

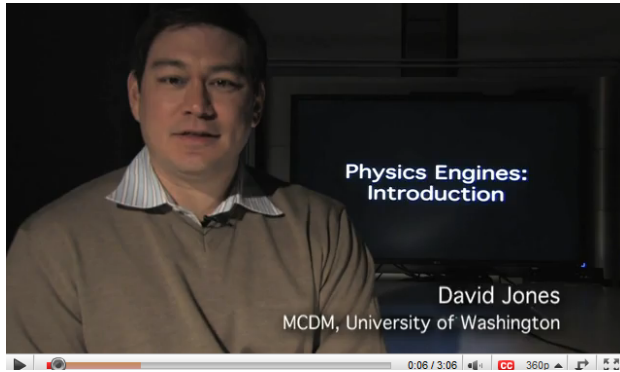
# 2010 & Beyond

## Video Game Physics Engines



Matthew Franco, David Jones, Chelsey  
Glasson and Shane Suzuki  
Com 546: Evolution and Trends in Digital  
Media  
3/18/2010

## Introduction



**Click Picture to Access Intro Video**

The video game industry has grown at an incredible rate over the past six decades and several advancements in gaming have been made over the years. Games are increasingly becoming more realistic as a result of higher quality graphics, developments in physics engines, and overall better game play.

Physics engines are libraries of code that perform approximate simulations of physical systems (Bolton, 2006). There are two types of physics engines: high-precision engines and real-time engines. High-precision engines require more processing power, so they are used more for scientific modeling rather than video games. The physics engines found in video games are real-time engines, which provide simulations performed at a rate appropriate for game play. (“Physics Engine,” n.d.) Real-time engines are the focus of this paper as they are a major driving force behind the realism in video games today.

In the following pages, a discussion concerning the history of video games and video game physics occurs, followed by a review of current trends and practices in video game physics. Finally, a look into the future of game physics engines is provided.

## Game Physics: History



In order to successfully forecast the future of video game physics engines, it is essential to examine the history of video games and video game physics. While the following history is by no means comprehensive, it highlights several landmark events that have greatly influenced the evolution of video game physics engines.

**[Click Picture to Access Video](#)**

### *1950s: The Rise of the Electronic Game*

It can be speculated that games have been in existence for equally as long as the human race. For centuries, games have served a variety of functions, including: providing a means to hone skills, social stimulus, psychological development, and entertainment. Although games in general have a deeply rooted history, electronic games, the precursor to the modern-day video game, did not come into existence until the late 1940s. Credited as the first electronic game, *Tic-Tac-Toe* was built in 1949 by University of Cambridge Ph.D. student A.S. Douglas. According to Nielsen, Smith and Tosca (2008), Douglas used an Electronic Delay Storage Automatic Calculator, a primitive British computer, to create the game. *Tic-Tac-Toe* and other electronic games programmed primarily for academic research purposes paved the way for the eventual creation of *Tennis for Two*, the first electronic game built using basic physics calculations. Programmed by physicist Willy Higinbotham in 1958. *Tennis for Two* was played on an oscilloscope using technology that could predict the path of bouncing balls and missile trajectories (Nosowitz, 2008).

### *1960s: The Birth of the Video Game*

The introduction of primitive electronic games is noteworthy because these games inspired three MIT employees to create *Spacewar!*, considered the first official video game, in 1962.

*Spacewar!* was born out of a love of science fiction and was revolutionary for its time for its use

of physics and mathematical calculations to compute ship motion, gravity, and the position of stars and the sun (“Spacewar!”, n.d). According to Barton and Loguidice (2009), “*Spacewar!* introduced real-time action, an arsenal of weapons, special moves, variable game conditions, physics, and a virtual world. It demonstrated that computers were far more than just expensive calculators.” Within weeks of its invention *Spacewar!* received attention and notoriety from researchers and entrepreneurs who wanted to study and modify the game. *Spacewar!* had an immense impact on the standards used in the making of subsequent video games and was the first electronic game to generate a profit, marking the initial promise of the video game industry.

Also significant to the history of video game physics is the creation of the first video game console, the Odyssey, by television engineer Ralph H. Baer in the late 1960s. The initial game pack made for the first generation Odyssey consisted primarily of sports-oriented games, including *Table Tennis*, a predecessor to the legendary game *Pong*. These games are among the first examples of programmers using basic physics calculations to create video games for the television (Winter, n.d.). However, opposition from cable companies prevented the Odyssey from hitting the market until 1972.

### *1970s: The Era of the Arcade Game*

According to Nielsen et al. (2008), “Although the previous decades had several possible beginnings of video games, the 1970’s saw them grow explosively. This decade marked the birth of video games as an industry...” (pg. 52.) In the 1970s arcade games became extremely popular as evidenced by the success of Atari, the maker of the first widely successful arcade games, including *Computer Space* and *Pong*. Through arcade game sales, Atari earned \$3 million in gross income in 1973 and \$40 million in gross income in 1975.

Although the 1970s did not mark substantial improvements in video game physics, it is important to recognize that the decade set up the framework necessary for the evolution of game physics engines. The success of arcade games in the 1970s led to the realization of the massive market for game consoles for the home. Upon finally being released to the public in 1972, 200,000 Odysseys were eventually sold at the price of \$100 each. Seeing Odyssey's success, Atari invented and successfully launched Home-Pong, a single-game console, in 1975. Home-Pong resulted in Atari's gross income increasing to an astounding \$200 million in 1978 (Nielsen et al, 2008). The 1970s also introduced several different genres of games, which developed consumer expectations and demands that later influenced trends in video game physics.

#### *1980s: Video Games as Home Entertainment*

Several exciting developments in video games occurred during the 1980s. In 1980, the first video game to incorporate three-dimensional (3D) elements, *Battlezone*, was created. With the goal of improving upon the limited physics of motion offered by earlier arcade games, the makers of *Battlezone* used vector graphics, which consisted of thin lines plotted mathematically on a high-resolution monitor. Interestingly, *Battlezone* was later enhanced and used by the United States government for training exercises ("Battlezone", n.d.).

In 1981, U.S. arcade revenues reached \$5 billion (Kent, 2001). 1981 was also the year that Nintendo became a serious competitor in the video game scene with its release of the arcade game *Donkey Kong*, the first "jumping" video game invented. Inspired by the success of the Odyssey and Home-Pong, Nintendo used revenue collected from *Donkey Kong* and other arcade games to build its first game console for the home, called the Nintendo Entertainment System (NES) in North America. *Super Mario Brothers*, the game shipped with the NES upon its public

release in 1985, featured the jumping action in *Donkey Kong* while it also integrated principals of friction and momentum. According to Karsemeyer and Henry (2008), “Using software to emulate real world physics made the game intuitive and even players who had never played a video game before found it enjoyable.” Sega also released its first video game console in 1983, the SG-1000; however, the SG-1000 and other game consoles introduced in the 1980s were not nearly as successful as the NES.

For over two decades, *Super Mario Brothers* was the best-selling video game of all time, before being outsold by *Wii Sports* in 2009. *Super Mario Brothers* had a massive impact on the video game world and a large portion of video games programmed in the 1980s and early 1990s expanded upon the popular “real world” physics models the game implemented. Some of these games include: *Zelda II* (1988), *Prince of Persia* (1989), *DuckTails* (1989), and *Sonic the Hedgehog* (1991). It is important to note that programming the physics models initially introduced in *Super Mario Brothers* was an expensive and time intensive process that occurred in-house in the 1980s and early 1990s. Nevertheless, in the aftermath of the success of *Super Mario Brothers*, it was not an option for video game developers to disregard game physics.

By 1984, video game consoles resided in one in four homes in America (Nielsen et al, 2008). A variety of new video game genres that entered the market in the 1980s—such as fighting, adventure, and racing —along with an expanding market, advances in video game graphics, and fierce industry competition, resulted in significant developments in the video game industry in the 1990s. These developments forever impacted video game physics.

*1990s: Here it Comes, the Game Engine*

For several reasons, the 1990s were a revolutionary time for video games and video game physics. Although game physics was not the primary focus of the game world in the 1990s, several events during the decade triggered the push for better physics in video games and the development of the first generation of physics engines.

In the 1990s the new multibillion-dollar personal computer industry presented new challenges for the video game industry. According to Kent (2001), “Sega and Nintendo found themselves facing a new and increasingly more dangerous opponent in the early 1990s—PC Computers....[which] threatened to eclipse the new generation of video game manufacturers as the era of multimedia began” (pg. 455). Not only did PCs offer better sound quality than video game consoles, they also offered better graphics and physics processing. This proved to be popular with consumers as illustrated by the success of early PC games such as *Myst* (1993), *Doom* (1993), and *Quake* (1996); which were all ground-breaking for their time in their use of graphics, physics, audio and animation. Real-time strategy and multiplayer games also became popular in the 1990s as a result of PCs and the introduction of the Internet.

As the PC game market began to grow in the 1990s, the game console market began to shrink. In fact, according to Nielsen et al (2008), a 17-percent drop in the U.S. console market in 1994 was followed by an even worse drop of 19-percent in 1995. Influenced by the success of the new PC video game market, the focus of all major video game manufacturers in the 1990s quickly shifted to improving video game graphics. Realizing the added costs of rendering better graphics in-house, Nintendo announced plans in 1994 to collaborate with Silicon Graphics to incorporate better graphics capabilities into its newest game console, the Nintendo 64 (Kent, 2001). This was among the first attempts at outsourcing a portion of game technology development, rather than doing all of the work in-house. Following Nintendo’s lead, games such as *Doom II* (1994),

*Quake III Arena* (1999) and *Unreal* (1998) were designed with this approach in mind, marking the introduction of the game engine.

By the late 1990s using an in-house or third-party game engine to speed up development and reduce the costs associated with programming graphic intensive video games was standard practice for many game developers. The first generation of third-party graphics game engines was dominated by three players: BRender from Argonaut Software, Renderware from Criterion Software Limited, and Render Morphics' Reality Lab ("Game Engine," 2010). These engines paved the way for the growth of third-party physics engines in the 2000s.

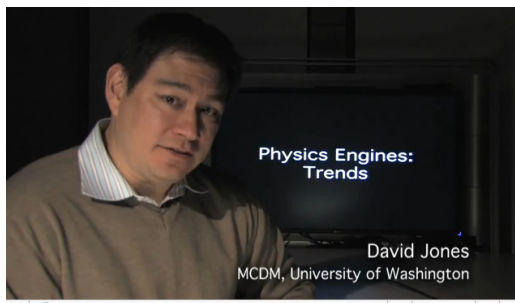
Ragdoll physics were also introduced in the 1990s with the release of the video game *Jurassic Park: Trespasser* (Kent, 2001). Ragdoll physics allows for the implementation of somewhat realistic, unique death scenes in video games based on a character's skeletal system. Prior to the introduction of ragdoll physics, video game programmers used preexisting, scripted animations for death scenes.

#### *2000s: The Physics Engine Market Explosion*

The 2000s was perhaps the most competitive decade to date for video game companies, with the Xbox 360, Playstation 3, and Wii all vigorously fighting for market share. In the 2000s it became common for video game developers to spend as much as \$25 million to produce a video game (Nielsen et al., 2008). Not surprisingly, it also became increasingly standard in the 2000s for video game developers to rely on third-party game engines to save time and money in the video game development process.

Some of the biggest players in video game physics entered the market in the 2000s. For example, Havok's first software development kit was officially unveiled in 2000 and Ageia, the inventor of the physics engine PhysXs, was founded in 2002 (purchased by Nvidia in 2008). An overview of these and other physics engines occurs in a subsequent section of this paper. However, it is important to mention that these physics engines were behind the advanced physics seen in popular games programmed in the 2000s, such as *Half-Life 2* (2004), *Gears of War* (2006), *Halo 3* (2008), and *Bioshock* (2007). The trend of third-party physics engine makers focusing on very specialized middleware products also began in the 2000s. In 2008, Havok became a leader in specialized middleware products upon its release of Cloth and Destruction. Cloth simulates the soft body dynamics of video game characters while Destruction deals with rigid body destruction.

### Current Trends in Video Game Physics Engines



The importance of the physics engine to the “serious” game industry and the gaming experience cannot be overstated. According to ATI’s Platform Technology Marketing Director, “Physics acceleration is the next

big leap in gaming; it’s the next step in evolving the immersive game experience beyond what gamers know today” (Cheng, 2006). Ever increasingly, AAA video gamers have an expectation for greater realism in their games, both visually and behaviorally. As discussed previously, comparative advantage in the late nineties was driven by who could build games the fastest, with the best graphics using one of the dominant game engines of the day. The more realistic the worlds and characters looked, the better. Now, realistic visuals are no longer enough. Games and game studios are being judged by how well they create great characters, storytelling, and how

realistically the characters and worlds behave within the game environment. Examining some of the reviews of last year's most popular game titles at the [E3 Expo](#), reveals that the games are hyped just as much for their authentic environmental destruction such as *Metal Gear Solid 4's* collapsing buildings and bridges, as they are knocked for the short comings of their physics evident in technical glitches such as *Mass Effect 2's* occasional issue with characters getting stuck in invisible objects and jittering back and forth.

It's important to note that game physics are not only important in the "serious" game space, but also becoming popular in the casual game space as well. Perhaps the most popular example is [Crayon Physics](#), a game built entirely around the concept of drawing working two-dimensional physics puzzles.

Companies like Havok and Nvidia continue to extend their engines (Havok and PhysX respectively) to provide increasingly elaborate possibilities. In this section, we will briefly discuss some of the current trends and emerging techniques for incorporating physics into the serious games segment that these modules are enabling. Specifically, this section will focus on destructible environments, more authentic representations of human behavior and characteristics, cloth behavior, and how technology is shaping the way that developers bring physics into their games.

### *The Physics Challenge*

The serious games industry is extremely challenging yet rewarding. Currently, the cost for building a serious video game is comparable to the budgets of some smaller Hollywood blockbusters. Yet with some video games outperforming movies exponentially in revenue generation, a wave of new game studios is flooding the industry – many backed by non-

traditional players like movie studios. The market is competitive and even major game studios are susceptible. Electronic Arts (EA) recently announced that it would be closing its subsidiary Pandemic, the studio responsible for the *Star Wars: Battlefront* series and *Mercenaries: Playground of Destruction* (Cabral, 2010). High season for new game releases is the holiday season. The net effect of this perfect storm of high production cost, more competition and one major selling season in which to compete for limited attention and dollars means the risk for failure is high.

This backdrop is important to understand because game physics is now presenting game studios with an increasingly diverse set of challenges. Game developers must establish priorities considering the specific game's physics needs while also bearing in mind the hardware limitations, processing tradeoffs and the overall cost to build and deliver the title.

### *To Physics or Not To Physics*

As the possibilities for bringing more realism into games increase, game developers must consider what should and should not be incorporated into a game. To what degree will more physics add to the game play experience rather than detract from it? In an interview with Guardian Games Blog writer Keith Stuart, Richard Hackett, Technical Director at Blitz, poses the question another way: How should developers balance the need for more realism provided by physics with the unrealistic behavioral aspects that make games fun (Stuart, 2009)? Physically impossible moves—taking two gun shots to the chest and surviving—and outrageous character animations are all desirable aspects of games. Furthermore, Luke Schneider, Project Design Architect for *Red Faction: Guerrilla*, suggests one concern for developers is integration across the areas of game play that were once considered separate elements. Theoretically, physics could

touch nearly every aspect of video game design; from character looks, to artificial intelligence (AI) of non-player characters (NPCs), to game flow control. Accordingly, choosing which elements of physics to invest time and money to integrate into a game and how they all work together can be critical to success or failure (Stuart, 2009).

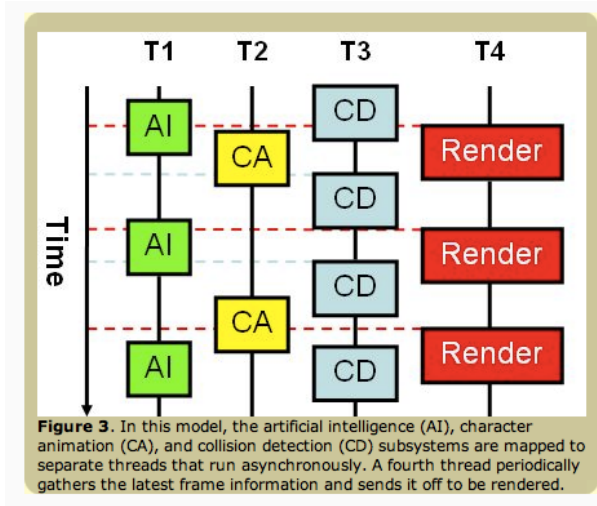
Another key concern for bringing physics into game play is one of technology. Developers must carefully consider how physics use may impact their addressable market. The tradeoff for more physics is greater processing requirements that may negatively impact game play. Lack of standardization across user hardware creates challenges for creating a game experience that is optimized for the largest possible audience. Currently physics algorithms can be processed by core processing units (CPUs), graphics processing units (GPUs), and/or physics processing units (PPUs). For systems that do not have dedicated physics accelerators, “game play” and “effects” physics need to be processed on the CPU, and consequently may result in fewer frames rendered per second. This can result in screen stutters interrupting game play. The introduction of multi-core processors has helped increase processor speeds on CPUs, but it also changed the processor architecture completely by enabling multi-threaded processes which in turn forced developers to re-write code for this new architecture to address parallel processing and the resulting challenges (Shrout and Davies, 2009). To further complicate the lack of standardization situation, the two major physics engines providers (Nvidia and Havok) take completely different approaches to how physics are handled with Nvidia focusing on dedicated graphics accelerators and Havok accelerating physics via the CPU. Consequently, a battle is being waged for the hearts and minds of game developers.

Of course, there remains an inherent problem with real-time game play, and that is that frame play is by nature sequential. Use of multiple subsystems (e.g. physics engines) creates

dependencies upon one another, and since processing time varies by subsystem this will create inefficiencies in rendering:

The inherent complexity of game engines makes them more difficult to thread than other applications. Various subsystems [e.g., artificial intelligence (AI), physics, character animation, collision detection, rendering, audio, etc.] manage aspects of the game engine. As a player moves through the virtual world of the game, a video forms. However, there is no predefined sequence of events. The actions of the player are effectively random from the game's perspective. The game is a real-time, event-driven process. This creates a problem for threading—the frame sequence is inherently sequential. In other words, each frame depends on the preceding frame. Therefore, multiple frames cannot be computed independently and in parallel to boost frame rates and improve performance. Techniques to predict more than a frame in advance would introduce latency and negatively impact the player experience (Gabb and Lake, 2005).

Gabb and Lake (2005) provide the below diagram as a visual example of the challenges



developers face in calibrating subsystem processing times in a multi-threaded environment. This diagram demonstrates the dependencies between subsystems and the parallel processing inefficiencies that can result. In the example, collision detection computation is indicated by the blue dotted line, and the frame render computation is

indicated by the red dotted line. Looking at the third frame render computation, we see that three collision detection computations have taken place, but the second collision computation is not animated before the third collision computation begins, thereby nullifying the second collision computation. The result is wasted processing power and time.

This illustrates how game developers are able to create increasingly immersive game experiences due to advances in technology and physics capabilities, yet these same advances are presenting developers with a whole new set of challenges.

### *Destructible Environments*

It can be argued that physical destruction in video games is a major selling point and has been one of the biggest beneficiaries of physics engine evolution. Over the last several years environmental destruction has evolved from primarily predefined animations triggered at specific points in game play to today in which designers enable game objects to break in a realistic manner using physics. According to Mike Enoch, Lead Coder at Ruffian Game studios, “We’re starting to see a lot more detail in how materials are simulated. Traditionally, it’s all been smoke and mirrors, a wooden crate would break up in predefined ways depending upon how the developer thought it would be smashed. It’s now possible to physically simulate the properties of the wood, so the crate would splinter and break up dynamically depending on the force and direction of an impact” (Stuart, 2009).

The recurring question is, how much physics should be utilized? As Nadeem Mohammad, Nvidia PhysX Product Manager suggests, “as Nvidia APEX (which will allow artists to more easily implement physics elements into game designs) starts to roll out, expect to see destructible environments which have real game impact, perhaps opening up new paths within the game and highly realistic responses to the forces applied by different weapons” (Stuart, 2009). The implication here is that game players can impact their individual game experience more than ever based upon the actions they take. Today, the available paths through a game’s levels are carefully planned and defined by the game’s designers. As the properties of more game objects

become physically simulated, there is the real possibility that the player could alter the pre-defined path through a level that the game designer intended. A building that once had to be navigated around could possibly be destroyed and moved over; a door that previously could not be unlocked could be blown open by rocket launcher. Likewise, a player could inadvertently blow up a building creating insurmountable obstructions to the only available path through the level. This could result in an inability to control the pacing of the game, and a potentially poorer game experience for the player.

Another potential challenge with deciding how much realism is necessary in environment destruction is the depth to which physics would need to be applied to properties of an object. Using the example of a building again, would it be necessary to simulate all the properties of a building in order for it to collapse accurately? As Enoch suggests, added physical realism to an object's properties "gives much better feedback to the player, empowering them by creating realistic reactions for every situation...with pipes bending, walls smashing, and large structures collapsing when you knock out some of the supports" (Stuart, 2009). Yet, what is the time and cost implication for the game developer to enable this kind of realism? For example, will the developer be forced to create and apply physics to more of a building's actual structural elements such as pipes, wiring, sheetrock, among other elements in order for the building to react accurately to destructive forces (Stuart, 2009)?

Certainly, the impact of integrating more realistic physical destruction in environments goes beyond what we have addressed here, both for the developer and the game player. Developers will have to think through and design more holistic worlds to account for the potential free roaming of a player through levels. For gamers, the added emotional impact can be substantial.

For example, a player's visceral reaction to destruction will be directly tied to their choice of weapon.

The topics we have discussed in this section only skim the surface of benefits, challenges and implications for today and the future of environmental destruction in game play.

### *Clothe, Hair and Weapons*

Although not as critical to game play, more focus is being placed on creating greater realism in the behavior of secondary character elements as a way to add cinematic value to a game's character animation. This concept is known as *secondary motion*. Secondary motion refers to the coordination between layered character systems, such as hair and cloth, with a character's body movement to create accurate overall character motion (Stuart, 2009). This motion is powered by applying physics to these elements and then making them interact with other physics-supported objects, as well as traditionally animated objects.

The impact secondary elements make to a character's movement is minimal and potentially adds little value to the overall game play, yet can add significantly to the visual stimulation a game can provide. For example, imagine a character whose hair reacts to head movement or gusts of wind. Dave Gargan, Principle Engineer at Havok, expects secondary motion to become a buzzword over the next couple of years and offers the following example why: "if a character runs and stops suddenly, momentum will be carried through their garments and this secondary motion that preserves momentum is visually convincing" (Stuart, 2009).

The ability to apply secondary motion consistently and realistically is challenging, yet the potential for real game impact as techniques and capabilities improve is substantial. Today, the focus on secondary motion physics is simply to create some natural secondary movement of the



**Figure 1**

**Figure 2**



object based upon its weight, and the velocity and direction that a character is moving. This would create more visual realism, but have no real game impact. Consider, however the example of a weapon and how the weight or shape of that weapon may impact a character's movement in terms of pace, smooth motion, and ability to navigate tight environments. Today many first-person shooters (FPS) offer a wide variety of weapons with no penalty for character performance. By applying real-world characteristics to these items, players would be forced to think more strategically about their choice of weaponry. Choosing more destructive weapons like a Barrett Sniper Rifle (Figure 1), a smaller rocket propelled like the M72 LAW (Figure 2), or even a heavier weapon like the M60 (Figure 3) while running through the game world would no longer be possible, and have to be based more on realistic conditions. The prospects for both positive and negative impacts on game play and satisfaction are evident.

### *Life-Like Characters*

As photo-realism becomes more of a reality in animation, the need for greater realism in human character responses to the game environment increases. Consumers are being conditioned by the wellspring of increasingly sophisticated animated films of the last ten years to expect realistic human reaction and representations of emotion, and physical attributes such as smooth body motion, muscle flexion, sweat, and even life-like eyes that emote. Inaccuracies are more obvious and off-putting as they take us out of our immersion into the film world. This expectation of realism in human characters has crossed over to the video game world as well.

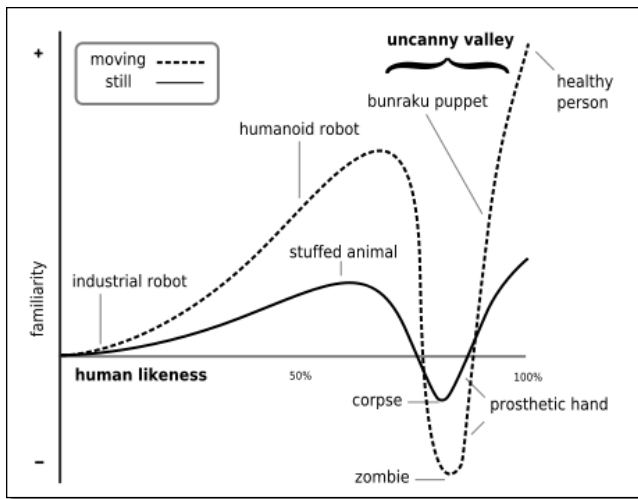
Developers are relying more on procedural animation techniques over predefined, "canned" animation or motion-capture techniques to portray realistic character behavior. Procedural animation also helps address the increasing challenges of longer and more diverse game play that

would make it nearly impossible to predict and create all the predefined character animations that that would be traditionally required. Procedural animation uses physics algorithms to automatically generate animations in real-time, allowing characters to react to different stimuli such as physical surfaces or forces against their body.

Natural Motion's [Euphoria](#) engine is leading the industry in Dynamic Motion Synthesis (DMS), the term they have coined that refers to the application of the abovementioned techniques, factoring in muscle and motor nervous system responses to stimuli in order to affect realistic character response. *Backbreaker*, a football title, has been hyped for its ability to showcase real-time human movement and completely unique tackles rather than the predefined tackles that game players have been exposed to previously. Other popular titles that make use of DMS are *Grand Theft Auto IV*, *Star Wars: The Force Unleashed*, and the forth coming *Red Dead: Redemption*.

Another challenge in bringing more realistic human behavior into video games is the coordination of the various elements involved in the human character such as hair, skin, muscles and skeleton. EA Sports has taken a step in this direction with their boxing title, *Fight Night 4*. Rather than create the entire body as one object, which then reacts to applied forces, the developers at EA have created character bodies made of layers, each layer reacting to whichever specific force is applied. Additionally, they have attempted to make these elements “aware” of one another in order to create a realistic coordinated response. To add to the realism, the developers created a system of muscle and fat movement that could be applied to the animation being played. The result was realistic muscle tension in the character striking his opponent, and fat movement in the character being punched (Stuart, 2009).

The prospects for video characters to appear more “life-like” are very real, and as capacity to accurately simulate realistic looks and behaviors increases, developers will need to be cognizant of the potentially negative emotional response of players to the game characters. For instance, a study in robotics has demonstrated that as androids become increasingly human-like, a level of familiarity is reached when people begin to exhibit positive emotional responses and empathy toward the android. However, there is a point of inflexion where the response becomes quickly negative when a look, movement, behavior or some other action is counter to expectations. This dip in the graphical representation of positive, negative familiarity is the source of the name, the *Uncanny Valley* theory first postulated by robotics engineer Masihiro Moiri, and is depicted in figure 1 below (Ishiguro, 2007).



**Uncanny Valley**

The ability to create affect with gamers is due in part to the connection gamers have with game characters which is facilitated by advances in photo-realism and physics (along with great storytelling). Character connection in turn contributes to deeper game immersion for the player, which is a critical component to the success of any video game.

There are already clear signs that the industry is continuing to drive toward greater realism and more direct interaction between humans and video game characters, as exemplified by Lionhead Studio’s work on [Milo and Kate](#) for Project Natal. As characters become more life-like the risk articulated by the Uncanny Valley theory is real and could have a serious impact on a game, and

the industry's success.

## **The Future of Video Game Physics Engines**

As stated earlier in this paper, for every new physics problem solved, another new physics problem is introduced. For gamers this means as graphics become more realistic and storytelling becomes more evolved, the game play and environmental physics are often the last piece to ruin the immersive feeling of many modern video games.

The most recent advance in gaming was 3 dimensional (3D) modeling. By allowing players to move in a virtual 3D world, game developers were able to create a new gaming experience that was truly unique. This advance spawned the explosion of the FPS genre, the creation of the “Open Sandbox” adventure/action game, and made sports and driving games more realistic than ever before (Lilly, 2009). However, as graphics and game play options continued to improve, the need for realistic (or at least consistent) physics became more important than ever.

To solve this physics problem, a handful of approaches are being taken by game developers and the developer support industry. These include: hardware dependent, software dependent, open source and inclusive game engine design.

At first, game developers continued the practice of coding their own physics models into the game engine, however a number of third-party developers were working on plug-in physics engines that would allow game designers to concentrate less on making the game physics accurate and more on making games fun.

*All Inclusive Engines*

There is no doubt that id Software, a Texas-based game developer, was the catalyst for both 3D games and a focus on the physics in games. The release of *Quake* in 1996 ushered in the modern era of games, true 3D modeling and more importantly, player influenced physics that impacted game play (Waveren, 2009). Using what would later be dubbed the id Tech engine, this and similar all-in-one game engines such as Frostbite and Unreal allowed game designers to allocate resources in level design, story and playability of their games instead of developing new game frameworks.

Although very popular in the early 2000s, these models have decreased in popularity as more a la carte options have been developed in the last few years. This is not to say these models are not important and influencing modern game development—some of the most popular games in recent years are based on id Tech, Frostbite and newcomer CryTek. But as processors became more powerful, the ability to handle multiple threads of information at the same time reduced the needs for one software package to manage system resources during game play. Increased processing power meant that individual processes could be specialized; resulting in the growth of third-party programs managing everything from collision detection to how sheets of cloth waved in the wind (Lilly, 2009).

#### *Commercial Physics Engines: Software Dependent*

Havok, an Irish company founded in 1998, released their first commercial physics engine in 2000 (Havok Physics, n.d.) By this time 3D graphics had gone from rare to standard and the basic 3D software engines had been incorporated into the operating systems and become non-hardware specialized. During this same time, graphics processing units (GPU) became standard accessories for gamers wanting to play the latest and best-looking games. By offloading the processing of graphics from the main processor chip to the GPU, game developers could now

allocate processor resources into other game play systems such as physics or artificial intelligence.

Havok took advantage of this by providing developers with a highly customizable engine that could manage collision detection, environmental interaction, character movement and object interaction. This third-party approach to managing basic behind-the-scenes game play functions proved to be exceptionally popular. The Havok engine has been used in more than 150 major games and has won numerous awards for its contributions in making games more realistic and immersive (Havok Physics, n.d.)

The other benefit to this approach was that as long as the game system guaranteed certain basic equipment standards, the Havok engine would be able to function on any system without a need for specialized cards. As mentioned previously, developers have to balance the desire to make the most technologically advanced game with the need to make their games as widely playable as possible. Havok helped by making their engine extremely flexible and portable to many different hardware configurations making it very robust and easy for developers to use.

#### *Commercial Physics Engines: Hardware Dependent*

While Havok was taking a software approach to game physics, a company named Ageia began work on a hardware solution to in-game physics support in 2006 (PhysX, 2010). The end result was PhysX, a physics processing unit (PPU) that plugged into the motherboard similar to a GPU. In theory, the PhysX card would take the physics calculation workload off of the main processor allowing for better physics and smoother game play.

Although the technology was promising, few games were able to take advantage of the hardware because of a relatively small install base. In 2008, video card manufacturer Nvidia bought Ageia effectively ending the era of the independent PPU.

Under the management of Nvidia, PhysX has now become a viable competitor to Havok. PhysX is primarily designed to offload physics calculation away from the main processor. However, instead of a specialized physics processor, Nvidia has created updated software that allows any of its newer GPUs to manage the PhysX engine. PhysX is still not as popular as Havok in terms of numbers of games using the engine, but Nvidia has signed agreements with many of the big names in game and console design to use their PhysX solutions including Sony and Microsoft (“NVIDIA PhysX SDK Features”, 2010)

Although Nvidia has since released a software-reliant version of PhysX similar to Havok, the cornerstone of their product is the ability to easily upgrade the physics experience of the game for users who own a newer Nvidia GPU. Since Nvidia controls more than 50% of the GPU market, game developers have been quick to use the PhysX Software Developers Kit as a middleware physics solution. (“PC Video Card Market Share in the Last 30 Days”, 2010) For gamers who have the latest hardware, their game play experience will be exceptional. However, since the SDK does a very good job (on par with Havok’s basic engines) without the newer GPUs, developers can use PhysX without concern that gamers will not buy the game because of stringent hardware requirements.

### *Open Source Physics Models*

Similar to OpenGL becoming a major player in the 3D graphics engine world, a number of entities are working under open source models to create viable physics engines. Two of them,

Bullet and ODE (Open Dynamics Engine) have been adapted by major game and film studios to help create AAA titles such as *BloodRayne 2* and *Grand Theft Auto 4*; and films like *2012* and *Hancock* (“Bullet (Software)”, 2010).

Although free for use, these open source physics engines generally require a large amount of technical knowledge and customization by the end user. This creates a barrier for some game developers who are looking for ready-made physics engines that can be used essentially right out of the box (Smith, 2007).

#### *What this all Means for Gamers*

While the final section of this paper looks at where the industry is heading, there are a few generalities that can be learned from these approaches. The largest studios will create all in one solutions that can be adapted across platforms and genres. This allows one engine to be used on many different games, however the cost of developing one of these engines is prohibitive for all but the largest game studios. Proprietary engines such as Havok and PhysX are dominating now, but as the trends section shows, more and more of the functions of physics engines are being specialized into smaller plug-ins, meaning developers can mix and match according to their needs. Finally, open source may one day become a standard, but until the physics engine matures, most developers will rely on more robust and proven technologies.

#### *Future Trends in Game Physics Engines*

By using the rise of 3D graphics as a model, one can make certain assumptions about the future of a game technology based on managing processing resources. The main limiters of physics implementation at this point in time are platform-independent standards and pure processing power.

The problem of processing power in home and game computing is merely a waiting game.

Moore's law of computing power shows no immediate signs of slowing down now that multiple processor cores are being installed in almost every personal computer and game system.

Meanwhile, the power capabilities of Graphics Processing Units are advancing even faster.

Because of their specialized design, current generations of GPUs are using more than 2.1 billion transistors to process graphic textures, shading, motion, 3D renderings and real-time physics.

This will pale in comparison to the next generation of GPUs, which will have close to 3 billion transistors providing advanced graphics and video calculations (Atwood, 2008).

As processors get more and more powerful, the need for game developers to prioritize specific game elements (graphics, artificial intelligence, physics, sound, etc) will decline. Just how 3D graphics became ordinary and expected soon after GPUs became standard, we expect physics to follow the same path as processing increases over the next decade.

While processors continue the march toward faster and faster chipsets, the makers of the actual physics engines are being forced to find new ways to differentiate their products. Actual physics calculations and the formulas to determine how different objects will react in motion are mostly developed at this time. The ability to do these calculations efficiently in real-time is very close as well. We predict a trend away from dedicated physics engines and a movement toward middleware that manages smaller and more refined aspects of game play.

As the standard hardware in computers continues to advance, the two big players in physics engines will have to adapt to make their products more diverse. Havok is doing this by creating a suite of game development tools similar to the Adobe Creative Suite of image creation software.

PhysX does this by following a freemium model – they give away their SDK in hopes that if more games use their advanced features Nvidia will sell more GPUs.

However, we predict that in less than 5 years none of this will matter. We predict that while these two companies continue to refine their engines and solve the problems that arise as physics become more precisely modeled in video games, physics engines and their basic functions will be incorporated into a standard library similar to how 3D graphics and sound were incorporated into Microsoft's DirectX development library.

While the basic physics tools become standardized, the door will open for more specialized physics tools to become incorporated into game design. Although Havok currently offers a more robust suite of tools to help game developers manage artificial intelligence, path finding and environmental destruction; PhysX is quickly maturing and offering a number of middleware solutions beyond the basic rigid/soft body dynamics, vehicle movement and volumetric options they currently offer. This, along with strategic partnerships with Nintendo, Sony, Microsoft and other major players in the game industry, lead us to believe that Nvidia's PhysX will eventually become the standard physics middleware suite. This does not mean that other physics engines will necessarily disappear. Actually, the market will expand to solve very specific physics problems that are not covered in the generic library - similar to light and texture shaders that are developed on top of the standard DirectX library.

The long-term projections for game design and physics are exciting, as game design has seen little innovation since 3D gaming was introduced in the mid-1990s. Most games are still based on getting from one physical space to another by navigating puzzles, mazes and obstacles. However, if players can fully interact with and destroy the environment, maps and mazes are no

longer necessary. Game developers will have to account for this and come up with new ways to direct the player toward objectives. Some games are experimenting with moral problems that guide players through games, but with few exceptions it seems merely added on as opposed to intrinsically linked to good design and game play.

The impending changes in game design will ultimately reflect advances made in physics technology, similar to the boon in first-person shooters following the successful release of *Quake's* 3D gaming. Completely open worlds with interactive and adjustable objects will continue to evolve until game worlds fully represent a physically relatable world.

### *Conclusion*

Advancing physics in games is the most groundbreaking technology to affect game play and design since 3D. Although right now the market is full of companies trying to determine the best way to solve the problems raised by better physics awareness, soon computing power and software standardization will create a new norm that games will have to live up to. Similar to how few people still play traditional two-dimensional platforms since games have shifted to a 3D environment, gamers desires and expectations will adjust to physics improvements.

The history of video games is filled with different game styles and game technologies. As the hardware has improved, software and game design has improved to take advantage of it.

Occasionally, a new hardware or software technology comes along that drastically alters how games are made and modern physics engines will prove to be one of these demarcation points in the history of video games.

### Immediate Future

1. Frostbite enters the game as a new all in one solution by the biggest company in the industry (Digital Illusions Creative Entertainment, a subsidiary of Electronic Arts).

2. Euphoria and Digital Molecular Matter (DMM) continue the trend toward specialization.
3. Open Source is still a fringe player.

### 5 Years

1. Either PhysX or Havok (or some combination of the two) become the winner of the middleware competition. PhysX seems to currently have better market position because of their corporate ties to Sony and Microsoft.
2. Basic physics dynamic libraries are incorporated into DirectX, meaning physics engines cease to do all the physics calculations, just the specialized needs of that specific game.
3. Open Source is still fringe although some large studios adapt and heavily modify open source engines (similar to *Grand Theft Auto*). Smart studios will incorporate support for these engines (similar to DirectX support for OpenGL).

### Long Term

1. Almost all basic physics calculations have become a standard in DirectX.
2. New computer architecture further separates the CPU from the GPU making physics an afterthought – hardware limitations no longer limit physics calculations.
3. Graphics and game play mechanics are no longer “wow factors” in purchasing a game, much like 3D is no longer a “wow factor” in buying a game today. It will be expected.
4. Physics and game engines will merely be tools to a larger medium – similar to film editing software. The tools and processes are different, but the end result in many cases is indistinguishable from one program and the next.

## References

- Atwood, J. (2008, June 16). *Physics Based Games*. Retrieved March 5, 2010, from Coding Horror: <http://www.codinghorror.com/blog/2008/06/physics-based-games.html>
- Barton M., & Loguidice B. (June 10, 2009). *The history of Spacewar!: The best waste of time in the history of the universe*. Retrieved 02/17/2010:  
[http://www.gamasutra.com/view/feature/4047/the\\_history\\_of\\_spacewar\\_the\\_best\\_.php](http://www.gamasutra.com/view/feature/4047/the_history_of_spacewar_the_best_.php)
- Battlezone (n.d.) Retrieved 02/17/09 from Classic.1up.com's Essential 50 List:  
<http://www.1up.com/do/feature?cId=3133873>
- Beckman, B. (2007, June 8). *The Physics in Games - Real-Time Simulation Explained*. Retrieved March 5, 2010, from Channel 9: <http://channel9.msdn.com/posts/Charles/Brian-Beckman-The-Physics-in-Games-Real-Time-Simulation-Explained/>
- Bolton, D. (October 22, 2006). *What is a physics engine?* Retrieved 03/09/2010:  
<http://cplus.about.com/b/2006/10/22/what-is-a-physics-engine.htm>
- Bullet (Software)*. (2010, February 28). Retrieved March 2010, 5, from Wikipedia:  
[http://en.wikipedia.org/wiki/Bullet\\_%28software%29](http://en.wikipedia.org/wiki/Bullet_%28software%29)

Cabral, Matt (January, 2010). *The Saboteur: Sean Devlin sees pandemic out on a high note*: pg. 61-61.

Cheng, Godfrey (July, 2006). *Effects physics and gameplay physics explored*. Interview by Brent Justice, Chris Seitz, and Jeff Yates. Retrieved 02/26/2010:

[http://enthusiast.hardocp.com/article/2006/07/10/effects\\_physics\\_gameplay\\_explored/2](http://enthusiast.hardocp.com/article/2006/07/10/effects_physics_gameplay_explored/2)

Egenfeldt, S. N., Smith, J. H., & Tosca, S. P. (2008). *Understanding video games: The essential introduction*. New York, NY: Taylor & Francis.

Evans, S. (2009, November 5). *Building An Analytical Physics Engine - Pt.1*. Retrieved March 5, 2010, from Gamasutra:

[http://www.gamasutra.com/blogs/StuartEvans/20091105/3492/Building\\_An\\_Analytical\\_Physics\\_Engine\\_\\_Pt1.php](http://www.gamasutra.com/blogs/StuartEvans/20091105/3492/Building_An_Analytical_Physics_Engine__Pt1.php)

Gabb, Henry, and Adam Lake (November 17, 2005). *Threading 3D game engine basics*.

Retrieved from *Gamasutra* 02/27/2010:

[http://www.gamasutra.com/features/20051117/gabb\\_01.shtml#](http://www.gamasutra.com/features/20051117/gabb_01.shtml#)

Game Engine. (February, 2010). In *Wikipedia, The Free Encyclopedia*. Retrieved 02/26/2010:

[http://en.wikipedia.org/w/index.php?title=Game\\_engine&oldid=344187337](http://en.wikipedia.org/w/index.php?title=Game_engine&oldid=344187337)

*Havok Physics*. (n.d.). Retrieved March 5, 2010, from Havok:

<http://www.havok.com/index.php?page=havok-physics>

Holz Korn, P. (2008, February 10). *Physics Simulation in Games*. Retrieved March 5, 2010, from

<http://stud3.tuwien.ac.at/~e0426262/pf/portf/psig.pdf>

Ishiguro, Hiroshi. "Scientific Issues Concerning Androids." *The International Journal of*

*Robotics Research* 26.1 (2007): 105-17. Print.

Karsemeyer, J. & Henry, C. (May, 2008). *Physics in mass market games*. Retrieved 02/18/2010:

[http://www.gamecareerguide.com/features/534/features/534/physics\\_in\\_mass\\_market\\_.php](http://www.gamecareerguide.com/features/534/features/534/physics_in_mass_market_.php).

Karsemeyer, J., & Henry, C. (2008, March 6). *Playing Dead: Physics in Pop Games*. Retrieved March 5, 2010, from Half-Life Havoc: <http://www.hlhmod.com/>

Kent, Steven L. (2001). *The ultimate history of video games*. Roseville, CA: Prima Publishing.

Lilly, P. (2009, July 21). *Doom to Dunia: A Visual History of 3D Game Engines*. Retrieved February 14, 2010, from Maximum PC:  
[http://www.maximumpc.com/article/features/3d\\_game\\_engines](http://www.maximumpc.com/article/features/3d_game_engines)

Lilly, P. (2009, May 19). *From Voodoo to GeForce: The Awesome History of 3D Graphics*. Retrieved February 14, 2010, from Maximum PC:  
[http://www.maximumpc.com/article/features/graphics\\_extravaganza\\_ultimate\\_gpu\\_retrospective](http://www.maximumpc.com/article/features/graphics_extravaganza_ultimate_gpu_retrospective)

Luban, P. (2007, December 4). *Physics in Games: A New Gameplay Frontier*. Retrieved March 5, 2010, from Gamasutra:  
[http://www.gamasutra.com/view/feature/2798/physics\\_in\\_games\\_a\\_new\\_gameplay\\_.php](http://www.gamasutra.com/view/feature/2798/physics_in_games_a_new_gameplay_.php)

*NVIDIA PhysX SDK Features*. (2010, January 17). Retrieved March 5, 2010, from nVidia Developers Zone: [http://developer.nvidia.com/object/physx\\_features.html](http://developer.nvidia.com/object/physx_features.html)

Nosowitz, D (November 8, 2008). *Retromodo: Tennis for two, the world's first graphical videogame*. Retrieved 02/17/2010: <http://gizmodo.com/5080541/retromodo-tennis-for-two-the-worlds-first-graphical-videogame>

*PC Video Card Market Share in the Last 30 Days*. (2010, March 5). Retrieved March 5, 2010, from Video Card Benchmarks: <http://videocardbenchmark.net/30dayshare.html>

Physics engine. (n.d.). Retrieved 03/17/2010: [http://en.wikipedia.org/wiki/Physics\\_engine](http://en.wikipedia.org/wiki/Physics_engine)

*PhysX*. (2010, March 5). Retrieved March 5, 2010, from Wikipedia.com:

<http://en.wikipedia.org/wiki/PhysX>

Shrout, Ryan, and Leigh Davies (October, 2009). *Who moved the goal posts? The rapidly changing world of CPUs*. Retrieved 02/27/2010:

[http://www.gamasutra.com/view/feature/4168/sponsored\\_feature\\_who\\_moved\\_the\\_.php](http://www.gamasutra.com/view/feature/4168/sponsored_feature_who_moved_the_.php)

Smith, R. (2007, May 28). *ODE*. Retrieved March 5, 2010, from Open Dynamics Engine:

<http://www.ode.org/ode.html>

SpaceWar! (n.d.) Retrieved 02/16/2010 from the Computer History Museum's on-line exhibition of Digital Equipment Corporation's PDP-1 computer: <http://pdp-1.computerhistory.org/pdp-1/?f=theme&s=4&ss=3>

Stuart, Keith (June, 2009). *The truth about game physics: Part one*. Retrieved 02/17/2010:

<http://www.guardian.co.uk/technology/gamesblog/2009/jun/22/games-physics>

Stuart, Keith (June, 2009). *The truth about game physics: Part five*. Retrieved 02/18/2010:

<http://www.guardian.co.uk/technology/gamesblog/2009/jun/25/games-gameculture1>

Stuart, Keith (June, 2009). *The truth about game physics: Part four*. Retrieved 02/18/2010:

<http://www.guardian.co.uk/technology/gamesblog/2009/jun/25/games-gameculture>

Stuart, Keith (June, 2009). *The truth about game physics: Part three*. Retrieved 02/18/2010:

<http://www.guardian.co.uk/technology/gamesblog/2009/jun/23/games-physics1>

Waveren, J. v. (2009, August 9). *id Tech 5 Challenges - From Texture Virtualization to Massive Parallelization*. Retrieved March 5, 2010, from <http://www.siggraph.org/s2009/>

Winter, D. (n.d.). *Magnavox odyssey: First home video game console*. Retrieved 02/16/2010:

<http://www.pong-story.com/odyssey.htm>.

## Interview with Chris Zimmerman

Below is a copy of an interview conducted for our project (via email) with Chris Zimmerman, Development Director for Sucker Punch Productions, a video game developer based in Bellevue, Washington. Sucker Punch was established in 1997 and its first release, *Rocket: Robot on Wheels*, was considered one of the earliest games to implement a simple physics engine. The company has since released other titles, most notably *inFamous*, a next generation super hero action-adventure, exclusively for PLAYSTATION 3, which has garnered awards and industry recognition including “Best Story of 2009” by IGN Entertainment.

Q: I read a little about Rocket: Robot on Wheels and the recognition received regarding the video game and its physics engine. Can you explain the physics engines used for that game? What elements did the engine add to the game play?

A: We built the physics engine from scratch. It’s a simple rigid-body physics simulator, without lots of the bells and whistles that most commercial physics engines provide these days. We didn’t support real constraints, for instance—the simulation runs on rigid bodies only, so articulated bodies aren’t supported. We did fake support for some simple constraints, so there are hinged platforms, and wheels which are constrained to the parent cars.

The physics engine helped in Rocket in two ways. First, we had puzzles where we used the physics engine directly. For instance, in one of the levels, you collect pieces of candy in various sizes and drop them on a scale. You need to put exactly the right total weight on the scale to solve the puzzle, which means getting the right combination of pieces of candy. The scale actually works like a real scale... there’s a spring force, which restores its position, and the dial spins proportionally to the distance the scale is pushed down.

Second, the physics engine adds a level of physical verisimilitude to everything in the game. Rocket himself moves because of friction between his wheel and the ground. If he stands on

something that isn't a rigid part of the world—say, a floating buoy—then that object reacts to the friction as well, making things feel weighty and realistic.

Q: Was Rocket: Robot on Wheels one of the earlier games to use a physics engine?

A: Yes. There were some other physics-based games that came out around the same time. The PC game *Trespasser* is the best-known example.

Q: What do you currently use for your physics in the games you develop?

A: We're still using our own internal engine, an outgrowth of the first Rocket physics engine.

Q: How important have physics engines become for the video game industry?

A: They're pretty common in games these days, but I wouldn't say they're crucial. Most of the bigger games in our category (third-person action) have physics integrated to one degree or another. Typically it's used only for certain things: exploding boxes, tumbling cars, maybe a rag-doll version of a character when they're hurt.

In some genres, a custom physics engine is more common than a generic, off-the-shelf one. For instance, there's an underlying physics model in every racing game that tries to incorporate the car's behavior as it drives around the track.

Q: How has the demand for physics in video games affected your company?

A: It hasn't, really. If there's a "demand for physics in video games", it ranks much lower than the demand for great graphics and sound, interesting and appealing characters, beautiful environments, a compelling story, and so on. Physics is neat, but it's a secondary feature for most of our audience.

Q: Do you use a physics engine that is developed in-house now? If so, can you tell us what aspects of game play it focuses on and why? If not, what do you use and why?

A: We're still using our own engine. We've moved away from using it for everything—for instance, we don't use it to move our characters around anymore. We still use it to detect collisions between objects, for handling projectiles, for tumbling cars, and so on.

Q: When do you think physics became such a big part of game development and why?

A: It happened pretty slowly. The biggest change was the availability of competent third-party physics solutions, like the ones from Havok and Ageia. They're pretty easy to integrate (second-hand knowledge), and this made it easier for people to add physically simulated objects to their games.

Q: Where do you see physics engines in the next five to ten years? How may these potential developments in physics engines affect the industry?

A: The feature set of physics engines has been pretty static for the last few years, so I doubt that it they'll change much in the next five to ten years. Companies like Havok are having a hard time adding more features to the basic physic engine, since they've solved that problem pretty well. Instead, they've been growing outwards from a physics core, adding things like animation systems to their products.

The thing that is changing is that hardware gets faster every year. With more CPU power, it's easier to add physical simulation to more things.

Q: What are the advantages for software processed physics engines? Hardware processed engines? Which do you prefer?

A: By hardware-processed engine, I assume you mean things like PhysX, where there's special-purpose hardware to support the simulation. In short, PhysX was a bad idea. There was never enough pull from customers to make this a viable market. Customers didn't care enough about physical simulation to buy the boards, and the improvements over software-based solutions weren't obvious enough anyhow. Ageia drew a false parallel with the graphics board market—the parallel was false because customers really cared about graphics, and the improvements over software renderers were massive.