteSimpleTrans
depth		Position in display list
Return value		0 = no Error -1 = Error (already on display list)

2.2.4 SPRModifySprite

`int SPRModifySprite(SPRITE * sprite, int x, int y, int angle, int xscale, int yscale, int type, int tint, int depth)`

Modify sprite parameters (parameters see **`SPRAddSprite`**).

2.2.5 SPRRemoveSprite

`int SPRRemoveSprite(SPRITE *sprite)`

Remove sprite from display list.

2.2.6 SPRDeleteSprite

`int SPRDeleteSprite(SPRITE *sprite)`

Deletes a sprites and frees up the used memory resources.

2.2.7 SPRSetSpriteSource

`int SPRSetSpriteSource(SPRITE * sprite, int x, int y, int w, int h)`

Changes the sprite’s source coordinates (parameters see **`SPRCreateSprite`**).

2.2.8 SPRSetSpriteXY

`int SPRSetSpriteXY(SPRITE * sprite, int x, int y)`

Changes the sprite’s destination coordinates.

2.2.9 SPRSetSpriteScale

```
int SPRSetSpriteScale(SPRITE * sprite, int xscale, int yscale)
```

Changes the sprite's X and Y scale.

2.2.10 SPRSetSpriteRotation

```
int SPRSetSpriteRotation(SPRITE * sprite, int angle)
```

Changes the sprite's rotation angle.

2.2.11 SPRSetSpriteType

```
int SPRSetSpriteType(SPRITE * sprite, int type)
```

Changes the sprite type.

2.2.12 SPRSetSpriteTColor

```
int SPRSetSpriteTColor(SPRITE * sprite, int tint)
```

Changes the sprite's transparent color (only for sprite type **`kSpriteSimpleTrans`**!).

2.2.13 SPRSetSpriteDepth

```
int SPRSetSpriteDepth(SPRITE * sprite, int depth)
```

Changes the sprite depth. This will effect the drawing order. Sprites with a smaller depth value will be drawn first, therefore will be behind sprites with a bigger depth value.

2.3 Rendering API

2.3.1 SPRSetDestScreen

```
int SPRSetDestScreen(void *scr, long dmaflags, int minx, int miny, int maxx,
int maxy, int bgColor)
```

Defines the destination screen and clipping coordinates.

2.3.2 SPRDraw

```
int SPRDraw(int clrScreen, int wait)
```

Processes the display list and draws all sprites. It is important to wait for all renderers to finish drawing before calling this API again or modifying sprite parameters. However is might be useful to do other, non-sprite related computations while the renderers are busy. In this case it makes sense to set the $wait$ parameter to 0 and use an explicit **`SPRWait`** call.

Parameter	**Values**	**Comments**
clrScreen		1 = fill screen with background color (given in **`SPRSetDestScreen`**)
wait		1 = wait for all render MPEs finish drawing
Return value		Sprite handle 0 = Error

2.3.3 SPRWait

`int SPRWait(void)`

Wait for all render MPEs to finish drawing.

2.4 Low-Level API

2.4.1 *SPRBlitter*

`int SPRBlitter(char *src, long src_dmaflags, int src_x, int src_y, int src_w, int src_h, char *dest, long dest_dmaflags, int dest_x, int dest_y, int scalex, int scaley, int angle, int clip_min_x,int clip_min_y,int clip_max_x,int clip_max_y, int type, int transColor)`

This function is a low-level access to the sprite routines. It bypasses the sprite management and can be use to directly draw on the screen. It is like a software blitter and should only be used if all render MPEs are idle (see **`SPRWait`** and **`SPRDraw`**).
The parameter list is long, since it is necessary to specify all information normally stored in the internal sprite management structures.

Parameter	**Comments**
src	Source buffer (NUON DMA format)
src_dmaflags	Source DMA flags
src_x src_y src_w src_h	Source rectangle
dest	Destination buffer (NUON DMA format)
dest_dmaflags	Destination DMA flags
dest_x dest_y	Destination coordinates
scalex scaley	Scale factors (**`SPRAddSprite`**)
angle	Rotation angle (**`SPRAddSprite`**)
clip_min_x clip_min_y clip_max_x clip_max_y	Clipping rectangle
type transColor	Sprite type and transparent color (**`SPRAddSprite`**)
Return value	<0 = Error 0 = Error

3. Sample Program

Below a simple example drawing one sprite.

```
#include <stdio.h>
#include "sprite.h"
#include <nuon/bios.h>
#include <nuon/dma.h>
#include <nuon/video.h>
#include <nuon/mml2d.h>

#define SCREEN_WIDTH 360
#define SCREEN_HEIGHT 240
#define DMAFLAGS  (DMA_PIXEL_WRITE | ((SCREEN_WIDTH/8)<<16) | (4<<4) | (1<<11))

extern char SpriteImage[];

void *FrameBuffer[2];

int main()
{
    SPR_IMAGE_INFO img;
    SPRITE *sprite;
    VidDisplay display;
    VidChannel mainchannel;
    int i;

    /* Setup video */

    FrameBuffer[0]=_MemAlloc(SCREEN_WIDTH*SCREEN_HEIGHT*4,512,kMemSDRAM);

    display.dispwidth = SCREEN_WIDTH;
    display.dispheight = SCREEN_HEIGHT;
    display.bordcolor=0x20100000;
    display.progressive=0;
    for(i=0;i<6;i++)
        display.reserved[i]=0;

    mainchannel.dmaflags=DMAFLAGS;
    mainchannel.base=FrameBuffer[0];
    mainchannel.dest_xoff = -1;
    mainchannel.dest_yoff = -1;
    mainchannel.dest_width = 720;
    mainchannel.dest_height = 480;
    mainchannel.src_xoff = 0;
    mainchannel.src_yoff = 0;
    mainchannel.src_width = SCREEN_WIDTH;
    mainchannel.src_height = SCREEN_HEIGHT;
    mainchannel.clut_select=0;
    mainchannel.alpha=0;
    mainchannel.vfilter=VID_VFILTER_2TAP;
    mainchannel.hfilter=VID_HFILTER_4TAP;
    for(i=0;i<5;i++)
        mainchannel.reserved[i]=0;

    _VidConfig ( &display, &mainchannel, 0L, 0L);

    /* Use MPEs 0,1 and 2 for rendering, 16 pixel slices */
    SPRInit(0,2,16);

    /* Set to the framebuffer */
    SPRSetDestScreen(FrameBuffer[0], DMAFLAGS,  0,  0,  359,  239, kBlack);

    /* Install the TGA image */
    SPRInstallTGAImage(SpriteImage, 1, &img, 0xf0f0ee00);

    /* Create a sprites */
    sprite=SPRCreateSprite(&img,1,1,26,26);

    /* Add sprite to display list*/

    SPRAddSprite(sprite, 180, 120, 0, 0x10000, 0x10000, 0x40000000 | kSpriteSimpleTra

    SPRSetDestScreen(FrameBuffer[0], DMAFLAGS,  0,  0,  359,  239, kBlack);

     /* Process display list */
    SPRDraw(1,1);

    /* Done */
    for(;;);
}
```

Index