How physics is used in video games

David M Bourg

David M Bourg & Associates, LLC, Kenner, LA 70065, USA

E-mail: david@davidbourg.com

Abstract

Modern video games use physics to achieve realistic behaviour and special effects. Everything from billiard balls, to flying debris, to tactical fighter jets is simulated in games using fundamental principles of dynamics. This article explores several examples of how physics is used in games. Further, this article describes some of the more important technical details of how physics is actually incorporated in games.

Immersiveness

I was 16 years old driving my white Escort up a winding road along the Mississippi River on my way to work in the local mall. It was hot and drizzling—typical Southern Louisiana weather—but my window was rolled down since I had no air conditioner. The road wasn't too busy, and I was speeding with "I'll stop the world and melt with you" blaring on my stereo.

I couldn't carry a tune but was singing my lungs out as raindrops dotted the left side of my face. I was day-dreaming about getting off from work and hitting Bourbon Street. It was Saturday and the French Quarter was sure to be hopping.

As I started into a sharp bend in the road, I suddenly lost control. With the back end of my car swerving out sideways I was headed towards a canal. My heart was pounding. There was nothing I could do to stop.

CRASH!

Time slowed to a crawl as I watched the horizon through my windshield spectacularly rotate and then invert. I didn't even feel the impact. It seemed like forever, but mere seconds later I found myself on the roof of my car buried under my seats and piles of debris—spare change, books, floor mats and trash. I was turned around, upside-down in the canal.

At near panic speed, I scurried out of the window and ran 50 yards in any direction certain that my car was about to explode. Surprisingly, a police officer was already on the scene; he was on his way to the mall too when he saw the whole thing. First thing he asks me: "can you crawl back in there and turn that music off please…".

I am sure this is a rather unusual beginning to a technical article, but I am telling you this story to illustrate a point. My story is complete with visual and emotional imagery as well as a realistic description of the crash itself. Now, what if I ended the story with something like "…and my car bounced 15 metres into the air, landed upright and kept going?" It would spoil the story wouldn't it? Video games are no different. To paraphrase Steven Collins, CEO of Havok: It's all about immersiveness.

Game developers tell their stories using stunning graphics, mood-setting music, rich sound effects and realistic behaviour. The idea is to immerse you in visual, emotional and physical realism—to bring you into the game without jolting you back to reality with a ridiculous special effect or quirky stunt.

So how do game developers create such physical realism? They borrow knowledge and techniques from other fields, in this case physics. More specifically, the techniques used to realistically simulate car crashes belong to the subject of classical dynamics.

Here's how it's done. While your virtual car is racing down the highway, the game's so-called physics engine constantly tracks the acceleration, velocity and position of the car. At the same time, the physics engine is also constantly checking the car to see if it has run into anything else in the game world, for example another car, a road sign or even a telegraph pole. When a collision is detected, the physics engine kicks into overdrive.

Each bounce, flip and roll is precisely calculated by the physics engine to yield a smooth, realistic crash event. When done properly, the results are impressive—the horizon jostles, rotates, even inverts as your car goes tumbling through the air just like it would in the real world. I should know.

These calculations are based on the application of fundamental impulse and momentum principles along with a healthy dose of mathematics. At the instant of impact an impulse force is calculated and applied to the car, and whatever it ran into, at the points of impact. This impulse force is a function of the relative velocity between the car and whatever it ran into—the higher the relative velocity the higher the impulse. Once this impulse is calculated, the car's acceleration, velocity, position and orientation are recalculated thousands of times to step the crash event through time.

The game's physics engine is tied to its graphics engine so that you can see the results in all their glory. The beautifully rendered 3D scenes combined with realistic physical behaviour yield a truly immersive experience second only to the real thing, but a whole lot safer and a heck of a lot more fun.

So, when your lead-footed, speed-hungry students are breaking all known virtual traffic laws in their favourite driving game, they might be getting a good lesson in the laws of physics. With that in mind, the remainder of this article explores some of the technical aspects of how physics is used in modern video games.

'Real' physics examples

So-called 'real' physics is used in video games to achieve a wide variety of realistic effects. Some applications of real physics are almost obvious, while other applications may surprise you.

Flight simulations

Flight simulations are probably one of the first game genres that come to mind when you think of how real physics can be used in games. These games use real-world physics to simulate the lift and drag on the aircraft's components and control surfaces in an effort to mimic the real plane's capabilities and handling characteristics. Some flight simulation games strive for ultra realism where the details of the flight model are meticulously tuned so as to be as realistic as possible. Other games aren't so concerned about realistically modelling any particular aircraft and instead aim to yield a simple, enjoyable virtual flight experience. Such a game could be one where the plane is a paper airplane that you fly around the interior of a house.

Billiards

A game like billiards is another good example of a game where real physics is crucial to the playability and believability of the game. Billiards simulations use real physics to model the striking of the cue ball and its impact with other balls on the table. The physics does not stop there. Realistic friction between impacting balls and rolling resistance are critical in order to properly account for spin. Without properly handling spin and rolling, shots that use 'side' (or 'english' as it is known in north America) would not be achievable.

Projectiles

Projectiles such as sports balls, grenades, cannonballs, bullets, rocks or anything else that can be shot, launched or thrown in a video game use realistic ballistics or projectile motion.

Figure 1. Examples of projectile motion.

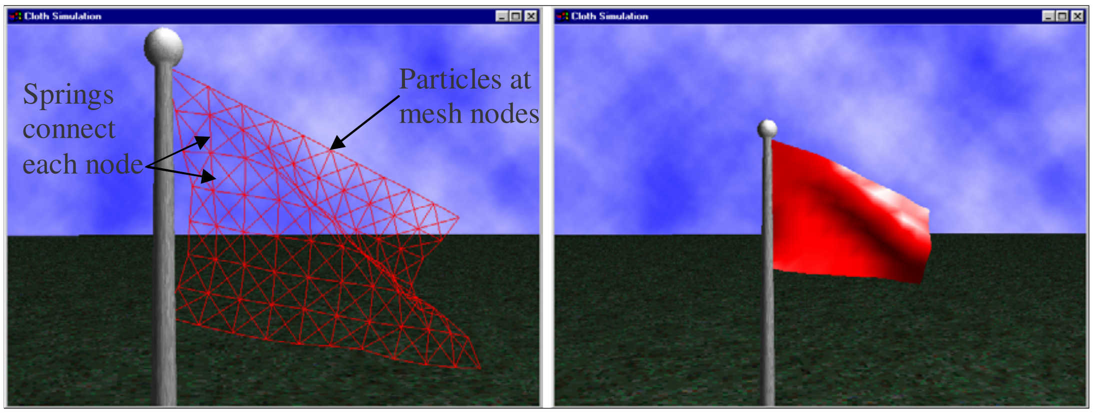


Figure 2. Cloth simulations.

Projectile motion isn't limited to these more traditional ballistic scenarios. Figure 1 shows a few other examples of how projectile motion can be used to achieve some interesting special effects.

In this adventure game, the fires shown in the fireplace and under the large kettle are simulated using buoyant particles that rise and change colour. The bubbling material shooting up from the kettle consists of a collection of particles that are launched upward at some velocity and fall back down under the influence of gravity. The Beholder beast in the upper right of the figure is casting a fireball spell that is essentially another collection of particles launched towards the player and falling under the influence of gravity.

Cloth

Collections of particles and springs based on the well known Hooke's law are used to simulate complex motion of non-rigid materials such as cloth. Figure 2 illustrates a network of particles and springs that form the basis for a waving flag simulation. Such a flag may appear on top of a castle in an adventure game. Or perhaps the robes of a wizard character would be modelled in this way to achieve a realistic flowing motion as the wizard ran or waved his arms while casting a spell.

Artificial intelligence

Perhaps an even more unexpected use of physics in games is in the area of artificial intelligence (AI). A good example of how physics can be applied for game AI is in potential-based movement. Potential functions, like the well known Lennard-Jones intermolecular potential function, are used to model repulsive and attractive forces between game agents. The potential function produces both attractive and repulsive forces depending on the proximity of the two molecules being acted upon. It is this ability to represent both attractive and repulsive forces that makes this function useful for game AI.

These forces can be used to cause one agent to chase or evade another, resulting in seemingly intelligent behaviour by application of a few simple calculations. Potential functions are also used to allow game agents to avoid obstacles in their path. Here the obstacle would repel the approaching game agent such that the agent would essentially steer around the object. Potential functions can even be applied to collections of moving agents to achieve a swarming effect.

Other examples

These are just a few examples of how physics is applied in video games. Other examples include simulating the motion of barrels or boxes being pushed along the floor in a first-person action game; simulating the physics of skateboard stunts such as vertical jumps and Ollies (raising all four wheels of the board off the ground); and simulating the flight dynamics of sports balls such as baseballs in order to achieve curve balls. Clearly the use of real physics in games is quite diverse. I should point out that while the aim of incorporating real physics is very often to achieve realistic behaviour, it isn't always the first priority.

Ultimately games must be fun and sometimes realistic behaviour just isn't as fun. Take Tony Hawk's Pro Skater game series as an example. In this skateboard game players can perform the usual skateboarding stunts with realistic-looking results. However, it's far more fun to be able to Ollie over a bus than a roadside kerb. Likewise, it's much more exciting to make a 18 m vertical jump as opposed to a more realistic 3 m jump. The developers realized this and so they built-in a 'moon gravity' option that allows players to perform their tricks in a gravity field that is about one-sixth that of the Earth's. Clearly, this isn't realistic since the game is set on Earth, but it is a whole lot of fun!

Newton's laws

While the examples discussed so far all aim to achieve different effects, they all share one thing in common, which is that they all rely on Newton's laws of motion. All of the effects mentioned earlier can be broken down into interacting masses, applied forces and torques, and resulting accelerations. This is how game developers are able to apply the equations of motion based on Newton's laws to model the motion of particles and rigid bodies representing various game elements.

Let's take a closer look at the equations of motion:

$$\sum \boldsymbol{F} = m\boldsymbol{a}$$
$$\sum \boldsymbol{M} = \mathbf{I}\boldsymbol{\alpha}.$$

The first equation shown here is the familiar $\boldsymbol{F} = m\boldsymbol{a}$ equation for linear motion[1]. For particles, this equation is all that is required. However, for rigid bodies that can rotate as well as translate you must also consider the angular equation of motion.

Clearly, dealing with particles is far easier than dealing with rigid bodies—you only have mass and translation to deal with. Particles are great for modelling things such as bullets or grenades. They're also quite useful for modelling more complicated things such as fire, smoke and cloth among others. For other objects such as tables, chairs, boxes, airplanes, cars or hovercraft, rotation is very important for realism so rigid bodies are a better choice.

Rarely is the angular equation of motion as shown earlier ever applied in that form for rigid-body motion in games. The reason for this is that the inertia tensor, $\mathbf{I}$, is expressed in global, Earth-fixed coordinates in that equation. Thus, the inertia tensor would have to be recomputed as the object moved. This computation is too costly for real-time video games, so a different form of the angular equation of motion is used instead:

$$\sum \boldsymbol{M} = \mathbf{I}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} + \left(\boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega})\right).$$

In this case, the inertia tensor is expressed in body-fixed coordinates and need only be calculated once for the object (unless it changes mass or shape at some point). The $\boldsymbol{\omega}$ term in this equation is the angular velocity of the object in terms of body-fixed coordinates. The $\mathrm{d}\boldsymbol{\omega}/\mathrm{d}t$ term is the time rate of change in angular velocity, that is, its angular acceleration.

With these equations of motion in hand, game developers must tally the mass properties—mass, centre of mass, mass moment of inertia—of the simulated game elements and then compute all of the forces and torques acting on them in order to solve the equations of motion. Ultimately, developers need to know the new positions of the moving objects so the next frame can be drawn.

Solving the equations of motion

Solving the equations of motion relies on fundamental kinematic relationships between acceleration, velocity and displacement. These are summarized in figure 3.

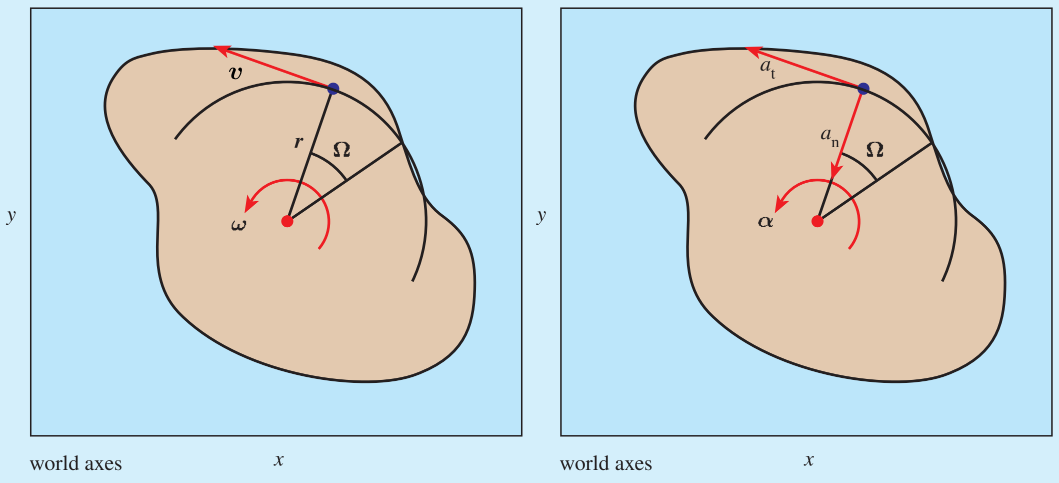
These relations provide the means to work backward from instantaneous accelerations to velocities to changes in position. This approach is in fact how developers simulate object motion in games—they work backward from acceleration to changes in position and rotation for each object. Accelerations are obtained by solving the equations of motion.

The steps in this process are illustrated in figure 4 for the linear equation of motion; the steps are similar for the angular equation of motion.

Notice that these steps discretize the time step from an infinitesimally small $\mathrm{d}t$ to something more finite, though still small. This is an important point. The equations of motion are not solved analytically, instead they are solved numerically and the procedure described here relies on numerical integration techniques.

If you were to implement the procedure for numerical integration exactly as developed in

[1] Variables shown in bold are vector quantities while those shown in normal type are scalars. The exception to this rule is the bold variable $\mathbf{I}$, representing mass moment of inertia, which in 3D is a second-rank tensor—a 3×3 matrix.



$$\boldsymbol{v} = \mathrm{d}\boldsymbol{s}/\mathrm{d}t, \quad \text{and} \quad \boldsymbol{v}\,\mathrm{d}t = \mathrm{d}\boldsymbol{s} \Rightarrow \Delta\boldsymbol{s} = \int \boldsymbol{v}\,\mathrm{d}t$$

$$\boldsymbol{a} = \mathrm{d}\boldsymbol{v}/\mathrm{d}t, \quad \text{and} \quad \boldsymbol{a}\,\mathrm{d}t = \mathrm{d}\boldsymbol{v} \Rightarrow \Delta\boldsymbol{v} = \int \boldsymbol{a}\,\mathrm{d}t$$

$$\omega = \mathrm{d}\boldsymbol{\Omega}/\mathrm{d}t, \quad \text{and} \quad \omega\,\mathrm{d}t = \mathrm{d}\boldsymbol{\Omega} \Rightarrow \Delta\boldsymbol{\Omega} = \int \omega\,\mathrm{d}t$$

$$\alpha = \mathrm{d}\omega/\mathrm{d}t, \quad \text{and} \quad \alpha\,\mathrm{d}t = \mathrm{d}\omega \Rightarrow \Delta\omega = \int \alpha\,\mathrm{d}t$$

Figure 3. Kinematic relations.

Equation of motion:	$\boldsymbol{F} = m\boldsymbol{a}$
Rewrite it as:	$\mathrm{d}\boldsymbol{v} = (\boldsymbol{F}/m)\mathrm{d}t$
Use discrete time step:	$\Delta\boldsymbol{v} = (\boldsymbol{F}/m)\Delta t$
Estimate new velocity:	$\boldsymbol{v}_{t+\Delta t} = \boldsymbol{v}_t + (\boldsymbol{F}/m)\Delta t$
Estimate new position:	$\boldsymbol{s}_{t+\Delta t} = \boldsymbol{s}_t + \boldsymbol{v}_{t+\Delta t}\Delta t$.

Figure 4. Basic steps to solve the linear equation of motion.

the previous discussion you would end up with what is known as Euler's method for numerical integration. This would be a rather casual development of Euler's method, albeit relatively easy to grasp. A more rigorous development of Euler's method would consider Taylor's theorem. Taylor's theorem provides a way to predict the value of some function at some point, knowing only the value of the function at another point and something about the derivatives of the function at that point.

Without going into too many details, the Taylor series expansion of a function is an infinite series consisting of terms involving successively higher order derivatives of the function whose value you're trying to find. Euler's method involves taking the partial sum of the series only to the terms including the first derivative of the function. This corresponds to the process I described earlier.

Euler's method is conceptually very important but it is not the most accurate or stable numerical integration method. In practice, using Euler's method usually requires taking very small step sizes (Δt). This results in more computations, which is not good for real-time video games.

There are other methods that give more stable and more accurate results. There are far too many to consider all their details here; however, they all have in common the aim of improving accuracy, stability and efficiency by allowing larger step sizes. You can refer to any good textbook on numerical methods for examples of a variety of methods and the strategies they employ.

Ultimately, game developers must choose a numerical integration scheme that gives them stable and efficient performance for their particular application. Stability is crucial since it would be very difficult to recover gracefully from a divergent solution in the midst of gameplay, not to mention the subsequent loss of realism that would surely ruin the immersive experience for the gamer. Efficiency is crucial because real-time games operate at relatively high frame rates required for smooth animation.

Summary

In practice, solving the equations of motion for game objects is a great deal more involved that what I've summarized in this article. For each object being simulated, developers must first select or calculate appropriate mass properties. Further, depending on the object being simulated, for example an airplane or a race car, a great deal of time is spent developing appropriate models for all of the forces and torques that will act on those objects during the simulation. These are crucial considerations since the forces and torques essentially define how the object will behave.

If collisions are to be modelled, then developers must also prepare accurate geometric models and collision detection algorithms to enable them to detect when objects collide and extract appropriate data on the collision. Dealing with collisions adds a great deal of complexity to simulating objects in games above and beyond that already inherent in solving the equations of motion as described earlier.

Finally, once all of these models and algorithms are developed, game developers spend a good bit of time tuning their simulation to make sure it performs satisfactorily. This step is not looked upon lightly as it involves a great deal of trial and error to yield stable, efficient, believable and, most importantly, fun results.

Received 8 July 2004
PII: S0031-9120(04)83062-4
doi:10.1088/0031-9120/39/5/002

Further reading

Bourg D M 2002 *Physics for Game Developers* (Sebastopol, CA: O'Reilly) www.ora.com
Bourg D M 2002 *Five Steps to Adding Physics Based Realism to Your Games* (Sebastopol, CA: O'Reilly) www.ora.com
Bourg D M 2003 *Physics for Game Developers Course Notes* (Game Institute) www.gameinstitute.com
Bourg D M and Seemann G 2004 *AI for Game Developers* (Sebastopol, CA: O'Reilly) www.ora.com

David Bourg performs computer simulations and develops analysis tools that measure such things as hovercraft performance and the effect of waves on the motion of ships. He teaches at the University of New Orleans School of Naval Architecture and Marine Engineering, and also teaches the online course 'Physics for Game Developers' for the Game Institute.