The pleasures of immersive interactive entertainment increasingly demand game designers deliver a physically consistent and convincing experience—solving fundamental computational problems along the way.

Physics in Computer Games

Chris Hecker

I don't know about you, but when I catch air in my Ferrari 312 racing down the straightaway at Monaco, I look forward to the gyroscopic effects of the wheels and engine rotating the car in mid-flight. I especially enjoy it when I crash into a wall or another car, flip over, and the wreckage—not to mention my standing in the 1967 Grand Prix circuit—careens down the track, tumbling and bouncing as it goes. In contrast, I don't like it at all when I can't stack up a bunch of crates to get over the wall of the castle in which I'm being held captive. Or, worse, it really bugs me when I can't even move the crates, because they're stuck fast to the floor through no means I can discern.

In the virtual world of today's increasingly immersive and addictive computer games, the physics of the environment—the very rules governing how the entities in the game interact—is subject to the whim of a game designer, the skills of a team of game programmers, and the speed of the CPU on which the game is running. As these skills and CPU speeds improve, and as player expectations increase, game "physics simulators" have to keep pace by becoming increasingly realistic and advanced. Therefore, game developers are turning to serious algorithms from the computer science research community to bring their virtual worlds to ever higher levels of sophistication. Here, I survey the use of physics and

SCREENSHOT FROM PAPYRUS DESIGN GROUP'S "GRAND PRIX LEGENDS"

its role in the multibillion-dollar interactive entertainment industry, then prognosticate on where game physics is likely to go in the near and not-so-near future.

Physical Simulation in Games

Every computer game involving moving objects, from the simplest puzzle games like the classic Tetris, even checkers, to the most incredibly realistic auto racing or flight simulation, depends on a physics simulator. Each of them needs some core algorithm to determine the rules under which the objects in its game world move around. Is a falling Tetris shape blocked on one side by another piece? Can the checkers piece move to the left, or does it bump up against the edge of the board? Can the wheels of my race car exert enough force on the ground to accelerate the car without slipping? What happens when I adjust the trim on my ailerons? These queries and lots more are answered by the physics simulator, to varying degrees of sophistication, and the results are displayed to the user.

Although the physics simulator in Tetris or checkers would not be recognized as such by most academic simulation researchers, it contains all the necessary attributes: It takes inputs and current positions of objects in the environment; computes the movement of the objects based on certain rules; and updates the object positions when all interactions are resolved. Then it repeats. It doesn't solve differential equations of motion or use F=ma but does produce motions consistent with the rules of the virtual world (if you'll allow me to call the checkers board a virtual world).

The complexity of the rules of motion and the ways interactions are resolved are the only things differentiating the Tetris simulator from its counterpart in the Grand Prix car racing game. The most important question for a game developer to ask is whether the physics simulator in a particular game is a good match for the design. Does it produce the motions the player expects and needs in order to have fun playing the game? Under this consistency metric, the Tetris simulator is flawless.

Unfortunately, most current game simulators score miserably under this metric. Even the best of the carefully tuned racing and flying games contain "holes" in their environments, where the game's physical laws act in nonintuitive ways, confounding and frustrating the player.

The problem is that when game designers try to create a virtual world more complex and subtle than Tetris, the limitations of the hardware, software, and their own brainpower quickly come into play. No one knows how to write simulation algorithms that simulate the real world, or even the slightly cartoon-like world of Indiana Jones. And even if developers did know how to create these algorithms, it's unclear how much processing power they would consume if the world were filled with hundreds of objects. So they simplify, then simplify some more. Eventually, however, they will come up with algorithms approximating the minimum number of physical rules they decide are necessary to play their particular games and hope they haven't introduced any gaping holes and inconsistencies in the environment players might fall through.

Consistency is crucial for keeping players immersed in game environments; it's okay to be able to run 100 miles per hour and jump 20 feet—as long as the rest of the world reacts appropriately.

Although conventional wisdom in the game industry says the goal of the games is reality itself, especially as infinitely cheap and powerful computers become commonplace, the real goal is consistency. Reality is boring; escaping reality is why people play computer games in the first place. Consistency, on the other hand, is crucial for keeping players immersed in game environments; it's okay to be able to run 100 miles per hour and jump 20 feet—as long as the rest of the world reacts appropriately. In effect, each computer game makes a contract with its players when they start playing; any inconsistency violates this contract, reminding the player "it's just a game." Getting stuck *in* a wall due to bad collision detection is not something a player wants to happen when adventuring in a dungeon. When inconsistencies occur—like when a boom microphone floats into a movie frame during

an emotional scene—the creators of immersive environments have failed.

Anatomy of a Game Simulator

Before exploring further the kinds of physical inconsistencies plaguing today's games and how the industry handles them, I briefly outline the flow structure of a modern game, looking at the various stages constituting a physics simulator.

Computer games run in a loop. They take input from the player, figure out what the non-player-controlled entities in the game are supposed to be doing, then combine this information into a coherent new state. Finally, they render the result. The physics simulator, which is a central part of this loop, consists of four main stages: contact detection; contact resolution; force computation; and state integration. Some researchers might argue with my imprecise decomposition and note that algorithms differ from simulator to simulator, but these stages do the job for this exposition.

Contact detection. Contact detection is the mostly geometric task of figuring out where the objects in an environment are relative to one another. Objects can be in any of a number of relationships, including disjoint, penetrating, resting on each other, and colliding violently. The oft-heard phrase "collision detection" is a subset of contact detection.

Contact resolution. The contact-resolution stage determines what the physics simulator has to do about the collisions and other contacts detected in the contact-detection stage. Are the objects touching now but supposed to be moving apart? Can they do both? Does the simulator need to do something to prevent penetration, or are a pair of objects penetrating one another and thus need to be separated to be consistent?

Force computation. All the forces influencing the objects are calculated during the aptly named force-computation stage. Here, the simulator considers physical forces like the wind resistance on a car, the recoil from a grenade launcher, and tractor beams from enemy spaceships, or mundane forces like gravity pulling objects down to the ground. This stage interacts with the contact-detection stage in some simulators, because one way to prevent the interpenetration of objects is to have them exert forces on one another.

State integration. Finally, the forces in the force-computation stage are applied to the objects, and their states are advanced over time. To do so, the simulator numerically integrates the equations of motion—a highly mathematical process. Basically, state integration answers the question of how far an object moves when a force acts on it. After state integration, the updated positions of the objects are ready to be rendered in their updated positions by the game's graphics subsystem. If these new positions are slightly different from the old positions, and the game loop can execute these four stages fast enough, the objects look to the player as though they really are moving across the screen.

Consistency Problems

If the simulation algorithm is that straightforward, where do the problems come in? Each stage in a game's physics simulator is rife with simplifications, each of which can lead to consistency problems. For example, in order to increase the performance of the contact-detection stage, the game uses simple approximating shapes for the objects, even though such objects may be rendered with more detailed representations. These approximations speed up the geometric queries, even during contact detection. A human character rendered with thousands of polygons might be approximated in the physics simulator as a sphere or a bounding box. This simplification can lead to inconsistencies whereby it visually appears the player can squeeze through an area, though the simplified physical representation makes it impossible to do so. Approximations are also made in the contact-resolution stage. Because it's so difficult for a programmer to compute the correct way to keep a stack of boxes upright, it's sometimes easier for the designer to not let them stack up in the first place. It's also important to be able to "put objects to sleep," so the game doesn't spend time computing physics for objects that aren't important to the game experience at a particular moment, though this too can lead to inconsistencies.

Play any game that seeks to deliver some degree of realism, and you'll find these inconsistencies and worse. It could involve ways of solving problems that simply don't work, because they weren't anticipated by the game designer and hard-coded into the game as solutions.

Some game genres generally provide better physics than others. For example, games simulating specific machines, such as flight simulators or car races, have pretty solid physics simulators delivering a convincingly realistic and consistent experience. These domain-specific games have an easier time than games that try to simulate human beings walking around in cluttered rooms. Simulating a speeding race car may seem more complicated than simulating a walking human, but it's actually much easier, because it's a more defined problem. There are well-known physics equations governing the limited number of ways cars behave in motion; these ways of behaving are called the car's "degrees of freedom." Humans, by contrast, have hundreds, if not thousands, of degrees of free-

dom, and the human brain exerts incredibly subtle control over each one of them.

Physical Control

That last sentence suggests a daunting problem—physical control—facing future developers of game physics. Even though I've complained about the state of affairs of physics simulation, some types of simulator technology are getting close to the point at which they satisfy a game's demand for consistency. I'm confident that specific simulation problems, such as those involving rigid bodies, will be solved theoretically by graphics and physics researchers in the next five years. There may be performance issues, but as long as game designers are careful about the number of objects they allow to interact at a given time, algorithms will work consistently. Rigid bodies will be able to model most of the physical objects games need for at least the next several years, including the usual crates, barrels, tables, chairs, swords, and other inflexible shapes. They'll also be able to model articulated figures, including robots, even humans, by constraining lots of bodies together with joints (see Popović's "Controlling Realistic Character Animation" in this section).

It remains an open question how game designers can tell a dramatic story with its natural twists and turns when everything in the game is controlled by physical laws.

Physical control in both computer game programming and leading-edge research is a completely unsolved problem. "Control" is the work of exerting forces and torques on objects to get them to do what the game designer wants them to do, like chase the player. Physical control is about knowing which forces need to be computed during the force-computation stage. It's fine to have a robust simulator with detailed articulated figures for each character in the game. But which torques does the physics simulator have to put on the joints to make the characters stand up and walk like flesh-and-blood humans? A game populated by rag dolls that just fall over and sit on the floor is no fun.

You might be wondering how game characters manage to move around at all in today's games if physical control is such a difficult problem. Most games today use canned animations to control their characters' movements. They involve a very simple physical model for each character, as in the box and sphere approximations mentioned earlier, and the physical model moves around the world while the character's graphical representation animates. The character's limb movements don't affect the physical simulation in any way at all; such prescripted animations are usually either hand-animated or motion-captured from a human performer.

While hand animation and motion capture are fine for noninteractive entertainment like movies and television, they provide yet another place for inconsistencies to crop up in an interactive environment. Say you have an animation of a character swinging a sword in a hallway. In a movie, the performers can make sure they don't hit their swords against the walls as they swing. If the actors accidentally do hit the wall, the director stops the performance and films another take. In a video game, the sword-swinging animation has to be created before the player has even thought about buying the game. Imagine that a player's character is in the hallway and ready to swing the sword. What if the character is too close to the wall because of where the player happened to place him before the swing? The player hits the swing-the-sword button and the character's sword flies toward the wall. At this point, one of two things usually happens: The sword goes right through the wall as if it's not there, or the sword touches the wall and the animation stops prematurely. Neither of these options is convincing, consistent, or immersive, but when the movements are all prescripted, the game developer rarely has another choice.

There are many kludgey ways of partially fixing this problem, but physical control is the only general solution. When a physically controlled character swings his or her sword and hits the wall, the sword collides with the wall, and the impact force ripples back up the character's arm. It looks consistent, and the world feels solid and real. The player's immersion is not compromised.

A lot of work remains to be done—in the game industry and in the research community—before physical controllers can be leveraged effectively in games. Game developers usually draw from research papers as soon as computers are fast enough to run the published algorithms in real time. For physical controllers, however, the researchers don't yet know how to make humans walk convincingly—let alone run down a flight of stairs while dodging bullets. Physical control is a very difficult problem—too difficult for game developers to solve in the midst of a product cycle.

They need to rely on control theory and robotics researchers to solve this problem first. In the early 1990s, these researchers focused on the field of physical control for locomotion. Today, however, fewer of them seem to still be working on this problem, although physical control is more important than ever, as interactive entertainment becomes an increasingly popular medium. While some of them continue in this area, I implore others to get into it and help advance the state of the art.

The Future of Games

Over the next few years, I expect most video games to completely embrace physical control and simulation. Interactivity—the key factor differentiating games from other forms of entertainment—will demand it. The interactivity and choice of what to do and where to go that players will increasingly have in games means the computer has to be able to generate consistent results from an infinite variety of inputs. There's simply no way to prescript or anticipate all the options. Because game developers cannot know ahead of time how close to a wall players will be when swinging their swords, the resulting movement has to be generated algorithmically.

As you can probably tell from what I've written here, the best algorithms for generating movement are those that obey a consistent set of physical laws. These algorithms allow for players' physical intuition to step in and help them stay immersed, sparing them unpleasant surprises. However, it remains an open question how game designers can tell a dramatic story with its natural twists and turns when everything in the game is controlled by physical laws. Every time a physical algorithm is added to replace a prescripted sequence, the game designer loses some artistic control. Game design as an art form will have to evolve to handle this loss of control, just like movie directing evolved to handle movable cameras and other cinematic innovations.

Many hurdles have to be overcome before games convincingly simulate physical environments and human beings. In the near term, ever-pragmatic game developers will use bits and pieces of physical simulation to add realism and consistency to their worlds, without messing up popular and proven game designs. Explosions, objects tumbling, and other easy-to-simulate effects will continue to raise the quality of computer games until the day game characters are completely interactive and physically reactive to their worlds. C

Chris Hecker (checker@d6.com) is technical and art director of Definition Six, Inc., Oakland, Calif., and editor-at-large of *Game Developer Magazine*.