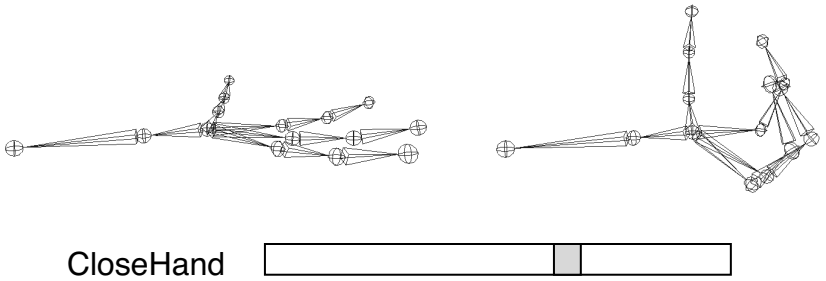

Figure 7-37. A single driven key can control a more complex set of actions, such as the closing of a hand.

fine-tuning the deformations of a skinned surface on your character model. As we discuss in sections 4.9 and 7.13, skinning surfaces onto an inverse kinematic skeleton causes the IK skeleton to control the deformations of the surfaces. If the deformations that result aren't quite satisfactory, one possible solution is to make the position of an IK effector the driver, and the positions of several surface points the driven. With such a driven key defined, you could keyframe the IK effector of your character, and then keyframe the slider bar of your driven key to adjust the deformation of the surface to suit you.

Expressions and driven keys provide powerful tools for producing animations that might otherwise be difficult to create. Still, the capabilities of these techniques are quite simple compared to those of a full-blown programming language. To approach that level of programming power, you must work with a related technique called procedural modeling and animation, discussed in section 5.4.

7.6 Motion Dynamics: Principles, Rigid Bodies

In certain situations the movement of an object, such as a ball falling from a height and bouncing on a surface, can be predicted very precisely. Physicists have studied this sort of phenomenon for a long time and have developed very accurate mathematical formulae to describe what happens in this situation. Knowing the effect of gravity and various properties of the ball and of the surface, they can calculate, using the laws of physics, how quickly the ball will fall, as well as how high and how often it will bounce.

This sort of mathematically precise and detailed study of the motion of objects is called **motion dynamics** and can be of great use to animators. The ball dropping straight down and bouncing off a flat surface is not very difficult to animate using standard keyframing techniques, but a more complex situation, such as a box being dropped and bouncing down a stairway (Figure 7-38), is very difficult to animate in a naturalistic way using keyframing. To estimate realistically how far the box bounces with each bounce, how much

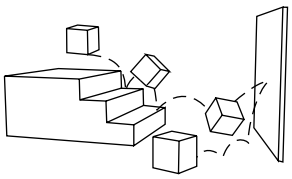

Figure 7-38. The motion of some objects can be calculated precisely according to the laws of physics.

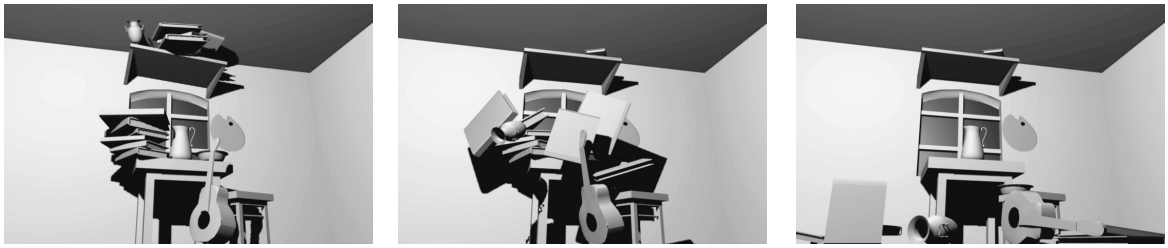

Figure 7-39. Motion dynamics applied to rigid bodies can produce an animation that would be quite difficult to do with keyframing. (© 2001 Sophia Rotchko.)

and in what directions the box rotates as it flies through the air, which edge or corner of the box strikes the surface of the stairs first, and so on, is a *very* complicated piece of animation. The physical laws of motion dynamics, however, are precise enough for you to calculate the motion of the box.

3D animation packages include this sort of motion-dynamics capability and provide user interfaces that make the technical complexity of the underlying calculations transparent. Typically you use a mouse device to open a menu and make several selections. These selections define the various parameters that control the motion of the object to be animated. Once you have defined these parameters, you make a selection that instructs the system to perform the calculations and play back the resulting animation on the screen. This result is called a **simulation**, because it simulates, or mimics, the behavior of objects in a real-world situation behaving under the influence of real-world physical laws.

Motion dynamics calculations can be applied to many kinds of animation problems, as we will see later in this chapter. In this section we present the basic principles common to all motion dynamics applications, and examine their simplest application: the movement of **rigid bodies**, nondeformable objects whose movement is calculated by motion dynamics. Figure 7-39 is an example of the power of motion dynamics as applied to rigid bodies. In this sequence of animation frames, the first frame shows a collection of books and other objects. In the middle frame, the vase on the highest shelf has fallen and hit the pile of books, causing them to fall in turn. In the last frame, nearly all the objects have fallen to the floor. One hundred percent of this animation was calculated as a motion dynamics simulation of rigid bodies. In subsequent sections we will discuss motion dynamics as applied to soft (deformable) bodies (section 7.7), to particles (section 7.8), and to cloth (section 7.9).

PHYSICAL PROPERTIES

In setting up a simulation you first define the **physical properties** of the object to be animated. That is, the motion-dynamics system must know certain characteristics of the object in order to calculate the simulated movement of the object. For example, a ball made of solid iron bounces much differently than does a ball made of balsa wood.

In physics—and in motion dynamics—one of the most basic physical

properties of an object is **mass**. Mass is related to, but not quite the same as, weight. That is, the weight of an object is the result of gravity pulling down on the massiveness of the object. On Earth, gravity is effectively the same everywhere, so mass and weight have a direct, one-to-one correspondence. However, since weight is linked to gravity, the weight of an object *can* change—for example, if the object were in outer space—although the mass of the object remains the same wherever the object is. An object with a large mass is very difficult to start moving and very difficult to stop once it begins moving. This is why astronauts approach docking with a space station very slowly. Even though both the space vessel and the space station are weightless, they are extremely massive and could cause great damage to each other if moving too fast when they make contact.

This same property of mass directly affects how an object moves in a motion-dynamic simulation. For example, if you roll a massive ball down a ramp and onto a level surface, the ball tends to keep rolling for a long time because the massiveness overcomes the forces that otherwise would make the object stop (Figure 7-40a). If you roll a less massive ball down the same ramp, it tends to stop much more quickly on the flat surface (Figure 7-40b).

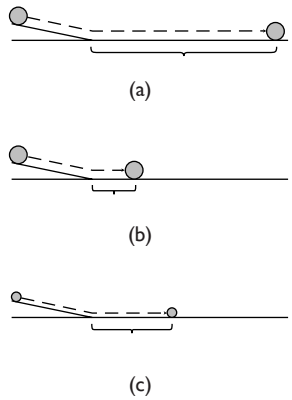

Figure 7-40. The mass of an object plays a key role in how the object behaves as it moves. Mass is a function of both the density of the material and the size of the object.

What makes the one ball more massive than the other is not merely the size of the objects but also the density of the materials from which they are made. A balsa-wood ball is less massive than an iron ball because balsa wood is less dense than iron. A solid clay ball of the same size is more dense and more massive than a balsa-wood ball, but less dense and less massive than an iron ball.

Size does play a role, however, in determining mass, which is a function of both the density and the size (more exactly, the volume) of an object. A small iron ball (Figure 7-40c), even though it has the same density as a larger iron ball (Figure 7-40a), has a smaller mass because it has a smaller volume. Consequently, the little, less massive ball does not roll as far as the larger, more massive ball (Figure 7-40b).

Since the mass of an object depends on both volume and density, most people cannot readily estimate it. Consequently, some systems allow you to define mass indirectly by specifying density. By thinking in terms of material (iron, wood, clay, etc.) you can readily give the system a density parameter value—larger numbers for more dense materials, smaller numbers for less dense materials. The software then automatically calculates the volume of the object and then, from the combination of volume and density, the mass.

The density of an object also is important to consider when simulating objects lighter than air. A gas-filled balloon, for example, rises into the air because the gas inside the balloon is of a very, very low density. This causes the total mass of the balloon to be so little that gravity will not even keep it on the ground.

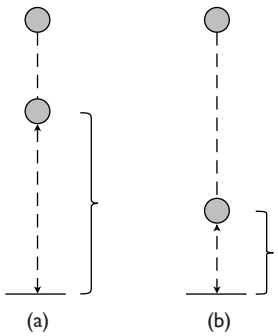

Figure 7-41. The elasticity of an object is a measure of how much the object bounces when it hits a surface.

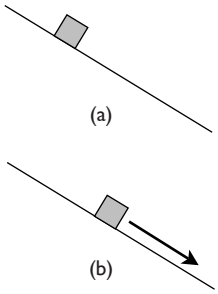

Figure 7-42. The static friction of an object is a measure of how readily the object, when stationary, will start to move after being subjected to some force.

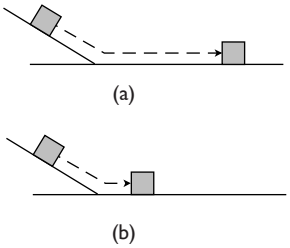

Figure 7-43. The kinetic friction of an object is a measure of how easily the object, when moving, can be stopped by some resisting force.

Another important physical property of objects, called **elasticity**, reflects the way objects bounce. Imagine that two balls have exactly the same mass, but that when you drop these balls, one ball bounces much higher than the other one does (Figure 7-41a and b). A solid rubber ball, for example, tends to bounce much higher than a solid plastic ball—even though, as in this example, the masses of the balls happen to be exactly the same—and a glass marble dropped on a concrete floor bounces much higher than you might expect. Objects that have a high degree of elasticity, such as the solid rubber ball and the marble, tend to bounce a lot. Objects with lower elasticity bounce less.

The term "elasticity" can be a bit misleading, however. In day-to-day usage, the word describes objects easily stretched or deformed. A rubber band and a soft rubber ball that you can squeeze between your fingers are elastic in this sense. As it is used in motion dynamics, however, "elastic" means something different. A glass marble is extremely elastic because it bounces very high, even though it is very hard and does not deform between your fingers at all. In motion dynamics, elasticity refers to the amount of energy retained or lost when an object makes contact with something else. If an object is very elastic, very little energy is lost and the object bounces a lot. If the object is very inelastic, a great deal of energy is lost and the object therefore bounces only slightly.

You must also define the **friction** created by an object. Since friction is a function of how smooth or rough a surface is, this property is sometimes referred to as **roughness**. There are two distinct types of friction. The first is **static friction**, which prevents stationary objects from sliding down inclined planes. It is called "static" because it refers to nonmoving, or static, objects. If, for instance, the static friction of a box placed on a ramp is very great, the box will not slide at all (Figure 7-42a). If you decrease the static friction, the box will begin to slide (Figure 7-42b).

The second type of friction, **kinetic friction**, causes moving, or kinetic, objects to come to a stop. For instance, suppose you place two boxes identical in mass on two ramps identical in slope. What causes one box to stop sliding after only a short time is the kinetic friction, or roughness, of the surface of the box (Figure 7-43b). If the other box is made of some very slickly polished material and has very low kinetic friction, it will continue sliding for a longer distance (Figure 7-43a).

Static friction and kinetic friction function independently and therefore can be adjusted independently in most systems. For example, if you set the static friction of a ball rolling on a surface to zero, the ball will slide, rather than roll, along the surface. At the same time, you can adjust the kinetic friction of the ball up or down. If you increase the kinetic friction, the ball will slide (because the static friction is set to zero) and stop quickly (because the

kinetic friction is high). If you decrease the kinetic friction, the ball will slide (because the static friction is still zero) and come to a stop slowly (because the kinetic friction is low).

FORCES AND FIELDS

So far, we have discussed a number of physical properties important for a motion-dynamics calculation. In order to complete the definition of motion dynamics for an object, however, you also need to describe the **forces**, sometimes also called **fields**, that act upon the object. What causes the object to move in the first place?

The most common force that motion dynamics takes into account is the force of **gravity**. Except in very rare circumstances, gravity affects all objects. Even if no other forces cause an object to move, gravity will keep it on the ground. Technically speaking, gravity is the attraction between two objects, caused by the masses of the objects and the distance between them. A ball falls to Earth because the mass of Earth is so much greater than the mass of the ball that the ball is attracted to Earth. This is why a ball in Antarctica will fall "upward" toward the Earth.

From the point of view of most animation, however, you can think of gravity in simpler terms—as a force pulling all objects downward. Consequently, most motion-dynamics systems define gravity as a simple force that pulls all objects downward at a specific rate in the negative Y direction. This force usually exists by default (just as it does on Earth) and affects all objects equally and automatically. Thus, even if you define no other forces and define no other animation, a ball that starts out suspended in the air at frame 1 of a simulation immediately begins falling in frame 2 because of gravity.

The fact that gravity causes objects to fall at a specific rate affects the dimensions you use when modeling objects in a motion-dynamics system. Because the speed at which an object falls is measured in terms of the distance the object travels per second, the measurements of distance you use in the modeling process must be consistent with the measurements of distance used by the motion-dynamics calculations for gravity. If the software you use measures gravity in feet per second, then the units of measurement for your models must also be feet. If gravity is measured in meters per second, then your models must be modeled in metric units. Normally, a motion-dynamics software package sets up default units of measurement, so that the units are consistent in both the modeling and simulation sections of the software. However, if you change the units of measurement, the gravitational force of the motion-dynamics system will not pull objects downward at a physically accurate rate.

Many animators are less interested in physically accurate simulations than in visually interesting animation, and motion dynamics can be used to good

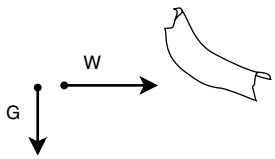

Figure 7-44. Motion-dynamics systems require you to define the forces, such as gravity and wind, that act on objects.

effect for this. If, however, you do want to create a physically accurate simulation, you must use physically accurate numbers for gravity. On Earth, gravity causes falling objects to **accelerate**—that is, gain speed—at a rate of 9.8 meters per second, which is equivalent to thirty-two feet per second.

In addition to gravity, another force that many systems allow you to define is **wind**, which can be very useful, for example, in animating a curtain blowing in a breeze, or the branches of a tree bending slightly in the wind. These are subtle kinds of movement, difficult to model convincingly with keyframing techniques. A wind force is usually defined as a vector having a location, a direction, and a strength. In Figure 7-44, the arrow pointing to the right represents the wind force. The arrow pointing downward is the default gravity force. The curtain model has been defined to have a certain mass. When the wind force hits the curtain, it tends to push the curtain to the right. At the same time, however, the gravity force tends to pull the curtain straight down. Both of these tendencies are affected by the mass of the curtain itself. The result of these forces pushing and pulling against each other is that the curtain rises and falls and flaps "in the breeze."

A wind force is defined in most systems to affect an entire scene equally from a given direction. For example, several curtains in a scene, each located some distance from the others, are affected to the same degree by the wind.

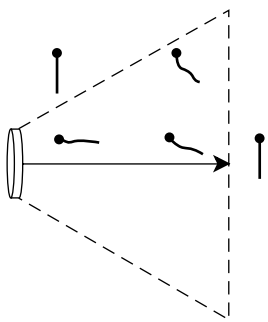

Figure 4-45. Unlike wind, which is omnipresent, a fan force has a limited range.

A variation on the wind concept is a **fan** force. Unlike wind, it is possible to limit both the distance over which a fan force will carry and the radial area that it will affect. In Figure 7-45, the fan force is represented by the cylindrical form on the far left. The arrow emanating from the center of that form and traveling to the right represents the distance over which the force of the fan will carry. At the base of the arrow—that is, at the cylindrical fan icon itself—the force of the fan is strongest. This is why the string close to the base of the arrow is being blown quite a bit. As the force of the fan travels down the arrow, the strength of that force decreases, which is why the next string is not being blown as much as the first. Beyond the tip of the arrow, the force of the fan dies out completely and the string does not move at all.

The force of the fan not only fades as it travels in the direction of the arrow; it also fades as it moves radially away from the center of the fan. The dotted lines emanating from the fan icon demarcate the radial range of effectiveness. The string at the top left, which is completely outside the radial range of the fan, is not moving at all. The string directly to the right lies just inside the radial limit and therefore is being affected slightly. Notice, however, that this string is not being affected as much as the string directly below it, which, even though it is at the same horizontal distance from the fan, is closer to the radial center of the force field.

Turbulence, another force offered by most packages, pushes objects in random directions. The amount of randomness is usually controlled by several

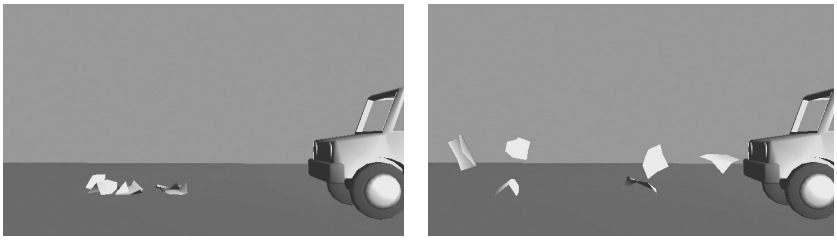

Figure 7-46. Several forces can be applied to each object. Here, vortex, turbulence, and gravity forces are applied to the swirling papers.

noise parameters, while a **magnitude** parameter controls how strong the turbulence is. You can also set an **attenuation** parameter controlling how quickly the force fades with distance. Being at the center of the turbulence subjects you to much more force than being farther away, at its edge. And if you are far enough away, the turbulence will not affect you at all.

Many packages also offer a **vortex** force, which moves objects in a spiraling whirlwind pattern. As with turbulence, you can control the magnitude and attenuation of the force. You can also control the **direction** of spin.

In setting up a motion dynamics simulation, you typically define several forces, some of which, such as gravity, might affect all objects, while others might affect only certain objects. To avoid unnecessary calculations, you **link** each force to specific objects. For example, in Figure 7-46, the papers blown about on the left side of the frames are affected by three forces—a vortex force spinning them in a rising circular pattern, a turbulence force which adds randomness to their motion, and a gravity force which pulls them back to earth. At a later moment in the animation, boxes falling out of the back of the truck are affected by the same gravity force, but are linked to neither the vortex nor turbulence forces (Figure 7-50).

A final force that can be very important in setting up a motion-dynamics simulation is the acceleration an object has as the result of some keyframed action: **keyframed acceleration**. Acceleration is a measure of how fast the speed of an object changes, and the most basic way of giving an object some sort of speed or movement in a three-dimensional animation system is keyframing (described in Chapter 4).

All software packages allow you to combine keyframed animation movements with motion-dynamic simulation. Thus, you might keyframe an animation of a ball translating from one position to another over ten frames (Figure 7-47). This operation involves the standard keyframing techniques of setting a keyframe for the ball as it appears in frame 1, then setting another keyframe for the ball as it appears in frame 10. Having defined the keyframed animation from frame 1 to frame 10, you then direct the simulation software to take over the calculations from frame 11 through frame 40. When the simulation takes over, two forces are at work: the default gravity force, and the acceleration that

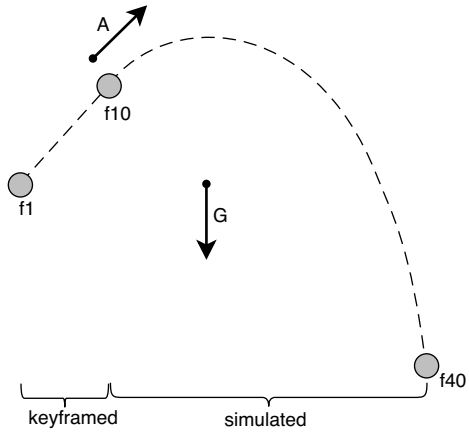

Figure 7-47. Some systems allow you to combine standard keyframed motion with motion-dynamics simulation. Here, the keyframed acceleration of an object is calculated into the motion dynamics.

the ball has as a result of being translated from the position at frame 1 to the position at frame 10. The combination of this keyframed acceleration and gravity causes the ball to continue rising for a while and then to fall.

COLLISIONS

One of the areas in which motion dynamics can be most useful is in simulating **collisions**. The reason the animation of a box falling down a flight of steps (see Figure 7-38) is so complex, for example, is that the box collides with the steps and bounces off them when it does. It is the precise calculation of these collisions, called **collision detection**, that makes a simulation of this animation so effective.

The calculation of collisions, however, can be extremely time-consuming for a computer system, and most motion-dynamics systems therefore allow you to elect whether or not to calculate collisions for a given object. If you know that an object, such as an isolated curtain (see Figure 7-44) won't hit anything in a particular animation sequence, you can turn off collision detection for that object.

If you want to calculate collisions for your model, however, most systems offer several options, all intended to minimize the amount of calculation necessary for a successful simulation. The first option is to define which other objects in the scene might be **obstacles**—that is, which objects the animated object might collide with. When the software does the collision-detection calculations, it will consider only these defined obstacles.

For example, suppose you add a lamppost to the scene in which the box falls down the steps (Figure 7-48). Given the position of the lamppost behind the steps, the box will never collide with it, so you can eliminate it from the list of obstacles that the motion-dynamics software needs to consider in collision-detection calculations. Only the steps, floor, and wall need be considered obstacles.

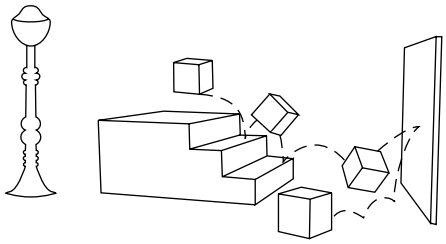

Figure 7-48. Only those objects that the falling box might hit need to be defined as obstacles for that box. This reduces the calculations necessary for collision detection.

Another area in which most systems offer collision-detection options is in the shape of the objects involved. The collision calculations for a complex shape—a telephone, for example—are more complicated than the calculations for a simple shape, such as a box. Consequently, many systems allow you to use a simpler shape as a stand-in for the purposes of collision calculations. Assuming that a falling telephone is shaped like a simple box, for example, greatly speeds up the collision-detection calculations, although you still see the actual telephone falling. In many animations, the accuracy of the collisions calculations does not need to be exact in order to be visually convincing.

Among the number of simplifying shapes that motion-dynamics systems permit is the **bounding box**, or the smallest rectangular box into which an object will fit (Figure 7-49a). If you use the bounding box of the telephone for collision-detection calculations, the resulting motion is similar to what it would have been if you had used the actual shape of the telephone, since the shape of the bounding box and the shape of the telephone are similar. Likewise, some objects can be conveniently and fairly accurately approximated with a **bounding sphere**, or the smallest sphere into which they will fit (Figure 7-49b).

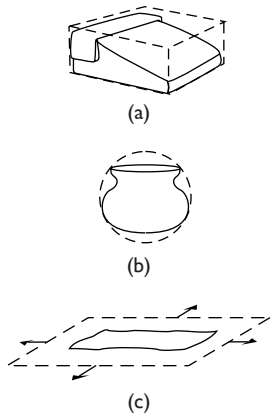

Figure 7-49. Bounding boxes, bounding spheres, and bounding planesÑall represented by broken lines in this illustrationÑcan be used to simplify collision-detection calculations.

Bounding boxes and bounding spheres may be used to simplify either the colliding object itself or the obstacles with which the object will come into contact. Sometimes it is possible to use a **bounding plane** as a simplification technique for obstacles. A flat surface that extends infinitely in all directions, a bounding plane could successfully approximate a bumpy surface, for example (Figure 7-49c).

To control the precision with which your software performs collision detection calculations, you can adjust a **collision tolerance** parameter. The smaller this number, the more precise the collision calculations. More precise calculations, of course, slow down the software.

Obstacles are sometimes stationary, as they are in the staircase example of Figure 7-48. Sometimes, however, you want an **animated obstacle**. In Figure 7-50, the animated truck bed has been defined as an obstacle for each of the boxes. As the truck bed rises, gravity pulls the boxes down, colliding them with the rising bed, and eventually sliding them off the back.

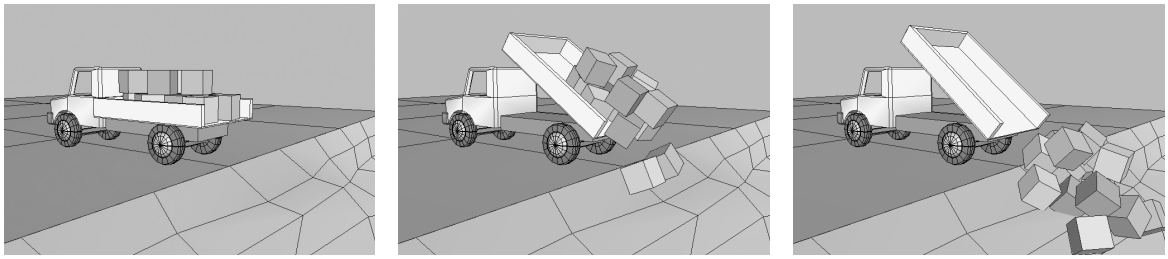

Figure 7-50. Obstacles can be animated with keyframing. Here, the passive-body truck bed rotates upward, causing the active-body boxes to tumble out.

In some motion dynamics simulations, there may be a great many objects colliding with each other, as in the dump truck animation of Figure 7-50. Not only does each of the boxes collide with the truck bed and the ground, but each box can collide with every other box. Collision detection calculations for such a scene can become quite complex and time-consuming, even when the objects are limited, as here, to nondeformable rigid bodies.

In an effort to minimize the complexity of these calculations, most software packages offer a distinction between two types of motion dynamics objects. A **passive body** is one that contributes to the motion dynamics calculations, but itself is not moved by them. Examples of this are the truck bed and ground of Figure 7-50. As passive bodies, both of these objects are obstacles and are therefore contributing significantly to the simulation calculations. Neither of them is affected by those calculations, however. The ground object is not moved at all when the boxes bang into it. The truck bed is moving, but all of its movement is a result of standard keyframed animation, not of motion dynamics calculations. By contrast, an **active body** is an object that not only contributes to the simulation calculations, but itself can be moved by them. In our dump truck example, each of the boxes is an active body. All the movement of each box is purely the result of the motion dynamics calculations. Also, and very importantly, each active body is automatically an obstacle to all other active bodies. Thus, each box can collide with any other box.

ALGORITHMS AND PLAYBACK ISSUES

Motion dynamics algorithms are based on precisely defined principles of physics and mathematics. These have important consequences for the user of motion dynamics software, and it is important to understand several aspects of the underlying science of motion dynamics. Whereas in the real world time is continuous, so that there is no break between one instant and another, in motion dynamics time is treated discretely. That is, time is broken up into a fixed number of measurable increments—for example, thirty increments per second, or one hundred increments per second. These increments are called **time steps**. The smaller the time steps, the more closely they approximate the continuous nature of time, and therefore the more accurately they simulate an animated scene. However, smaller time steps also mean more calculations.

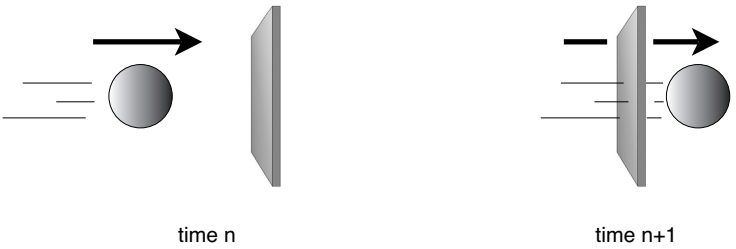

Figure 7-51. If the time step is too large, a fast-moving object might pass through an obstacle in the interval between one time step and the next.

Consequently, you can define the size of the time steps, giving you the option of trading off accuracy of simulation for speed of calculation if necessary.

The size of the time step becomes especially critical when you have objects that are moving extremely fast and should collide with an obstacle (Figure 7-51). If the time step is too large, the object might be on the left side of the obstacle at time step *n*, and have passed right through it as of time step *n + 1*. Not only does this look incorrect, of course, but it can also cause the software to freeze, since interpenetrating objects are impossible according to the physics of motion dynamics. The solution to this situation is to decrease the size of the time step until the calculations succeed. In extreme situations, the time step may have to be decreased so much that the calculations become impractically slow. In this case, you may have to redesign your animation to avoid the problem in the first place.

Because of the mathematics of motion-dynamics calculations, two simulations of a given scene that use different time steps will not produce the same motions. The path of an object can differ substantially when calculated under two different time steps. For this reason you may not find it useful to define a larger time step to sketch out a preliminary version of an animation, with the intention of switching to a smaller time step for the final animation.

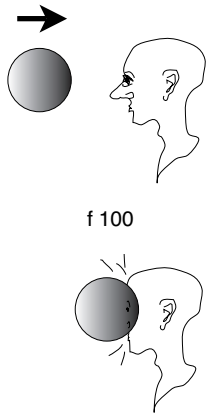

Figure 7-52. An objectÕs momentum at one moment contributes to how it behaves in the next.

Another important issue in motion dynamics algorithms is **time interdependence**, the dependence of every moment on the preceding moment. What an object will do in the next instant depends on what it is doing now. Imagine two situations, as illustrated in Figures 7-52 and 7-53. In both situations the large ball is in exactly the same location at frame 100—just in front of the character's face. In Figure 7-52, the ball is also moving forward very quickly toward the character as of frame 100. In 7-53, although it is in the same location, the ball is stationary at frame 100, hovering in the air. The subsequent actions of the two balls will therefore be very different. The ball that had been moving quickly toward the character in Figure 7-52 will continue to do so because of its **momentum**—the tendency of a moving object to keep moving unless something stops it—and will crash into the character's face in frame 101. By contrast, the ball that had been hovering in mid-air in Figure 7-53 has no such momentum, and will begin dropping straight down at frame 101.

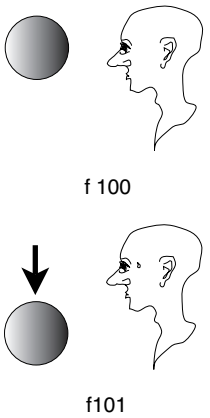

Figure 7-53. The movement of an object at any one moment is dependent upon what it was doing the previous moment.

The dependence of one simulated moment on the previous simulated

moment has important implications for **playback** of a motion dynamics simulation. In order for motion dynamics software to know the positions of objects at frame 101, it must know where they were, and how fast and in what direction they were moving at frame 100. But in order to know that for frame 100, it must first know the same information for frame 99. In short, in order to know the positions of simulated objects at any given frame, the motion dynamics software must go all the way back to frame 1, and recalculate the simulation from the beginning each time.

Different software vendors offer different ways around this problem, as we will see in a moment, but the implications are significant in any motion dynamics software. One implication is that you cannot drag your frame counter through your timeline to view a given frame. If you do so, you will skip some frames as you drag, and the calculations will be inaccurate. Also, when playing back your animation, you cannot force the playback to synchronize to a specific playback rate—for example, video or film. If you do, the playback will skip some frames, and therefore cause inaccurate simulation calculations. And finally, when rendering your frames, you cannot pick a frame in the middle of your sequence and start rendering there. You must always start at frame 1 and render every frame in sequence from there.

Clearly, these restrictions are extremely inconvenient for the animator. As a consequence, motion dynamics packages offer several solutions. One solution is to create temporary **cache data** in RAM. The word *cache* means to hide or store, so cache data is data that has been stored for future use. When you turn on this option, your software will take longer to calculate the simulations the first time through, because it will also be storing this information in RAM. Once the simulations have been calculated, however, the cache data allows you to view your simulation as if it were a normal animation—that is, you can scrub through the timeline, and you can play back the animation, synching it to a particular playback rate. If you make any changes to your simulation setup, however, your must delete the old cache data and calculate new data. For example, if you reposition one of your objects, or change the mass of another object, you must delete the old cache data and recreate it.

A related issue is that, since cached data is stored in RAM, it is not saved permanently with your files on your hard drive. This means that when you return to work on your file the next day, the cache data will have been lost and will have to be recalculated. It also means that if you try to render your frames in several stages—for example, starting at frame 1, then restarting the next day at frame 600—you will have the same problem. To address this, most packages allow you to create a **permanent cache**, sometimes also called, rather picturesquely, a **baked simulation**. In this case, cache data is written to the hard drive and saved with your scene file.

Permanent cache data stores the positions of every object at every frame

as **raw data**—that is, as a keyframe at every frame. This can result in data files that are quite large. There are several advantages to creating such permanent cache data, however. Once you have created permanent cache data, you can, if you choose, delete all your motion dynamics elements—forces, physical properties, and so on—because the permanent cache data has stored the positions of every object at every frame. Very importantly, you can also view the raw data curves in your parameter curve editor (section 4.4), where you can thin the curves and then edit them as you would any other parameter curve.

All the motion-dynamics techniques discussed in this section are very powerful and can yield stunningly realistic simulations of an event. Sometimes, however, the most effective thing you can do is to combine the more standard, keyframe-based animation techniques with these sophisticated motion-dynamics calculations. For example, the human figure in Figure 5-13 was animated using standard keyframing and hierarchical animation. The bowling balls, on the other hand, were animated using motion dynamics to achieve a high degree of realism as the balls flew through the air and bounced on the ground about the feet of the character.

7.7 Soft-Body Dynamics

In the previous section we discussed the principles and concepts common to all motion dynamics, and focused on how they are applied to rigid bodies, objects whose shapes do not deform. When motion dynamics calculations are applied to an object whose shape can deform, such an object is called a **soft body**. For a soft body, as for a rigid body, you define physical properties such as mass and elasticity. You also define forces such as gravity or wind, as well as friction and collision obstacles. In addition to all these parameters, there are also several other important issues related specifically to the deformability of soft bodies. We focus on these issues in this section.

Soft-body dynamics can be used for many applications. One example is a soft rubber beach ball falling under the force of gravity, colliding with a surface, and deforming as it bounces (Figure 7-54). Another example is a flag waving in the wind, with the wind and gravity causing the fabric of the flag to ripple in the breeze. A third example of soft body dynamics might be a modeled lock of hair on a CG character's head, with the up and down movement of his walk causing the lock of hair to bounce up and down.

We saw in section 7.6 that when a rigid body moves in a simulation, the motion dynamics calculations act upon the object as a whole. If a cube is bouncing down a stairway, as it is in Figure 7-38, the motion of the entire cube as a unit is simulated, and the entire cube as a unit bounces off the stairs.

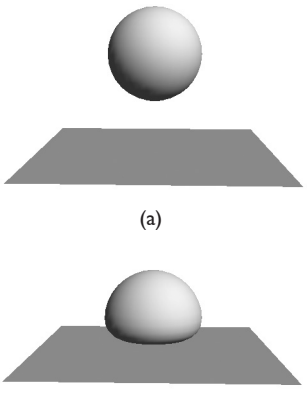

Figure 7-54. A soft body is a deformable object moving under motion dynamics.