The physics of Angry Birds

Angry Birds has a great physics engine, says **Julian Bucknall**, but how does it work?

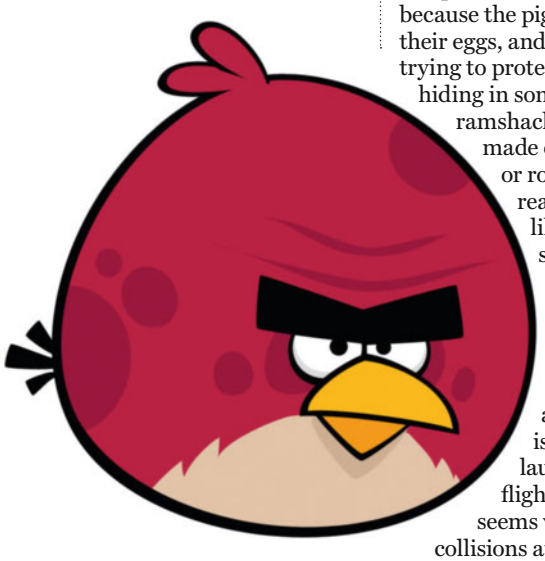
What's covered

SIMULATION OF PHYSICAL LAWS

It seems a bit daft to talk of realism when discussing Angry Birds. After all, it's about trying to destroy a bunch of pigs cowering in rickety edifices by launching flightless birds at them from a catapult. Nevertheless, if you look closely, you can see the game elements obey standard physical laws of gravity, momentum, springs, and collisions. Newton is hiding in the game.

I'm sure, without a doubt, that you know the game Angry Birds. I'm equally sure that you've played it – it's available for purchase on innumerable device platforms, and even for free on Google Chrome and Android, albeit with annoying ads. And I'm pretty certain you're better at it than I am.

Just in case, the basic plot of the game is to knock out a bunch of green pigs by firing flightless birds at them from a catapult. The birds are angry because the pigs have stolen their eggs, and the pigs are trying to protect themselves by hiding in some remarkably ramshackle structures made of wood, glass, or rock. And that's it really. Put baldly like that it doesn't seem like much, but in reality it's quite addictive.



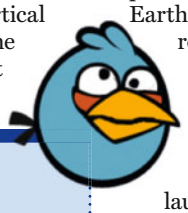
I think that part of the reason for its addictiveness is that the launching and the flight of the birds seems very natural, the collisions authentic, and the wobbliness of the pigs' constructions real. In other words, the game jives with our knowledge of how the real world works, and so we use our experience of throwing balls and stones to work out how the birds will fly as they're launched from the catapult. We make judgments about how the momentum of the red bird will be transferred to the planks and sheets of glass from our familiarity with colliding snooker balls.

Velocity

So how did the developers at Roxio, the company that produces Angry Birds, write such realistic interactions into the game? It all boils down to some fairly simple algorithms.

Let's first take a look at the flight of the red bird (to recap, the red bird acts like a cannon ball: it flies and then crashes into something). We'll take it from the point the bird leaves the catapult: it has a certain velocity at a certain angle. What happens next?

There are two components to the velocity: the vertical and the horizontal. The horizontal component is constant; there are no forces acting on the red bird horizontally (I'm sure that Roxio's programmers ignored the friction due to air resistance). The vertical component on the other hand is subject to the acceleration due to gravity, g, continually pulling the bird down to Earth. Let's assume that g is roughly 10 meters per second squared, as it is at ground level.



If we say that the vertical velocity is v m/s upwards at the point of launch, then a short time later – let's say 1/10 of a second – the vertical velocity will be (v-1) m/s. (In other words, the reduction in velocity is 1/10 of 10 m/s^2.) Another 1/10 of a second later the vertical velocity will be (v-2) m/s, and so on. At some point, gravity will slow down the vertical velocity to zero, after which point, the vertical velocity becomes negative (in other words, the red bird will accelerate downwards – we defined v to be a velocity ▶

3D collisions

In many games, the objects you interact with have two forms. The first is the form you see on the screen, formed from a complex mesh. The other form bounds the first in a simpler shape like a sphere. This form is known as the collision geometry and is used to calculate possible collisions with other objects. Collision detection requires a lot of computation, so the simpler the objects, the faster the computation. The computation has to be done between frames of the resulting game so the frame rate won't suffer during collision detection.

Sometimes the bounding box geometry is only used as a coarse measurement. The more complex mesh is used once the coarse method indicates that a collision could have occurred in order to refine the possibility. ■

Spotlight on... Using games to teach physics

Rhett Allain, associate professor of physics at Southeastern Louisiana University, has used Angry Birds as a pedagogical exercise to help students explore and understand basic physics.

In a fascinating series of articles for Wired, he's probed the physics of the Angry Birds world as it's written, rather than, as I did here, how it *should* be written. To aid in this investigation he used Tracker, a video analysis tool that's designed to help in physics education. By analysing videos of Angry Birds with Tracker, he was able to deduce the physical laws describing the birds' flight.

In another article he discussed the physics of the game's blue bird. After it's fired, the blue bird will, when you tap on the screen, split into three smaller birds. To me there seems to be a speed increase – maybe the three new birds have a jetpack – but Allain shows that, if momentum is conserved, their combined masses are greater than that of the original bird. In the Angry Birds universe, mass can be created from nothing.

I'm sure you can think of lots of other Angry Bird physics that you could test. My favourite would be the white bird. This bird will lay an egg in flight when you tap on the screen. Not only that, but the egg goes straight down and the bird bounces off in another direction. Is momentum conserved in this case? ■

upwards). So, in short, the vertical velocity of the red bird will decrease from v to 0 upwards, and then increase from 0 downwards. The theory of kinematics provides the formula $w = v + at$ for the velocity w at a time t given a constant acceleration a and an initial velocity v.

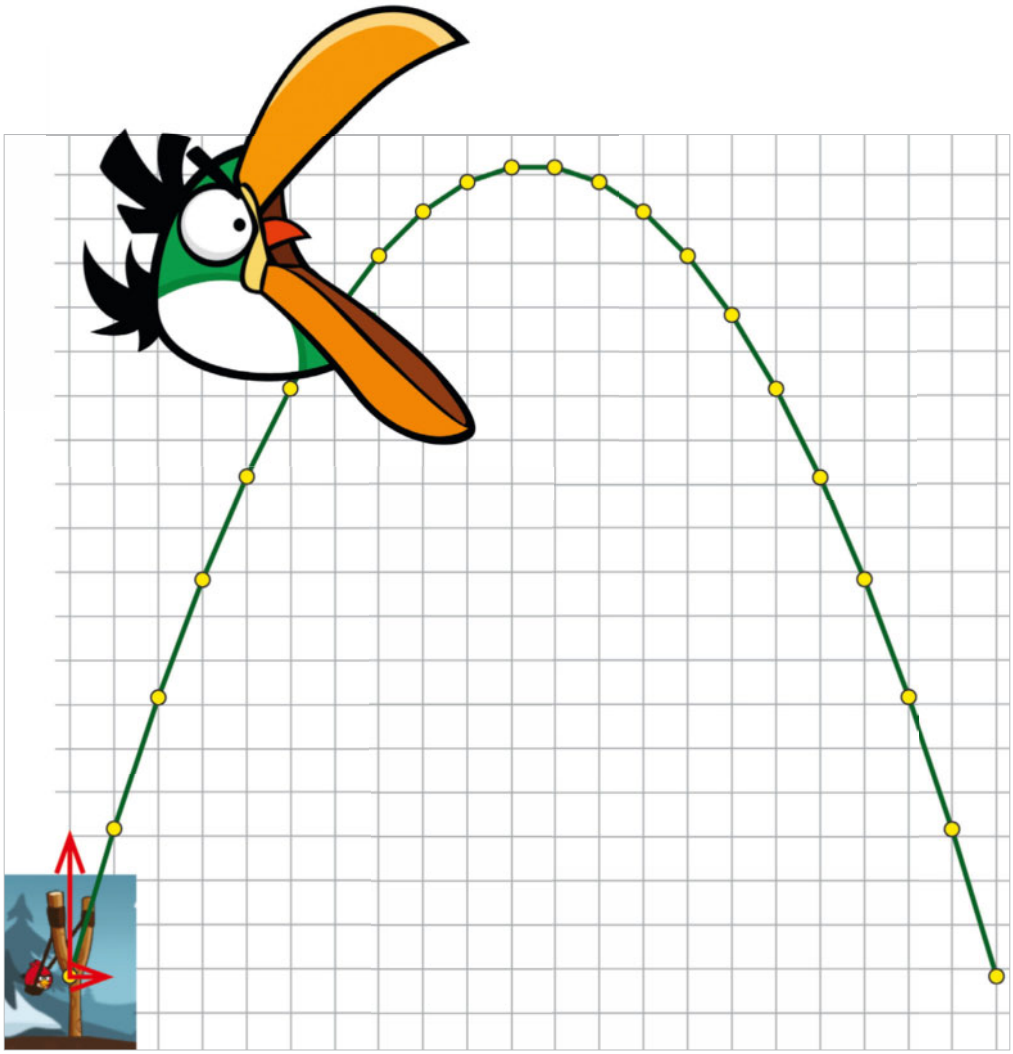
Go the distance

Now that we understand how the velocity changes with time, what about the distance travelled? Again we have two components: the vertical and the horizontal distance. The horizontal distance is easy; the distance travelled horizontally is simply the constant horizontal speed component multiplied by the time.

Vertically, there are a couple of ways we can simulate this: a step-wise algorithm, or by using the relevant kinematics formula. Since we're emulating reality on a fairly small screen, the step-wise algorithm will work perfectly well. After all, we're going to be displaying the red bird on the screen at one position and then a very short time later at another to give the illusion of motion.

Let's assume that our 'steps' are 1/10 second apart again, and g has the same value. We'll make the assumption also that the speed at the start of a step is maintained throughout the step (in reality we know that this is an approximation – the speed changes continuously). Using these assumptions, the distance travelled upwards in the first step is v*1/10, in the second (v-1)*1/10, the third (v-2)*1/10, and so on. Eventually the distance travelled goes negative, in other words, downwards.

Now that we have calculated the distance travelled both vertically and horizontally for each step, we can plot this on the Angry Birds screen and see the parabola we'd expect for the path travelled by the red bird. In fact, the way Roxio displays the parabolic path uses little puffs of 'smoke' at regular intervals as if they were calculating the path in this step-wise manner.



▲ **Figure 1: Parabolic path traced by the red bird at a steep angle**

Figure 1 (above) shows this conceptually. In the lower left corner I show the initial components of the velocity in red: a large vertical component (subject to gravity) and a smaller horizontal one (at the start it's roughly in the ratio 3:1). The grid spacing represents the distance travelled horizontally in 1/10 second (you can see that the yellow dots are spaced equally from left to right). As you can see, the vertical distance travelled gets shorter and shorter upwards every time slice until we reach the top, and then it gets longer and longer downwards. The segments between the yellow dots are all straight; I did not attempt to apply a curve.

For completeness, I'll add the formula from the theory of kinematics to calculate the position q at time t: $q = p + v*t + 0.5*a*t\hat{}2$, where p is the initial position, v the initial velocity, and a the constant acceleration. As you can see, this is a quadratic formula in t; that is, a parabolic path.

Stretch of imagination

Let's now take a look at that catapult. It's an elastic cord and the further we pull it, the more tension is applied, and the more rapid the acceleration when we let go and the cord snaps back. This in turn imparts the initial velocity to the bird once the acceleration due to the tension is dissipated. In essence, the further back we pull the cord the greater the initial velocity.

We *could* simulate the cord snapping back to rest. The relevant pieces are Hooke's law: the force exerted from the stretched cord is proportional to the stretched length, and Newton gave us $F=ma$, or the force is equal to mass times the acceleration. In reality though, the player wouldn't be able to see anything – the action is over so quickly.

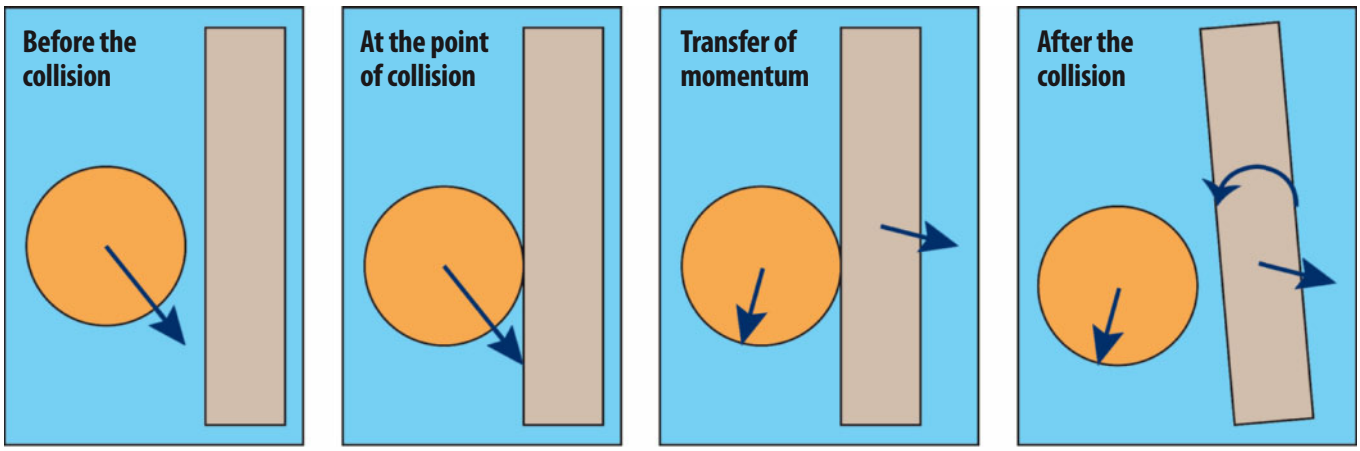
It's easier from a programmer's point of view to code up a simple formula: the initial velocity is equal to the length of the stretched cord times some constant. Work out a good value for the constant through experimentation and move on to the next problem to simulate. I would guess that the game player will always apply the maximum stretch to get the maximum initial velocity – this will provide the maximum damage to the pigs' edifices on contact.

Collision physics

Since we're talking about what happens on contact, we should take a look at the physics of collisions. Here our old friend Isaac Newton is the master.

There are two parts to collisions when simulating them in a game on a computer. The first is how to detect a collision between two objects. This, to put it bluntly, is hard. In Angry Birds, all collisions are between a moving object and a stationary one, the easiest case to simulate. Furthermore, I'm going to postulate that the reason nearly all the birds are circular is that it makes it a bit easier to detect a possible collision.

Rather than provide a detailed discussion here of



▲ **Figure 2: Example sequence showing a collision. Notice the red ball hits below the block's centre of mass causing a rotation as well**

what's required to detect a collision, I'll just illustrate the problems. First of all, the objects have a centre of mass, and it is the centre of mass that describes the path that the object takes. The object has a shape that extends around that centre of mass: to detect a collision you have to track the shape as the centre of mass moves along its trajectory. The easiest shape to track is a circle, and it's probably easiest to just track the portion of the object in front. If you play a lot of Angry Birds, you'll have noticed that the game doesn't track the extremities perpendicular to the line of flight very well – some birds will fly close enough to an object to hit it, but will continue onwards without contact or deflection.

The second part of a collision is what happens because of the collision. This is known as the collision response. Here again we makes use of some simplifying assumptions. The first assumption is that the objects colliding are treated as being rigid. What this means is that we assume that the objects do not deform when they collide. The reason for this is to avoid all those tricky calculations about how much collision energy could be absorbed by something crumpling or denting or compressing. In other words, the collisions in the game are not like crashing your car into a wall, but are instead similar to two snooker balls colliding.

There is a fundamental principle at play here: Newton's law of the conservation of momentum. What this says in essence is that the sum of the momentums of both objects just before they collide is equal to the sum of the momentums just after. For this law to apply, the momentum is assumed to be a vector – that is, the momentum has a direction as well as a magnitude (if you think about it, velocity is a vector as well, as is acceleration). The momentum of an object is its velocity multiplied by its mass.

This is now where we fudge things a little in the game. For a rigid body to collide with another, there's going to be a change of direction for both of them. Their velocities will change. In order to change a velocity we have to apply an acceleration, which in turn implies that a force has to be applied over a period of time. But, since 'rigid' implies 'non-deformable' we have no time. The change in velocity is immediate.

To counter this problem we use a new quantity called the impulse. This is equivalent to a very large force applied over a very small time, and is essentially a way for us to get around the idealisation of a rigid body. (Think about what happens at the atomic level when two snooker balls collide: there is some complicated interaction between the various electric fields of the atoms of the two balls to cause a repulsive force. Just because we want a perfect rigid ball doesn't mean that it actually is.) We can then calculate the impulses at the collision point and apply them to change the two bodies' velocities without them deforming.

Act on impulse

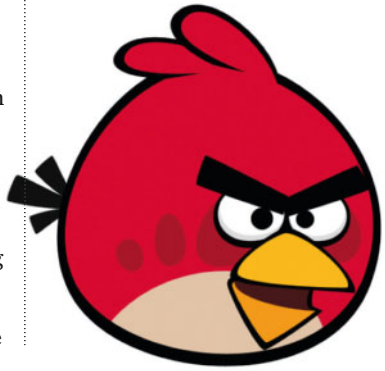
The way to calculate the impulses is determined by the realism you want to achieve. The simplest model to use is known as Newton's Law of Restitution. Here we postulate a coefficient of restitution that models the elasticity of the collision and defines the relation between the incoming and outgoing velocities or the amount of energy absorbed by the collision. A perfect elastic collision has a value of one and describes the collision between two perfect snooker balls. A perfect inelastic collision has a value of zero and basically describes the collision between a lump of wet clay and a wooden floor: splat.

Without going into detail, you have to calculate the perpendicular to the point of collision, the *normal*. It's along this line the collision occurs, and with Angry Birds we're generally talking about a circle – the bird – hitting a straight edge – the plank, sheet of glass, and so on. The normal is perpendicular to the straight edge. Using some relatively straightforward mathematics, we can work out the relative velocities along the normal, the impulses that apply, and hence the new relative velocities after the collision.

Figure 2 shows a stylised view of a collision between a red bird and a plank of wood. This example also shows that the momentum transferred to the wood can also cause a rotation, creating angular momentum as well.

Angry Birds does fudge some of this detail to a certain extent: there are explosions, smashings, bonus points, clouds of feathers and the like, all of which help to disguise the somewhat unrealistic collisions. All in all though, Angry Birds is an excellent example of how to use physics to produce realistic and engaging two-dimensional gameplay. PCP

Julian M Bucknall spent entirely too much time playing Angry Birds as research for this article. His usual collision response is to call the AA.
feedback@pcplus.co.uk



Moving in steps

There are two main ways to detect collisions. The simpler and less computationally intensive of these is known as discrete collision detection. In this algorithm, the movement of objects proceeds in a step-wise fashion, usually linked to the frame rate of the game. The positions of all moving objects are adjusted for the small delta in time, and then the engine checks to see if any of them now intersect or are just about to intersect. If so, those objects are deemed to be colliding.

The main problem with this method is if you have any fast-moving objects in the game. If they are fast enough (and care isn't taken in the implementation), they may pass right through another object without any collision being detected. ■