What is Strafe Jumping?

idTech3 and the Game Engine as Software Platform

Dylan Lederle-Ensign & Noah Wardrip-Fruin

INTRODUCTION

Strafe jumping is a technique by which players can break the "speed limit" of games built on the Quake family of engines, and achieve up to double the normal movement speed. It exploits a bug deep in the physics engine, where the player's ground friction is calculated, to minimize this friction and speed up the player's movement through successive jumps. This speedup drastically increases the pace of gameplay and contributes to the sense that *Quake III Arena* (id Software, 1999) is a twitch shooter which rewards quick reflexes (Juul, 2005). Although it was possible to fix the glitch, the player community intervened to preserve it. It is a strange example, in which a glitch enters into the game design space, and is eventually adopted by the player community (followed by some designers and developers) as a key game mechanic.

This paper argues that a full account of strafe jumping requires an understanding of the context in which it emerged, both socially and technically. The distinct features of software platforms, specifically the game engine, must be taken into consideration. This paper characterizes game engines as software platforms, and uses this to conduct a brief platform study of idTech3, the engine underneath *Quake III Arena*. This study includes a code reading of the function that enables strafe jumping, and references John Carmack's extensive development notes to provide social and historical context. We consider the significance of strafe jumping to the player community's play experience, profiling the DeFrag movement mod. Finally, we

consider the lessons this study can provide to future work on software platforms and game engines.

WHAT IS STRAFE JUMPING?

> Strafe jumping is an exploitable bug. Just because people have practiced hard to allow themselves to take advantage of it does not justify it's existance (*sic*).
> — John Carmack, .plan June 3, 1999[1]

The precise technique for strafe jumping is difficult to describe, but in essence the player coordinates pressing the directional keys while jumping and moving the mouse to specific vectors. These vectors are miscalculated by the engine, and the normal friction which slows a player down after landing on the ground is not applied. By repeating this process, a player can accelerate to very high velocities and navigate around a level environment in new ways, bridging gaps which were previously impassable.

Strafe jumping, and associated movement techniques like circle jumping, were originally enabled by bugs dating from the first *Quake* (id Software, 1996). The player community mastered their use and incorporated them into the metagame. By the release of *Quake III Arena* three years later, official maps were being designed to specifically incorporate high skill "jumps," which allow shortcuts through a level, but require practice to hit the correct vectors every time.

Our first question, in this paper, is: Within the context of game studies, how should we think of strafe jumping? With our current historical and interpretive frameworks, what is it, and where did it come from? Unfortunately, this question can be difficult to answer.

For example, strafe jumping might be thought of as an element specific to particular id Software implementations of the common operational logics of physics and navigation. Mateas and Wardrip-Fruin define operational logics as a "representational strategy, supported by abstract processes or lower-level logics, for specifying

the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience" (2009). However, to be seen as representational strategies, operational logics must be positioned as *authored,* and strafe jumping was not authored. Instead, it emerged from the interactions of the complex system underlying *Quake III Arena*. This is in some ways similar to another well-known emergent property from *Quake*, rocket jumping. Rocket jumping is another movement phenomenon, but unlike strafe jumping it emerged from an unexpected combination of explicitly authored rules (in particular the properties of the weapon). In the case of strafe jumping, the framework of operational logics must be revised or set aside to understand the phenomenon.

From a different perspective, Juul includes the laws of physics in the rules of physical sports, arguing that *FIFA 2002* "requires that the laws of physics … be explicitly implemented in the programming *on the same level as the explicit rules of the game*" (2005, original emphasis). This is necessary for *FIFA* to accurately simulate a real game of soccer, and Juul argues that "therefore it makes sense to see the laws of physics on the same level as the conventional rules in soccer" (*ibid*). However, while real world physics influence the dynamics of soccer, viewing them as another system underlying all physical sports is clearer than characterizing them as rules. Physics is the "platform" on which the rules of sports are designed, and play in virtual worlds is shaped by the ways that their platforms implement and tune physics.[2] We could attempt to position this implementation and tuning as a set of rules, but this again implies authorship, or at least enumeration, and strafe jumping is neither an authored rule nor explicitly encoded anywhere in the software. It arguably isn't even an "emergent phenomenon" from the interactions of the rules, but a difficult-to-correct bug (given the way the platform's physics rules were encoded). Placing it on the same level as rocket jumping disregards the differences between the unintentional results of authored rules and the accidental nature of this glitch.

Alternately, we could turn to another popular approach in game studies and attempt to position strafe jumping as an example of game

rules and behavior being socially negotiated. But strafe jumping is precisely the result of the characteristics of a technical artifact. Its uses are of course socially negotiated, but we cannot use such a framework to understand its origins or examine its nature.

In short, strafe jumping is difficult to account for with current approaches to game studies. To understand where strafe jumping comes from and how it functions requires investigations at the platform level. To understand strafe jumping's importance and staying power requires an understanding of play phenomena, game design practice, and the influence of player communities — all in the context of game platforms. The primary platform for *Quake III Arena* is its engine, known as idTech3. The rest of this paper will develop a theory of software platforms, in an attempt to fully understand strafe jumping. Taking the game engine as a platform, it will explore the relationship between engines, genres, and their communities.

SOFTWARE **PLATFORMS**

Platform studies is an emerging field, but as Dale Leorke pointed out in a review of the eponymous MIT Press book series (2012), it is already verging towards a predictable formula: "One can imagine an endless production line of books—one on the Magnavox Odyssey or Sega Dreamcast, another on Java or Microsoft DOS—that are valuable in themselves, but which don't expand on the established formula of the series." While Leorke's statement suggests a relatively even balance between hardware and software platforms, in fact all the published books in the series (and, to the best of our knowledge, all but one of the forthcoming books) focus on hardware platforms. This paper instead is an initial look at a software platform — and examines some aspects of software platforms, particularly game engines, which have not yet received critical attention. The study of game engines provides insights into details of game genres, which are narrower and more specific than the constraints imposed by most hardware, as well as more malleable due to the flexibility of software. As might be expected in such an initial look, this paper identifies more issues than it seeks to resolve.

In Montfort and Bogost's diagram of the layers of digital media (2009), *platform* is at the bottom, followed by *code* above. Software platforms complicate this boundary, which is comparatively distinct and taken for granted when dealing with hardware. Software platforms are both made up of code to run on a variety of physical devices, as well as capable of executing code. This navigation through layers of abstraction is a strength of Montfort and Bogost's model, and studying software platforms helps us to understand the tangled, recursive relationships between code and platforms.

As this suggests, we see it as key that software platforms are less clearly defined than commercial game consoles and other hardware platforms. Just as the boundary between code and platform blends and becomes difficult to pin down, even the border between different platforms is fuzzy. As our below discussion of idTech3 will demonstrate with a proliferation of modified engines, software platforms can be used piecemeal by developers, and extended well beyond their original capacities in ways that hardware platforms rarely are. How much shared code is necessary to be considered the same platform? If two software platforms share an architecture, but have significantly differing implementations, how do we characterize this relationship? If code written for one software platform runs on another, are they different realizations of the same platform? These are questions that past studies of hardware platforms have not addressed.

Murray and Salter, in their study of Flash (Forthcoming) situate it within the tangled platform of the Web. Flash extends the native capabilities of Web browsers, and must co-exist with a multitude of technologies like CSS, HTML and JavaScript. Flash is clearly a platform of its own, with a full development suite and very different affordances from the native Web. However, its development and existence are also closely intertwined with the Web, as Murray and Salter explain in a chapter considering Flash's future in an HTML5 Web. Tangled, simultaneously existing platforms seem to be a characteristic of software platforms, one which deserves further study.

This is an incomplete characterization of software platforms. As noted above, it raises more questions than it answers. However, it gets us closer to an understanding of strafe jumping. This phenomenon is a property of its engine, which can be characterized as a specific type of software platform. Game engines are the key software platform for video games.

Game Engines as Platforms

Game engines are the infrastructural software and tools which allow developers to manage the vast complexity of modern games. They commonly provide physics simulations, networking, and graphical primitives as well as often handling portability across physical computing platforms. They are just as important for shaping the playable experience as the hardware itself, perhaps more so. Decisions made in the design and implementation of the engine clearly constrain the games made on them.[3]

Henry Lowood identifies *DOOM* (id Software, 1993) as the first game to use the term "game engine." He associates the engine with development efficiency, and argues that in 1993 id imagined "the licensed game engine could become a platform upon which diverse games would be constructed" (Lowood, 2014).[4] He continues on to state that "the development of engine technology traces the growth and maturation of the game industry" (*ibid*). The efficiency enabled by game engines allowed the industry to develop games much more rapidly than before.

Game developers will license another company's engine to build their game on, but the exact requirements for each game are different. Due to the flexibility of software, the underlying engine can in some cases be modified as necessary. While hardware modifications can be done by individuals, they are not typically done by commercial game studios. Instead, hardware platforms are extended with peripherals.[5] Though it is possible to modify the game engine, the challenge of understanding the internals of a system as large and complex as a modern 3D game engine makes it difficult to make the changes

developers desire. It is easier to treat the engine as a black box to interface with, and utilize the official tools provided for developers to achieve their goals.

Andrew Hutchinson has called working inside the technological limits provided by platforms the “pragmatic expression” of games (2008). Hutchinson writes that both *Doom* and *Myst* (Cyan, 1993) originally aimed for a similar immersive 3D style. However, due to the limited computational power available in 1993, both were forced to make compromises and engineering decisions that influenced the aesthetics of the games. Hutchinson explains that “*Myst* went the visual ‘high and slow’ road, and *Doom* went the ‘low and fast’ road” (2008). Despite their flexibility, software platforms are still constrained by the hardware platforms below them.

In “Untangling Twine” (2013), a paper about the hypertext story platform Twine, Jane Friedhoff argues that the documentation, community and discourse around the Twine platform make it uniquely suited to experimental playable media experiences that push at the boundaries of what games are. Friedhoff rightfully emphasizes that Twine’s profiling by independent game maker Anna Anthropy “likely shaped the initial crop of games created with the platform.” Friedhoff argues that both the free, web-based nature of Twine and its promotion by Anthropy make the platform appealing for marginalized people. She notes that the official documentation explicitly tries to appeal to writers rather than coders saying: “rather than answering ‘how would you make a game with this?’, the official Twine reference manual focuses on answering ‘why would you make a game at all?’” Friedhoff’s work emphasizes that documentation and community are key to a platform’s growth and the aesthetic that emerges around them.

Combining Hutchinson’s writing on the technical limitations that game engines address with Friedhoff’s about the community around them, we can start to see a clearer picture of the significance and influence of game engines as software platforms. Engines, like hardware platforms, have a particular set of affordances — which

make certain kinds of game creation comparatively easy for developers. They also generally have an ecosystem of tools for people contributing in different manners (e.g., as level designers, artists and coders) to interact with the engine in particular ways. These combine to align engines so closely to particular genres and styles of game that technical research into game engines has questioned whether engines can be separated from their genres (Anderson et al., 2008).

Game Engines and Game Genre

Game genre is notoriously messy and marketing defined (Aarseth, 2004). Game genres are usually defined by the dominant play activity, while genres in other media are typically defined by shared iconography (Wolf, 2001). To address this disconnect, David Clearwater proposes three aspects of genre categorization: formal/aesthetic, industrial/discursive context, and social meaning/cultural practice. While the industrial/discursive aspects are certainly influential in shaping game engines, this paper addresses formal dimensions of genre.

Game engines are made to support certain prototypical games. In the case of idTech3, this was *Quake III Arena*, but even engines which are not made for a single game have a particular type of game in mind. The support for these games takes the form of implemented features, which we will characterize as particular operational logics (Mateas and Wardrip-Fruin 2009). These easily usable logics make it straightforward for developers to make games similar to the prototypical game. However, because game engines are also modifiable and extensible, developers find other uses for these logics and ways to adjust them. This is one source of difficulty regarding genre. Two games may share many common formal elements, and perhaps substantial source code, but may have very different play dynamics because of extensions or modifications to the original engine.

This also leads to a major difference between a game engine and a more general purpose hardware platform like a game console.

Engines are closely aligned to particular prototypical games. This leads to the surprising characterization of the Atari VCS as a game engine implemented in hardware. As Bogost and Montfort (2009) detail extensively, the console was designed to support Pong. The necessary operational logics (most notably collision detection) were then appropriated to form the myriad other games published for the console.

Other authors have characterized video game genres as common interfaces (Douglass, 2007), but they can also be thought of as collections of common operational logics. Game engines are tightly tied to specific genres because they implement a large number of these shared operational logics (Wardrip-Fruin, 2009). Close study of the implementation choices for a particular logic is one way to learn more about it and how it functions across different games. This characterization of genre is not meant as a complete overhaul or recharacterization of existing genre literature. Engines are one way to clarify that messiness, operational logics are clear

Game Engines and Community

Game engines, like all platforms, have complex communities of users around them. The original developers, commercial licensees, amateur modders, and players all have a stake in the engine's success and direction. John Carmack, in his .plan from December 31 2004, discussed the delayed open sourcing of idTech3 due to a recent licensing agreement: "Previous source code releases were held up until the last commercial license of the technology shipped, but with the evolving nature of game engines today, it is a lot less clear. There are still bits of early Quake code in Half Life 2, and the remaining licensees of Q3 technology intend to continue their internal developments along similar lines, so there probably won't be nearly as sharp a cutoff as before." The original creator, Carmack, wanted to open source this technology to make it easier for modders and amateurs to create games using it. However, his company had commercial agreements with other studios. Releasing it for free would devalue their purchase of a license. The network of actors with

a stake in the future of idTech3 was complex, and Carmack wanted to serve as many competing interests as possible.

As Carmack mentions, some small pieces of Quake code survive, both in the current generation of idTech engines as well as those of their licensees, most notably Valve's Source engine (used in Half Life 2). The tangled nature of game engines makes drawing clear lines between them difficult, and perhaps pointless. It is easier and more descriptive to classify them into "families," such as the Quake family (of which Source could be a branch), the Unreal family, the Crysis family, and so forth. Each family has a distinct style and "feel." This shared sense of feel can be traced back to low level decisions in the physics or graphical simulations that form the engine.

Game designer and teacher Robert Yang noted on his blog the influence that the Source engine has had on his aesthetic: "When I'm trying to tune movement physics in other games, am I just trying to replicate the feel of Half-Life because that's what feels 'right' to me? (Unreal Engine games almost all universally feel 'chunky' to me, in comparison. I'm sure people who grow up using Unreal would disagree with me, and argue that Half-Life or Quake-lineage games are too loose.)" (Yang, 2013). This is one of the key reasons for studying game engines. More than any other single piece of software, they exert influence over the design of the games and genres that are built on them.

In short, we argue that understanding game engines as software platforms is useful for studying game genres, operational logics, and the games built on them. This characterization provides technical insight into the way specific logics work, and includes the social factors by which communities shape their platforms, but requires setting aside a strongly authorial view of all aspects of logics as intentionally representational. We follow this, in this paper's next section, with a short platform study of the idTech3 engine.

IDTECH3

The history of id Software has been told more widely and often than most video game developers'. It is an American success story, of the small independent creator striking it rich with a good idea and devoted work ethic. David Kushner tells this story in the popular history *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture* (2003). Kushner's book is framed around "the two Johns," Carmack and Romero. Their creative chemistry and interpersonal drama leads to the rise and fall of id. This paper is not about their legacy, but about their technology, specifically the engine from *Quake III Arena*, idTech3.

Focus on Mod Programming in Quake III Arena (Holmes, 2002), a technical manual for aspiring game programmers, describes the influence of John Carmack as follows: "John Carmack, lead programmer at id Software, is the man responsible for creating the technology that drives all the latest and greatest games. Not only do his 3D engines power id Software's games, such as the *Quake* series, it also powers many other companies' games as well, thanks to licensing agreements." As seen in Figure 1 (Wikipedia user Tei, 2013) the Quake family of games and engines is expansive. While this diagram is from Wikipedia and has some problems (it does not distinguish clearly between mods, engines, and commercial games, and it does not indicate how closely related linked entities are) it does give a sense of the connections between the idTech engines and their descendents. It is also clear that after idTech3 there were significantly fewer licensees than the earlier games. Despite the numerous games built in the Quake-engine family, Carmack claims it was not their intent: "It's interesting when you look at our technology licensing — it was never really a business that I wanted to be in" (Graft, 2011). While idTech4 was licensed to a handful of outside studios, idTech5 is solely for id Software use. idTech3 was the last of the highly influential Quake engines.

According to a popular Quake 3 source code review from Fabien Sanglard, "the engine is mostly an evolution of idTech2" (Sanglard,

2012). Building on the groundbreaking work that they did with the first Quake, id Software's engineering team, led by John Carmack, created a complex game engine that balanced the goals of speed, security and portability across physical computing platforms. Internally known as "Trinity", the engine introduced the Quake Virtual Machine (QVM), which Holmes' emphasis allowed for increased security for running mods. This was, in fact, one of Carmack's main goals, as noted in his .plan entry from November 3, 1998. The idTech2 architecture supported mods in the form of potentially dangerous .dlls. In order to continue supporting mods, Carmack went to the significant engineering effort of developing the QVM, running a subset of the C programming language, QuakeC. idTech3 also moved to fully hardware rendered graphics, and featured an improved network model (Lederle-Ensign, 2013).

idTech3 exemplifies the diversity and extensibility of software platforms. There are a number of engines that are direct descendents of idTech3, including ioquake3, Quake III w/ Uber tools, and qFusion. Uber tools were a set of proprietary extensions to the engine from Ritual Entertainment, used in *American McGee's Alice* (Rogue Entertainment, 2000), *Star Trek Elite Force II* (Ritual Entertainment, 2003), and several other games. The improvements seem to mostly be in scripting for single player experiences, an area that was not a concern for *Quake III Arena*. QFusion and ioquake3 are both open source forks of the engine. Ioquake maintains a modernized version of the engine, which supports a number of total conversion mods released as standalone games, such as *Urban Terror* (Silicon Ice, 2000) and *World of Padman* (Padworld Entertainment, 2007). QFusion was originally a port of idTech2, but now supports idTech3 data formats. It was mainly developed for the competitive arena shooter *Warsow* (Warsow Team, 2005).

The diversity of the idTech family of engines is a trait of software platforms. Software platforms are more flexible than hardware platforms, and are easily extended by third parties. This leads to a proliferation of slightly modified platforms, and adds challenges for

scholars studying them. Frequently software platforms are less stable objects and more groups of related concepts and software objects.

The history and technical details of the idTech3 platform provide context for strafe jumping. It came from a fast moving start-up studio, which iterated quickly on its technology. While a distinct platform in its own right, idTech3 cannot be understood in isolation from the engines that came before it, or those that followed. In the next section, the site where strafe jumping is encoded is traced through these different engines.

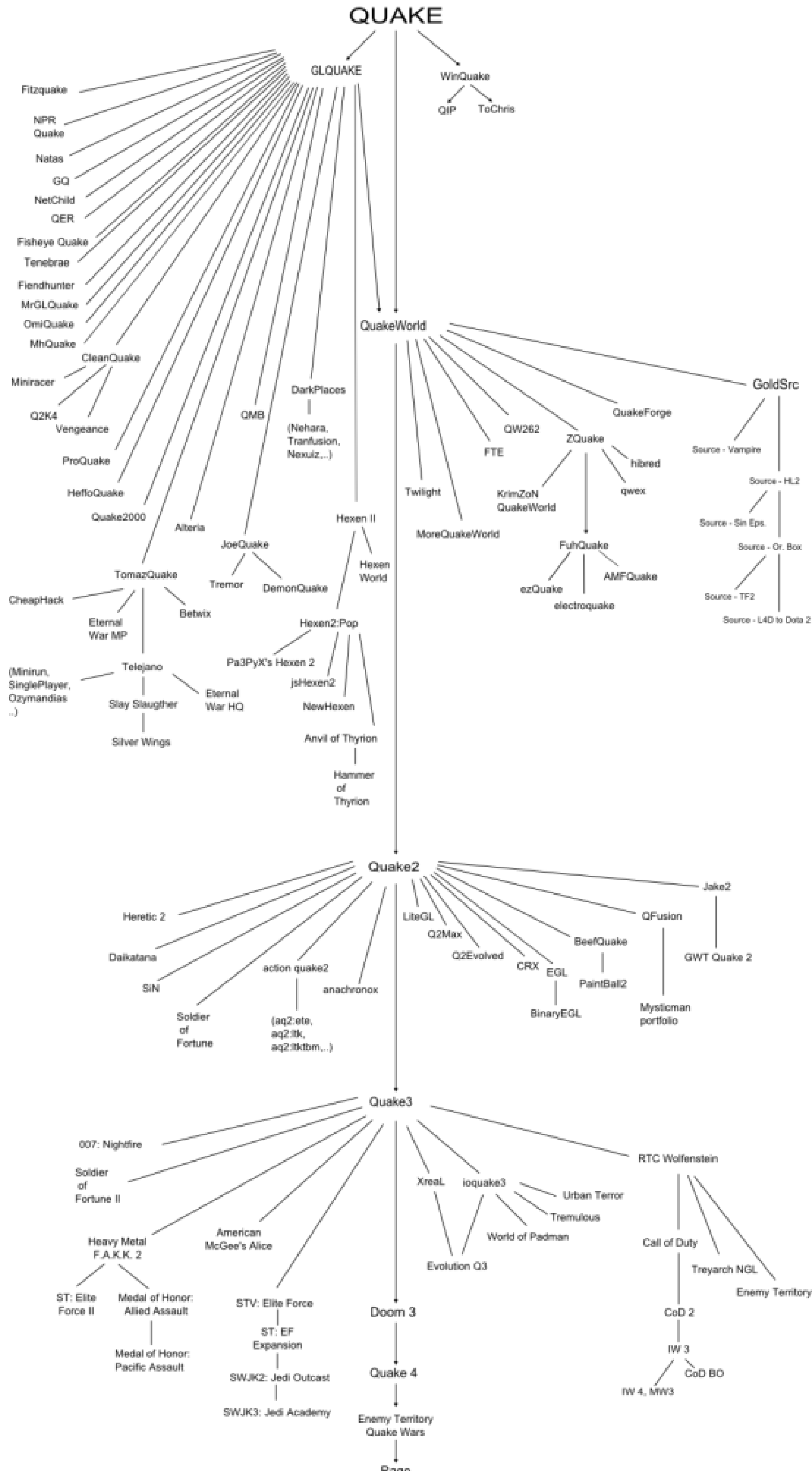
QUAKE
GLQUAKE
WinQuake
QIP
ToChris
Fitzquake
NPR Quake
Natas
GQ
NetChild
QER
Fisheye Quake
Tenebrae
Fiendhunter
MrGLQuake
OmiQuake
MhQuake
CleanQuake
Miniracer
Q2K4
Vengeance
ProQuake
HeffoQuake
Quake2000
Alteria
QMB
DarkPlaces
(Nehara, Tranfusion, Nexuiz,..)
JoeQuake
Tremor
DemonQuake
TomazQuake
CheapHack
Eternal War MP
Betwix
Telejano
(Minirun, SinglePlayer, Ozymandias ..)
Slay Slaugther
Eternal War HQ
Silver Wings
Hexen II
Hexen World
Hexen2:Pop
Pa3PyX's Hexen 2
jsHexen2
NewHexen
Anvil of Thyrion
Hammer of Thyrion
QuakeWorld
Twilight
MoreQuakeWorld
FTE
QW262
ZQuake
hibred
qwex
KrimZoN QuakeWorld
FuhQuake
ezQuake
electroquake
AMFQuake
QuakeForge
GoldSrc
Source - Vampire
Source - HL2
Source - Sin Eps.
Source - Or. Box
Source - TF2
Source - L4D to Dota 2
Quake2
Heretic 2
Daikatana
SiN
Soldier of Fortune
action quake2
(aq2:ete, aq2:ltk, aq2:ltktbm,..)
anachronox
LiteGL
Q2Max
Q2Evolved
CRX
EGL
BinaryEGL
BeefQuake
PaintBall2
QFusion
Mysticman portfolio
Jake2
GWT Quake 2
Quake3
007: Nightfire
Soldier of Fortune II
Heavy Metal F.A.K.K. 2
ST: Elite Force II
Medal of Honor: Allied Assault
Medal of Honor: Pacific Assault
American McGee's Alice
STV: Elite Force
ST: EF Expansion
SWJK2: Jedi Outcast
SWJK3: Jedi Academy
Doom 3
Quake 4
Enemy Territory Quake Wars
XreaL
ioquake3
Urban Terror
Tremulous
World of Padman
Evolution Q3
RTC Wolfenstein
Call of Duty
Treyarch NGL
Enemy Territory
CoD 2
IW 3
CoD BO
IW 4, MW3

Figure 1: Games descended from idTech1 Engine

Strafe Jumping in Code

When I tried fixing the code so that it just didn't work, I thought it changed the normal running movement in an unfortunate way. — John Carmack, June 3, 1998

Carmack viewed strafe jumping as a bug. Thanks to id's open sourcing of *Quake III Arena* in 2005, we can pinpoint exactly where this bug occurs. In the file "/code/game/bg_pmove.c" we find the following function:

```
//Handles user intended acceleration

static void PM_Accelerate( vec3_t wishdir, float wishspeed, float accel ) {

#if 1

// q2 style

int          i;

float    addspeed, accelspeed, currentspeed;

currentspeed = DotProduct (pm->ps->velocity, wishdir);

addspeed = wishspeed – currentspeed;

if (addspeed <= 0) {

return;

}

accelspeed = accel*pml.frametime*wishspeed;

if (accelspeed > addspeed) {
```

```
accelspeed = addspeed;

}

for (i=0 ; i<3 ; i++) {

pm->ps->velocity[i] += accelspeed*wishdir[i];

}

#else

// proper way (avoids strafe jump maxspeed bug), but feels bad

vec3_t      wishVelocity;

vec3_t      pushDir;

float       pushLen;

float       canPush;

VectorScale( wishdir, wishspeed, wishVelocity );

VectorSubtract( wishVelocity, pm->ps->velocity, pushDir );

pushLen = VectorNormalize( pushDir );

canPush = accel*pml.frametime*wishspeed;

if (canPush > pushLen) {

canPush = pushLen;

}

VectorMA( pm->ps->velocity, canPush, pushDir, pm->ps->velocity
);

#endif
```

}

Following Carmack's mention that his fix for strafing changed "normal" running for the worse, we believe that PM_Accelerate is the main place where the physics necessary for strafe jumping are implemented. This function is called when the player wishes to accelerate, taking in the player's intended direction, her intended speed, and an acceleration multiplier. The precise vector math bug is not as important as the comments around the code. First, note that the first block, inside of "#if 1", will always execute. In C, and many other programming languages, 1 is equivalent to true, so this is just a way of block commenting out the second half of the function so it is not evaluated. As the comment in the second block alludes, this code takes out support for strafe jumping. However, it does so at the cost of the game's "feel."

Small choices of how to write highly specific simulations become hugely important and influential to the game design space. As more code and game assets are built on the assumption of a particular behavior, it becomes calcified and cannot be easily changed. This complicates the idea that software is flexible. It also demonstrates that the modern game engine is so complex that even the lead developer on the project cannot always determine the consequences of low level changes. At the platform level, code is difficult to change.

This function can also be found in several other related code bases. We can find the identical function in the ioQuake and qFusion. This makes sense, as those projects are trying in some way to build on the idTech3 source releases. We can also go backwards and find identical code in the *Quake II* (id Software, 1997) source code. In fact, while not identical, we can find extremely similar code in the original *Quake* source code. In the *Doom 3* (id Software, 2004) codebase, running on idTech4, we find a function with the same code structure, with the same comments, but updated to C++.

In the open-sourced codebase for *Return to Castle Wolfenstein* (2001)

we can find the same function, nearly identical except for a snippet code preceded by this comment:

// Ridah, variable friction for AI's

"Ridah" was the nickname for a programmer at Gray Matter Interactive, one of the studios that worked on RTCW. The code appears to be a simple modification, not significantly altering the effects of strafe jumping, but it is an example of something that you can find throughout the RTCW codebase. When the developers have modified something "deep" in the engine, code which was originally written by id, they usually noted the change with a signed comment. These signatures are not found in other files that were newly authored for this game, only the engine code. It is a clear indication that idTech3 is a substrate for this new game, and an indication of how different software platforms are from hardware platforms. While hardware modding is certainly possible, it is not a typical, commercial activity. With the source code of the engine available, there is nothing *at a technical level* stopping developers from completely changing the behavior of PM_Accelerate and removing strafe jumping from their games. However, the weight of the surrounding system's complexity makes any change a risky endeavor, hence the cautious signing of any modifications.

Indeed, complexity would not be enough to stop Carmack from "fixing" strafe jumping if it were a normal glitch. However, this was a glitch which had many vocal defenders in the Quake community.

Strafing, the Player Community and Software Platforms

In his .plan file from Jun 03, 1999 Carmack posted this about strafing:

Some reasonable messages have convinced me that a single immediate jump after landing may be important to gameplay. I'll experiment with it. Strafe jumping is an exploitable bug. Just because people have practiced hard to allow themselves to take advantage of it does not justify it's existance (*sic*). When I tried fixing the code so that it just didn't work, I thought it changed the normal running

movement in an unfortunate way. In the absense (*sic*) of powerups or level features (wind tunnels, jump pads, etc), the game characters are supposed to be badasses with big guns. Arnold Schwartzenegger (*sic*) and Sigourney Weaver don't get down a hallway by hopping like a bunny rabbit. This is personal preference, but when I play online, I enjoy it more when people are running around dodging, rather than hopping. My personal preference just counts a lot.

His references to action movie stars give a sense of the aesthetic goals that id had for *Quake III Arena*. They were aiming for a cartoony, Hollywood style action adventure. They wanted to empower their players to act out fantasies of being "badasses with big guns." This is an extraordinary example of a developer responding to community wishes on something as fundamental as movement physics. The Quake community loved the challenge and blazing speed made possible by strafing.

As Carmack mentions when he discusses "normal running", the "feel" of movement is immensely important for player enjoyment of a game. An episode from the development of *Quake II* illustrates how strongly people feel about slight changes to the movement physics. In the post-release patch 3.15, the following change was made by an id programmer nicknamed "Zoid," who was maintaining the game: "Player movement code re-written to be similiar (*sic*) to that of NetQuake and later versions of QuakeWorld. Player has more control in the air and gets a boost in vertical speed when jumping off the top of ramps." In a .plan entry from July 4, 1998 entitled "Here is the real story on the movement physics changes" Carmack addresses concerns of the community surrounding the apparently controversial change. After defending the rights of Zoid to make changes he wanted, Carmack acknowledged that "The air movement code wasn't a good thing to change in Quake 2, because … subtle physics changes can have lots of unintended effects." He goes on to note that: "None of the quake physics are remotely close to real world physics, so I don't think one way is significantly more 'real' than the other. In Q2, you accelerate from 0 to 27 mph in 1/30 of a second, which just (*sic*) as unrealistic as being able to accelerate in midair." The next day, the

change was made optional for servers to enable or not, and Carmack closed the issue by reflecting (presumably sarcastically) on “the joy of having a wide audience that knows your email address.”

This movement can be a pleasure in its own right, exemplified by the *Quake III Arena* mod “DeFrag,” which is a movement-based mod in which players navigate obstacle courses as fast as possible. The mod offers several movement styles, including the “vanilla” Q3A, as well as CPMA from the Challenge ProMode Arena mod, which adds increased movement control for skilled players. DeFrag also distributes official map packs to challenge players. These maps are designed to exploit strafe jumping, as well as emergent phenomenon like rocket jumping, in ways that are not found in the conventional deathmatch maps. The goal is to move from one end of the level to the other as fast as possible, and when it was active the community held contests for who could achieve the lowest times on officially sanctioned maps. For some players, DeFrag functions as a training mod, with features which let you keep track of how fast you are moving in order to practice and improve your strafing abilities. The DeFrag scene also embraces and celebrates “tricking” or moving about the level in non-intuitive ways, reminiscent of digital parkour.

The DeFrag community distributes demos that are run in-engine, but they also edit their exploits into machinima. Their movements are synced to music, typically electronic, and the effect is extremely evocative of dance. The most viewed DeFrag video on YouTube is “Event Horizon 2 – Quake 3 Team Trick Jumping” with over 380k views at the time this article was written. Rather than a solo video promoting a single player’s skill, it promotes a “tricking crew.” The bulk of the video’s 15 minutes is devoted to elaborately choreographed group tricks, such as having multiple players firing rockets at the same spot in order to propel another further into the air than is possible alone. It feels very much like a dance troupe.

While DeFrag players push the limits of strafing with their precision, nearly all *Quake III Arena* players who hope to be competitive must learn the basics or be left in the dust. One particularly famous

example is the Bridge-to-Rail jump on the map Q3DM6, or “The Camping Grounds.” This jump allows players to access the map’s only rail gun, a particularly powerful and useful weapon, in a little under 3 seconds from a particular bridge. Without the jump, it takes more on the order of 10-15 seconds, a lifetime in Quake. This jump was clearly designed into the map, and is key to successful play. As evidenced by the numerous YouTube videos demonstrating in detail how to learn the jump, it has become a rite of passage for players.

This demonstrates that while John Carmack and other id programmers may originally have wished to eliminate the “bug” of strafing, by *Quake III Arena* id’s level designers treated it as just another affordance of the engine. They exploited it in their designs, elevating the behavior from glitch to mechanic. In fact *Quake Live* (2010), the browser based version of *Quake III Arena*, features official tutorials on strafe jumping techniques.

Strafe jumping is an important feature for the Quake player community, as exemplified by the DeFrag mod. The community is also central to the persistence of the glitch. id Software fixed many glitches in the course of developing idTech3, but this one survived. It survived in large part because the player community advocated for it as an important part of their play experience.

CONCLUSION

In the course of investigating the phenomenon of strafe jumping, we have developed an account of game engines as software platforms. This allows us to more fully understand the context in which strafing developed and understand strafe jumping itself. Strafe jumping emerged from the complexity of modern game development and the attempt to manage this complexity by abstracting common processes into a game engine. These processes make up the infrastructure that games are built on. As this phenomenon illustrates, small implementation details can have wide ranging effects on the play experience of games using the engine. Game engines are particularly opinionated development tools, ones which are tightly tied to game

genres. Genres require common operational logics to be implemented across games, which creates a problem that game engines solve. Some of these logics, which are below the level of the rules which a game designer specifies, nevertheless exert a large influence over the play experience. As our code study demonstrates, sometimes fully understanding these logics requires careful consideration of their implementation — and a setting aside of the assumption that every aspect of implemented logics should be seen through the lens of authorship.

This study raises questions about the nature of software platforms, particularly with regard to their flexibility or stability. We believe idTech3 to be a distinct platform in its own right, but we have also traced nearly identical code through several generations of idTech engines. While hardware platforms surely share common characteristics between generations, the textuality of code makes it simple to trace specific implementations through different code bases, and the open sourcing culture of id Software made this study possible. A strong argument could be made that all the idTech engines are the same platform; merely iterations on a theme. While we chose to focus on one specific code base, the ease of patching and updating software does complicate platform boundaries.

The social negotiation highlighted in Carmack's open development diaries is another important element. Software's flexibility allowed Carmack to take input and implement changes rapidly, without the manufacturing time and cost associated with hardware. However, the complexity of the engine made it difficult to "fix" strafe jumping. This complicates the idea that hardware is stable and software is flexible. This tension points to further work for software studies beyond games.

Strafe jumping is an unanticipated phenomenon from the platform layer. It is a possibility enabled by specific implementations of the operation logics of navigation and physics, creating a new means of interactive movement through a simulated 3D space. While it originated below the level of the authored rules, some games have

incorporated it as a mechanic and designed game features around it. It was discovered, embraced and advocated for by an active player community. Ultimately, it became a distinctive feature of games built on idTech engines.

END NOTES

[1] With implementations dating back to the 1970s, the "finger" command on some network computer systems allows the querying of a particular user or network resource's status. For users this can include items such as full name, email address, and special files ".project" and ".plan" — which in some contexts served similar purposes to the (micro) blogging and status updates that are common in today's social networking approaches. At id, John Carmack used his ".plan" to share information with the public about his current work and ideas, which we reference here as they appear in the archive at http://floodyberry.com/carmack/plan.html/.

[2] Further, we understand real world physics easily and unconsciously, but fully comprehending implemented physics requires study.

[3] Beyond a certain point of working against the assumptions and constraints built into an engine, it is more effective for developers to use a different engine or write a new one. For this reason, the decision to use an engine (which may be made by executives rather than developers) is the decision to accept a certain level of constraint from its architecture.

[4] While the term "game engine" may have come into use around the development and release of *DOOM*, the separation of data and process in game development, allowing multiple games to be developed by substituting new data, certainly had prior precedent in the industry — including in the practices of Infocom and LucasArts. However, we are aware of no previous examples of engines licensed by game developers to outside game developers.

[5] In some cases, physical hardware has also been extended through additional chips embedded in game cartridges and other approaches that may not immediately come to mind when hearing the term "peripherals."

BIBLIOGRAPHY

Aarseth, E. (2004). Genre Trouble | Electronic Book Review. Retrieved April 07, 2015, from http://www.electronicbookreview.com/thread/firstperson/vigilant

Anderson, Eike Falk, Steffen Engel, Peter Comninos, and Leigh McLoughlin. 2008. "The Case for Research in Game Engine Architecture." In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, 228–231. Future Play '08. New York, NY, USA: ACM. doi:10.1145/1496984.1497031. http://doi.acm.org/10.1145/1496984.1497031.

Apperley, T. H. (2006). Genre and game studies: Toward a critical approach to video game genres. *Simulation & Gaming*, *37*(1), 6–23. doi:10.1177/1046878105282278

Bogost, I., & Montfort, N. (2009). *Racing the Beam: The Atari Video Computer System*. MIT Press.

Carmack, John. "The John Carmack .plan Archive." http://floodyberry.com/carmack/plan.html.

Douglass, Jeremy. 2007. "Command Lines: Aesthetics and Technique in Interactive Fiction and New Media". Dissertation, University of California, Santa Barbara. http://jeremydouglass.com/dissertation.html.

Graft, Kris. 2011. "E3: Id's Carmack, Willits Happy To Be Done With Engine Licensing."*Gamasutra*. June 8. http://gamasutra.com/view/news/125324/E3_ids_Carmack_Willits_Happy_To_Be_Done_With_Engine_Licensing.php.

Holmes, Shawn. 2002. *Focus on Mod Programming in Quake III Arena*. Boston :; Independence :: Course Technology ; CENGAGE Learning Distributor.

Hutchinson, Andrew. 2008. "Making the Water Move: Techno-Historic Limits in the Game Aesthetics of Myst and Doom."*Game Studies* 8 (1). http://gamestudies.org/0801/articles/hutch.

id Software. 1993. *DOOM*. https://github.com/id-Software/DOOM.

———. 1996. *Quake*. https://github.com/id-Software/Quake.

———. 1999. *Quake III Arena*. https://github.com/id-Software/Quake-III-Arena.

———. 2004. *Doom 3*. https://github.com/id-Software/DOOM-3.

———. 2010. *Quake Live*. http://www.quakelive.com.

Jane, Friedhoff. 2013. "Untangling Twine: A Platform Study."http://www.digra.org/wp-content/uploads/digital-library/paper_67.pdf.

Juul, Jesper. 2005. *Half-real: Video Games Between Real Rules and Fictional Worlds*. Cambridge, Mass.: MIT Press.

Kushner, David. 2004. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. New York: Random House Trade Paperbacks.

Lederle-Ensign, Dylan. 2013. "Lags, Frags and John Carmack: a Platform Studies Analysis of the Quake III Network Module". Conference Talk presented at the Digra 2013.

Leorke, Dale. 2012. "Rebranding the Platform: The Limitations of 'Platform Studies'." *Digital Culture & Education* 4 (3): 257–268.http://www.digitalcultureandeducation.com/uncategorized/dce1073_leorke_2012_html/.

Lowood, Henry. 2014. "Game Engines and Game History."*Kinephanos* (History of Games International Conference Proceedings).http://www.kinephanos.ca/2014/game-engines-and-game-history/.

Mateas Michael and Wardrip-Fruin, Noah. 2009. "Defining Operational Logics."http://www.digra.org/wp-content/uploads/digital-library/09287.21197.pdf.

Montfort, Nick, and Ian Bogost. 2009. *Racing the Beam: The Atari Video Computer System*. 2nd ptg. The MIT Press.

Murray, John, and Anastasia Salter. 2015. *Flash: Building the Interactive Web*.

Padworld Entertainment. 2007. *World of Padman*.

Ritual Entertainment. 2003. *Star Trek: Elite Force II*.

Rogue Entertainment. 2000. *American McGee's Alice*.

Sanglard, Fabien. 2012. "Quake 3 Source Code Review." June 30.http://fabiensanglard.net/quake3/index.php.

Silicon Ice. 2000. *Urban Terror*.

Tei. 2013. "File:Quake – Family Tree.svg." *Wikipedia, the Free Encyclopedia*.http://en.wikipedia.org/wiki/File:Quake_-_family_tree.svg.

Wardrip-Fruin, Noah. 2009. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. The MIT Press.

Warsow Team. 2005. *Warsow*.

Wolf, Mark J.P. (Ed.). (2001). *The Medium of the Video Game*.