

Draw to the screen?

Setup

There's quite a bit going on in the setup, but it thankfully is mostly boilerplate.

The functions you want, in order, are:

- `VIDEO_Init()`: This needs to be called before any other VIDEO functions. Documentation says it should be done "in the early stages of your `main()`", but we've had success in moving it to a different function that is called from main.
- `VIDEO_GetPreferredMode()`: This returns you your `GXRModeObj` struct pointer (a "rendermode object") defining the screen layout and mode. What you get from this depends on the console settings, and if it's PAL or NTSC, things like that. The parameter also takes a `GRXModeObj` and seems to overwrite it if present? Not sure why you'd want that, just pass `NULL`.
- `SYS_AllocateFramebuffer()`: Allocate cacheline aligned memory for the external framebuffer based on the passed rendermode object. Returns a 32-byte-aligned pointer to the framebuffer's start address. This allocates the memory for our framebuffer, but we cannot use it yet.
- `MEM_K0_TO_K1()`: This is a macro to cast a cached virtual address to an uncached virtual address. In this case, we use this to convert the address of the allocated framebuffer to one we can use to write to it.
- `VIDEO_Configure()`: Configures the "VI" (video interface?) with the given rendermode object. Internally, this is setting a lot of video registers with our rendermode settings
- `VIDEO_SetNextFramebuffer()`: This sets some video register to point to our allocated framebuffer memory. In effect, we are telling the video hardware where the framebuffer is.
- `VIDEO_SetBlack()`: TRUE to black out the screen, FALSE not to.
- `VIDEO_Flush()`: When we make changes to the video hardware registers in the above functions, we do not *actually* make those changes to the hardware until we flush them with this function. This in effect commits our changes.
- `VIDEO_WaitVSync()`: Waits for the next vertical retrace.

Putting all that together, a simple (minimal) video setup would look like:

```
#include <gccore.h>
```

```

static void* xfb = nullptr;
static GXRModeObj* rmode = nullptr;

void video_initialise()
{
    // Initialise the video system
    VIDEO_Init();

    // Obtain the preferred video mode from the system
    // This will correspond to the settings in the Wii menu
    rmode = VIDEO_GetPreferredMode(NULL);

    // Allocate memory for the display in the uncached region
    xfb = MEM_K0_TO_K1(SYS_AllocateFramebuffer(rmode));

    // Set up the video registers with the chosen mode
    VIDEO_Configure(rmode);

    // Tell the video hardware where our display memory is
    VIDEO_SetNextFramebuffer(xfb);

    // Make the display visible
    VIDEO_SetBlack(FALSE);

    // Flush the video register changes to the hardware
    VIDEO_Flush();

    // Wait for Video setup to complete
    VIDEO_WaitVSync();
    if (rmode->viTVMode & VI_NON_INTERLACE)
        VIDEO_WaitVSync();
}

```

Text output

Getting console output text out to the screen is quite easy. We need an additional setup function, `CON_Init()` (sometimes seen with the old name `console_init()`). This takes as input:

- The framebuffer (the same parameter we gave to `VIDEO_SetNextFramebuffer()`, `xfb` in examples).
- The x, then y-coordinate to start output from. This allows for a border around the edge of the screen.
- The full width, then height, of the screen. Get these from the rendermode object (`rmode` in the example).
- The *stride*, which is the size of one line of the framebuffer in bytes. This is the width, multiplied by the constant `VI_DISPLAY_PIX_SZ`, which is the size of one pixel in bytes.

And then that's it! You can then use `printf()` (from `<stdio.h>`) normally and it prints to the screen.

The console also accepts [VT terminal escape codes](#), notably the `cursorpos` one to set the console position. The statement `printf("\x1b[2;0H");` sets the cursor to row 2, character 0. Remember the padding border in the init function, a pos of 0,0 is not going to be the top-left of the screen if you have a border set.

An example of printing to the console looks like:

```
int main()
{
    // Initialise the video interface.
    video_initialise()

    // Initialise the console.
    CON_init(xfb, 20, 20, rmode->fbWidth, rmode->xfbHeight, rmode->fbWidth * VI_DISPLAY_PIX_SZ)

    // Set the cursor to row 2.
    printf("\x1b[%d;%dH", 2, 0);

    // Print the text.
    printf("Hello World!");
}
```

Drawing to the framebuffer

Drawing to the framebuffer directly is the simplest method of drawing to the screen. All that's required is writing the pixel values to the array directly inbetween clears each frame.