CHAPTER 2

시뮬레이션 기초

2.1 모델 및 시뮬레이션

물리적으로 생성하는 데 두 가지 핵심 구성 요소가 필요합니다. 애니메이션 : 모델 및 시뮬레이션. 모델은 무언가가 어떻게 작동하는지 또는 운영 하는지를 통제하는 예외로서, 시뮬레이션은 프레임 워크에 모델을 캡슐화하여 시간 경과에 따른 이러한 행동의 예측을 가능하게합니다.

물리적 기반 모델은 시스템 동작 방식의 물리학을 정의하는 모델입니다. 즉, 객체의 동작, 환경의 동작, 객체 간의 상호 작용 등을 관리하는 규칙입니다. 일반적으로 물리적 기반 모델은 실제 세계의 동작, 우리 세계가 실제로 어떻게 작동 하는지를 관리하십시오. 그러나 "실제"모델조차도 거의 항상 단순화된다는 것을 기억하는 것이 중요합니다. 사소한 것으로 간주되는 요소는 종종 무시되며 모델은 모든 세부 사항없이 대규모 동작을 설명 할 수 있습니다. 예를 들어, 자동차 시뮬레이션은 모든 원자를 추적 할 필요는 없지만 대신 자동차의 대형 부품의 동작을 설명하는 데 집중할 것입니다.

또한, 애니메이션 목적으로, 실제 세계의 물리학에 반드시 부합하지 않는 물리학을 정의하는 것이 종종 바람직합니다. 예를 들어, Wile E. Coyote는 낭떠러지를 달리는 것으로 유명합니다. 잠시 동안 대기 상태에 빠져 자신의 상황을 파악한 다음 실제 물리학이 지시하는 것보다 훨씬 빠르면 "잃어버린 "떨어지는 시간! 이와 같은 행동을 포착하기 위해 애니메이터는 물리적 법칙을 재발 명하고 만화 물리학을 구축하여 원하는 효과를 얻을 수있는 기회를 갖습니다 [Hinckley, 1998].

물리적 기반 시뮬레이션은 물리적 기반 모델과 일부 초기 조건 (시작점 및 속도)을 취하여 모델링되는 객체가 시간이 지남에 따라 작동하는 방식을 결정하려고합니다. 다양한 시뮬레이션 기법이 있으며, 하나를 선택하는 것은 정확성, 속도, 견고성 및 특정 모델에 대한 적합성을 조사하는 것입니다.

모델링과 시뮬레이션의 아이디어와 일반적인 시뮬레이션 프로세스를 설명하기 위해, 우리는 모든 사람들이 실제 생활에서 익숙해지는 간단한 예제를 통해 시작합니다 : 중력의 힘에 따라 공을 떨어 뜨리는 것. 나중에 우리는 공기 저항과 바람의 영향을 포함하도록이 예를 확장 할 것입니다.

2.2 뉴턴의 운동의 법칙

이 예제에 대한 실제 기반 모델을 실제로 정의하기 전에 먼저 물리적 기반 시뮬레이션의 기본 원리 인 몇 가지 기본 법칙을 살펴 보겠습니다. 뉴턴의 3 가지 운동 법칙. 대충 말하면, 뉴턴의 법칙은 다음과 같습니다 :

1. 움직이는 물체는 계속 움직이거나, 바깥에있는 물체는 외부의 힘이 작용하지 않는 한, 휴식을 취합니다.
2. 물체에 작용하는 힘은 가속도의 질량과 같습니다. 이것은 종종 $F = ma$로 쓰여집니다.
3. 하나의 물체가 두 번째 물체에 힘을 가하면 첫 번째 물체에는 균등한 반대의 힘이 작용합니다.

이 세 가지 법칙은 물리적 기반 시뮬레이션이 실제로 어떻게 작동하는지 이해하는 데 중요합니다. 우리는 이 장의 두 가지 법칙 중 처음 두 가지를 다룰 것이고, 이 책의 뒷부분에 세 번째 법을 보게 될 것입니다.

첫 번째 법칙에 포함 된 핵심 개념은 관성입니다. 물체는 외력이 가해지지 않는 한 고정 된 속도와 고정 된 방향으로 계속 이동합니다. 물체의 관성을 극복하는 것은 물체의 질량에 의해 결정됩니다. 관성과 관련된 운동량은 물체의 질량과 속도의 곱으로, $P = mv$로 표시됩니다.

두 번째 법칙은 우리에게 가속도라고 부르는 운동량과 운동량의 변화율의 정확한 관계를 제공합니다. 일반적으로 일련의 힘을 결정하여 물리적 기반 모델을 정의합니다. 이러한 힘들로부터 뉴턴의 두 번째 법칙에 의한 가속도를 결정할 수 있으며, $a=\frac{1}{m}F$와 같이 쓰여진다. 가속도와 초기 상태에서 속도와 위치를 결정할 수 있습니다.

차별화 란 시간이나 공간에 따라 변화하는 속도를 정의하는 수학적 용어입니다. 물리적 기반 애니메이션에서 우리는 일반적으로 시간이 지남에 따라 무언가가 변하는 속도에 관심이 있습니다. 위치 $\mathbf{x}$, 속도 $\mathbf{v}$ 및 가속도 $\mathbf{a}$는 모두 속도 변화가 위치 변화율로 정의되고 속도가 속도 변화율로 정의되기 때문에 차이를 통해 서로 관련됩니다. 수학적으로 표현하면, 속도는 위치 변화의 시간 비율이며,

$$\mathbf{v} = \frac{d\mathbf{x}}{dt},$$

가속도는 속도의 시간 변화율이고,

$$\mathbf{a} = \frac{d\mathbf{v}}{dt}.$$

따라서 가속도는 위치의 두 번째 파생어이며 우리가 작성한 것입니다.

$$\mathbf{a} = \frac{d(\frac{d\mathbf{x}}{dt})}{dt} = \frac{d^2\mathbf{x}}{dt^2}.$$

이 책 전체에서 점 표기법은 시간 도함수를 나타내는 속기로 사용됩니다. 변수 위에있는 한 점은 첫 번째 시간 도함수를 나타내고 두 개의 점은 두 번째 시간 도함수를 나타냅니다. 결과적으로 위치, 속도 및 가속도와 관련된 미분 표현식을 다음과 같이 작성합니다.

$$\mathbf{v} = \dot{\mathbf{x}}, \mathbf{a} = \dot{\mathbf{v}}, \text{ and } \mathbf{a} = \ddot{\mathbf{x}}.$$

적분은 시간의 경과에 따라 작은 변화가 축적되어 전체 변화가 발생한다는 것을 설명한다는 점에서, 적분과 반대입니다. 수학적으로 말하면 통합은 차등화 연산의 반대입니다. 이와 같이 보았을 때 가속으로 시작하면 속도를 파악하기 위해 한 번 통합 한 다음 위치를 찾기 위해 속도를 통합 할 수 있음을 알 수 있습니다. 이것은 물리적 시스템의 수치 시뮬레이션에서 핵심 아이디어 중 하나입니다. 우리는 가속으로 시작하여 속도를 발견하고 이를 통합하여 위치를 파악합니다.

수학적으로 표현하면 속도는 가속도의 시간 적분이며,

$$\mathbf{v} = \int \mathbf{a}\, dt,$$

그리고 위치는 속도의 시간 적분이며,

$$\mathbf{x} = \int \mathbf{v}\, dt.$$

미적분 과정에서, a와 v의 표현식이 단순하다면 닫힌 대수 형태로 이러한 적분을 계산하는 방법을 배웠을 것입니다. 그러나 애니메이션 시뮬레이션에서 발생할 수있는 복잡한 동적 문제에서 미적분의 원리를 사용하여 이러한 적분을 해결할 수있는 경우는 거의 없을 것입니다. 대신에 수치 적 통합 기법을 사용하여 이러한 적분을 대략 계산하는 것입니다. 접근법은 특정 시점에서 알려진 위치와 속도로 시작하여 가속도를 계산하기 위해 이들을 사용하고, 시간적으로 짧은 간격을 앞으로 나아가도록 수치 적으로 통합하여 새로운 위치와 속도를 제공합니다. 이 프로세스는 반복적으로 반복되어 일련의 짧은 타임 스텝에서 시작 시간부터 원하는 종료 시간까지 시간을 앞당긴다.

과정을 설명하기 위해, 가속도 a가 일정한 경우를 보자. 이 매우 간단한 경우, 통합 프로세스가 간단하기 때문에 문제에 대한 닫힌 대수 솔루션을 실제로 찾아 낼 수 있습니다. 초기 위치는 x0이고 초기 속도는 v0라고 가정합니다.

$$\mathbf{v} = \int \mathbf{a}\, dt$$

그런 다음 한 번 통합하면

$$\mathbf{v} = \mathbf{a}t + \mathbf{v}_0. \tag{2.1}$$

이 식 통합하기

$$\mathbf{x} = \int (\mathbf{a}t + \mathbf{v}_0)\, dt$$

그러면 주어진다.

$$\mathbf{x} = \frac{1}{2}\mathbf{a}t^2 + \mathbf{v}_0 t + \mathbf{x}_0. \tag{2.2}$$

개념적으로, 가속도, 시작 속도 v0, 시작 위치 x0가 주어진다면, 미래의 어떤 시간 t에서 속도와 위치를 찾을 수 있습니다.

따라서 일반적인 프로세스는 다음과 같습니다.

1. (Modeling) 행동을 지배하는 힘의 모델을 정의하라. 뉴튼의 두 번째 법칙에 따라, 이 힘은 가속도에 의해 결정됩니다.

2. (Provide Initial Conditions) 초기 시작 위치와 속도를 정의하십시오.

3. (Simulation) 통합을 사용하여 시간이 지남에 따라 위치 및 속도를 푸십시오.

이 과정은 각각의 단계에 대한 다양한 단서를 가지고 훨씬 더 복잡해질 수 있습니다. 그러나 이 기본 개요는 우리가 이 책에서 볼 수 있는 대부분을 포함하여 먼 길을 가져올 수 있습니다.

2.3 DROPPING A BALL IN 1D

우리는이 과정을 설명하기 위해 사용하는 간단한 예를 살펴볼 것입니다 : 공을 떨어 뜨리는 것. 이 예제를 가능한 한 간단하게 유지하기 위해 우리가 관심있는 유일한 힘은 중력이라고 가정 할 것이므로 공기 저항을 무시할 것입니다. 우리는 단 하나의 차원 (높이)에 대해서만 걱정할 것이며, 객체는 단지 단일 점이라고 가정 할 것입니다. 우리가 1D에서만 작업하고 있기 때문에 우리의 모든 가치는 스칼라가 될 것이고 우리는 긍정적 인 방향으로 나아갈 것입니다. 우리의 첫 번째 단계는 우리가 사용할 모델을 정의하는 것입니다.

우리는 지상에있는 물건을 다루고 있다고 가정합니다. 이 경우 중력은 거의 일정한 하향 력을 발휘하며, $F = -mg$, 여기서, g는 중력 상수입니다. 간단한 예제에서, 이것이 우리가 신경 쓰는 유일한 힘입니다. F = ma이기 때문에 우리는 a = -g를 가지며 볼의 질량과는 독립적입니다.

중력 가속도 상수 g는 물론 지구상에서 대략 9.8m / s2 또는 32ft / s2로 잘 알려진 "실제 세계"값입니다. 구체적인 수치는 그렇게 중요하지 않습니다 : 우리는 미터 나 피트보다 다른 거리 측정을 사용하거나, 다른 행성에서 중력을 시뮬레이션하거나 인공 시뮬레이션을 설명하고자 할 수 있습니다. "현실 세계"상황을 시뮬레이션하기 위해,

단위를 정확하게 추적하고 모든 상수에 대해 올바른 값을 사용하는 것이 중요합니다. 사람들이 이것을 잘못했을 때 하나 이상의 값 비싼 재앙이 발생했습니다! * 그러나 일러스트레이션과 많은 그래픽 애니메이션에서 우리는 상수에 대한 "참"값을 고수 할 필요가 없습니다. 일을 단순하게 만들려면 g = 10 단위를 제곱당 제곱으로합니다.

이제 우리는 매우 간단한 모델 인 a = -10을 가지므로 몇 가지 초기 조건을 정의해야합니다. 우리가 휴식에서 시작한다고 가정 해 봅시다. 그래서 v0 = 0이고 100 단위의 높이에서 x0 = 100입니다.

2.4 운동의 미분 방정식

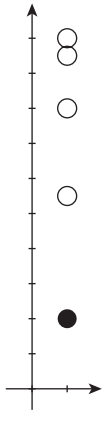
이 시점에서, 우리는 초기 조건과 함께 우리의 떨어지는 볼의 모션에 대한 모델을 가지고 있습니다. 우리의 목표는 t = (0, 1, 2, 3, 4)라고하는 일련의 시간 값에서 떨어지는 볼의 위치를 찾는 것입니다. 이러한 값을 애니메이션의 프레임을 렌더링 할 수 있도록 위치를 알고 싶을 때라고 생각할 수 있습니다.

이것은 매우 단순한 모델이기 때문에, Equa-tions 2.1과 2.2를 사용하여 실제로 통합 할 수 있습니다.

$$\mathbf{v} = at + \mathbf{v}_0 = -10t + 0 = -10t, \quad \text{and}$$
$$\mathbf{x} = \frac{1}{2}at^2 + \mathbf{v}_0 t + \mathbf{x}_0 = \frac{1}{2}(-10)t^2 + 0t + 100 = 100 - 5t^2.$$

우리가 일반적으로 이와 같은 좋은 닫힌 형태의 대수적 인 해법을 가지고 있지 않다는 것을 깨닫는 것이 중요합니다! 일반적으로, 우리는 벡터 값을 갖는 다중 힘에 대한 표현을 가지고 있으며 여기에서했던 것처럼 v와 x에 대한 닫힌 형식 표현을 얻기 위해 통합 될 수 없습니다. 다음 섹션에서는 우리가 좋은 공식을 갖지 않을 때 우리가 어떻게 수치 적으로 통합하는지 배우게 될 것입니다.

그러나이 경우에는 우리 모델에 대한 정확한 해답을 얻고 관심있는 각 시간 값에서 속도와 위치를 결정할 수 있습니다. 아래의 표는 5 개의 시간 단계에서이 값을 보여주고 오른쪽 그림은 볼의 위치가 맨 위의 시작 위치에서 바닥 근처의 4 초 위치까지 어떻게 변하는지를 보여줍니다.



	t	0	1	2	3	4
Exact	v	0	−10	−20	−30	−40
	x	100	95	80	55	20

다음으로, 시뮬레이션 루프 내에서보다 일반적인 통합을 처리하는 방법을 살펴 보겠습니다.

* 가장 유명한 예로 1999 년 Mars Climate Orbiter가 있습니다. 이 행성은 미터법 (MW) 단위와 영국식 단위 (파운드)의 혼합으로 인해 행성에 너무 가까이 다가 갔을 때 파괴되었습니다.

2.5 A BASIC SIMULATION LOOP

In a more general problem, we will have an expression for forces that does not allow us to integrate to get closed-form solutions like those in Equations 2.1 and 2.2. Instead, we must integrate to find velocity and position using numerical techniques. There are a variety of numerical integration techniques, several of which we will encounter later, but for now we will introduce the most basic technique—*Euler integration.*

Numerical integration techniques all rely on the notion of a *timestep.* They work by starting with a given state at a particular time, and then computing the state one timestep into the future. Although the notation used in the physically based simulation literature can vary from publication to publication, the timestep is usually denoted by h, and we will follow that convention in this book.

First, we must understand what it is we are wanting to simulate. The *state* of a simulation refers to the values of the changing simulation parameters at an instant in time. In our simple example the state is just the velocity $\mathbf{v}$ and position $\mathbf{x}$ of the ball. We have an initial state ($\mathbf{v}_0$ and $\mathbf{x}_0$) at time $t = 0$, and we want to determine the state at the various time values $t = (1, 2, 3, 4)$.

Using this notion of state, the general simulation loop will take the following form:

current-state = initial-state;
$t = 0$;
while $t < t_{max}$ **do** *// note: here* current-state *is the state at time t*
    Determine forces at current-state;
    Determine accelerations by Newton's second law: $\mathbf{a} = \frac{1}{m}\mathbf{F}$;
    new-state = Integration of accelerations over timestep h;
    current-state = new-state;
    $t = t + h$;
end

Notice that at the beginning of each loop iteration, the state and the current time t are in-sync.

For notation purposes, we will use a superscript in square brackets on a variable to indicate the loop iteration we are on, and we will use n to denote the "current" iteration. The value of a variable at the initial time will use a superscript [0], after one timestep it will be [1], etc. Notice that the superscript refers to the number of iterations, not the actual time. If the iteration number is n, the actual time is $t = nh$.

The simplest numerical integration technique is *Euler integration.* The basic assumption of Euler integration is that all values being integrated in the simulation stay constant during a timestep. Thus, if we know the acceleration at the beginning of a timestep, we can assume that it is constant throughout the timestep. Referring back to Equation 2.1, we can see that this means that the new velocity is just the old velocity increased by acceleration times time. Likewise, we assume velocity is

constant over the course of the timestep, so the new position is found similarly. So, Euler integration to determine a velocity and position from the old velocity and position can be written

$$\begin{aligned} \mathbf{v}^{[n+1]} &= \mathbf{v}^{[n]} + \mathbf{a}^{[n]}h, \\ \mathbf{x}^{[n+1]} &= \mathbf{x}^{[n]} + \mathbf{v}^{[n]}h. \end{aligned} \tag{2.3}$$

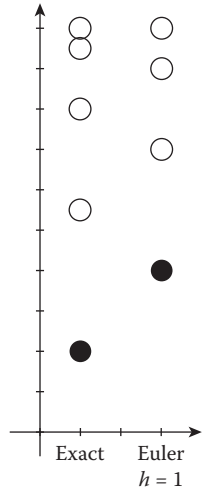
The simulation loop for Euler integration for our basic ball example will look as follows:

$\mathbf{v}^{[0]} = 0$; $\mathbf{x}^{[0]} = 100$;
$t = 0$; $n = 0$;
while $t < t_{max}$ **do**
  $\mathbf{a}^{[n]} = -10$;
  if *t is an output frame time* **then**
    Output $\mathbf{v}^{[n]}$, $\mathbf{x}^{[n]}$;
  end
  $\mathbf{v}^{[n+1]} = \mathbf{v}^{[n]} + \mathbf{a}^{[n]}h$;
  $\mathbf{x}^{[n+1]} = \mathbf{x}^{[n]} + \mathbf{v}^{[n]}h$;
  $n = n + 1$; $t = nh$;
end

The **if** statement is used because it can be that not every simulation step needs to be output as an animation frame. Typically, a simulation will run for several iterations to cover the time taken by one frame. For example if we are producing 20 frames per second for display purposes, and our simulation timestep is $h = 0.005$ seconds, then we only want to output a frame every 10 iterations.

For this example, let us pick a timestep of $h = 1$, and output every new state. The resulting ball positions are shown in the figure, compared with those of the exact solution. The corresponding values are

	t	0	1	2	3	4
Exact	v	0	−10	−20	−30	−40
	x	100	95	80	55	20
Euler	v	0	−10	−20	−30	−40
$h = 1$	x	100	100	90	70	40



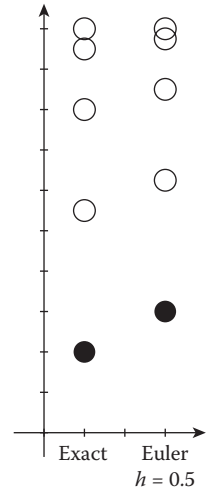
Unfortunately, this is not giving us quite the correct solution. It is getting some of the general behavior—the velocity values match exactly, and the ball is dropping downward at an ever-increasing rate—but over the length of the simulation, the Euler solution is lagging behind the exact solution. The assumptions made in Euler integration give us a clue as to why this is happening. We are assuming that both acceleration and velocity remain constant over the timestep. While this is true for acceleration, it is not true for velocity, which

is increasing continually as the ball falls. Thus, the integration of acceleration to determine velocity is exactly correct, but the integration of velocity to determine position always yields a result that is underestimated. In the next section, we will see some ways we might get better results.

2.6 NUMERICAL APPROXIMATION

Given that Euler integration was not as close to the correct answer as we would like, we want to think about some ways that we might get more accurate numerical approximations. We will briefly look at three possible options.

First, let us consider the step size. In our earlier example, we set $h = 1$—that is, that the step size was the same as the rate at which we wanted to know results. What if we took a smaller step size, say $h = 0.5$? In this case, we will take two simulation steps for every output value. The results are shown in the figure and in the table below.



	t	0	0.5	1	1.5	2	2.5	3	3.5	4
Exact	v	0		−10		−20		−30		−40
	x	100		95		80		55		20
Euler	v	0		−10		−20		−30		−40
$h = 1$	x	100		100		90		70		40
Euler	v	0	−5	−10	−15	−20	−25	−30	−35	−40
$h = 0.5$	x	100	100	97.5	92.5	85	75	62.5	47.5	30

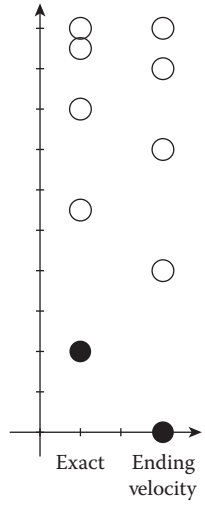
This approach has brought us quite a bit closer to the correct answer, however it also required twice as many simulation steps, and thus twice as much computation time. In fact, as we continue to shrink the step size (i.e., take more intermediate steps between each output time), we will get closer and closer to the correct solution. This approach works in general (the smaller the step size, the more accurate the integration) for any problem. If you recall from calculus the way derivatives and integrals are developed, they are usually expressed as a limit when some value goes to zero. By reducing the timestep to smaller and smaller values, we are in effect approaching that limit, making the integration more accurate. Reducing the timestep is usually the first approach to try when you are worried that your integration is not giving good enough results—shrink the step size and see if things improve. Reducing timesteps will get us closer and closer to the correct answer, but never quite there.

Besides decreasing the step size, we might consider whether a different integration scheme entirely could give better results. One thing that you might notice is that with Euler integration, no change in the position is seen until after a full timestep. This is because we are using the velocity value from the previous state, which will not change from 0 until after that full timestep. Instead, we could try

using the value from the end of the timestep, so that we get a change immediately. Equations 2.3 become

$$\begin{aligned}\mathbf{v}^{[n+1]} &= \mathbf{v}^{[n]} + \mathbf{a}^{[n]}h,\\ \mathbf{x}^{[n+1]} &= \mathbf{x}^{[n]} + \mathbf{v}^{[n+1]}h.\end{aligned} \tag{2.4}$$

Notice that we keep the same integration scheme for acceleration that we used previously to determine velocity. Trying to use acceleration at the end of the time step would be more difficult since that acceleration is not known until we first know the state at the end of the timestep (a chicken-and-egg problem)! If we integrate using velocity at the end of the timestep, with the original timestep $h = 1$, we get the following:



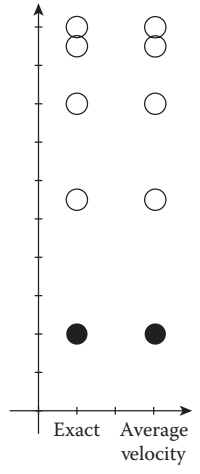
	t	0	1	2	3	4
Exact	**v**	0	−10	−20	−30	−40
	x	100	95	80	55	20
Euler	**v**	0	−10	−20	−30	−40
$h = 1$	**x**	100	100	90	70	40
Ending	**v**	0	−10	−20	−30	−40
velocity	**x**	100	90	70	40	0

That gave us different results, but really no better than in our original Euler example. Instead of undershooting on position we are now overshooting—moving too far in one timestep.

As a third approach, let us try using the average of the Euler approach and the ending value approach we just tried. Now, our equations are

$$\begin{aligned}\mathbf{v}^{[n+1]} &= \mathbf{v}^{[n]} + \mathbf{a}^{[n]}h,\\ \mathbf{x}^{[n+1]} &= \mathbf{x}^{[n]} + \frac{\mathbf{v}^{[n]} + \mathbf{v}^{[n+1]}}{2}h.\end{aligned} \tag{2.5}$$

Using this average step approach gives us the following results shown in the figure and table.



	t	0	1	2	3	4
Exact	**v**	0	−10	−20	−30	−40
	x	100	95	80	55	20
Euler	**v**	0	−10	−20	−30	−40
$h = 1$	**x**	100	100	90	70	40
Average	**v**	0	−10	−20	−30	−40
velocity	**x**	100	95	80	55	20

Now we are getting the same values as the exact calculation. In fact, this approach turns out to be an exact way to integrate a set of equations with constant acceleration. For a more general problem, it will give a better answer than simple Euler but it is also more complicated to compute. We will see more details and discuss why this is the case in Chapter 7. For now, though, the thing to notice is that by selecting a different integration technique, we actually achieved better results than we would have achieved with Euler, regardless of the step size.

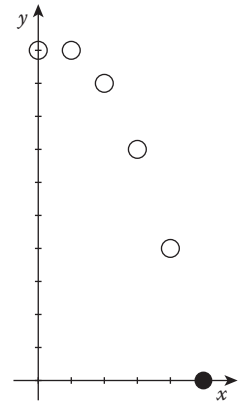
2.7 3D MOTION IN AIR

For the example so far, we have illustrated a ball dropping under the force of gravity. Although all of this example and our illustrations were in just one dimension (height), the methods and equations used apply equally well when we extend our simulation to two or three dimensions. From this point forward, we will generally be describing our equations in 3D, recognizing that the simplification to 2D is usually straightforward.

2.7.1 Tracking Three Dimensions

We will still keep track of a position $\mathbf{x}$ and a velocity $\mathbf{v}$; however, each of these will be a 3D vector, rather than a single scalar value. Our gravity vector will be $\mathbf{g} = [0\ -g\ 0]^T$ for some constant value g. Given this model, all of the equations described earlier apply exactly the same way. The height y of the ball will follow a path just as previously described. If the only force is from gravity, in the horizontal direction (x and z) there will be no change in velocity since the gravitational acceleration has no x or z component.

As a quick illustration, let us assume that the initial velocity is $\mathbf{v} = [10\ 0\ 30]^T$, the initial position is $\mathbf{x} = [0\ 100\ 0]^T$, and the gravitational constant $g = 10$. If we were to use simple Euler integration (with $h = 1$) as in Section 2.5, we would calculate the motion:



t	0	1	2	3	4	5
v	$\begin{bmatrix}10\\0\\30\end{bmatrix}$	$\begin{bmatrix}10\\-10\\30\end{bmatrix}$	$\begin{bmatrix}10\\-20\\30\end{bmatrix}$	$\begin{bmatrix}10\\-30\\30\end{bmatrix}$	$\begin{bmatrix}10\\-40\\30\end{bmatrix}$	$\begin{bmatrix}10\\-50\\30\end{bmatrix}$
x	$\begin{bmatrix}0\\100\\0\end{bmatrix}$	$\begin{bmatrix}10\\100\\30\end{bmatrix}$	$\begin{bmatrix}20\\90\\60\end{bmatrix}$	$\begin{bmatrix}30\\70\\90\end{bmatrix}$	$\begin{bmatrix}40\\40\\120\end{bmatrix}$	$\begin{bmatrix}50\\0\\150\end{bmatrix}$

The figure, showing just the x and y components, illustrates the positions computed in the table. The motion of the ball follows a parabolic arc downward.

The object continues to move at a constant rate in the horizontal direction, while accelerating downward continuously.

Just like in Section 2.6, we could use different integration schemes and smaller timesteps to get more accurate results.

2.7.2 Air Resistance

We have a ball moving through the air under the force of gravity, however this is still not a very interesting simulation. Since gravitational force is proportional to mass, the acceleration is independent of mass, i.e. a feather will fall at the same rate as a bowling ball. To make our model more interesting, and to get the difference in behavior across different types of objects, we need to add another factor to our model: *air resistance*.

We will use a very simple model of air resistance here. Real air resistance is actually a very complicated physical effect, involving the linear and angular velocity of the object, the geometry and orientation of the object, and the properties of the fluid (i.e. air) that it is inside of. While an engineer designing an airplane needs to take all of this into account, for our purposes a very basic model will suffice and will help us illustrate how an animator might create forces to generate a desired effect.

Air resistance is a force that pushes against the ball opposite the direction of its motion. So, the direction of the force will be in the direction opposite current velocity, i.e., $-\hat{\mathbf{v}}$. Two factors contribute to its magnitude. First, the air resistance force will increase the faster we are going. So, as a first approximation, we will make it proportional to the magnitude of the velocity $\|\mathbf{v}\|$. As velocity doubles, the wind resistance will also double. We will also assume that there is a constant d that accounts for the overall factors (such as geometry) contributing to air resistance: for a sleek, slippery object, d would be small, while for a rough object that catches a lot of wind, d would be large. This user-tunable constant gives an animator more direct control over the amount of air resistance desired.

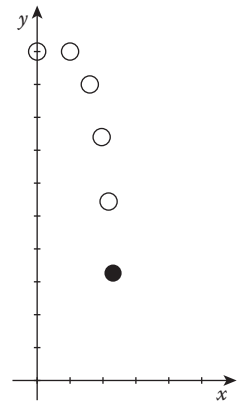
Our wind resistance force is thus $\mathbf{F}_{\text{air}} = -d\,\|\mathbf{v}\|\hat{\mathbf{v}} = -d\mathbf{v}$.* If we couple this with our gravitational force $\mathbf{F}_{\text{gravity}} = m\mathbf{g}$, we get:

$$\begin{aligned} \mathbf{F} &= \mathbf{F}_{\text{gravity}} + \mathbf{F}_{\text{air}}, \\ m\mathbf{a} &= m\mathbf{g} - d\mathbf{v}, \quad \text{or} \\ \mathbf{a} &= \mathbf{g} - \frac{d}{m}\mathbf{v}. \end{aligned}$$

This acceleration accounts for both gravity and air resistance.

*Note that the notation here $d\mathbf{v}$, which is the product of constant d with the vector $\mathbf{v}$, should not be confused with the differential $d\mathbf{v}$.

Let us look at an example to see the effect. We will use the same conditions as the previous example, but now assuming that $d = 0.4$ and that $m = 1$. Thus, our equation for acceleration is $\mathbf{a} = [0\ -10\ 0]^T - 0.4\mathbf{v}$. Using a basic Euler simulation gives us the following results:



t	0	1	2	3	4	5
v	$\begin{bmatrix}10\\0\\30\end{bmatrix}$	$\begin{bmatrix}6\\-10\\18\end{bmatrix}$	$\begin{bmatrix}3.6\\-16\\10.8\end{bmatrix}$	$\begin{bmatrix}2.2\\-19.6\\6.5\end{bmatrix}$	$\begin{bmatrix}1.3\\-21.8\\3.9\end{bmatrix}$	$\begin{bmatrix}0.8\\-23.1\\2.3\end{bmatrix}$
x	$\begin{bmatrix}0\\100\\0\end{bmatrix}$	$\begin{bmatrix}10\\100\\30\end{bmatrix}$	$\begin{bmatrix}16\\90\\48\end{bmatrix}$	$\begin{bmatrix}19.6\\74\\58.8\end{bmatrix}$	$\begin{bmatrix}21.8\\54.4\\65.3\end{bmatrix}$	$\begin{bmatrix}23.1\\32.6\\69.2\end{bmatrix}$

In this example, both the horizontal and the vertical motion are slowed considerably by the air resistance.

If the simulation were to continue, the x and z velocities would gradually approach 0, while the vertical velocity would approach a constant. Thus, accounting for air resistance has allowed us to simulate the idea of *terminal velocity*. Terminal velocity occurs when the force of air resistance exactly compensates for the force of gravity, meaning that an object no longer accelerates downward, but rather moves at a constant speed downward. Setting $\mathbf{a} = \mathbf{0}$ and solving for $\mathbf{v}$ in the equation above, it is easy to see that terminal velocity occurs when $\mathbf{v} = \frac{m}{d}\mathbf{g}$. In the above example, the terminal velocity in the y direction would be -25. We can see that even in the few timesteps we have in the table, we are getting close to that terminal velocity.

2.7.3 Wind

One final point should be noted about air resistance. In many animations, an animator is interested in applying forces to guide the simulation in a particular way, and one common way to do this is using "wind" forces. Figure 2.1 shows a frame from a falling leaves animation that was simulated using wind forces. The air resistance model we have described here does not distinguish what the source of the air pressing on the object is, and thus a wind force can be handled just like air resistance. In other words, wind blowing in some direction on a stationary object is the same as if the object were moving opposite that direction in still air. In either case, the force is acting to slow the object relative to that wind direction. The wind force, then, will be given by

$$\mathbf{F}_{\text{wind}} = d\,\mathbf{v}_{\text{wind}},$$

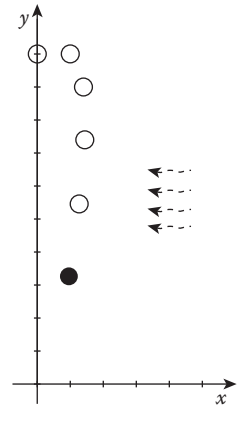
so the overall acceleration is

$$\mathbf{a} = \mathbf{g} + \frac{d}{m}(\mathbf{v}_{\text{wind}} - \mathbf{v}).$$

FIGURE 2.1 Falling leaves blown by the wind. (Courtesy of Lana Sun.)

The term $(\mathbf{v}_{\text{wind}} - \mathbf{v})$ in this equation is known as the relative velocity of the object with respect to the wind.

Building on the previous example, if we had a wind velocity of $[-12.5\ 0\ 0]^T$, our simulation would result in the following:



t	0	1	2	3	4	5
v	$\begin{bmatrix}10\\0\\30\end{bmatrix}$	$\begin{bmatrix}1\\-10\\18\end{bmatrix}$	$\begin{bmatrix}-4.4\\-16\\10.8\end{bmatrix}$	$\begin{bmatrix}-7.6\\-19.6\\6.5\end{bmatrix}$	$\begin{bmatrix}-9.6\\-21.8\\3.9\end{bmatrix}$	$\begin{bmatrix}-10.8\\-23.1\\2.3\end{bmatrix}$
x	$\begin{bmatrix}0\\100\\0\end{bmatrix}$	$\begin{bmatrix}10\\100\\30\end{bmatrix}$	$\begin{bmatrix}11\\90\\48\end{bmatrix}$	$\begin{bmatrix}6.6\\74\\58.8\end{bmatrix}$	$\begin{bmatrix}-1.0\\54.4\\65.3\end{bmatrix}$	$\begin{bmatrix}-10.6\\32.6\\69.2\end{bmatrix}$

While the motion in y and z is the same as in the previous example, there is a wind force in the $-x$ direction, that overcomes the ball's original velocity in the $+x$ direction, and begins moving the ball backward.

Just as there was a vertical terminal velocity counteracting gravity, the constant wind force will cause there to be a horizontal terminal velocity, matching the horizontal wind speed of -12.5. Thus, if left to run indefinitely, the simulated ball would eventually approach a constant velocity of $[-12.5\ -25\ 0]^T$.

Figure 2.2 illustrates the three examples we have shown, side by side, so that it is easy to compare the effects.

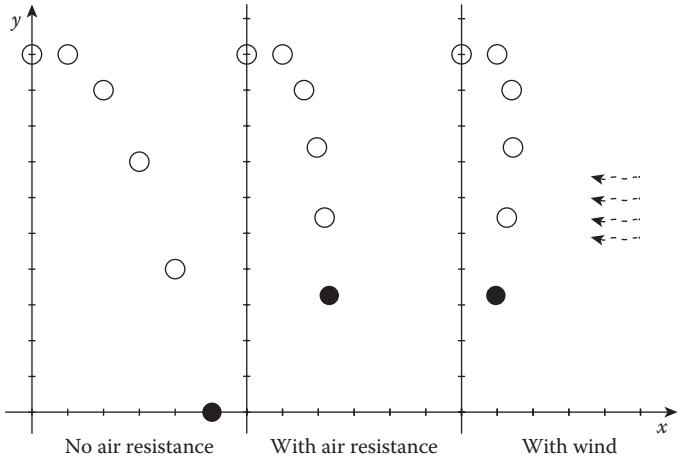


FIGURE 2.2 Ball drop showing the effects of air resistance and wind.

2.8 SUMMARY

Here is a brief review of some of the main concepts to take away from this chapter:

- The general process for simulation involves three stages: defining a model, establishing initial conditions, and simulating.
- Typical simulation involves computing forces based on the current state of the system.
- From these forces we obtain accelerations using Newton's second law written $\mathbf{a} = \frac{1}{m}\mathbf{F}$.
- We use these accelerations to project forward in time to update velocities and positions. This projection process is numerical integration.
- Euler integration (see Equation 2.3) provides a simple way to perform numerical integration.
- We can improve our integration results by reducing the timestep at the cost of increased simulation time, or by using a better integration technique.
- A simple gravitational model hides the effect of mass on acceleration, because the force due to gravity and the force needed to accelerate an object are both proportional to mass.
- Adding air resistance to the gravitational model brings the effect of mass back into play, and provides a convenient mechanism to build wind effects into a simulation.