NEW MEDIA AS MATERIAL CONSTRAINT
An Introduction to Platform Studies

Ian Bogost, Nick Montfort

We introduce platform studies, a family of approaches to digital media. In platform studies, close consideration is given to the detailed technical workings of computing systems. This allows the connections between platform technologies and creative production to be investigated. Two short studies of the Atari VCS (2600) and the Nintendo Wii show how close consideration of this sort can inform our understanding of the history, present, and future of new media.

Platforms have been around for decades, right under our video games and digital art. Those studying new media are starting to explore the low level of code to learn more about how computers are used in culture, but there have been few attempts to go even deeper, to the metal — to look at the base hardware and software systems that provide the foundation for computational expression. Platform studies is such an attempt, investigating the relationships between the hardware and software design of standardized computing systems — platforms — and the creative works produced on those platforms.

INTRODUCING PLATFORMS

The hardware and software framework that supports other programs is referred to in computing as a platform. A platform in its purest form is an abstraction, simply a standard or specification. To be used by people and to take part in our culture directly, a platform must manifest itself materially. This can be done in the chips, casings, peripherals, and other components that make up the hardware of a physical computer system. A platform may also include an operating system. It is often useful to see a programming language or environment on top of an operating system as a platform, too. Whatever the programmer takes for granted when developing, and whatever, from another side, the user is required to have working in order to use particular software, is the platform. In general, platforms are layered -— from hardware through operating system and into other software layers —- and they relate to modular components, such as optional controllers and cards. Studies in computer science and engineering have addressed the question of how platforms are best developed. Studies in new media have addressed the cultural relevance of particular software that runs on platforms. But little work has been done on how the hardware and software of platforms influences, facilitates, or constrains particular forms of computational expression.

By choosing a platform, new media creators simplify development and delivery in many ways. Their work is supported and constrained by what this platform can do. Sometimes the influence is obvious: A monochrome platform can't display color, a video game console without a keyboard can't accept typed input. But there are many more subtle ways that platforms interact with creative production, due to the idioms of programming that a language supports or due to transistor-level decisions made in video and audio hardware. In addition to allowing certain developments and precluding others, platforms also encourage and discourage different sorts of expressive new media work with much more subtlety. In drawing raster graphics, the difference between setting up one scan line at a time, having video RAM with support for tiles and sprites, or having a native 3D model can end up being much more important than a few numbers representing resolution and color depth.

Particular platform studies may emphasize different technical or cultural aspects and draw on different critical and theoretical approaches, but to deal deeply with platforms and new media, these sorts of studies will all have to be technically rigorous. The detailed analysis of hardware and code can connect to the experience of developers who created software for a platform and users who interacted with and will interact with programs on that platform. Only the deep investigation of computing systems will reveal the interactions between these systems and creativity, design, expression, and culture.

New media studies, focusing on artifacts, games, and works of digital art and literature, have been undertaken on many different levels. The ones we describe next have been discussed in the context of a specific video game before, but here we briefly explain how they are relevant to digital media studies overall.[1] This provides some context for our focus on the lowest level, that of *platform*.

Reception/operation is the level that includes reception aesthetics, reader response theory, psychoanalysis, desensitization to violence studies, and empirical studies of interaction and play. Only interactive media are explicitly operated, but all sorts of media are received and understood, so insights from other fields can often be usefully adapted to digital media at this level.

Interface studies include the whole discipline of human-computer interface, comparative studies of user interface done by humanists and literary critics, and approaches from visual studies, film theory, and art history. Remediation concerns itself with interface, although reception and operation are concerns of that approach, too. This is not unusual. Many new media studies span multiple levels, but there is often a focus on one.

Form/function is the main concern of cybertext studies and of much of the work in game studies and ludology. Narratology, previously used to understand literature and cinema, is an approach that deals with form and function and which has been applied to new media as well. Because these approaches deal with the same level, it is at least meaningful to imagine a narratology/ludology debate — whether or not any such debate has occurred — while it makes much less sense to think about a psychoanalysis/ludology debate or a remediation/narratology debate.

Code is a level that has only recently been explored by those investigating new media. Code studies, software studies, and code aesthetics are not yet widespread, but a few interesting books and panels dealing with the code level signal an increasing interest in the way creative work is programmed and understood by programmers. The discipline of software engineering is a related field that concerns itself with the code level as well as with organizational and individual capabilities for software development.

Platform is the abstraction level beneath code, a level which has not yet been systematically studied. If code studies are new media's analogue to software engineering and computer programming, platform studies are the humanistic parallel of computing systems and computer architecture, connecting the fundamentals of new media work to the cultures in which they were produced and the cultures in which coding, forms, interfaces, and eventual use are layered upon them.

Our focus here in on this platform level. We hope that studies at this level will help to fill in our overall understanding of new media and will benefit the humanistic exploration of computing. We also want to emphasize that we see all of these levels, not just the top one, as being situated in culture, society, economy, and history. Because of this, we seek to describe how platforms have come about as well as how they influence further cultural production. This awareness of contexts informs our approach to platform studies, just as it has informed the best new media studies at other levels in the past.

Next, we discuss three examples from different eras of computing to explain the general relevance of platform studies in new media. Our examples include two video game consoles and one general-purpose personal computer system. They are the 1977 Atari Video Computer System (later called the Atari 2600), the 1991 Multimedia Personal Computer, and the 2006 Nintendo Wii.

1977: THE ATARI VCS (2600)

The Atari VCS (renamed the Atari 2600 in 1982) was the first successful cartridge-based video game console. We describe the elements of the Atari VCS: The 6507 processor, its Television Interface Adapter (TIA), its interchangeable ROM cartridges, and a variety of 8-pin controllers.[2] These can help why VCS games imitated some aspects of arcade games while they left other aspects aside. We conclude this section with a detailed discussion of how sprite graphics worked on the VCS and how this influenced the development of cartridges.

Pre-VCS Gaming

The Atari VCS appeared at a time when the vast majority of video games were played in bars, lounges, and arcades. Today, the arcade cabinet is a rare sight, but in their heyday coin-operated games generated more income than today's home console-dominated market.[3] At a time when coin-ops ruled the market, part of the appeal of the home console system was its promise to tap a new market of kids and families.

That year Atari, eyeing the home market for video games, also designed a home *Pong* and arranged for Sears to sell it exclusively — which they did, moving 150,000 units in 1975.[4], Atari's triumph was short-lived, however. In 1976 General Instrument released its $5 AY-3-8500, a "PONG-on-a-chip" that also contained simple shooting games. It, along with other cheap processors, allowed even companies without much electronics experience to bring *Pong*-like games to market. They did — there were 75 available by the end of 1996, "being produced in the millions for a few dollars apiece." Even if Atari had cornered the market for home *Pong*, having that system in a home wouldn't have done anything to lead to future sales. How many *Pong*s could one house have needed? Atari looked, instead, to model some features of the nascent personal computer market with a home console that used interchangeable cartridges, allowing the system to play many games. There would be an important difference from home computing, as Atari saw it, though: All of the cartridges for the system would be made by one company.

The tremendous success of *Pong* and the home *Pong* consoles suggested that Atari produce a machine capable of playing *Pong*-like games. The additional success of *Tank* by Kee Games (a pseudo-competitor that Atari CEO Bushnell created to give the sense of an industry) suggested similar design possibilities for what would become the VCS. *Tank* featured two player objects, each controllable by a separate human player, and projectiles that bounced off walls — a computational model almost identical to *Pong*, and one that would become the inspiration for *Combat*, the title that was included with the original VCS package. These simple, existing elements would be the basis for the console's capabilities.

On the other hand, previous attempts at home machines that used interchangeable cartridges, such as the Magnavox Odyssey and the Fairchild VES/Channel F, suggested the potential benefits and risks for such a system. Released in 1972, the Odyssey played twelve games, but required players to attach plastic overlays to the screen in lieu of a computer graphics background. The machine had no memory or processor, and the experience of playing the Odyssey was certainly that of a video game, but perhaps too simplified, even for the time, and reminiscent of board game play. Even though Magnavox sold $22 million worth of Odysseys by 1975, the product posted losses of $60 million due to distribution and marketing problems — many customers thought they needed a Magnavox TV to play it.[5] Fairchild's Video Entertainment System, released in 1976, was the first programmable, interchangeable cartridge system, with an onboard processor and RAM. (The system had a rapid name change when Atari's VCS was released, and is better-known as the Fairchild Channel F.) Even before Fairchild's system was market tested, Warner Communication purchased Atari in 1976, largely based on the commercial promise of an interchangeable, programmable home console. It was this acquisition that provided the capital Atari needed to bring the VCS to market.

The Design of the VCS

The engineers developing the VCS needed to take into account both of these goals — imitation of known successes and versatility — as they designed the circuitry for a special purpose microcomputer for video games. Material factors certainly influenced the design, most notably the high cost of hardware components. The Fairchild system used the complex Fairchild F8 CPU, a specialty processor created by future Intel founder Robert Noyce. In 1975, MOS Technology released a new processor, the 6502. The chip was the cheapest CPU on the market at the time by far, and it was also faster than competing chips like the Motorola 6800 and the Intel 8080. The 6502's low cost and high performance made it an immensely popular processor for more than a decade. The chip drove the Apple I and Apple II, the Commodore PET and Commodore 64, the Atari 400 and 800 home computers, and the Nintendo Entertainment System (NES).

Steve Mayer and Ron Milner were chiefs at Cyan Engineering, a consulting firm Atari had purchased in 1975. They selected a chip for the VCS project that was very similar to the 6502, but stripped-down and even less costly. The two used the 6507, which came in a cheaper package with only 13 address lines, used to designate which byte in memory will be read or written. This was reduced from the 6502's 16 address lines. So while the low-cost 6502 could address 2^16 = 64KB of memory, the even lower-cost 6507 was only capable of addressing 2^13 = 8KB. But the memory that was to be addressed was on cartridge ROMs, and, again in the interests of economy, the VCS was designed with a cartridge interface had one fewer line than the processor would support — limiting access to 4KB of cartridge ROM at once. Bill Gates may have thought that 640KB should be enough for anybody; Mayer and Milner figured that 4KB would do. The 6507 was available for less than $25; similarly capable Intel and Motorola chips went for $200. Using the chip enabled the VCS to have a very low initial retail price of $199 — just above the consle’s manufacturing cost. This was an unusual move by Atari, but the company was counting on profiting from cartridge sales.

While important, the 6507 CPU was only one component — the VCS still needed additional silicon for memory, input, graphics, and sound. For sound and graphics, the VCS was to use a custom chip, the Television Interface Adapter (TIA). Joe Decuir and Jay Miner designed the TIA, codenamed Stella — a name also used for the VCS itself. Of course, the two sought to simplify the hardware design as much as possible, reducing its complexity and cost.[6] Input from the two player controls and the console switches were managed by an interface called RIOT. The VCS also sported 128 bytes of RAM — not enough to store this ASCII-encoded paragraph. Contemporary home computers usually have more than a thousand times as much memory, but this wasn’t an unusual amount for a video game system of the late 1970s. The VCS, like other cartridge-based systems, ran programs without loading them into RAM. The VCS’s 128-byte memory was twice as large as the VES/Channel F’s standard RAM; the later NES has only 16 times as much RAM, 2KB.

Drawing the Screen

The bare-bones nature of the TIA makes seemingly basic tasks like drawing the game’s screen complex. An ordinary television picture of the late 1970s and early 1980s was displayed by a cathode ray tube (CRT). The CRT fires patterns of electrons at a phosphorescent screen, which glows to create the visible picture. The screen image is not drawn all at once, but in individual scan lines, each of which is created as the electron gun passes from side to side across the screen. After each line, the beam turns off and the gun resets its position at the start of the next line. It continues this process for as many scanlines as the TV image requires. Then it turns off again and resets its position at the start of the screen.

Modern computer systems offer a frame buffer, a space in memory to which the programmer can write graphics information for one entire screen draw. This facility was even provided by many systems of the late 1970s — including the Fairchild VES/Channel F that preceded the VCS. In a frame buffered graphics system, the computer’s video hardware automates the process of translating the information in memory for display on the screen, also managing graphical administrativa such as screen synchronization.

The VCS does not provide such services for graphics rendering. The machine isn’t even equipped with enough memory to store an entire screen’s worth of data in a frame buffer. The VCS offers 128 bytes of RAM total — not even enough to store one 8-bit color value for every *line* of the VCS’s 191-line visible display, let alone for multiple elements per line such as individual pixels, or, at a higher level, backgrounds, players, and missiles. Additionally, the interface between the processor and the television is not automated as it is in a frame buffered graphics system. Instead, the VCS programmer must draw the screen manually, synchronizing the 6507 processor instructions to the television’s electron gun via the TIA. The programmer has a small amount of time to change the TIA settings via its numerous addressable registers when the electron beam resets to draw a new line (this period is called Horizontal Blank), or a new screen (this period is called Vertical Blank). However, the programmer must also manually instruct the TIA to initiate or wait for the horizontal and vertical blanks themselves, which involves keeping track of how much time the instructions take to execute on a single line, between lines, and between frames. Programming the VCS, then, effectively means drawing every line of the television display individually, making decisions about how to change the display on a line-by-line basis rather than a screen-by-screen basis.

Sprites

Many of the common techniques on the VCS are rooted in the manual nature of graphics programming for the device. Despite its simplicity, combinations of hardware and software techniques have produced a wide variety of visual, audio, and gameplay effects in many hundreds of games created in the three decades since the console's release. Rather than try to describe all of these in cursory detail, we focus on one aspect of VCS games: sprites.

In computer graphics, a sprite is a two-dimensional image composited onto a two- or three-dimensional scene. The VCS was designed to support two sprites, each a single byte in size, set via two memory-mapped registers on the TIA (named GRP0 and GRP1, respectively). The influence of *Pong* and *Tank* can be seen clearly here. Such games feature two opponents, each controlled by a human player. The VCS provided a facility for a single-pixel *Pong*-like ball, single-pixel *Tank*-like missiles, and the player sprites that were common to both games.

Sprite Graphics

When the programmer stores a value in the GRP0 or GRP1 register, the TIA displays that 8-bit pattern on-screen. A VCS sprite is thus always 8-bits wide, although the TIA provided a few ways of modifying the appearance of sprites on screen.

```
Bit      7 6 5 4 3 2 1 0         Sprite:
Line 0   0 0 1 1 1 1 0 0           XXXX
Line 1   0 1 1 1 1 1 1 0          XXXXXX
Line 2   0 1 0 1 1 0 1 0          X XX X
Line 3   1 1 1 1 1 1 1 1         XXXXXXXX
Line 4   1 0 1 0 0 1 0 1         X X  X X
Line 5   1 0 0 1 1 0 0 1         X  XX  X
Line 6   0 1 0 1 1 0 1 0          X XX X
Line 7   0 1 0 1 1 0 1 0          X XX X
Line 8   0 1 0 0 0 0 1 0          X    X
```

Figure 1: VCS sprite pattern from *Space Invaders.*

Figure 1 shows the pattern for a sprite — a 2D image, but one that is drawn, like everything on the VCS, one line at a time. Each sprite register can only contain the one byte of data that it needs for a single line of on-screen graphics. To draw the sprite shown above, the programmer would have to load the byte of graphics for the alien invader that corresponds with the current line on the television display and store that value in the sprite graphics register during the horizontal blank, in between the drawing of two lines. To position a sprite vertically, the programmer would have to keep track of which lines of the display have sprites on them, and compare the current line to that value in memory before drawing.

Sprite Colors
The TIA also provides a register to set sprite colors, one for each sprite (named COLUP0 and COLUP1, respectively) In early VCS games like *Combat*, sprites colors were usually set once for the entire game. In later games, programmers stored a different color value in one or both sprite color registers along with a different bitmap value. Multicolor sprites, such as the player character in Activision's *Pitfall*, allow for more visually interesting graphics. The careful observer can note color banding in most of these sprite graphics, though, which is not seen in the true bitmapped graphics of later platforms like the NES. This style of "stripe-colored" sprites is a particular trademark of VCS games.

Sprite Variations
To allow for variations in sprite graphics, the TIA offers two Number-Size registers that enforce automatic modifications to the sprites when drawn on screen (named NUSIZ0 and NUSIZ1, respectively). In particular, the programmer can change the number of sprites drawn on a single line, as well as the size of the sprites. The size of missile graphics, which are always comprised of a square shape corresponding in color to its parent sprite, are also adjustable. Adjustments to the sprites are made by setting one or more of the lowest three bits on the Number-Size register register. Table 1 shows a summary of the size and number adjustments afforded by this register.

D2	D1	D0	Description
0	0	0	one copy
0	0	1	two copies - close
0	1	0	two copies - med
0	1	1	three copies - close
1	0	0	two copies - wide
1	0	1	double size player
1	1	0	3 copies medium
1	1	1	quad sized player

Table 1: Effects of setting the VCS Number-Size registers.

The Number-Size register offers an easy way to modify the appearance and behavior of player sprites. The most transparent use of this technique is in *Combat*, which uses the Number-Size settings as the basis for many of its 27 game variations. The bi-plane and jet plane variations that double, triple, or stretch one or both sprites use the Number-Size register to accomplish what would otherwise have had to be done through complex on-the-fly graphics processing or by storing additional sprites in precious ROM — *Combat* is a 2k cartridge. For example, variation 19 is "2 vs. 2 Bi-Plane," in which each player controls two planes which fly in formation. This variation does nothing more than setting NUSIZ0 and NUSIZ1 to the binary value %00000001, which corresponds with "two copies - close" in the Number-Size register table above. Variation 20 is "1 vs. 3 Bi-Plane," in which player one controls a large plane and player two controls three small ones in formation. This variation sets NUSIZ0 to %00100111 (quad sized player) and NUSIZ1 to %00000110 (three copies - close).

Variation 20 demonstrates the opportunities and limitations of the Number-Size registers for gameplay modification. Player 1 is at a disadvantage, since his plane is larger and therefore more vulnerable to fire. To counterbalance, this variation increases the size of the missile so that player 1 does not have to be as accurate: The third flipped bit in %00100111 increases the size of player one's missile to 4 TIA clock cycles, or roughly 4x the size of player 2's missiles. However, when player 2's sprites triple, the TIA automatically triples its missiles as well, making it even easier for player 2. A more appropriate orthogonal design approach for this variation might have been to speed up the larger player and/or his missile, thereby offsetting player 1's increased target footprint. However, to do so would have required changes in the game's logic, not just in the data settings that map variation to sprite appearance. The tradeoffs in such a decision are typical of VCS game programming.

Multiple Sprites

As we discussed above, the VCS shared the video game marketplace with coin-op arcade games, most of which were built on much more sophisticated technical infrastructures. The VCS exchanged graphical complexity and specificity of circuit design for multiple cartridge home play. But the massive popularity of arcade games like *Space Invaders* and *Pac-Man* suggested a special opportunity for the VCS: home versions of these popular coin-op games were bound to be hits.

Combat, with its tank variations based on *Tank*, showcases the hardware affordances of the VCS more clearly than almost any other game. For example, it uses two sprites, each of which fire a corresponding missile. But games like *Space Invaders* were not designed with the peculiarities of the VCS in mind. Sprites were different in many post-1977 arcade games. Notably, there were often more than two per screen! When faced with the rows of aliens in *Space Invaders* or the fleet of ghosts that chases *Pac-Man*, VCS programmers needed a way to draw more than two sprites, even though only two one-byte registers were available.

One method comes from an exploit in the way sprites are positioned on screen horizontally. A VCS programmer positions a sprite vertically on screen by comparing a counted television line number with a variable stored in RAM to see if a sprite needs to be drawn there. This technique is grounded in the nature of the CRT television: the horizontal blank offers a natural break in screen drawing during which a few cycles of processing can be accomplished. But no similar natural mapping exists for horizontal positions on screen. To allow the VCS programmer to position sprites horizontally, the TIA exposes a set of horizontal motion registers for each of the sprites, the missiles, and the ball (named HMP0, HMP1, HMM1, and HMBL, respectively). Any object that is not intended to move must have 0 set in its corresponding register. To move an object, the programmer writes an offset value from -8 to +7 to move that object by the corresponding number of TIA color clocks. The TIA also exposes another register called HMOVE to execute changes in horizontal motion. These registers were primarily intended to be set during a vertical blank — that is, between screen draws. For example, *Combat* repositions both player and missile horizontal positions each frame, then updates variables in RAM to insure that the objects are drawn on the appropriate lines, then updates the horizontal motion registers once at the start of the frame.

Larry Kaplan, one of the first developers to work on the Stella prototype, reasoned that sprite data could be reset more frequently than once per frame.[7] Because the VCS requires the programmer to control every line of the television screen, it was also be possible to change the sprite graphics values and their horizontal positions more than once per frame. He first used this technique in *Air Sea Battle*, one of the console's launch titles. In the game, multiple rows of enemies, one per row, pass back and forth across the screen. Each player controls a turret on the ground that can be aimed and fired to destroy the enemies in the air. The multiple targets are accomplished by resetting the sprite graphics multiple times down the screen. Finally, when it is time to draw the ground, the sprite graphics and horizontal positions are reset for the player turrets.

Another variation of the horizontal movement technique helped bring *Space Invaders* to the VCS. The trademark feature of the popular arcade game were its rows of slowly descending aliens – which the TIA, of course, didn't support in any obvious way. Kaplan's *Air-Sea Battle* technique allowed multiple sprites to appear down the screen, but *Space Invaders* required multiple sprites in a horizontal line. Rick Mauer, the programmer for the VCS port of *Space Invaders*, discovered that strobing HMOVE while a line was being drawn would reposition objects immediately, even if they had already been drawn earlier in that line.[8] The TIA, lacking memory of what it has already done, will begin drawing the data from its sprite graphics registers to the screen any time that HMOVE is reset. After one row of aliens was drawn using this technique, Mauer read and wrote new sprite graphics values from ROM to create a new row of aliens with a different appearance.

These two techniques, combined with the VCS's lack of a frame buffer and subsequent requirement that the programmer draw every scanline, allowed the VCS to overcome the apparent limitation of only supporting two sprites on screen. Rather than changing both sprites and their positions every frame, one or both could be changed every line. Together, these approaches extended the game design space on the VCS, making it capable of playing games very different from the *Pong* and *Tank* arcade titles that had been the hits of the mid-1970s. The importance of these exploits was not lost on Atari executives, either. Discussing this technique in 1983, after he had become Atari VP of Product Development, Kaplan commented, "Without that single strobe, H-move, the VCS would have died a quick death five years ago."[9]

Despite the cleverness of these techniques, both vertical positioning and horizontal strobing required sprites to move together in vertical unison. Some variations of *Air-Sea Battle* moved different enemy sprites at different rates of speed by writing new values to the horizontal motion registers, but the objects only moved horizontally, never along both horizontal and vertical axes. After the VCS port of *Space Invaders* enjoyed considerable success, partly rescuing Atari from the losses of 1977-78, the company became even more interested in arcade ports. One obvious target was *Pac-Man*, whose U.S. arcade success in 1980 made it ideal for home console adaptation. But the four *Pac-Man* monsters need move horizontally and vertically, and independent of one another, as had not been done before. Just as *Space Invaders* would have been unrecognizable without its characteristic rows of invaders, so *Pac-Man* would have been unrecognizable without its characteristic quadruplet of monsters.

To accomplish this task, Tod Frye relied on a technique called flicker. Each of the four ghosts was moved and drawn in sequence on alternating frames; Pac-Man himself is drawn every frame using the other sprite graphic register. The TIA synchronizes with an NTSC television 60 times per second, so the resulting display showed a solid Pac-Man, maze, and pellets, but ghosts that flickered on and off every quarter of a second. The phosphorescent glow of a CRT television takes a little while to fade, and the human retina retains a perceived image for a short time, so the visible effect of the flicker is slightly less pronounced than it really is. The fact that the monsters in Pac-Man were commonly referred to as "ghosts" apologized somewhat for the flicker, which suggests the dimness of an apparition. Nevertheless, the flicker technique was widely criticized by players. Later ports of games in the *Pac-Man* family, including the 1982 *Ms. Pac-Man* and the 1987 *Jr. Pac-Man*, used less visually intrusive techniques to draw the ghosts. While these last two examples have been arcade ports — providing strong, specific motivation to programmers who are charged with imitating an existing game's visual appearance and behavior — the development of original VCS games was also affected by the nature of the system's sprite capabilities and the development of techniques to exploit this capability in previously unseen ways. Not only *Air Sea Battle* but also *Adventure, Freeway, Star Wars: The Empire Strikes Back,* and many other games were developed by programmers who carefully explore and exploited the sprite drawing capabilities of the system, and who learned, directly or indirectly, from what programmers before them had done.

2006: THE NINTENDO WII

The Wii from Nintendo offers low raw processing power but an innovative controller, recalling the way that the Nintendo DS relates to the Sony PSP. We describe the Wii's controller system, which uses accelerometers and radio frequency communication with the console base to map user gestures onto a three-dimensional space. We consider how the Wii platform relates to past controller development.

Low-Power, High Activity

When Nintendo announced the Wii in 2005, calling it the Revolution at that point, it turned away from its major competitors in the console videogame industry. Competing consoles from Sony (PlayStation 3) and Microsoft (Xbox 360) focused particularly on improved graphics, including support for higher-resolution high definition (HD) TV displays. Both Sony and Microsoft's entries into this generation of machines included higher powered processors and increased RAM, providing support for larger, more detailed game worlds. Nintendo took a very different tack, not trying to keep pace with these massive improvements in processing and memory power. Whereas the Xbox 360 boasts 512MB RAM and a triple-core processor running at 3.2 GHz, the Wii's IBM processor runs at 728MHz and couples to 88MB of total RAM.[10] This is an improvement over the GameCube's 485 Mhz IBM PowerPC processor and 43MB RAM, doesn't compare to the step up that the Xbox 360 and PlayStation 3 made from their predecessors. Instead of adding general computational power or much more highly-powered graphics, Nintendo focused on a new type of intuitive gestural interface. Nintendo also announced a focus on simpler games intended for players of all ages, suggesting that the Wii's gestural interface would afford more intuitive, facile control in games.

Influences

Physical interfaces for video games are nothing new. Understood in the broadest sense, arcade games and pinball machines had physical interfaces that required players to stand and jostle vigorously as they played. Custom physical control peripherals for the home console can be traced back to the early 1980s: Amiga's 1982 Joyboard was a plastic platform the player stood upon, rocking in all directions for control. In 1982 Atari planned an exerbike controller for the VCS codenamed "Puffer," although the device was never released commercially. LJN followed in Amiga's footsteps with the 1987 Roll 'n Rocker, a balance board controller for the Nintendo Entertainment System. Other types of physical interfaces include pad controllers and camera controllers. Pad controllers are best known today thanks to the success of *Dance Dance Revolution* but had their origins in early devices like Exus's 1987 Foot Craz for Atari VCS and Bandai/Nintendo's 1988 Power Pad for NES. The canonical camera controller is Sony's EyeToy, first introduced for PlayStation 2 in 200X, which uses computer vision to translate a player's gross motor movements into in-game game actions.

But the Wii uses different technology. Instead of converting physical movement into joystick direction as the Joyboard does, responding to floor-level touch sensors as a dance pad does, or using difference filters to detecting changing movement patterns in a video image, the Wii uses a combination of gyroscopes, accelerometers, and infrared sensors to accept user input. All of these sensors are built into the main controller, the "Wii remote." Even though Nintendo's marketing rhetoric made claims for radical innovation (as in the code name "Revolution"), both the technology and its application in games had already been tested in previous titles, mainly by Nintendo itself.

Gyroscopes and Accelerometers

Nintendo first experimented with motion controls in a title for the Game Boy Color handheld, *Kirby Tilt 'n' Tumble,* which was released in August 2000 in Japan, hitting the US market in April 2001. The well-received action/puzzle game has a gyroscope built into the cartridge and senses when the unit is tilted or jerked upward; Kirby moves left or right, or pops up into the air, in response to these movements. This technique was refined in cartridges for the GameBoy Advance (GBA). The third installation of the company's popular *WarioWare* series, *WarioWare Twisted* (Japan 2004/US 2005), offers microgames that require that the player turn, shake, and twists the entire GBA by applying yaw to the device. The second title, *Yoshi Topsy-Turvy* (Japan 2004/US 2005), adapted classic 2D platformer gameplay for a gyroscope controller. Players twist the handheld itself and also use the more traditional d-pad and button interface to move the character Yoshi through obstacles. Twisting the device alters the game world's gravitational center, allowing the player to solve physical puzzles by moving platforms or changing world's apparent floor.

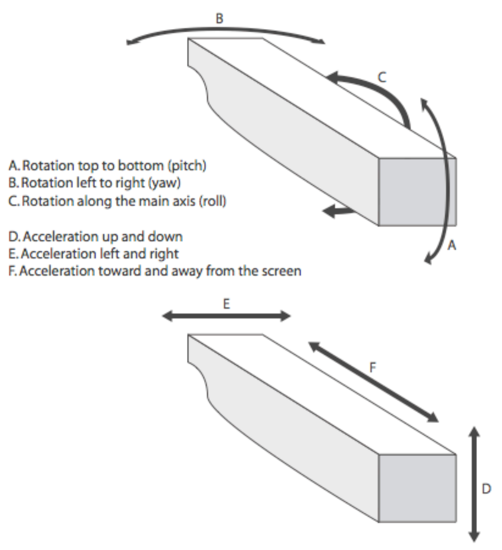
B
C
A
A. Rotation top to bottom (pitch)
B. Rotation left to right (yaw)
C. Rotation along the main axis (roll)
D. Acceleration up and down
E. Acceleration left and right
F. Acceleration toward and away from the screen
E
F
D

Figure 2: Categories of movement detected by the Wii remote.

These titles do not provide as rich a system of control as the Wii remote does, but they record how Nintendo used "cartridge hacks" as a way of prototyping gyroscopic control, both as an abstract interface principle and as a product in the market. The Wii extends the idea of earlier motion-sensitive cartridges both conceptually and technically. The Wii remote contains gyroscopes and accelerometers capable of measuring rotational movement and acceleration along the device's major axes. The basic motions shown in Figure 4 are the ones that are possible to detect and interpret in software.

In addition to the gyroscopes and accelerometers, the Wii also connects to an infrared (IR) strip sensor that detects when the Wii remote is pointed directly at the screen. The user configures the Wii for the location of the sensor — above or below the screen — and the sensor couples its readings with the top/bottom and left/right gyroscopic rotation sensors to create a pointing device. A cursor is displayed onscreen, and the Wii uses this cursor for primary selection in console menus, and some games also use it for gameplay.

Intuitive Interfaces

The Wii promises a more intuitive interface for video games. Both marketing and popular praise for the device announces that players can simply move the device as they would a prop like a tennis racket or bowling ball, and the game will respond accordingly. But the machine doesn't actually understand complex gestures that correspond directly to the basic components of real world physical activities. Instead, it understands rotation and acceleration along the axes just described. As a result, the software opportunities for the machine must either translate complex combinations and sequences of rotation and acceleration, or rely on simpler motion detection with the assumption that the player will interpret those simpler actions as more complex ones. The Wii's apparent ability to recognize gross motor gestures comes from combining combinations of the more basic motion detection methods.

The translation of the Wii's motion detection capabilities into basic gameplay mechanics are most easily observed in *WarioWare: Smooth Moves*, the first Wii game in the previously mentioned microgame-based series. In *Smooth Moves*, the player learns a series of very simple gestures, each in turn used for a set of microgames. With few exceptions, these gestures are constructed of one or two of the basic mechanical readings afforded by the device. The game gives the gestures themselves clever names (it calls them "forms," recalling basic postures in a martial art) both to telegraph interaction methods to the user and to hide the relative simplicity of the gestures.

For example, the game's first gesture, called "The Remote Control" asks the player to point the device at the screen like a remote control. This gesture mimics the recognition method of the Wii's basic pointing mode. Another gesture, called "The Umbrella," asks the user to hold the Wii remote upright and then move it's tip down toward the screen at the proper time. This gesture relies on gyroscopic pitch recognition. Another gesture, called the "Pencil" asks the user to hold the Wii remote on its sides like a pencil, and to move it toward the screen. This gesture relies on toward/away from the screen acceleration recognition. Yet another gesture, called "The Mohawk," asks the user to hold the Wii remote on his head and to move his body up and down by bending at the knees. This gesture relies on up/down acceleration recognition. And another gesture, called "The Waiter," asks the user to balance the Wii remote in the palm of his hand, moving the hand on a plane parallel to the floor. This gesture relies on left/right (and sometimes also on toward/away) acceleration recognition. Much as *Combat* offers a window into the hardware affordances of the VCS, *WarioWare Smooth Moves* shows the basic capabilities of the Wii.

Despite the platform's novelty, its controller is not a magical gesture recognizer, as some critics have noted.[11] As the actual technical affordances above attest, the device is no more capable of detecting the actions a player is actually performing than one might be capable playing charades with a man behind a curtain. More "realistic" games like *Red Steel* or *The Legend of Zelda: Twilight Princess* ask the player to swing the Wii remote like a sword. But the console does not judge these gestures based on their swashbuckling quality; rather, it simply looks for up/down and left/right acceleration. Clever players might quickly realize that sword swinging can be accomplished equally well by slouching on the couch, beating the cushions occasionally with the Wii remote. Such exploits are not defects in the platform; rather, they expose the technical underpinnings of the Wii's gesture system as it is actually implemented in hardware.

Nintendo took the opportunity on two portable platforms to load cartridges with additional sensors. Finding that there were development opportunities and that players were receptive helped to justify the risks of the Wii's unusual controller scheme. While new controllers can be developed for the Wii, the irony may be that as cartridge-based R&D systems such as the Game Boy Color and Game Boy Advance disappear, there will be fewer opportunities for low-cost, per-title, *ad hoc* interface growth. The more polished Wii, with innovative interfaces built in, may not provide room for the very sort of new controller experimentation that made the platforms possible.

CONNECTING PLATFORMS, PAST AND PRESENT

The platforms discussed, and the approaches used to understand them, help to connect the technical underpinnings of new media to what has been created. The bare-bones graphics system of the VCS and the techniques developed to exploit that system in unanticipated ways has an influence on specific games and on whole genres, such as 2D platformers and shooters. The Wii shows that controllers, as well as core functions, have a history and can be considered in a platform studies approach, and that even "peripheral" elements have the potential to power innovation in game development.

The cases studies here are the result of two different platform studies approaches to two different platforms. They were not undertaken in the hopes that they would lead to a single insight about one aspect of new media. What they show, instead, is that platform studies is a rich approach that can provide a variety of insights about new media's evolution. Studies of the material history of texts have shown how the technical specifics of writing and printing technologies can inform our understanding of literary history; we believe this sort of examination is even more important in new media, which is based on complex technologies that are capable of general computation, of response to user input, of storage and retrieval of information on a large scale, and of multi- and metamedium function. Not only do our three short studies fail to exhaust what platform studies can do in the specific cases of these platforms — they do not even begin to illustrate all the major categories of platforms or possible areas of technical focus.

EPILOGUE

We are hopeful that our efforts in platform studies will be of good use to those interested in creativity and computing, and that others in the digital media community will join us in this studies. To this end, we are happy to announce a new Platform Studies book series we are co-editing at the MIT Press. Potential writers should consult the series website at http://www.platformstudies.com.

NOTES

[1] Nick Montfort, "*Combat* in Context," *Game Studies* 6:1 (2006), http://gamestudies.org/0601/articles/montfort.
[2] Tekla E. Perry and Paul Wallich, "Design case history: the Atari Video Computer System," *IEEE Spectrum* 20:3 (1983), 45-51.
[3] Harold L. Vogel, *Entertainment Industry Economics* (Cambridge: Cambridge Univ. Press, 2001).
[4] Martin Campell-Kelly, From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry (Cambridge, MA: The MIT Press, 2004).
[5] Scott Cohen, *Zap!: The Rise and Fall of Atari.* (New York: McGraw-Hill, 1984).
[6] Perry and Wallich.
[7] Ibid.
[8] Ibid.
[9] Ibid.
[10] Marshall Brain, Jennifer Hord, "How the Wii Works." How Stuff Works. http://electronics.howstuffworks.com/nintendo-revolution.htm; Matt Casamassina, "IGN's Nintendo Wii FAQ," *IGN.* Sep 19. 2006, http://wii.ign.com/articles/733/733464p1.html.
[11] Erik Sofge, "Nintendon't: The Case against the Wii," *Slate.* November 20, 2006, http://www.slate.com/id/2154157.