John Carmack Archive - .plan (2007)

http://www.team5150.com/~andrew/carmack

November 9, 2007

Contents

Chapter 1

November

1.1 Technology and Games (Nov 02, 2007)

Source: http://blogs.ign.com/OrcsandElves/2007/11/02/70591/

Most of my time lately is spent working on Rage, id Software's Id Tech 5 based game that runs on PCs, Macs, 360s, and PS3s. A modern high-end game is a really breathtaking consumer of computation and storage resources for anyone that has been around computers for any length of time. Our target platforms have at least 512 mb of ram, almost 20 gb of media storage, and many tens of gflops of computation, but the development environment involves an even more massive deployment, with a terabyte of raw data being generated before the final culling and compression is done. It is easy to be a little nonchalant about the continuous pace of improvement with computing, but I still take the time to feel a sense of awe about it all.

I started programming on a Commodore VIC-20 with 4k of ram and a tape drive, and I remember writing absurdly long lines of basic code to save the couple bytes that a new line number would consume. As the years went by, and my projects moved from the Apple II to the PC and the early consoles, I continued to gain new insights and perspective on different problems, and I often thought that it would be fun to go back to one of the early small systems. Now that I ”knew what I was doing”, I could do a lot more within the tight constraints than I was ever able to before. I actually carried a bunch of old systems around with me from house to house for many years before I reached the conclusion that I never was going to spend any real time on them again, and finally tossed them out.

As technology continued to rapidly advance, I saw a lot of good programmers sort of peel off from high end game development, and migrate to other platforms where their existing skill sets were still exactly what was needed. There was a large contingent of hard core assembly language programmers that never wanted to ”get soft” with the move to C for more complex game development, and many of them moved from PCs to platforms like the super Nintendo, and eventually into embedded systems or device firmware. There was another contingent that never wanted to move to windows, and so on.

There is an appeal to working with tighter constraints. The more limited the platform, the closer you can feel you are getting to an ”optimal” solution. On a modern big system, there are dozens of ways to accomplish any given task, and it just isn't possible to evaluate all the tradeoffs between the implementations of hundreds of different tasks. On a little system, you have to constrain your design to have a much smaller total number of tasks, and the available options are a lot more reduced. Seeing if something is The Right Thing is a lot easier.

I probably had my personal ”moment of truth” around the beginning of Doom 3's development, when it became clear that it is no longer possible to deeply understand every single part of a modern application. There is just too much. Nevertheless, I found that I could still enjoy my work when confined to a subset of the entire project, and I have thus remained committed to the high end. However, the appeal of smaller systems still lingers.

A couple years ago, almost on a whim, I got involved in developing games for mobile phones. The primary instigator was when I ran a bunch of games on a new phone and was left frankly appalled at how poor they were. I was thinking to myself ”I wrote better games than this each month before we founded Id”. I downloaded the SDK for the phone, and started tinkering around a bit. A proof of concept demo and a plan for a game play style specifically tailored for cell phones followed, and we wound up with DoomRPG, and later Orcs & Elves which turned out to be big hits.

In an ideal world, where I could either stop time or clone myself, I would act as a lead programmer for some smaller projects. In the real world, I can't justify spending much time away from the high-end work, so the low-end work gets done in a few short bursts of engine creation and foundation laying, which is then handed over to the Fountainhead team to actually build a great game. After that, Anna mostly uses me as a threat – if her programmers tell her that something she really wants in a game can't be done, she threatens to call me up and have me tell them how straightforward the problem really is, which usually fuels them to figure out how to do it on their own.

Mobile development was fun in a resource constrained design sort of way, but the programming didn't have the "tight" feel that early gaming had, due to the huge variability in the platforms. When I was messing around on my own phone, I spent some time doing java bytecode disassembly and timing, but it was fairly pointless in light of the two hundred or so different phones the game would wind up running on.

Enter the Nintendo DS.

We had initially looked at possibly moving one of the cell phone titles over to the GBA, but it didn't look like the market would support it, and technically it would have only turned out somewhere in between the low end and high end cell phone versions. With the success of DS, and a suspicion that the players might be a little more open to new third party titles, EA decided that they would support developing a really enhanced version of Orcs&Elves for the DS. This is going a bit out on a limb – most successful Game Boy / DS titles have been either first-party Nintendo titles, or titles with a strong movie / toy / history tie in. While Orcs & Elves is doing well on mobile, it is still very far from a recognized brand.

The resource limits on the DS make it almost perfect for a small team development. The hardware is fun to work with, you can directly poke at all the registers, the tool chain works well, and the built in limitations of cartridge memory keeps the design decisions fairly straightforward. Going one step farther up to the PSP with a UMD brings you into the realm of large media sizes that can rapidly consume multi-million dollar development budgets.

Once the decision was made to go for it, it was my job to figure out what we could reasonably hope to accomplish on the platform, andbring up a first cut at the 3D rendering engine.

Up next: all the technical details

1.2 DS Technology (Nov 08, 2007)

Source: http://blogs.ign.com/OrcsandElves/2007/11/08/71156/

The actual implementation decisions for Orcs&Elves DS were driven by the hardware, the development timeline, and the budget. With a five person team, we had six months to bring it all together on a new platform. I wrote the code for the hardware accelerated 3D renderer and for the remainder of the project, I was technical advisor.

The basic compute power of a 32 bit, 66 mhz arm processor and four megs of ram is a pleasant size to work with, basically about what we had on the PC back when the original Doom was written. You are intrinsically limited to a design that is compact enough that you can wrap your head around every aspect of it at once, but you don't wind up mired in crazy size decisions like trading a couple functions of code for an extra graphical icon.

Going back to fixed point math is always a chore, and while the DS has hardware acceleration for fixed point math, it doesn't automatically integrate with C/C++ code. The compiler / linker / debugger tool chain worked just fine, and I never felt that I was fighting the development environment like it used to be with the really early consoles. The DS SDK takes my preferred approach of both documenting the hardware fully and providing a helper library with full source code that you can pull apart as necessary.

The baseline spec we started with was a 16 meg cart for the game. I was sure we could get all the features we were discussing to fit in there, but in hindsight, we really should have pushed for a 32 meg cart, because it would have allowed us to add a lot more high quality 2D art to the game, and include some pre-rendered cinematics to help set the mood and tell the story. Anna had pushed for this from the beginning, but I was worried that we wouldn't have enough time to create the additional media, and I didn't want to eat the extra manufacturing costs on a speculative game release of an unknown IP.

At first glance, a 16 meg cart is over eight times as large as our high end cell phone distribution, but that turns out to be misleading. Everything is highly compressed on the mobile versions, but because of the need to directly DMA many assets in a usable form on the DS, and sometimes due to the tighter ram limits, a lot of the media takes up more space on the DS. The game is still a lot bigger, but not 8x.

Interfacing with the DS 3D graphics processor was my major contribution to the project. The DS is an unusual graphics machine, with no direct analog before it, but the individual pieces were close enough to things I had experience with that I was able to be effective pretty quickly. While there are a few things I wish would have been done differently, I still found it a lot of fun to work with.

The geometry engine is nice and clean, implementing a good subset of the basic OpenGL pipe in fixed point. It is also quite fast relative to the rest of the system. I had originally laid out the code to double buffer all the command traffic so that the geometry engine could completely overlap with the CPU, but it turned out that we would run into the polygon limits on the rasterizer before the geometry engine worked up much of a sweat, so I just saved memory and let the geometry engine run with a single command buffer. The game logic tended to take more time to process than the geometry engine, so it was almost never a gating factor. A title that heavily used vertex lighting and matrix stack operations might be able to load up the geometry pipeline a bit, but with just texture and color at the verts, it has margin.

Unlike classic OpenGL, you can change the matrix in the middle of specifying a primitive, allowing single bone skinned model rendering to be performed with a conventional vertex pipeline. This was a novel approach that I hadn't seen anywhere else. It was a little odd to see an explicitly separate model and projection matrix in hardware, since they are usually pre-multiplied into a single MVP matrix, but it allows the slow main cpu to avoid having to do some matrix math.

The question of 3D models versus sprites was one of the most significant decisions for Orcs&Elves. There are several situations in the game where you may be facing six or eight monsters, and I was concerned about how low poly the enemies would have to be to avoid problems with the rasterizer, especially when you consider that monsters can chase you into scenes that may tax the rasterizer all by themselves. Coupled with the fact that developing a new skeletal animation system, building all new models, and animating them would almost certainly have busted our development timeline, in the given circumstances, we decided sprites would do a better job.

The rasterization side of things on the DS is... quirky. Almost all 3D rendering systems today use a frame buffer and a depth buffer that is stored in dedicated graphics memory and incrementally updated by the graphics processor. The DS essentially renders as it is being displayed, with only a small set of lines acting as a buffer. This saves memory, and can give speed and power savings, but it has some significant tradeoffs.

Failure modes are bad – If you overload the polygon list, the remaining polygons just don't show up. This can be mediated by drawing the nearby and important things first, so if anything disappears it is hopefully in the distance where it isn't very noticeable. If you overload the fill rate, horizontal bars appear across the screen. In a perspective view, the greatest complexity and density tend to be in the middle of the screen, so if your scene gets too bad, a jittering color bar tends to appear in the center.

In a conventional rendering engine, overloading complexity just tends to make the game go slower. On the DS, overloading looks broken. This means that you need to leave yourself a lot more margin, and therefore use the hardware less aggressively. The plus side is that since the hardware is re-drawing the screen at 60hz no matter what you do, you are strongly encouraged to make sure the rest of your game also stays at 60hz.

The lack of texture filtering on the DS is the most obvious difference with other current platforms, and it does strongly encourages a cartoony art style for games. The art style for O&E isn't really ideal, and perhaps we should have stylized things a bit more.

You don't have a lot of texture memory available on the DS, less even than an original Playstation. In O&E, the environment graphics (walls and floors) are allocated statically at map load time, and must be balanced by the level artist. The character sprites are dynamically managed, with high resolution artwork for the monster directly in front of the player being swapped in every frame. The field of view and sprite positioning are carefully managed, along with a "pushback factor", to ensure that the sprites are at 1:1 scale when they are one tile away, and 2:1 scale when they are right in front of the player. Without bilinear filtering on the texturing, non-integral scales wind up looking very ugly. This is one of the advantages of the tile based play – if it was completely free form, the monsters would look a lot uglier. There isn't any intermediate step to be taken to improve the monster rendering without using a full 4x the memory to render an adjacent monster at a 1:1 scale. Even if we had more memory, I probably would have spent it on more animations instead of higher resolution.

The most disappointing mistake in the DS hardware is the lack of even the basic original blending modes from OpenGL. This was a mistake that was common in the very first generation of PC 3D accelerators, where so many companies just assumed that "blending" meant "alpha blending", and they didn't include support for add, modulate, and the other standard blending modes. No relevant company had made that mistake in a decade, and Nintendo's consoles have always done it right since the U64, so it was a surprise to see it in the DS. Additive blending in particular is crucial to most "3D flash" sorts of rendering, and various non-additive modulation modes are used for light mapping and other core rendering features.

I only got to spend four days actually writing all the 3D code for Orcs&Elves, so there are lots of potential directions that I am interested in exploring in the future. We plan to have two more DS projects in development next year, which I hope will let me try out a skeletal animation system, experiment with the networking hardware, and implement a more flexible high level culling algorithm than what I used in O&E.

John Carmack