Memories of a Vector World

Owen R. Rubin
Pacific Bell Video Services

While vector graphics -- non-memory-mapped displays consisting of lines drawn directly to the visible screen -- were once at the heart of the coin-op videogame industry in the early 1980s, it is quite possible that most younger game players, and most of today's game designers, have probably never even seen a vector display. I first saw one connected to a PDP-11 computer while attending University of California Berkeley in the early '70s. The first time I played with one, I knew I had to design games for a living someday. And I did.

Upon graduating, I joined Atari as one of their first software engineers. I wasted no time getting started, but -- I am sorry to say -- did not get started in the right direction. I wrote my first (non-vector) game, *Cannon Ball*, while sitting in my small office at a Model 33 teletype connected to a Motorola MicBug 6800 processor, both of which were connected to simple videogame hardware. I hand-assembled the entire program -- it was only 2K, but still took several months -- including self-test, saving the code on punched paper tape.

I recall my first review of the game. My boss said "Can we see the listings?" "Listings, what listings?" I replied. He was astonished to learn that I had not used the PDP-11 systems and the two computer operators which sat in the other building who would type in all your code, assemble and link the program and return you a listing and a paper tape. And so started my videogame career.

I worked on most of the typical coin-op game hardware through the next few years until we saw *Space War* by Cinematronics, which used vector graphics. Vector graphics -- sometimes called XY graphics -- were great for videogames because they provided very high resolution, razor sharp images in a day when 200x180 pixels was considered "hi-res." All the game designers had to have it. Since Atari designed and built all its own game hardware, have it we did, thanks to some very clever engineering by Lyle Rains, Howie Delman and others at Atari.

Rich Moore received the first vector hardware and created *Lunar Lander*, which was based on a well-known mini-computer game. I recall seeing it on that vector system at Berkeley. The only real difference was that we had a nice cabinet with great controls. This was Atari's first vector game, released in August 1979. Ed Logg used the same hardware for *Asteroids*, which was released just three months later.

I wanted to create a game based on something I saw in the first *Alien* movie. I loved the planet landing effect at the beginning and thought "What a great game it would make to be able to fly down tunnels like that" -- or so I thought. Unfortunately, it wasn't much fun. Without hidden line removal on the XY system, the tunnels were almost impossible to see. Similar hardware was used for *Battlezone*, released November 1980, but that's another story.

The Hardware

The first vector system was rather simple, much like an Etch-a-Sketch. You controlled an electron beam which you could turn on, off or dim; draw lines (vectors); move to the center of the screen; scale vectors; or jump to a subroutine of vectors to draw and return. The viewable screen area was about 1024x768 pixels, though those pixels only "existed" on the display tube (i.e. there was no video memory as such, just a list of vector instructions). This viewable area was part of a 4Kx4K world to make clipping and scrolling much easier.

Since a vector generator was essentially an analog system, it was not very accurate over time. If you drew many vectors, the beam would "drift" and not always land where you expected it should. You constantly had to "zero" -- or center -- the beam, the only known reference point. Unfortunately, this took valuable drawing time, so a balance had to be struck between centering the beam for drawing accuracy and the total amount you wanted to draw. If you turn up the intensity on a vector game, you will see this great collection of lines running from the center of the screen to all the moving objects. It is also why you often find the center of the screens burned out. If you didn't adjust the intensity just right, the beam, constantly moving to the middle, would burn a little hole in the phosphor right there.

Unlike memory mapped games, you never had to erase the old object from the screen before drawing the new one. Once a vector was drawn, it started to fade: the screen erased itself. To move an object, you drew it somewhere else. The old object would simply vanish if not drawn again, and appear to move to the new location. Anything you wanted static on the screen (like scores, for example), had to be drawn every frame.

The obvious downside was that to have something on the screen, you *had* to draw it, and that took time. And the more you wanted to draw, the longer it took to draw everything. Unlike raster displays, there was no constant "frame time" or refresh rate; you had to update the display often enough so it looked stable to the viewer. A "frame" was based on the time it took to draw everything. If you drew a small amount, you could then wait until a specified amount of time had passed since you started drawing, giving you a constant refresh rate. But if you drew too much then you would take too much time and the screen would start to flicker. This was a difficult balance.

Color, when it was eventually added, was very simple from the software point of view (I am sure the hardware engineers had a different perspective on that!). You filled a 16x8 color RAM with the colors you wanted (three bits each). Then you set a register specifying which of the 16 colors you wanted to use, and all subsequent beams would be drawn in that color. You could change the color of a vector without having to change the drawing commands by simply changing the color RAM location. Or you could set the drawing color register and jump to a list of vectors to draw an entire object in various colors.

We did all our programming on DEC computers -- originally PDP-11's moving to VAX systems later -- and Dave Shepperd, Atari's king of systems, had rewritten the Macro 11 language used for programming on these machines to allow us to use it to write code for the 6502 and 6809 and other microprocessors. Macro 11, as you would expect, had a very powerful macro language built in, and we created a macro language for drawing vectors which these systems also assembled for us, making it much easier to define vectors, pictures and subroutines.

The actual graphics themselves were created using a set of macros for all the drawing commands needed. When assembled, these macros generated lists of vectors needed to draw an object. For example, "VCTR 10,20" would draw 10 pixels in the x direction and 20 in the y. Once an object was created this way, it could be used as a subroutine . To draw that object, you would center the beam, then position the beam to the object's location, set the color and then call the subroutine. Vector positions were all relative, so this worked great for moving an object.

3D Gaming, Circa 1980

Since we had simple hardware and small processors, 3D graphics were a challenge.

For several games, Atari used a thing called a "math box" to do some 3D calculations. This was originally created by hardware engineer Jed Margolin for a game I was working on with Ed Logg called *Malibu Grand Prix*. *Malibu* was a first-person driving simulator with a point of view from inside the car. Unlike other driving games of the time, you could drive anywhere and were not limited to staying on a fixed track. Unfortunately, because of the limited capacity of the hardware and the limited amount of displayed information, it was hard to tell where you were actually driving much of the time. This game was quickly junked.

The math box was later modified for use in *Battlezone* and *Red Baron* (an airplane combat game). But because the cost of this box was quite high, it wasn't used in many other games. Ed Rotberg, the programmer for *Battlezone*, recalls using the math box:

> *"The math box consisted of a set of 2901 bit-slice processors. In Battlezone it was used strictly to do the '3D' calculations for the display. In point of fact, we only did 2D rotations and hacked the very little motion that was done in the Y axis. The math box was passed a 2x2 rotation matrix and -- I believe -- a list of points to rotate. If I remember correctly translation was handled in the 6502, although this may have been done in the math box as well. It's been a long time."*

Mike Albaugh,[1] one of Atari's first game programmers, and still with Atari Games today, wrote the microcode for the math box. He explains its functions as follows:

> *"The math box appeared to the game programmer as a memory-mapped I/O device. 'Storing' to one of 32 locations would jam a micro-code address into the 'PC,' and result in some routine being executed.*
>
> *I am surprised, looking at it again after many years, that the internal coordinate system seems to have had X into the screen, with Y (H) and Z (V) on the surface of the screen. I can only assume that this was a holdover from Jed Margolin's original 'Malibu Mathbox,' for which this one was supposed to be a 'stop-gap replacement' :-)*
>
> *Anyway...the routines provided were pre-sub (X' = (X-E)*A+(Y-F)*B) and post-add (X' = X*A+Y*B+E) multiplies, with the latter 'falling through' (after computing X' and Y') to the Y'/X' 'perspective divide' (remember the coordinate system) Whew! Believe me, it was harder for me to extract that from the code and comments than it was for you to read it. The 'commands' were:*
>
> *"Set Y-High, start Multiply"*
>
> *"Set Y-High, start Multiply, post-add"*
>
> *"Start Y' multiply" (after reading X' from pre-sub multiply)*
>
> *"Divide Y'/X'"*
>
> *"Divide Z/X'"*
>
> *"Clip"*
>
> *"Distance" (approximate sqrt((X1-X2)^2+(Y1-Y2)^2)*
>
> *This was done with 256x24 bits of microcode and four four-bit 2901 bit-slice datapath elements."*

As you can see, videogame design was anything but normal. In fact, this example is typical of what happened at Atari. Because total cost was always a consideration, we were asked to get the most out of minimal hardware. Unlike the graphics systems in large computers -- like flight simulators -- we created smaller, but similar effects for considerably less money by cheating our way through. Since, unlike a real flight simulator, we knew exactly what the game needed to do, we could take shortcuts on the display and hardware by limiting the hardware to doing only what would be needed. In *Major Havoc* for example, 3D objects were precalculated because they would only be seen in certain positions. Thus, 3D hardware was not really needed. When all you have is a 6502, you forget about doing true 3D graphics.

A great feature of the vector hardware that helped with these cheats was the ability to apply a scale factor to the drawing. An object could appear to be flying off into the distance, but still be actually using the same vector subroutine. In *Major Havoc*, for example, there is a small map of the entire maze at the top of the screen. It actually uses the same drawing routines as the big maze, but is scaled down. And the space scenes appear to have a 3D perspective, yet they are drawn on a standard, flat XY grid. By changing the scale factor on the fly, the graphics seem to disappear into a vanishing point by getting smaller towards the top of the screen, thus a 3D effect. We added to this effect by adding prerendered 3D objects, also being scaled and animated. But collision and calculations were very easy, because to the program everything was on an XY grid.

Space Duel and Major Havoc

My return to vector game design was a year after *Battlezone* to work on the first color vector game, *Space Duel*. It was to be a two player knock-off of *Asteroids*, but in color. Unfortunately, I wasn't the only one at Atari that thought *Asteroids* was great enough to do a follow-on game. The color game was put on hold because *Asteroids Deluxe* (May 1980), a "newer, better" version of the *Asteroids* game on the original vector system, was to be released. The hardware for *Space Duel* and some of my code were transferred to Dave Theurer's *Tempest* (October 1981), and that, as they say, is history. But design of *Space Duel* slowly continued and was later released as yet another newer and better version of *Asteroids* in February 1982. A new gameplay idea was added in *Space Duel*: cooperative play. Not only did you have individual scores, but your extra lives were based on a combined score in some games, so it was competitive and cooperative at the same time. The game did quite well and was a lot of fun to create.

After *Space Duel*, I spent almost 18 months working on *Major Havoc*. In the meantime, the color hardware also moved to several other projects. *Gravitar* was released in August 1982, and was listed by *Replay* magazine as one of the worst games ever. They used to rate new games which were considered really bad with a rating of "move over *Gravitar*." (Actually, I liked the game and didn't think it deserved this bashing.) A version of the hardware was also licensed to a company in southern California. They removed the 6502 processor and added a Motorola 68000 to the board. The result was *Quantum* (November 1982). The original hardware also went to *Black Widow* (February 1983) and *Star Wars* (May 1983).

For *Major Havoc*, Doug Snyder beefed up the hardware, added some new special effects -- like a vector that sparkled -- and another processor, and we created a game that had a very strange cult following. To this day, I still receive regular emails and phone calls about the game. There is one person who is even creating more levels for the coin-op game by disassembling the code by hand.

Most games used one 6502, but *Major Havoc* had a pair. It was originally designed to have three processors because we thought we would do as much real-time 3D as possible. Even with "this much processor power," as my boss once said, there was no way to calculate objects on the fly, so the third processor was left off. And hidden line removal for 3D objects was impossible. So to do a rotating 3D object with hidden lines, I drew the object by hand, then removed all the hidden vectors by hand. The resultant picture was then translated, by hand, back into a new set of vectors that would draw that object.

[1] Editor's note: Thank you to Mike for his help in obtaining cover imagery from the original *Battlezone*.

Animation was also done by hand. Each step in the 3D animation was drawn and edited to create the desired effect. This was quite time consuming, but there were few alternatives. We later wrote programs on the same PDP-11 we used to assemble our programs to do some of this 3D and hidden line work for us at night. But by today's standards, it was quite slow. Calculations for the fighter spaceship in *Major Havoc* took all night, and we are really only talking about an object with 200 to 300 vectors and 50 different positions to be drawn. In fact, it took so long that a later change in the design of the ship was not completely rerendered, so if you watch closely, the ship actually changes design as it flies onto the screen.

The Close of the Vector Era

Major Havoc was one of the last of the vector videogames. The *Empire Strikes Back* (released over a year later in March 1985) was the final game, and had it not been a *Star Wars* product, would probably never have sold. Had *Major Havoc* been done in raster, it might have sold four to five times more. And I believe Atari killed vector games themselves. We discovered a problem a bit too late which caused the displays to blow out. It seemed, to correctly draw vectors with proper clipping, you needed to be able to draw a distance off the actual screen. This allowed scrolling and lines to disappear gracefully without the need for expensive clipping hardware. Unfortunately, if you drew too far off the screen, you could blow out the deflection amps in the monitor. Effectively, you were trying to draw on the back of the cabinet, and the hardware would let you try! We thought we had some hardware checks to prevent this, but they didn't always work. *Major Havoc* added software to check the vectors before drawing, and reduced the problem, but by that time, the game operators did not trust vector games at all. Earlier games were always blowing out the screen. *Space Duel*, *Space War* and other vector games were just too big of a service problem, and operators didn't want the headaches of a new vector game when there were so many other games available in the industry.

It is quite strange to no longer see XY technology used in displays these days. I feel it added a detail and crispness that you still do not get in raster displays, even with some of the best computer hardware. But then again, design of games seems to have changed as well. I feel, in the days of "limited" hardware, designers had to be smarter with gameplay. When I look at many of today's games, I see many of the same old games being recreated over and over again, but this time with better graphics, hardware and special effects. While there are a few exceptions to this rule, and there have been some wonderful games recently, they are just that, exceptions. I am glad to have had the chance to program games when I did. The challenge of creating good gameplay along with the challenge of pushing limited hardware to the breaking point was a simply a lot of fun.

Owen R. Rubin designs truly interactive video and digital TV networks for Pacific Bell Video Systems. He joined Atari, Inc., in 1976, and Bally in 1984, creating coin-op games for almost 14 years. He also spent seven years working on the Macintosh hardware and operating system at Apple.

Owen R. Rubin
Director, Technology & Media
Pacific Bell Video Services
2410 Camino Ramon, Suite 100
San Ramon, CA 94583
Email: **orrubin@pbvs.com**