

Nintendo DS Development Notes and Information

Andrew Cox
andrew.cox@cs.ucl.ac.uk
<http://www.cs.ucl.ac.uk/students/Andrew.Cox/>
<http://www.btinternet.com/~ahcox/>¹

March 12, 2006

¹Many thanks to everyone whose edited words are reproduced in this doc. That includes: dovoto, joat, ecurtz, DekuTree64, ficedula, and many others. Greetings and thanks to you all.

Contents

1	Distillation of Online info: May be Reliable or Not	2
1.1	3D	2
1.1.1	Basics	2
1.1.2	Polygon Limits	2
1.1.3	Display Capture	2
1.1.4	Antialiasing and Outlining	3
1.1.5	Characters and Skinning	3
1.1.6	Display Lists	4
1.1.7	Misc	4
1.2	Linking, Loading, and Memory	4
1.2.1	GBA Carts	5
1.3	Uncategorised	5
2	Information Scrapings from blogs, forums etc.	6
2.1	Uncategorised Ordered by Time	6
3	Other People's Ideas	8
3.1	3D	8

Chapter 1

Distillation of Online info: May be Reliable or Not

1.1 3D

1.1.1 Basics

1. I found that vertex coordinates only had a range from -8 to 8. so try rescaling your track to fit within an 16x16x16 cube and use `glScalef` to rescaling it to the proper size before rendering.
2. `TEXTURE_PACK` macro parameters are swapped (16 uppers bits should be V, while it's U [should be in low bits])

Yes, U and V are swapped and they know it. I was one week trying to know why my textures appeared messed until I found that [Webez: 06-03-12].

1.1.2 Polygon Limits

1. **poly limits** It's important to note that you can only display 2048 *triangles*, each quad is worth 4/3 triangles, so for example if you display only quads you can only show 1536.
2. My understanding is that the limit of 6144 vertices (in other words, 1536 quads or 2048 tris) applies after culling. See the BoxTest demo.
3. That limit is there because the DS sorta renders the polygons like sprites, on the fly per scanline. That's the size of the polygon register/buffer/something. The GPU reads that every scanline, and renders the polygons to a 256x1 buffer, that is then sent to the LCD memory.

1.1.3 Display Capture

1. Actually, now that I think about it I'm pretty sure you CAN do real 3D additive blending with capture. Just set the capture EVA and EVB both to full, and render alpha polygons.

2. you can fiddle with the buffer all you want. Unlike other modes, you CAN capture to a buffer and display it in framebuffer mode at the same time, so you can do buffer-modifying effects losing only one VRAM bank. At least, as long as you can finish your modifying within the VBlank following the capture.
3. It's said to be possible to empty and load new polygons into the buffer mid-render. Maybe MKDS (and some other games) do this. I can see this working on MK, since the camera does have some restrictions, even when it appears it doesn't. As example, if the camera is always a bit high and looking a bit down, the objects far away into the track will be in different scanlines than the racers and the track itself.
4. . It's also said to be possible to make the LCD "hold" it's image and draw more polygons on top of it into two or more passes. You wouldn't be able to do any Z-buffer tests against the previous content, though, since the LCD memory only stores colors, not Z-values. So you'd need to "split" your scene into "far" and "near" slices and render the "far" first and the "near" later. [DON'T KNOW POSTER's REP]

1.1.4 Antialiasing and Outlining

1. **Edge Antialiasing** Edge antialiasing couldn't be easier:

```
glEnable(GL_ANTI_ALIAS);
```

would appear to be all you need!

2. **Outlining** Outlining is a neat feature - it'll draw a single-pixel outline around a group of polygons with the same ID - for example giving a perfect outline around a cel-shaded object.

IDs range from 0-63, giving 8 groups of 8 IDs. (group 0 is IDs 0-7, group 1 is IDs 8-15, etc) Each group can have its own outline colour. de:

```
glEnable(GL_OUTLINE);
glSetOutlineColor(nIDGroup, RGB15(r, g, b));
```

```
glPolyFmt( [normal flags here] | POLY_ID(nId))
// draw polys ...
```

The outline will be drawn where a polygon edge meets a polygon with a different ID (even if in the same group).

3. Outlining and edge AA can be used together, but it doesn't look very good.
4. There's an annoying glitch with outlining - it'll always draw an outline at the edges of the screen where the polygon is clipped. Not sure if there's any way to prevent this - maybe you just have to cover it up with something

1.1.5 Characters and Skinning

1. **Hardware Skinning** *Does the DS have hardware support for (multiple bones + weights per vert)-type skinning?* I'm not sure, but probably no[ecurtz].
2. *Is the HL MDL format limited to one bone per vert ?* I'm not positive, but I think yes (for HL 1) [ecurtz]

3. ecurtz

Calculate the matrix for each bone and MatrixStore it in the matrix stack. Embed MatrixRestore commands into your display list when the active bone changes. Dump the display list onto hardware.

Using MatrixStore/MatrixRestore instead of MatrixPop/MatrixPush? i.e. Put all the matrices onto the stack before glBegin and change the active one. I THINK that should work.

1.1.6 Display Lists

Question ishraam

- Looking at the FIFO section in libnds videoGL.h, I can't see any macros specifically for texture swapping, or matrix operation. Precisely : can I simply use GFX_MATRIX macros in FIFO stuff ? Generally : what can / can't be done with NDS display lists ? Are we limited to those few operations the FIFO_xxxx are showing us ?

Answer ecurtz

You are correct that you can only do operations that can be built from the registers in the core control section.

However that definitely DOES include MatrixRestore, since I've used that, so I bet you can do all of the other Matrix operations as well. **Just define the new values with REG2ID() and try them out.**

1.1.7 Misc

1. Bit 12 of the poly format register affects z-far clipping. If its not set then polygons intersecting the far plane get culled instead of clipped.

I haven't yet seen how this affects the box test. But maybe the same thing is true, if your box is large and intersects the far plane, it would get culled instead of returning that its intersecting.

2. **Polygon Transparency** Try using 0xFFFF (pink) color and set poly_alpha(31), and that would make those parts transparent. Atleast it worked for me...
3. * 0.7If I remember right that 3d modes are 3d only... so i think no sprite no bg no printf Hardly. The 3D layer just replaces BG0[ecurtz].

1.2 Linking, Loading, and Memory

1. Tightly Coupled Memory Sections

Well, on ARM9, functions go into main RAM by default, so you do have to specify. The section you're after is .itcm, which is 32KB and even faster than IWRAM because it's internal to the ARM9 core, which runs at twice the main system frequency. There's also .dcm for data, which is 16KB.

Anyone done some work on keeping the itcm & dcm contents out of the caches?

Devkitarm does this by default[pepsiman].

2. There isn't any ROM, the DS doesn't run code directly from it's game cards.

3. The nds file contains both arm7 and arm9 binaries. The start addresses and lengths of each are also specified in the header. The loaders copy these binaries to the specified places.
4. **0x02400000-0x02800000** is an uncached mirror of **0x02000000-0x02400000**.
What this means is it will contain the exact same data (it is the same memory), but the CPU will always read and write it, instead of using its own internal copy.

1.2.1 GBA Carts

- (a) The RAM on the Supercard is in the GBA cart address space (8000000-9FFFFFFF). But !! beware. Just like vram, you can't do 8-bit writes to it
- (b) The RAM on the Supercard is made writable by writing 0xa55a to 009ffffe twice, then writing 00005 to 009ffffe twice. Write 00001 instead of 00005 to make it read only again.

1.3 Uncategorized

1. **Texture Formats:** Also the smallest texture format is 2 bit paletted. The screen colour depth is 15 bits, 5 for each channel. 5-bit alpha is only available for 8-colour textures (bringing the size up to 1 byte per pixel) and 3-bit alpha is available for 32-colour textures (once again 1 byte per pixel). Only 1-bit alpha is available for all other texture formats as far as I can recall. Also, texture filtering is rumoured to be available, but noone has managed to enable it, and I don't know of any commercial games that use it. For texture palette storage you have two options, you can store them in a 16k bank or a 64k bank. each colour takes 2 bytes, so in the 64k bank you would have room for 128 256-colour palettes, 2048 16-colour palettes, etc., more than enough room.
2. **Texture Mem:** Depending on which VRAM blocks A/B/C/D you use for tex memory, that's your limit. each block is 128KB, if you use all 4, then you have 512KB
3. Official developers aren't allowed to change the code running on the arm7, and the official bin takes care of things like playing audio, send/recv wifi packets, polling the touch and x/y buttons, etc [joat]

Chapter 2

Information Scrapings from blogs, forums etc.

2.1 Uncategorized Ordered by Time

1. ecurtz Joined: 11 Jan 2003 Posts: 142 Location: Seattle Posted: Tue Dec 20, 2005 7:40 pm

Doing 30 fps in two passes is certainly possible, drop me a line when you get back to DS hacking if you want some help getting it to work. It probably sucks up half your VRAM for buffers though. I can't figure out how to do it with only one bank. As far as I know you're correct about needing to sort yourself to some extent - I haven't heard of any way to store the Z-buffer between frames.

2. DekuTree64 Joined: 01 Jan 2003 Posts: 737 Location: Washington, USA Posted: Wed Dec 21, 2005 8:31 am

Actually if the original only ran at 15fps, you could even have 8000 tris :) With the display capture trick, you can do as many passes into the buffer as you want. You could also swap out some of your textures during VBlank between each pass. Not sure exactly how much time there is, but you could probably get at least 16-32KB through.

The only way I know of to get a sort of additive blending look with the hardware is by turning the emission color up nice and bright with regular alpha blending.

You'd probably have enough spare time for some software rendering if you're doing multiple passes though. You could use VRAM E (64KB) as an 8-bit bitmap for it.

3. dovoto Posted: Thu Nov 03, 2005 7:23 pm

I do not believe it is appropriate to represent the map itself as a display list but only the objects inside it. Mainly because you normally do not want to draw your entire map each frame.

I believe you can bind to textures within a display list so there is no need to separate objects based on texture (i guess i have not verified this yet but i see no reason why not). This means you can reference several textures from the same display list.

It is also normally desirable to separate your textures into the smallest size possible. There are a couple of reasons. 1) You can usually get away with a lower color depth with a set

of smaller textures vs one big one 2) One big one often will contain unused space 3) Small textures are much easier to load on the fly.

4. ficedula

That is good news ;) I assume that the way to render like this is to capture the frame into a buffer (96KB for a full screen 16 bit buffer), render into the buffer *again* next frame to add more details, then display that buffer on-screen in some sort of framebuffer mode? Then while it's displaying, render into another buffer to prepare the next (visible) frame.

I can see why that would require two banks of VRAM. Is there any possible use to be made of the top 32KB of each bank that's unused by the display, or can they not be effectively used while the rendering is going on?

(In order for this to work I guess you would have to have some way of 'redirecting' the rasterisation from the 3d engine to the buffer entirely, rather than rendering it to both the display AND a buffer - which was how I imagine the display-on-both-screens method working?)

Chapter 3

Other People's Ideas

3.1 3D

1. - **Asset storage** - Longer term, I have in mind coding the engine to generically load assets from whatever storage device you have available. CF cards if possible (I've got a GBAMP for development), flash carts otherwise (smaller space then, of course), or even streaming via wireless over WLAN or internet. The engine should cope with whatever you have. Theoretically. First we'll need the wifi libraries to be finished, of course, but that's looking promising. [**ficedula**]
2. -**2D backgrounds** - The 2d tiled background shots on my website are using additive blending for all the special effects, but that's a purely 2D effect so far. The DS (and the GBA, for that matter) supported additive blending between background layers (and sprites) as standard. I haven't yet run the tiled background in combination with 3D models, although obviously it should be possible. [**ficedula**]