**The (Un)Official GBAtemp Guide to 3DS Homebrew Development**

# --Table of Contents--

Preface

A Little More Info

Chapter 1: Before We Begin... (Requirements, Languages)

Chapter 1-2: Terms you need to know

Chapter 2: And Now We Start (Hello World Example and Breakdown)

Chapter 3: So... What and how did it even show up? (Framebuffers, VRAM)

Chapter 4: A picture is 1000 lines of code (.c source images)

Chapter 5: Buttons, Buttons, and more Buttons (HID, Input, Touch Screen)

# --Preface--

I set myself up for a lot of work. I have enough announced projects to deal with. It's complicated to make a tutorial about C while learning it. It'll pay off eventually. Hopefully more people will start making 3DS homebrew because of this, (which is my goal). Well, enjoy.

# --A little more info--

This book is based off of Ctrulib (The GitHub build was downloaded on May 7th, 2016). If you don't have the latest version in your devkitPro, you should update it. After each chapter I'll be introducing new terms that was seen in that chapter.

# --Chapter 1: Before We Begin--

Before we actually get into it this tutorial assumes that you:

1: Know the C Programming Language

2: Installed and setup devkitPro and the 3DS environment

3: Have a 3DS with a Homebrew Environment (\*hax and Homebrew Launcher) or Citra

There are many great C Tutorials and Books out there.

I recommend Derek Banas C Tutorial (16 Videos) on YouTube and

Cprogramming.com's C Tutorial (if you want to go quicker)

Once you have those two things ready, then move on with the tutorial.

Also, when creating 3DS homebrew, you need a Makefile, and a source folder, along with your Main.c file in it

Structure:

Project folder

-------------source folder

------------------------------main.c

-------------Makefile

The basic files (makefile and main.c) will come with this PDF.

# --Chapter 1-2: Terms you need to know--

C/C++ - Programming languages used in many homebrew scenes

Binary - The main language a computer understands that consists of 0 and 1

devkitPro –

# --Chapter 2: And Now We Start--

Welcome. I guess you’re here to learn how to make homebrew, or at least brush up on your skills (which is okay too).

I'll go over a lot of the fundamentals of 3DS homebrew, and try to cover many features, like utilizing the Camera (not much homebrew does that, if any), or using the buttons.

Now I'll show you some code, and then we'll break it down.

Template.c

#include <stdio.h>

#include <3ds.h>

int main(int argc, char \*\*argv)

{

//Initialize GFX (note: not needed if you're using SF2Dlib)

gfxInitDefault();

consoleInit(GFX\_TOP, NULL);

printf("\x1b[15;19HHello World!");

// Main loop

while (aptMainLoop())

{

hidScanInput();

//hidKeysDown returns information about which buttons have been just pressed (and they weren't in the previous frame)

//hidKeysUp returns information about which buttons are currently pressed (regardless if they were pressed or not pressed in the previous frame)

//hidKeysHeld returns information about which buttons are not pressed but were pressed in the previous frame

u32 kDown = hidKeysDown();

if (kDown & KEY\_START) break; // break in order to return to hbmenu

//Flush and swap framebuffers, this is needed for rendering these will not be needed when using SF2D lib

gfxFlushBuffers();

gfxSwapBuffers();

//Wait for VBlank, this is needed for rendering these will not be needed when using SF2D lib

gspWaitForVBlank();

}

gfxExit();

return 0;

}

This may be a lot of code, but the only thing it will actually do is display "Hello World!" on the screen. Let's go over it.

The first thing you see are the 2 preprocessors which include stdio.h (which holds printf and other main C functions), and 3ds.h (that has all of the 3ds features in it). Next you see the start of the main function, which is where every homebrew starts. Inside the main function, you see GFX being initialized. GFX are the graphics of the 3DS. Next you see the console being initialized for the words that are being printed. Inside the parentheses you see, GFX\_TOP and NULL which initializes the top screen with GFX\_TOP, but makes it empty with NULL. Next you see the printf function that prints "Hello World!" to the screen. The characters before Hello World are actually coordinates where to put the text. In "\x1b[15;19" the 15 can be replaced with a number that controls the y coordinate (vertical), the number 19 can be replaced with a number that controls the x variable (horizontal), but every other character has to stay. Next you see the main loop which everything in this loop is checked every frame. In the main loop, first you see the function hidScanInput() which scans the HID (or Human Input Device) for input which will be used in later code. Next you see the function “u32 kDown = hidKeysDown();”, and what this means is that it will scan to see if a button is pressed. On the above comments, it tells about all the different button functions. If you wanted to check which buttons were held you would add a function under “kDown” that will be “u32 kHeld = hidKeysHeld();” and the same for “hidKeysUp” and if you wanted to check what buttons are currently pressed. Under that, you see the START button being utilized to end the “Hello World” program and go back to the Homebrew Launcher. Here are the list of all the buttons you can use in code:

List of Buttons:

KEY\_A (A button)

KEY\_B (B button)

KEY\_SELECT (Select button)

KEY\_START (Start button)

KEY\_DRIGHT (D-Pad Right button)

KEY\_DLEFT (D-Pad Left button)

KEY\_DUP (D-Pad Up button)

KEY\_DDOWN (D-Pad Down button)

KEY\_R (R button)

KEY\_L (L button)

KEY\_X (X button)

KEY\_Y (Y button)

KEY\_ZL (ZL button, New 3DS only)

KEY\_ZR (ZR button, New 3DS only)

KEY\_TOUCH (Touch, not actually provided by HID)

KEY\_CSTICK\_RIGHT (C-Stick Right, New 3DS only)

KEY\_CSTICK\_LEFT (C-Stick Left, New 3DS only)

KEY\_CSTICK\_UP (C-Stick Up, New 3DS only)

KEY\_CSTICK\_DOWN (C-Stick Down, New 3DS only)

KEY\_CPAD\_RIGHT (Circle Pad Right)

KEY\_CPAD\_LEFT (Circle Pad Left)

KEY\_CPAD\_UP (Circle Pad Up)

KEY\_CPAD\_DOWN (Circle Pad Down)

You may have saw that the touch screen isn’t provided by HID, meaning we have to take extra steps to use it, but we’ll go over that later. Next, you see “gfxFlushBuffers();” “gfxSwapBuffers();” which Flushes and Swap the Framebuffers. The Framebuffers are flushed (reset) and swapped to prevent the framebuffers from messing up and to always draw the next frame without risking to alter what is currently on the screen. We’ll go over Framebuffers more in the next chapter, but for now, just know that Framebuffers are what displays every frame. After that, you see “gspWaitForVBlank();” which isn’t important now. Then you see the end of the main loop, the exit of the GFX, and the ending of the Homebrew. You can copy and paste this code in your IDE/Editor or use the Template.c file in the code files.

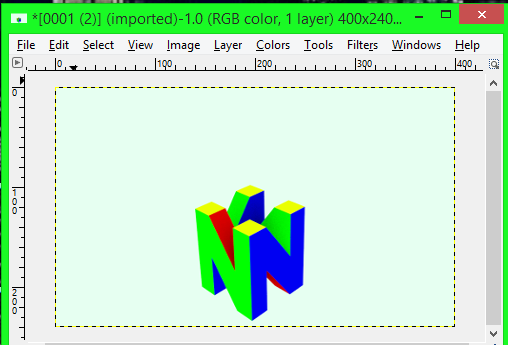
# --Chapter 3: So... What and how did it even show up?--

Well like described in the last chapter, the way this happened is because of something called VRAM, and it holds information of three screens (which is really 2 screens). The first screen, or the top-left screen, is the screen that the 3D mode uses (this screen, and top-right). The second screen, or the top-right screen, is the main screen that is the main screen (the one used in 2D mode), and the one for the right eye in 3D mode. The third screen is the bottom screen which is 2D only.

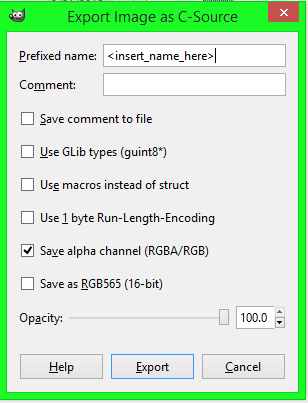
In each screen there is something that is called Framebuffers which are 2 different images that replace each other. The two images are called fb0 and fb1. When fb0 is shown on the screen, fb1 renders for the next frame, and when fb1 is shown, fb0 gets ready, and this happens 60 times a second. And after we finish the program, we flush and swap the Framebuffers, like we saw last chapter.

# --Chapter 4: A picture is 1000 lines of code--

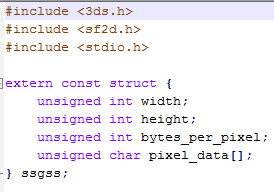
Well it's not really 1000 lines of code. We’ll be using the sf2d\_template file included in the code files. There are many different ways to use images in your homebrew, such as .BIN files, .PNG/.JPG files, or .C files which we’ll be going over. Before you start this chapter, you’re going to need the free program GIMP so you can export your images in .C source format. Once you have it, start GIMP and resize the image to 400x240 (the resolution of the 3DS screen). Just click Image and click Scale Image.



Next click the File tab and click Export As and where it says “All Export Images“, change it to a .C source file.



Keep these exact same settings except for the “<insert\_name\_here>” part. Change that to the name you want to use in code. Now open the C file you just exported and at the beginning of the file, change “static const struct” to just “const struct” or else it won’t work when compiling it. You have to do this with every .C file you use as an image. After you do that put it in your source folder with Main.c. Now in your Main.c that’s in the sf2d\_template, add this line of code right after the header files:



Don’t worry about the “ssgss”, that’s from another time. Just replace that with whatever you named your image in the above GIMP option. After the main function, but before the main loop you will to add:



Everywhere you see “ssgss”, you replace it with the name of the file you exported in GIMP where it said (<insert\_name\_here>). This comes after sf2d is initialized (“sf2d\_init();”) and when the background color is set (“sf2d\_set\_clear\_color(RGBA8(0x40, 0x40, 0x40, 0xFF));”). Next, after hidScanInput() and where you define kHeld, kUp, and kDown, you add this code:



And at the end right before ending the sf2d library (sf2d\_fini), you’ll add:



And now to go over the code.

The first part you see in the extern const struct is the parts that define the parts to use the images data (ssgss.width, ssgss.height, etc.). Next you see the variables that were just defined. Where you see “ssgss.pixel\_data[]” was defined and it means to use the pixel data from the image itself. Next you see ssgss.width and ssgss.height which both tell the program to use the width and height of the picture (400x240 for the top screen). “TEXFMT\_RGBA8” means to use the Texture Format of RGBA8 and “SF2D\_PLACE\_RAM” means to place the image in the system’s RAM. Next you see the 3 functions you added. “Sf2d\_start\_frame” means to start a new frame, and in the parentheses, you see GFX\_TOP, and GFX\_LEFT. If you remember from Chapter 2, that it means to use the top screen of the 3DS. Next is GFX\_LEFT and that says to use the left screen (or the main screen) of 3D mode. Next is “sf2d\_draw\_texture” which means to draw a texture to the screen. In the parentheses you see tex\_image (keep that, it just means to use the defined image above), and it starts on the 0 X coordinate and the 0 Y coordinate since it takes up the entire screen exactly. And then you end the frame with “sf2d\_end\_frame”. The last line you add is “sf2d\_free\_texture” and that means to stop displaying the image and the “tex\_image” means that it’s just doing it to the image. We’ll go over other things about images in a later chapter.