

Game Sprite Animation Tutorial

By Ari Feldman
ari@yogicflyer.com

Excerpted from Chapter 9 of my book *Designing Arcade Computer Game Graphics* by WordWare Publishing, Inc. (ISBN: 1-55622-755-8).

What is Animation?

Animation is the process that produces the illusion of movement. It works by displaying two or more image fragments called *frames* (also commonly referred to as *cells*). When these frames are displayed in rapid succession with subtle changes made to their content, our eyes register these changes as movement.

Animation is not a mystical art. Rather, it's a well-established process that combines the aesthetics of design with real-world physics in order to breathe life into what are otherwise static objects and scenes. This chapter will introduce the fundamental concepts behind animation to you so that you can create and implement animation in your own arcade game projects.

Animation Properties and Fundamentals

To be able to create effective animation, you must learn how to divide the elements of motion into their basic components. This means breaking them down into a sequence of easy-to-follow frames. However, before you can do this, you must first learn and master two things: the art of observation and the characteristics of basic motion.

The secret behind creating great animation lies in keen observation and the ability to focus on the subtle details of how different objects move. Every object exhibits certain peculiarities as it moves. Some of these idiosyncrasies are slight while others are more pronounced. As such, there are several characteristics of motion that you should be aware of before attempting to animate an object. These characteristics include such things as:

- Motion lines
- Motion angles
- Key-frames and in-betweens
- Weight and gravity
- Flexibility
- Secondary actions
- Cycles and loops
- Tempo

Motion Lines

A *motion line* (sometimes referred to as a *natural path*) is an invisible line created by an object as it performs a series of sequential movements.

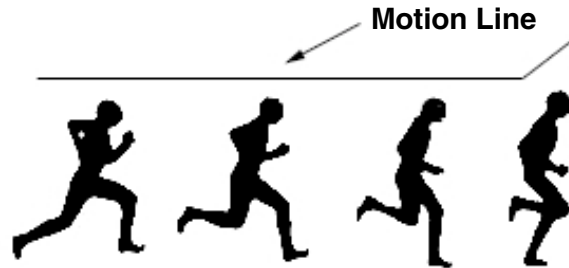


FIGURE 9-1: Motion Line Example

Motion lines are essential to creating effective animations, and manipulating the motion line can add realistic emphasis to animated objects. For example, you can create very smooth animations by making small alterations to the motion line. Conversely, you can produce very dramatic animations by making large or exaggerated changes to the motion line.

Even more interesting to the animator is how objects produce different shaped motion lines depending on how they move. For example, a bullet has a motion line that is straight and even while a bouncing ball has a motion line that is wavy and uneven. This being said, motion lines must be consistent with an object's real-world behavior in order to produce realistic-looking animation. Otherwise, the quality of the animation will suffer. Therefore, you can use an object's motion line as a means of determining whether or not it is being animated correctly and convincingly.

The best way to follow an object's motion line is to locate its *center of gravity*. The location of the center of gravity varies according to the type of object involved.

To help you accomplish this, Table 9-1 provides some examples of where the center of gravity is for a number of common objects. Using this information, you should then be able to identify the center of gravity for other types of objects.

TABLE 9-1: Location of the Center of Gravity in Different Objects

Object Type	Estimated Center of Gravity
2-legged animals	Head
4-legged animals	Chest
Flying animals	Torso
Humans	Head
Insects	Torso
Spaceships or airplanes	Hull or fuselage
Tracked or wheeled vehicles	Turret or body

Motion Angles

Motion angles are one of the most obvious clues as to an object's direction as it moves. It's important to point out that there is a direct relationship between an object's direction and its motion angle. Almost any change in an object's speed or direction will require a similar adjustment to its motion angle. Therefore, the sharper the motion angle, the faster or the more extreme the change in the object's motion or direction will be.

Motion angles are particularly useful for conveying a sense of realism in animated objects. For example, a jet fighter making a steep bank will have a motion angle that is sharper than a jet fighter that is flying straight and level as shown in Figure 9-2. In this case, you can use its motion angle to visually discern that it is traveling at a high speed, which ultimately helps to reinforce the illusion of realism.

Although the actual location of motion angles varies, most motion angles are located along the spine of an object, i.e., the backbone of a human being or the hull of a spaceship.

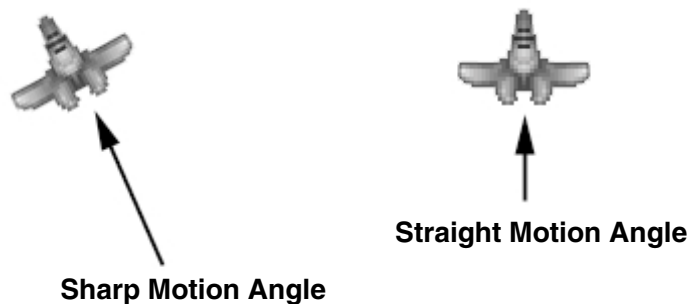


FIGURE 9-2: Motion Angle Example

Key-frames

Most people are aware of the extremes that occur during movement, i.e., such noticeable things as the flapping of wings or kicking of legs. In animation, we refer to these actions as *key-frames*.

Being able to determine which frames in an animated sequence are the key-frames is an extremely important part of the animation process. This is because key-frames serve as the framework for the entire animation sequence.

In addition, there is a direct relationship between the number of key-frames used and how smooth a particular animation appears. The presence of more key-frames in an animated sequence means smaller incremental changes in the animation and

results in smoother overall movement. Having fewer key-frames present, on the other hand, results in coarser and jerkier animation. Please keep this very important relationship in mind as it has a direct effect on the quality of any animation you create.

Key-frames are most effective when they incorporate very exaggerated or dramatic elements, since these actions can be used to emphasize the most critical movements in an animated sequence. In addition, exaggeration can help you to better determine the most effective starting, middle, and ending points for an animated sequence. For example, Figure 9-3 shows the two key-frames for a bird flying. Notice how they mirror the two extreme states of the action, i.e., the wing moving up and the wing moving down. These two frames are extremely exaggerated, which makes them ideally suited as key-frames because their differences are clearly distinguishable to the observer.



FIGURE 9-3:
Key-frame
Example

It's important to realize that the more key-frames a particular animation has, the more complex it is and the longer it will take to design. This is certainly something to consider when designing the artwork for an arcade-style game, especially when working under a tight deadline. In addition to taking longer to create, complex animations make it easier to introduce errors and mistakes into the animation sequence, which can have a negative impact on the animation's overall quality and effectiveness.

In reality, however, the actual situation dictates which approach to take and how many key-frames to use for a given animation.

There will be instances where you can get away with using fewer key-frames than in others. In most cases, you can use as few as two or three key-frames per object in arcade-style games, with little or no detrimental impact. However, you should look at each game on a case-by-case basis before deciding on a particular number of key-frames to use. If you don't, you run the risk of reducing the quality of your game's animations and, ultimately, the quality of the game.

Please refer to Table 9-2 for general suggestions on key-frame usage in different arcade game genres.

TABLE 9-2: Object Key-frame Quantity Suggestion Chart

<i>Arcade Game Type</i>	<i>Key-frame Usage Suggestions</i>
Pong games	2 per animated object
Maze/chase games	2-3 per animated object
Puzzlers	2-3 per animated object
Shooters	2-4 per animated object
Platformers	2-6 per object

NOTE: These are subjective estimates only. Each game situation will be different and will depend on the design time, the game's target platform, and the overall level of animation quality you want to achieve.

It's important to note that key-frames can occur at any point within an animated sequence. However, certain factors such as the type of animation involved and its relative complexity can influence where in the sequence they might actually appear. For example, non-repetitive motions such as explosions have many key-frames and tend to be located at several points within the animation sequence or wherever there is a major change. In comparison, repetitive motions such as walking or flying have only a few key-frames (i.e., two or three). These tend to be distributed at the start, middle, or end of the animation sequence.

After the key-frames of the animation are identified and established, the *in-between* frames must then be added. In-betweens are frames of animation that are used to smooth out the transition between individual key-frames.

In animation, key-frames are important for defining the object's primary movements, while in-betweens are responsible for making the entire animation look smooth and convincing. Introducing slight or subtle changes to each frame between key-frames creates in-betweens. This process is also known as *tweening*, and great care must be taken to ensure that these changes are small in order to maintain the illusion of continuous and realistic movement.

Figure 9-4 shows an example of how in-betweens co-exist with key-frames. This example shows an animation of a man swinging a stick. The start of the sequence shows the man with the stick up at the shoulder and ready to swing, while the final frame shows the end of the swinging process with the stick fully extended. These are the key-frames of the animation while the frames that occur in-between them are the in-betweens of the animation.

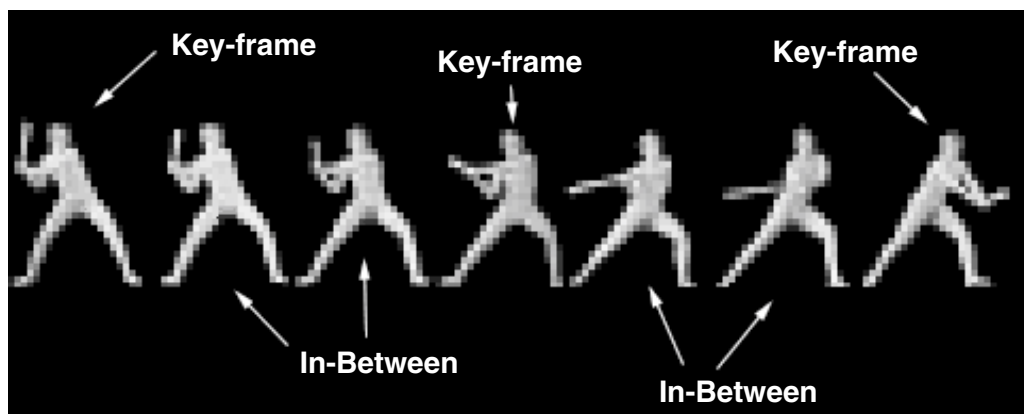


FIGURE 9-4: Key-frame and In-Between Example

Creating in-betweens is one of the most tedious and time-consuming aspects of designing arcade game animation. Fortunately, most painting programs allow you to easily duplicate existing frames so that each in-between doesn't have to be re-created entirely from scratch. In traditional animation, the process of duplicating an existing frame of animation to create a new one is called *onion skinning*.

Determining the number of frames necessary for creating a particular type of animated movement takes time and experience to figure out. To help you on your way, Table 9-3 lists the number of frames required to produce a variety of different animation effects. Use these as a general guide if the issue is ever in question in your own game projects.

TABLE 9-3: Common Frame Animation Frame Requirements

Object	Minimum # of Frames	Maximum # of Frames
4-legged animal running	4	16
Animal biting	2	5
Crawling	2	12
Explosions	5	16
Falling	3	5
Flying	2	12
Jumping	2	10
Kicking	2	6
Punching	2	6
Rotating/spinning	4	16
Running	2	12
Swinging (an object)	2	8
Throwing (an object)	2	6
Vehicle flying	2	4
Vehicle moving	2	8
Walking	2	12

Weight and Gravity

If you intend to create realistic-looking game animations, you must consider how the elements of weight and gravity can affect your work. When designing animation, you must remember that real-world physics should always apply to the sequences that you create. One of the most important of these influences is that of weight. Weight affects speed. So, for example, the larger the object, the heavier it is and the slower it is likely to move.

In addition to influencing the speed at which objects travel, weight can also affect how easily an object can move. For example, think about how fast a racecar moves in comparison to a battle tank. The tank will plod across the ground whereas the racecar will quickly skim across it. This is because the racecar weighs less.

Then there's the issue of gravity. Gravity also influences the motion of objects. In the real world, gravity will exert more force (resistance) against heavier or denser objects than it will against lighter ones. To see how this works, consider how quickly a bomb falls from the sky when compared to a feather, or how a rock bounces off the ground in comparison to a rubber ball. Remember that people aren't always easily fooled. Believe me, they are well aware of such things and failing to account for these behaviors in your animations can have a negative impact on how they are perceived in a game.

Incidentally, the principles of weight and gravity can be applied to virtually any animated object. Therefore, the more attention you pay to them, the more realistic your animations can be.

Flexibility

Flexibility is essential to producing convincing and fluid animation, particularly when depicting complex, jointed objects such as animals, insects, and human beings. Animations without proper flexibility can appear stiff and rigid, which for these types of objects is a less than desirable effect.

The key to adding flexibility to your animations is careful planning coupled with careful observation of how different objects behave when they move. Whenever you animate a jointed object, you must always determine which parts of the object (i.e., arms, legs, etc.) lead the movement and which ones follow it. It's very important to realize that not all body parts actually move at the same time on all objects. For example, consider how a swordsman might slash with his sword. The swordsman leads with his legs first to gain a solid footing and then follows through with his arm to complete the cutting motion. In animation, this flow of movement is called the *range of motion*.

When attempting to incorporate flexibility into an animation you must take care to account for the limits of anatomy. That is to say, never attempt to unrealistically extend the available range of motion that a given object has. Don't depict objects moving in ways that aren't possible in the real world. Examples of this might include, but aren't limited to, such things as bending a joint backward or in a position that is not normally possible for one to make.

Secondary Actions

As mentioned in the previous section on flexibility, not all parts of an object move simultaneously when animated. There are parts that lead the flow of movement and parts that follow it. The parts that follow the movement are called *secondary actions*.

Secondary actions are extremely important to the animation process because they add an extra dimension of realism to animations. Essentially, anything that is free moving can qualify as a secondary action. This includes everything from hair and clothing to eyes and lips. So, for example, if a character in a game is wearing a cape and walking, the secondary action of this action would occur when the character's cape bounces and sways as the character moves. Figure 9-5 shows this in frames 1 and 2.

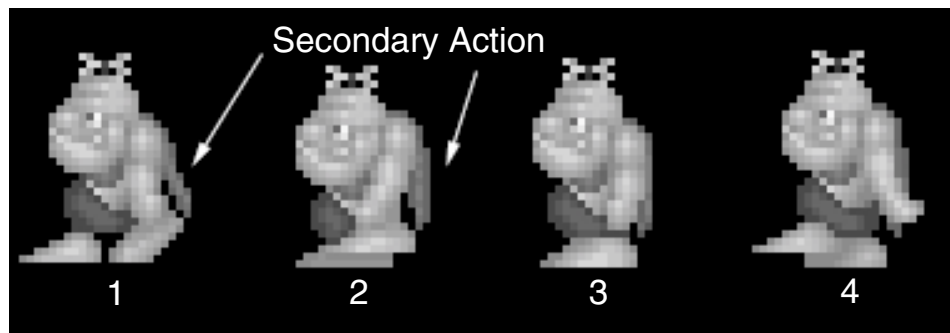


FIGURE 9-5: Secondary Action Example

Secondary actions are not limited to small details such as clothing or hair. It's important that you understand that they can be virtually anything in an animation sequence that isn't the main focus.

Cycles and Loops

In animation, *cycles* are the repetitious movements that many animated objects make when displayed in a sequence. As your ability to observe how objects move in nature improves, you will soon discover that many objects include cycles in their movements.

Cycles can be considered the time savers of the animation process. They help us avoid the tedium of constantly having to re-create frames to construct basic actions. For example, without cycles you might have to draw hundreds of frames of animation to show a bird flying across the screen. However, by using cycles, you simply have to identify the object's repeated motions and display them instead. So, if your bird animation used 30 frames to display the complete animation sequence you would only have to draw the three or four frames that best

represent the major points of movement. Although key-frames usually fall into this category, it's important to realize that there isn't always a direct one-to-one relationship between key-frames and cycles.

While cycles focus on repeating the occurrence of specific frames within an animation sequence, *loops* emphasize the repetition of the entire sequence as a whole. For example, an animated sequence of a man walking goes through a number of frames to produce the illusion of movement. By looping the sequence, you can simulate the effect of constant motion so that the walking sequence appears to be continuous. Therefore, it can be said that loops help us to keep animated movement constant.

However, this being said, it's important to understand that not all objects require constant motion while being animated. Loops are a device to help us achieve this but they shouldn't be used in all animations. In fact, quite a few types of animation don't rely on loops at all.

In addition to adding realism and continuity to your animations, looping can also save you time by preventing you from having to manually repeat the individual frames of the entire animation sequence.

Figure 9-6 shows an example of how cycles and loops work together in animations. Here, frames 1 and 5 are cycles because they are repeating the same frame of animation. When the sequence reaches frame 5, it would loop back to frame 1 to create the illusion of continuous movement.



FIGURE 9-6: Cycle and Loop Example

Tempo

Every animated object can display at a specified *tempo*, or speed. Tempo can be used to control the rate at which entire animation sequences are shown as well as the speed of the individual frames that comprise the sequence.

Tempo is important because it allows us to create more realistic-looking animation sequences. Its usefulness becomes immediately apparent when you consider that most real-world objects move at different rates during the course of their movements. For example, consider the average marathon runner. When running, a marathoner's legs move faster during mid-stride than they do when they are fully extended. You can account for this fact in your own animations by using the distance of the object between successive frames as a means of depicting animations

moving at varying speeds. For example, the animation of a jumping character would have objects in frames that are evenly spaced during the part of the jump sequence that is moving at a constant rate. When the sequence begins to slow, the objects in the frames that would appear closer together. When the sequence speeds up, the objects in the frames would move farther apart. The basic concept is illustrated in Figure 9-7.

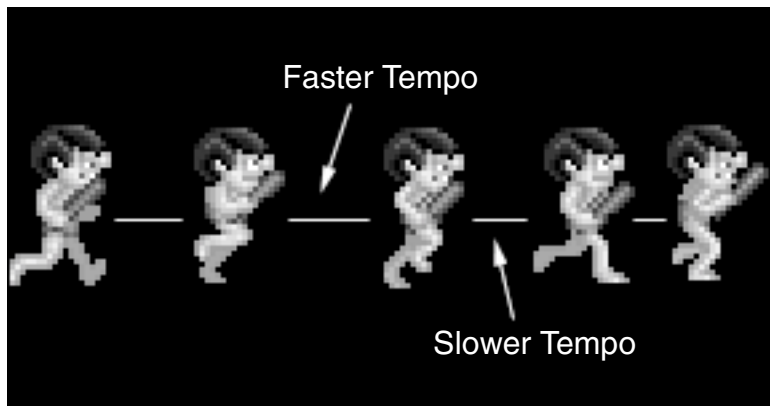


FIGURE 9-7: Tempo Example

We measure tempo in *frames per second (FPS)*, or the number of frames that can be displayed per second of time. The human eye can perceive movement (or the illusion of it) in as few as 12 frames per second. Generally speaking, the higher the frame rate (FPS), the smoother the animation. Therefore, it is preferable to display most forms of animation at a tempo that is greater than 12 frames per second.

Table 9-4 compares some of the more common animation frame rates for different forms of animation.

TABLE 9-4: Comparisons of Common Animation Frame Rates

Animation Type	Frame Per Second (FPS)
Computer video	15
Fast-action arcade game	30
Motion picture	24
Television	30

Unfortunately, FPS is not a universal constant when it comes to displaying animations on computers. Several factors, including the physical size, the number of frames involved, and the speed of the computer, can all adversely influence the rate at which animated sequences display. This is particularly true of the

animation that appears in computer games, as they tend to really tax the systems that they are played on.

Table 9-5 provides some suggestions on the common frame rates for each type of arcade game.

TABLE 9-5: Common Arcade Game Frame Rates

Arcade Game Type	Common Arcade Game Animation Frame Rates (FPS)
Pong games	15-30
Maze/chase games	20-30
Puzzlers	9-15
Shooters	20-50
Platformers	20-30

Most game developers consider 15 FPS to be the minimum acceptable frame rate for fast-action arcade games and consider 20 or 30 FPS to be a desirable frame rate. Certain arcade game genres demand higher animation frame rates than others. This is because the fluidity of their animation enhances the overall user experience.

Sprites

Sprites are special graphic objects that can move independently of the screen background. Sprites are used in arcade games to represent spaceships, bombs, aliens, soldiers, and so on. In addition, they can be either animated or static and can also be used to represent a variety of fixed game objects such as treasure and power ups as well.

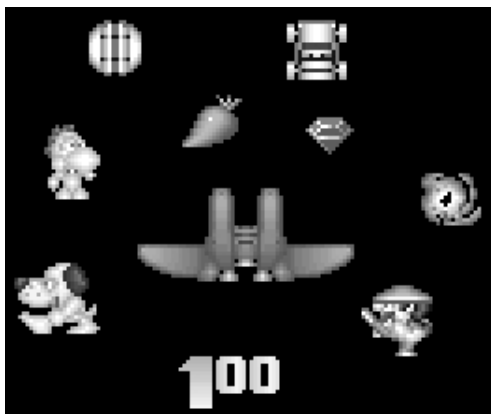


FIGURE 9-8: Sprite Examples

Sprite Properties

Sprites are used extensively by arcade games and have a number of interesting and distinct properties, including:

- Variable sizes and shapes
- Free range of movement
- Separate from background

Variable Sizes and Shapes


For the most part, sprites have no limits to either their size or shape. This being said, sprites can have rectangular, square, or even irregular shapes—it really doesn't matter. This versatility makes sprites useful for depicting virtually any type of object an arcade game may require. Sprites can also be of any size and they can change their size to respond to specific game events or actions as needed.

Free Range of Movement

Unlike other types of animated objects, sprites can move freely about the screen. This means that they are not restricted to displaying at specific areas of the screen. This characteristic makes sprite animations very effective in representing all types of moving game objects.

Separate from Background

Sprites exist as separate entities from the background of the screen. Sprites also support transparency, which, if you recall our discussion on transparency in Chapters 7 and 8, allows the background of an object to show through the foreground. This feature makes sprites particularly useful when used in games that have complex background screens since the contents of the background can be preserved as the sprite moves over it.

 **NOTE:** Practically any graphic object can be used as a sprite. As a result, sprites can be created using most of the painting programs mentioned in Chapter 6.

Despite their unique advantages, sprites are among the most challenging aspects of creating arcade game graphics. This is because designing most sprites involves a complex two-step process. First, the sprite must be created just as any other graphic image. However, unlike most graphic objects, sprites tend to face more restrictions in terms of their size and use of color, which can complicate their design and increase the time required to create them. Then, after doing all of that

work, the sprite needs to be animated, which, as you will soon see, is a very complicated process.

Grid Squares

As previously mentioned, the size of the sprite being animated can also impact the speed at which a sprite is animated. Smaller sprites will always animate faster than larger sprites, especially when there are lots of objects being displayed on the screen at once. This is because the computer has to manipulate less data with smaller sprites than it does with larger ones. Because of this, you should limit the size of your sprites whenever possible. One of the best ways to do this is to create your sprites in predefined size using a grid as a guide. Grid squares are miniature “screens” on which to draw sprites. As such, all grids have an *origin*, or a starting reference point. The origin helps us pinpoint the location of individual pixels within the grid when designing and editing individual sprite images. Like a graph, the origin of a grid square always starts at position (0,0), the position on the grid where X and Y are both equal to 0.

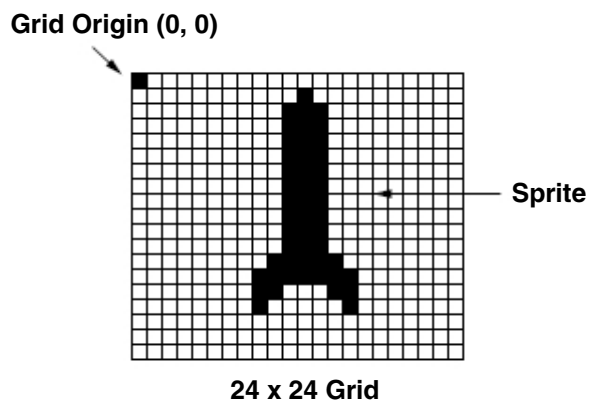


FIGURE 9-9: Grid Origin

Grid squares offer us a number of important advantages when creating sprites, including:

- Maintaining size consistency
- Assisting the animation process
- Optimizing sprites for implementation in games
- Optimizing sprites for screen performance

Maintaining Size Consistency

Grid squares are very useful for helping us to place constraints on the sizes of the sprites we create. This allows us to focus on packing as much detail as possible into the grid space that is available. As most sprite grids have visible borders that indicate their boundaries, this feature allows us to create sprites that are consistent and uniform in size. Keeping the sizes of your sprites consistent helps to improve their quality and better facilitates their incorporation into a game.

Assisting the Animation Process

Grid squares can also be quite useful for assisting us with the sprite animation process. This is due to the fact that grids allow us to arrange and organize sprites in a more consistent and predictable manner. This in turn makes them easier to manipulate than if they were haphazardly arranged on the screen. For example, you can use grids to arrange your sprites in the sequence you want an animation to appear. This makes it easier to work with your objects and improves your overall efficiency.



FIGURE 9-10: Example of Grid Containing an Animation Sequence

Optimizing Sprites for Implementation in Games

Because grid squares allow sprites to be easily arranged on-screen, they also make it possible to optimize the process of adding sprites to games. For example, many game development tools support the ability to import batches of sprites at one time. By arranging your sprite grids in a particular sequence and position inside a larger graphic image, you make it much easier to include them in a game, especially when dealing with large numbers of similarly sized objects. This can wind up saving you countless time, hassle, and effort during the course of a game project.

Optimizing Sprites for Screen Performance

Finally, grid squares offer programmers the opportunity to optimize their games around certain sprite sizes, which contributes to the overall screen performance of a game. You can find more information on this particular issue by referring to Chapter 2.

Table 9-6 highlights some of the more common sprite grid sizes used.

TABLE 9-6: Common Grid Square Sizes at Different Screen Resolutions

Screen Resolution	Common Grid Sizes
320x240	8x8, 16x16, 32x20, 32x32, 64x64, 96x64
640x480	16x16, 32x20, 32x32, 64x64, 96x64, 96x128, 128x128, 256x256
800x600	16x16, 32x20, 32x32, 64x64, 96x64, 96x128, 128x128, 256x256

NOTE: These sizes change in proportion to the screen resolution in which they are displayed. For example, an object animated at 16x16 at a resolution of 320x240 will appear as if it were created at 8x8 when shown at a resolution of 640x480. You can easily draw grid squares by using the Grid tool of your favorite painting program. This tool is described in more detail in Chapter 5.

You may have noticed that most of the common sprite grid square sizes are based on even multiples. This is intentional. In addition to helping with screen performance, evenly sized sprite grids also simplify the programming process because they allow for cleaner and more optimized math in programs. This being said, this doesn't actually mean that the sprites you design inside these grids have to be evenly sized. For example, a given grid square might be 32x32 but the sprite inside the grid square might actually measure 29x27.

Table 9-7 provides some examples of how these different grid square sizes might be used for different types of arcade game objects.

TABLE 9-7: Example Grid Square Sizes for Different Game Objects

Grid Square Size	Comments
8x8, 16x16	Useful for small objects such as bullets or missiles. Can also be used to represent on-screen text characters at lower screen resolutions (i.e., 320x200).
32x20, 32x32	Useful for objects such as spaceships, bonus items, icons, and other small on-screen objects in most screen resolutions.
64x64, 96x64, 128x128	Useful for larger spaceships and vehicles as well as most on-screen objects in most screen resolutions.
Greater than 128x128	Useful for very large objects, namely "boss" objects, or major characters that frequently appear at the end of game levels, etc.

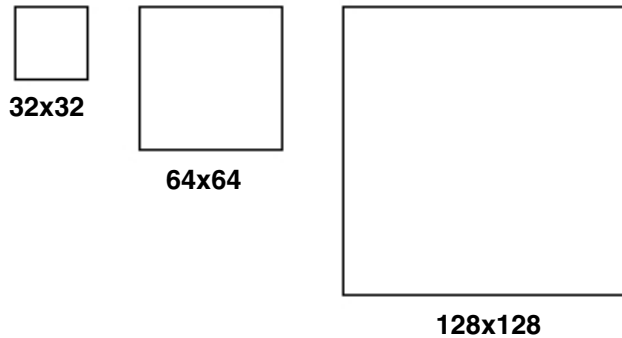


FIGURE 9-11: Comparison of Different Grid Square Sizes

Table 9-8 provides some suggestions for using sprite grid sizes for different types of arcade games.

TABLE 9-8: Suggestions for Using Grid Square Sizes in Different Arcade Game Genres

<i>Grid Size</i>	<i>Suggested Arcade Game Genre</i>
8x8, 16x16	Pong games, maze/chase, shooters
32x20, 32x32	Pong games, maze/chase, shooters
64x64, 96x64, 128x128	Pong games, maze/chase, puzzlers, shooters, platformers
Greater than 128x128	Shooters, platformers

Table 9-9 shows the grid square sizes that are most useful at different screen resolutions.

TABLE 9-9: Using Grid Square Sizes at Different Screen Resolutions

<i>Grid Size</i>	<i>Suggested Screen Resolution</i>
8x8, 16x16	320x200, 320x240, 640x480
32x20, 32x32	320x200, 320x240, 640x480, 800x600
64x64, 96x64, 128x128	320x200, 320x240, 640x480, 800x600
Greater than 128x128	640x480, 800x600

NOTE: The information presented in Tables 9-7, 9-8, and 9-9 is to be used as general guidelines only. You are certainly free to use grid sizes other than the ones mentioned here.

General Rules for Creating Grid Squares

There are a few items to consider before going to the trouble of designing your artwork using a specific grid square size. These issues include:

- **Programmer requirements**—In order to maximize screen performance, some games use sprites that are hard-coded to certain sizes. Therefore, never choose a particular grid square size to create your sprites without first consulting with your programmer! Otherwise, you're likely to create sprites that are incompatible with the game's animation code and will probably have to redo a large portion of your work.
- **Grid square size in relation to other objects**—During your initial graphics work, you may find that you need to make certain game objects larger or smaller to better fit the overall look and feel of the game. Therefore, always test the grid square size you choose before using it for all of your sprites. To do this, simply create a few key objects using the selected grid square size and see how they fit together. Place the objects side by side and in relation to the rest of the items in your game, i.e., backgrounds, status indicators, etc. After a few quick tests, it should become clear whether or not a particular grid square size will work for your game. Taking this extra step can save you a lot of extra time and effort later on.
- **Proper grid square scaling**—It helps if you use grid squares that are divisible by a power of two. Doing this has several advantages. First, it allows your sprites to be compatible with all common screen resolutions. Second, it facilitates their placement on the screen, as all common screen resolutions are also divisible by a power of two. Third and finally, doing this allows you to scale your sprites up or down to other sizes as the need arises with minimal loss in quality.

One last word about sprite grid sizes: be very careful when creating them! When the programmer specifies a grid size of 32x32, find out if the 32x32 dimensions include the grid border or just the contents of the grid cell. This detail is often overlooked and can result in game objects that do not fit together properly, such as background tiles.

Therefore, to be safe, always make your grids one pixel larger than what is specified in both the X- and Y-axes. For example, when tasked with making objects that are 40x40 in size, create a grid that is 41x41, as this will account for the grid's border.

Core Arcade Game Animation Primitives

Animated objects are at the heart of every arcade-style game. They represent the stylistic sequences that comprise everything from on-screen characters that walk,

run, and jump to spaceships and special effects such as flashes and explosions. It's these objects that ultimately make arcade games engaging and appealing to those who play them. Yet, despite all of the outward differences each arcade game exhibits in terms of their looks, the animations that give them unique character all share a common set of *primitives*, or techniques of producing animated sequences. This is a little-known fact but an extremely important one. Once you understand how these arcade animation primitives work, you will have an important insight into how arcade game animations are designed and created.

The purpose of this section is to highlight these core animation primitives and explain both how they work and how to use them effectively in your own arcade-style game projects.

For the purposes of this book, these key animation primitives can be grouped into three general categories. These categories include:

- Major arcade game animation primitives
- Minor arcade game animation primitives
- Complex arcade game animation primitives

Major Arcade Game Animation Primitives

This category includes seven of the most basic animation primitives that are used in nearly every arcade game. The techniques described here are relatively simple in nature and are applicable to both character and mechanical objects of all types. They include:

- The cylindrical primitive
- The rotational primitive
- The disintegration primitive
- The color flash primitive
- The scissors primitive
- The growing primitive
- The shrinking primitive

The Cylindrical Primitive

This primitive is used in arcade games to represent the spinning motion of round, cylindrical objects. Although this doesn't sound like much, you'd be surprised. In arcade games, these objects can be used to represent a diverse and extensive range of elements including ship's hulls, buildings, missiles, robots, and even car wheels.

The cylindrical technique is among the easiest animation techniques to master because unlike most forms of animation, it doesn't require any dramatic changes

to occur between frames to produce its intended effect. Rather, cylindrical animations rely on points or lines called *markers* that change gradually from one frame to the next as illustrated in Figure 9-12.



FIGURE 9-12: Cylindrical Animation Example

NOTE: Technically, this animation sequence can be completed in as few as four frames. However, the example in Figure 9-12 shows five in order to emphasize the full motion of a typical cylindrical effect.

When animated it will produce the illusion of the object rotating in place. Here's a quick breakdown of the motion that's taking place:

- **Frame 1:** The marker (the horizontal line) is positioned near the top of the object.
- **Frame 2:** The marker moves down slightly from the previous frame.
- **Frame 3:** The position of the marker moves to the center of the object. This provides an unmistakable visual clue that the object is starting to rotate since the marker has made a dramatic and visible positional change since frame 1.
- **Frame 4:** The marker moves down past the center. If you look carefully, you'll see that frame 2 and frame 4 are exact opposites of each other in terms of the position of the marker.
- **Frame 5:** The marker moves to the bottom of the object. At this point, a single rotation has been completed and the animation is ready to be cycled.

Creating effective cylindrical animations requires that you pay close attention to three things: color selection, proper positioning of the marker, and sequence length.

Because it doesn't rely on the broad or exaggerated changes used by the other forms of animation, the cylindrical primitive requires that you pick suitable colors to support the effect. These colors should be high contrast shades with the marker color being the most prominent of them. Making the marker brighter than the rest of the object serves two important purposes. First, it establishes the point of change on the object. This cues the user's eyes to follow the object as it is animated. Second, it serves as a highlight that reinforces the object's illusion of roundness.

The position of the marker can make or break a cylindrical animation. The marker should always travel from one end of the object to the other in a smooth and predictable fashion. If it doesn't, the illusion of motion won't be properly established

and the animation will fail. To ensure that the marker is always positioned correctly, move it in gradual increments as this will give you more control over its progression along the surface of the object and give you the opportunity to correct it during the design process. The correct positioning of the marker is one of the more difficult aspects of creating this type of animation. To help give you a better sense of how to do this, look at a real-world object such as a large coin. Turn the coin on its side and roll it in your fingers. Notice how the light moves along the edge of the coin. Move the marker in the same fashion and the animation will produce the desired result.

The effectiveness of cylindrical animations is also greatly influenced by the number of frames used to create the effect. To ensure that the animation is successful, plan on rendering the cylindrical animation using between three and ten frames. In my experience, seven works best, but more can be used without drawing out the effect too much. Never use less than three frames to represent any cylindrical animation since there won't be enough frames available to adequately show the flow of movement.

TABLE 9-10: Cylindrical Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Cylindrical animations should never have an uneven (non-linear) motion path. Cylindrical animations always have straight motion lines.
Motion angles	Motion angles are always straight.
Key-frames and in-betweens	Three to four key-frames with up to three in-betweens produce the smoothest and most effective cylindrical animation sequences.
Weight and gravity	These properties <i>may</i> influence the speed at which a cylindrical object rotates if it's large or heavy.
Flexibility	Does not apply and has no effect.
Secondary actions	Has no effect as all parts of the object move at the same time during cylindrical animations.
Cycles and loops	Cycles are used extensively in cylindrical rotation sequences. All cylindrical animations loop; otherwise, the effect won't seem continuous.
Tempo	Used extensively to adjust the speed at which the cylindrical action occurs. Be careful not to move objects too quickly when performing cylindrical animations. Doing so has a tendency to ruin the effect.

The Rotational Primitive

The rotational animation primitive is often used in arcade games to describe a variety of rotating object movements. Although it's often confused with cylindrical-style animation, it's really a unique technique unto itself. In arcade games, rotational animation is used to represent everything from rotating gun turrets to spinning asteroids, making it a powerful and versatile technique.



FIGURE 9-13: Rotational Animation Example

The rotational technique is popular because it's easy to implement. Basically, it works this way: an object is rotated in variable increments using a 360-degree scale. The object completes a single rotation when it progresses from 0 to 360 degrees over the course of successive frames.

Rotational animations can move either clockwise or counterclockwise. There's also a direct relationship between the smoothness of the animation and number of degree increments each frame represents. For example, an object such as a gun turret can complete a full rotation in as little as four frames if each frame represents a 90-degree rotational increment. At the extreme end, a rotating object can require as many as 360 frames if each frame uses only a 1-degree rotational increment.

Refer to Table 9-11 for some suggestions on how to handle common rotating objects in arcade-style games.

TABLE 9-11: Common Arcade Game Rotational Object Suggestions

<i>Arcade Game Object</i>	<i>Degree Increments per Frame</i>	<i>Total Frames Required</i>	<i>Comments</i>
Asteroids/meteors (coarse)	45°	8	Minimum required to produce convincing animation.
Asteroids/meteors (smooth)	25°	16	Sufficient to render convincing animation.
Gun turrets (coarse)	90°	4	Minimum required to produce convincing animation.
Gun turrets (smooth)	45°	8	Sufficient to render convincing animation.
Spinning objects (coarse)	90°	4	Minimum required to produce convincing animation.
Spinning objects (coarse)	45°	8	Sufficient to render convincing animation.

<i>Arcade Game Object</i>	<i>Degree Increments per Frame</i>	<i>Total Frames Required</i>	<i>Comments</i>
Vehicle/character facings (coarse)	90°	4	Minimum required to produce convincing animation.
Vehicle/character facings (smooth)	45°	8	Sufficient to render convincing animation.

TABLE 9-12: Rotational Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Typically, rotational animations have straight motion lines, but there will be some waviness depending on how smoothly the object is rotated.
Motion angles	Motion angles are almost uniformly straight. Rotational animations seldom have uneven motion angles.
Key-frames and in-betweens	The number varies depending on the amount of smoothness you want to achieve. Anything between two and four key-frames is effective for this type of animation.
Weight and gravity	These properties may influence the speed at which an object rotates if it's large or heavy but has little effect on the animation otherwise.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Cycles are used extensively in rotational animation sequences. All rotational animations loop; otherwise, the effect won't seem continuous.
Tempo	Used extensively to adjust the speed at which the rotation action occurs. Very rapid tempos can help reinforce the notion of an object moving at a high speed. Similarly, very slow tempos can help give the impression of an object moving slowly.

NOTE: Rotational animations are the basis for other animation effects such as *motion blurs*. Motion blurs are effects used to depict the movement and passing of air that is disrupted by the motion of another object. For example, motion blurs occur when helicopter rotors move or when a fighter kicks or punches.

The Disintegration Primitive

The disintegration primitive is most often used in arcade games to remove objects from the screen. For example, when a character dies in a game, it gradually disintegrates and disappears so a fresh character can replace it. There are two factors to pay attention to when designing disintegration animations: the method of

removal and the extent of the removal that occurs between each frame. Let's start with the removal method first. The three most common *removal methods*, or disintegration effects, are melting, dissolving, and color fading.

Table 9-13 identifies how each of these removal methods work and differ from each other.

TABLE 9-13: Common Object Removal Methods

<i>Removal Method</i>	<i>Effect Produced</i>	<i>How it Works</i>
Melting	Causes an object to appear as if it's melting, similar to a candle.	Gradually reduces the vertical area of an object and blends its pixels together to form an unidentifiable mess.
Dissolving	Causes an object to decay similarly to being dissolved by acid.	Gradually removes random patterns of pixels from the object over successive frames.
Color fading	Causes an object to slowly vanish.	Introduces gradual (or in some cases extreme) color changes to erase the object from the screen over time.

In order to remove an object effectively, once you select a removal method, it's imperative that you stick with the mechanics associated with the effect. In other words, if you choose to employ the dissolving method, don't use color fading as part of the removal process.

The example in Figure 9-14 uses the dissolving removal method.



FIGURE 9-14: Dissolving Removal Example

Here's a breakdown of the animation sequence:

- **Frame 1:** This is an unmodified version of the original object.
- **Frame 2:** The object starts to break up. Care must be taken so that this effect occurs gradually but remains distinctly visible to the eye.
- **Frame 3:** The breakup effect continues. The integrity of the object continues to hold but large areas of its surface begin to vanish.
- **Frame 4:** There is further progression of the effect. Notice how large areas of the object are now gone.
- **Frame 5:** The object is now practically destroyed with only a few small areas of its original structure and shape now visible and intact.

As mentioned, how much of an object is removed during each frame of animation can influence the overall quality of the intended effect. For example, if you remove too much of an object's content too early in the animation sequence, the effect will look fake. Similarly, if you remove too little, the effect won't be as convincing. Planning the right amount and rate of removal for an object can be tricky. Therefore, to ensure positive results, try using what I call the *percentage method*.

Using this system, a fixed percentage of the object is removed on each successive animation frame. Not only does this method produce consistent results but it also will help you determine the maximum number of frames required by the animation. So, for example, if you set the removal percentage to 25 percent, 25 percent of the object will be removed per each frame of animation. This means that it will require a total of four frames of animation to completely remove the object from the screen. Since each removal technique requires a different number of frames to produce convincing animation, consult Table 9-14 for some general guidelines for using this method under different circumstances.

TABLE 9-14: Percentage Removal Method Guideline Suggestions

<i>Selected Removal Method</i>	<i>Estimated Percent Removed per Frame</i>	<i>Total Frames Required</i>
Melting (coarse)	25	4
Melting (smooth)	10	10
Dissolving (coarse)	25	4
Dissolving (smooth)	10	10
Color fade (coarse)	12.5*	8*
Color fade (smooth)	6.25*	16*

* Denotes that frames used should correspond to available palette entries to produce the best results.

NOTE: It's entirely possible to combine different removal methods to enhance the disintegration effect. However, because this can get messy fast, it's only recommended once you've mastered creating and using each of the different removal techniques individually.

TABLE 9-15: Disintegration Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Have no direct application to the disintegration action but are inherited from the original object and bound to its particular behavior (i.e., if a bird is flying, the disintegration animation will continue to fly as well and it will theoretically inherit the same wavy motion line).

<i>Animation Property</i>	<i>Comments</i>
Motion angles	Have no direct application to the disintegration action but are inherited from the original object and bound to its behavior.
Key-frames and in-betweens	Application varies. Depends on the desired smoothness of the disintegration action. Refer to Table 9-14 for more information.
Weight and gravity	Have no direct application to the disintegration action but are inherited from the original object and bound to its behavior.
Flexibility	Does not apply and has no effect.
Secondary actions	Have no direct application to the disintegration action but are inherited from the original object and bound to its behavior.
Cycles and loops	Rarely used since the purpose of disintegration animations is to remove an object from the screen in a graceful manner. Therefore, such sequences only move once and never cycle or loop.
Tempo	Used extensively to adjust the speed at which the disintegration action occurs. Correct tempo is a major factor in the success or failure of most disintegration-style animations. When in doubt, move your objects faster. Slow tempos, particularly when displaying explosion animations, can ruin the illusion produced by the effect.

The Color Flash Primitive

Color flashes are commonly used in arcade games to produce a flashing effect on or behind an object such as the sparkling of a jewel, the flicker of a torch, the flash of a light, or the pulse of a rocket motor. Color flashes work by applying a subtle (or dramatic) color change between each frame of the animation. This color change is usually limited to a fixed area within each frame and can range anywhere from a single pixel in size to an intricate, multipixel pattern. Regardless, when done properly, the color flash technique can produce an extremely effective “quick and dirty” effect that requires minimal time to create but adds a powerful sense of realism and character to a scene.

This being said, creating an effective color flash effect relies on four distinct factors: proper color selection, the size of the flash used, the number of frames involved, and the position of the flash on the object itself. In most situations, color flashes require you to use intense, contrasting colors. For example, to use a color flash to simulate a rocket exhaust at the back of a spaceship’s engine, you should use shades of red with distinct contrast such as bright red, medium red, and dark red. When animated, these colors will exhibit the visible distinctions that will make the effect easier to see in the context of the overall object and game screen.

The size or radius of the color flash can also have a big effect on how convincing the effect looks when used on an object. A color flash that is too small will appear too subtle on-screen, whereas a color flash that is too large can ruin the effect and jeopardize the integrity of the object itself. For the best results, try to find a place

in the middle of these extremes. Determining the right size for the color flash takes experience, but after creating just a few implementations, you'll begin to notice a pattern and will be able to judge the size properly.

Color flashes are generally meant to happen quickly. Therefore, their animations should be short and should not extend beyond a few frames. Color flashes that are between two and five frames in length are ideal. Anything less won't work and anything more can seem too drawn out. Thus, be careful in this regard and try to keep your sequence lengths relatively short.

Finally, the position of the color flash itself can make or break the intended effect. To ensure that the effect works as expected, always position the color flash logically. In other words, place the color flash where the user expects it to be. For example, jewels sparkle at the point where they reflect light. So, if you shade a jewel with the light source along the upper left-hand corner, the color flash should appear there and not in the middle or on the right-hand side. Like anything else, failing to take into account physical laws can diminish the effect that you're trying to achieve.

Figure 9-15 illustrates a complex form of the color flash as used in an animated logo sequence for a game.



FIGURE 9-15: Example of the Color Flash Primitive

Here is a frame-by-frame synopsis of the effect in action:

- **Frame 1:** This is the original, unmodified logo.
- **Frame 2:** A subtle but noticeable light is placed behind the letter A.

- **Frame 3:** The light's intensity and flash radius increase by about 50%.
- **Frame 4:** This frame shows an increase in the light's size and radius by an additional 50% of what it was in frame 3.
- **Frame 5:** By the end of the sequence the light is several times larger and brighter than it was in frame 2. At this point, the animation should cycle back to frame 1 in order to restore the animation to its original state and create the sudden flashing effect that is the hallmark of this primitive.

TABLE 9-16: Color Flash Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Do not apply and have no effect.
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	Three to five key-frames and in-betweens are optimal. More can be used to smooth the effect but at the risk of appearing too drawn out.
Weight and gravity	Do not apply and have no effect.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Used frequently to produce the illusion of continuous action.
Tempo	Used extensively to adjust the speed at which the color flash occurs. Most color flash animations occur quickly; therefore, use a fast tempo whenever possible when creating animations of this type.

The Scissors Primitive

The scissors primitive is one of the most popular animation techniques used by arcade games and is used for effects that range from simple walking to creatures biting.

It's also one of the simplest animation techniques to create. The basic technique simulates the cutting action of a pair of scissors, hence the name. Using the scissors primitive, the animation starts in the closed position and gradually progresses to the open position by the end of the sequence. It works by taking advantage of the element of exaggerated motion. In other words, it relies on the introduction of vast changes from one frame to the next. This fools the eye into seeing the intended effect while using a minimum number of animation frames. Because of this, the scissors effect is ideal for animations that require use of as few frames as possible.



FIGURE 9-16: Example of the Scissors Primitive

Here's a breakdown of a typical scissors-based animation sequence:

- **Frame 1:** The first frame shows a fish with its mouth closed.
- **Frame 2:** The next frame shows the fish partially opening its mouth. This frame serves as the in-between between frame 1 (fully closed mouth) and frame 2 (fully open mouth) and makes the animation smoother and more realistic looking.
- **Frame 3:** The final frame of the sequence shows the fish with its mouth fully open. Should you want to make this a continuous motion, you could add cycling frames at this point of the animation sequence.

The effectiveness of scissors-type animation is entirely dependent on the number of animation frames used. Smoother scissors-like animation effects require more transitional frames while coarser scissors animations can get away with fewer transitional frames. However, scissors animations of between two and four frames are the most commonly used and encountered.

NOTE: Animations based on the scissors primitive can be horizontally or vertically oriented.

TABLE 9-17: Scissors Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Do not apply and have no effect.
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	Two key-frames at minimum. More frames will increase the smoothness of the animation. Three frames is the optimum.
Weight and gravity	Can influence the speed at which the animation occurs.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Cycles are frequently used in scissors animation sequences; however, their use depends on the complexity of the animation. Scissors animations make extensive use of looping to produce the illusion of continuous action.

<i>Animation Property</i>	<i>Comments</i>
Tempo	Used extensively to adjust the speed at which the scissors animation occurs. The object's tempo will vary according to the type of effect you're after.

The Growing Primitive

This primitive is commonly used in arcade games as a modifier for the other animation techniques described here. It essentially expands an object from one size to another such as during an explosion, when a character drinks a growth potion, or to simulate an object getting larger as it moves closer.

Figure 9-17 provides an example of this technique. Notice the progression of how the object changes its size.



FIGURE 9-17: Growing Primitive Example

When using this primitive, it's important to pay close attention to the element of *scale*. Scale determines how large an object can become during the course of the animation. Always make sure that you use a constant scale. In general, it's good practice to use a scale based on a multiple of two. This ensures that the object will scale consistently. Otherwise, the object in question won't look right when it grows during the animation sequence.

TABLE 9-18: Growing Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Generally tend to be uneven as the growing object "grows."
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	This effect requires two to three key-frames to create effective growing sequences. Five to seven total frames is the optimum for most purposes.
Weight and gravity	Do not apply and have no effect.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Cycles tend not to apply to these types of animations due to the fact that object growth is incremental in nature and each frame requires a larger version of the object. Growth animations rarely loop due to this fact but can if the effect is warranted.

<i>Animation Property</i>	<i>Comments</i>
Tempo	Used extensively to adjust the speed at which the growing effect transpires. The object's tempo will vary according to the effect you're after.

The Shrinking Primitive

This is the opposite of the growing primitive. It, too, is a commonly used modifier for the other animation techniques described here. It essentially contracts an object from one size to another such as during an explosion, when a character drinks a shrinking potion, or to simulate an object getting smaller as it moves away from view.

When it comes to the element of scale, what applies to the growing primitive applies to the shrinking primitive as well.

TABLE 9-19: Shrinking Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Generally tend to be uneven as the growing object "shrinks."
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	Two key-frames is the minimum. More frames create smoother shrinking sequences. Five to seven frames is the optimum for most purposes.
Weight and gravity	Do not apply and have no effect.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Cycles tend not to apply to these types of animations due to the fact that object shrinking is incremental in nature and each frame requires a smaller version of the object. Shrinking animations rarely loop due to this fact but can if the effect is warranted.
Tempo	Used extensively to adjust the speed at which the shrinking effect transpires. The object's tempo will vary according to the type of effect you're after.

Minor Arcade Game Animation Primitives

This set of animation primitives is used in many types of arcade-style games but appears far less often than those described in the major animation primitives category. These techniques are applicable to both character and mechanical objects. They include:

- The piston primitive
- The squeeze primitive
- The swing primitive

- The slide primitive
- The open/close primitive
- The bounce primitive
- The stomp primitive

The Piston Primitive

Objects that use the piston primitive appear to pump up and down or from side to side when animated. This effect is usually seen in mechanical objects such as engines, machinery, or robots.

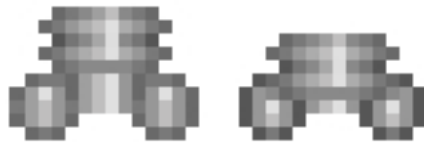


FIGURE 9-18: Piston Primitive Example

TABLE 9-20: Piston Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Piston-like animations tend to have non-straight motion lines.
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	Effective piston animations can be made with as few as two key-frames. More than eight frames is overkill for this type of effect.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in slower overall movement.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Piston animations make extensive use of both cycles and looping effects.
Tempo	Used extensively to adjust the speed at which the piston effect occurs. Piston animations usually occur quickly; therefore, use a fast tempo.

The Squeeze Primitive

Objects that use the squeeze primitive appear to compress themselves in a manner similar to an accordion when animated. This effect is usually seen in certain mechanical objects, such as spaceships and robots.

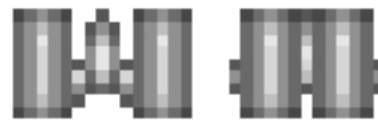


FIGURE 9-19: Squeeze Primitive Example

TABLE 9-21: Squeeze Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Squeeze animations almost always have constant and even motion lines.
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	Squeeze animations can be depicted in as few as two key-frames. Using more than four frames tends to minimize the effect as the squeeze primitive relies heavily on exaggeration.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in slower overall movement.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Squeezing animations make extensive use of both cycles and looping effects.
Tempo	Used extensively to adjust the speed at which the squeezing effect occurs. Squeeze animations usually occur quickly; therefore, use a fast tempo.

The Swing Primitive

Objects that use the swing primitive appear to swing like a pendulum either horizontally or vertically when animated. The swing primitive is actually a variation of the scissors primitive and is used by both mechanical and organic objects to represent movements from spaceship wings to chomping creatures.

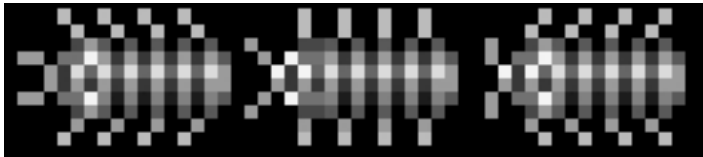


FIGURE 9-20: Swing Primitive Example

TABLE 9-22: Swing Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	They tend to have wavy motion lines that start out straight and sharply curve towards the center of the motion.
Motion angles	They tend to have very sharp motion angles
Key-frames and in-betweens	Swing animations can contain any number of key-frames. More key-frames will, of course, result in smoother swinging animation effects.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in slower overall swinging movement.

<i>Animation Property</i>	<i>Comments</i>
Flexibility	Does not apply and has no effect.
Secondary actions	Usually doesn't apply, but can in certain instances, particularly when animated walking objects can make use of secondary actions to enhance the drama of the effect.
Cycles and loops	Swinging animations make extensive use of both cycles and looping effects.
Tempo	Used extensively to adjust the speed at which the swinging effect occurs. The object's tempo will vary according to the effect you're after.

The Slide Primitive

The slide primitive is used to represent sliding and shuffling type movements. Such movements are frequently used to create simple walking and crawling animations with only a handful of frames.

The slide primitive works by taking advantage of excessive exaggerations. When done well, this has the effect of fooling the observer into thinking the object is walking or crawling even though no part of the object has left the ground.

Figure 9-21 demonstrates the sliding primitive in action. Notice how the character's arms and feet work in unison to give the impression that the character is sliding across a surface.



FIGURE 9-21: Slide Primitive Example

TABLE 9-23: Slide Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Sliding animations are almost exclusively straight and constant. However, occasionally you will encounter sliding animations that have uneven motion lines.
Motion angles	Tend to be very sharp for certain parts of the object being animated, i.e., legs and arms.
Key-frames and in-betweens	Sliding animations are extremely simple and usually only comprise two key-frames: object still and object sliding. Adding in-betweens will enhance the sliding motion and are encouraged.

<i>Animation Property</i>	<i>Comments</i>
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in a slower overall sliding movement.
Flexibility	Does not apply and has no effect.
Secondary actions	Occasionally applies to sliding objects, particularly during walking sequences.
Cycles and loops	Sliding animations make extensive use of both cycles and looping effects.
Tempo	Used extensively to adjust the speed at which the sliding movement occurs. Slide animations run more slowly than most other animation primitives; therefore, slow down the tempo when creating sliding effects.

The Open/Close Primitive

As the name implies, objects that use the open/close primitive have two states: open and closed. As such, this primitive is most commonly used to depict simple objects such as doors. However, this primitive can be used to produce emotion in detailed characters such as the blinking of an eye as illustrated by Figure 9-22.



FIGURE 9-22: Open/Close Primitive Example

TABLE 9-24: Open/Close Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Do not apply and have no effect.
Motion angles	Do not apply and have no effect.
Key-frames and in-betweens	Almost all open/close animations require at least two key-frames. Adding additional frames can enhance the effect but isn't required.
Weight and gravity	Do not apply and have no effect.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Open/closing animations make extensive use of both cycles and looping effects.
Tempo	Used extensively to adjust the speed at which the open/close action occurs. The object's tempo will vary according to the type of effect you're after.

The Bounce Primitive

The bounce primitive is commonly used in arcade games on objects to simulate simple objects traveling or bouncing back and forth between two endpoints. It should not be confused with the element of gravity. This primitive is generally limited to mechanical objects such as computers and robots.

Figure 9-23 shows the bounce primitive in action. In this example, the white dot at the center of the object “bounces” back to its original position when looped.

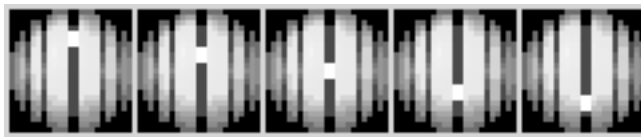


FIGURE 9-23: Bounce Primitive Example

NOTE: This primitive should not be confused with the cylindrical primitive as it is not intended to simulate the motion of an object spinning in place.

TABLE 9-25: Bounce Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Always have straight motion lines.
Motion angles	Always have straight motion angles.
Key-frames and in-betweens	Requires at least two to three key-frames to produce a relatively convincing effect. Adding more frames will improve the smoothness of the bouncing movement.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in a slower overall bouncing movement.
Flexibility	Does not apply and has no effect.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Bouncing animations are usually too simple to require cycles. However, bouncing animations make extensive use of looping effects.
Tempo	Used extensively to adjust the speed at which the bouncing movement occurs.

The Stomp Primitive

Objects that use the stomp primitive appear to flail from side to side much like a person stomping their foot. Because of this and the fact that the stomp primitive only has two states, up or down, it’s used to depict a simple and exaggerated form

of walking for characters, human and otherwise. In fact, alien invader type objects frequently use it.

Stomp-like animations are largely secondary actions since they typically only affect the feet or bottom of an object.



FIGURE 9-24: Stomp Primitive Example

TABLE 9-26: Stomp Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Have short and extremely wavy motion lines.
Motion angles	Have very sharp motion angles.
Key-frames and in-betweens	Requires at least two to three key-frames to produce a relatively convincing effect. Adding more frames will improve the smoothness of the bouncing movement.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in a slower overall stomping movement.
Flexibility	As all stomping animations deal with legs, care should be taken to ensure that the stomping movement is crisp.
Secondary actions	Do not apply and have no effect.
Cycles and loops	Stomping animations make extensive use of both cycles and looping effects.
Tempo	Used extensively to adjust the speed at which the stomping movement occurs. Stomping animations tend to run slower than most other animation primitives; therefore, slow down the tempo when creating these type of effects.

Complex Arcade Game Animation Primitives

This category of animation primitives encompasses the fundamental character and creature animation techniques that are used in platform-style games, regardless of their particular style or theme.

- The slinking primitive
- The flying primitive
- The walking primitive

- The running primitive (humans)
- The running primitive (animals)
- The jumping primitive
- The crawling primitive

NOTE: Several of the figures used in this section of the chapter are based on observations and computer drawings made from material presented in Eadweard Muybridge's books, *The Human Figure in Motion* and *Animals in Motion*. Both of these books do an excellent job of breaking down the complex movements of both humans and animals into easy-to-digest pieces. These books are excellent resources for anyone interested in creating more realistic arcade game animation.

The Slinking Primitive

Slinking is a form of locomotion that certain “lower” animals such as snakes and worms use. Therefore, it's commonly used in arcade-style games that feature such animals.

The basic slinking motion incorporates elements of the squeezing primitive in that these creatures move as a result of the momentum produced as their bodies squeeze and straighten. Slinking movements utilize a great deal of exaggeration in order to produce the best impression of motion with as few frames as possible.

Figure 9-25 demonstrates how a cartoon snake would move using a slinking motion.



FIGURE 9-25: Slinking Example

Here's a breakdown of how this particular slinking animation works:

- **Frame 1:** Shows the snake with its body straight and neck and head curled and braced for movement.
- **Frame 2:** Shows the progression of the movement as the snake's body curls upwards as if it's pushing the rest of the body forward. Frame 2 is also an excellent example of secondary action as the snake's head and tail also move, albeit at different points.

- **Frame 3:** Here we see the snake's body beginning to flatten. At the same time, the snake's head and neck lurch forward. This further reinforces the notion that the snake is moving by the sheer force of momentum.
- **Frame 4:** Here, the snake's body is nearly completely flat and its head and neck are in a similar position to how they were originally in frame 1. At this point, you would cycle the animation to continue the flow of movement.

TABLE 9-27: Slinking Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Always have slightly wavy motion lines.
Motion angles	Always have straight motion angles.
Key-frames and in-betweens	All four frames qualify as key-frames when using this type of primitive. Of course, smoother animation can be achieved by adding corresponding in-betweens for each key-frame used.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in a slower overall slinking movement.
Flexibility	Although snakes, worms, and snails don't have joints or backbones, you should pay attention to flexibility when applying slinking movements, as these will enhance the naturalness of the movement.
Secondary actions	Secondary actions occur very frequently in slinking animations. Pay close attention to heads and tails in addition to the main part of the object's body as it moves.
Cycles and loops	Depending on their complexity, slinking animation sequences can make use of cycles. However, all slinking motions require loops to provide the illusion of continuous movement.
Tempo	Used extensively to adjust the speed at which the slinking movement occurs. Slinking animations run slower than most other animation primitives; therefore, slow down the tempo when creating slinking effects.

The Flying Primitive

Animating the motion of birds and insects can be a fairly complex business that requires a lot of time and study on how creatures fly in order to do properly. Birds, and all flying creatures for that matter, move their wings from the down position to the up position and vice versa. This motion is then repeated, or cycled, to complete the sequence. It takes flying creatures several steps to transition the movement of their wings from these states.

Under most circumstances, it can take as many as 12 frames to create a realistic flying animation. However, by breaking the basic flying motion into its most exaggerated components, you can extract the key-frames and produce effective, if not somewhat less realistic, flying animations in as few as five frames.

NOTE: It's actually quite possible to produce passable flying animations with as few as two key-frames (up and down). However, you will need to make sure that you compensate for the lack of key-frames by using extensive exaggeration between the two frames and by using the proper tempo.

When studying a bird's flight, you should take careful note of motion path, motion angles, tempo, and secondary actions of the wings and body. Failing to do so can cause you to produce animations that are inaccurate and unconvincing.

To get a better idea of how the flying process works, let's look at a simplified example. Figure 9-26 shows a simplified flying sequence rendered in just five frames.



FIGURE 9-26: Simplified Flying Sequence

Here is a breakdown of the simplified flying sequence shown in Figure 9-26:

- **Frame 1:** The sequence begins with the bird's wings in the full down position. In this frame, the movement of the wings starts upward. The secondary action occurs at the head and tail with the head being fully raised and the tail pointing down.
- **Frame 2:** The wings are now level as if gliding. Here, the head begins to move down and the tail begins to level out. This produces an effect that is consistent with the flow of movement and cues the observer into believing the authenticity of the movement. The movement of the bird's head and tail between frames also produces a motion line consistent with flight. Since the exertion of movement and air turbulence both act against the bird, there's no way that it would have a perfectly straight motion line in the real world. Therefore, you wouldn't expect to have a straight motion line when creating its animation.
- **Frame 3:** Here the wings start to rise. The bird's head lowers farther and its tail begins to straighten out. Do you notice how the concept of the motion angle comes in play? The lowering of the head and leveling of the tail serve to visually emphasize the bird's acceleration as it starts to gain lift.
- **Frame 4:** The wings are nearing the completion of their upward movement. The tail and head begin to repeat their upward motion.

- **Frame 5:** The wings are now in a full upright position. At this stage, you could cycle the animation in reverse to restart the sequence and keep the sense of motion constant.

NOTE: You can start simplified flying animation sequences in either the up or the down state. It doesn't really matter as long as there is a beginning and an ending to the sequence.

Figure 9-27 represents a complex flying sequence. This is an exact frame-by-frame breakdown of how a bird really flies. If you compare it with Figure 9-26, you'll probably notice a number of similarities between the key-frames, the shape of the motion path, and the motion angles. For the most part, Figure 9-27 is identical to Figure 9-26 but it has seven more frames, or in-betweens, added in order to produce a smoother-looking animation sequence.

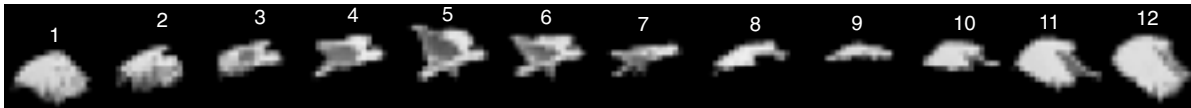


FIGURE 9-27: Complex Flying Sequence

Here's a breakdown of the complex flying sequence shown in Figure 9-26:

- **Frame 1:** The first frame starts with the bird's wings in the down position.
- **Frame 2:** This frame has the bird's wing and body slowing rising upwards.
- **Frame 3:** The third frame shows the bird's wing and body leveling out as if it's gliding.
- **Frame 4:** This frame shows the bird's tail drooping down and head and wings rising.
- **Frame 5:** Here, the bird's wings reach their uppermost point while its tail continues to move down. Notice how the first five frames show the unevenness of the bird's motion path.
- **Frame 6:** The bird's wings slowly move downwards. Its tail has now reached its lowest position in order to aid lift.
- **Frames 7 and 8:** These frames show the bird's wings moving down even farther.
- **Frame 9:** In this frame, the bird's body, tail, and wings are completely straight and level. Here, the bird's motion line also begins to straighten.
- **Frame 10:** The wings continue their downward progression.
- **Frames 11 and 12:** The wings once again complete their downward movement until they are fully extended. At this point, you could add cycles to keep the flow of motion constant.

As mentioned, it's very important not to neglect tempo when creating flying animations. Different birds fly at different speeds. For example, insects and hummingbirds tend to flap their wings at very high speeds, whereas most birds tend to flap their wings more slowly.

You should also remember to consider the effects of weight and gravity of the flying creature. For example, smaller birds and insects will tend to have very uneven motion lines because they flutter as they fly, but larger birds don't.

TABLE 9-28: Flying Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Always have wavy motion lines. Certain objects, namely complex bird and butterfly animation sequences, will have very wavy motion lines.
Motion angles	Always have straight motion angles.
Key-frames and in-betweens	Requires at least two key-frames. More frames will result in smoother flying sequences. In Figure 9-26, frames 1, 3, and 5 are key-frames while in Figure 9-27, frames 1, 5, 7, 9, and 12 are key-frames.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will result in a slower overall flying movement.
Flexibility	Flexibility should be emphasized around certain object parts such as the head, neck, and wings.
Secondary actions	Secondary actions occur very frequently in flying animations. Pay close attention to heads and tails in addition to the main part of the object's body as it moves.
Cycles and loops	Cycles are used extensively in all flying animation sequences. All flying animations make use of loops to provide the illusion of continuous and realistic flight.
Tempo	Used extensively to adjust the speed at which the flying movement occurs. Different animals fly at different speeds. For example, insects move their wings faster than birds. Along the same lines, smaller birds tend to move their wings faster than larger birds. Make sure you account for this and apply tempo properly.

NOTE: There is a direct relationship between the number of frames present in a flying animation sequence and the speed at which it moves. Slowly moving objects will require more frames than faster moving objects due to the way our eyes discern movement.

The Walking Primitive

Walking is used in almost every type of arcade game that involves humanoid characters and is particularly important for platform games. Unfortunately, walking is

also one of the more difficult types of character actions to animate, especially for beginners. This is because walking is a complex motion. You see, walking involves multiple body parts such as the head, arms, and body, as well as the legs. However, the process becomes much easier if you break the walking process down into smaller pieces by focusing on animating each body part that is affected by the movement individually. Thus, walking is also an excellent way to sharpen your skills in creating secondary actions.

However, in order to do this, we need to review the different components of the walking process:

- **Step 1: The legs**—Walking always starts off with the legs. One leg grips the ground and pulls the other one forward. This essentially produces a propelling type of movement. When creating a walking sequence, always concentrate on the legs first.
- **Step 2: The arms**—The arms swing back and forth along with the motion of the legs but in the opposite order, i.e., as the leg on one side moves forward, the arm on that side moves back and vice versa.
- **Step 3: The head**—Factor in how the head moves along with the rest of the walking motion. In the real world, the head bounces slightly up and down as the positions of both legs change and the weight of the body shifts.

Simple walking can be achieved with only two key-frames as shown in Figures 9-28 and 9-29. One frame has the object standing still while the other frame has the legs of object spread widely apart. Also notice the use of a modified and heavily exaggerated form of the scissors primitive and the addition of secondary action subtleties such as arm and head movement. This enhances the effectiveness of the walk despite the small number of frames used to depict it.

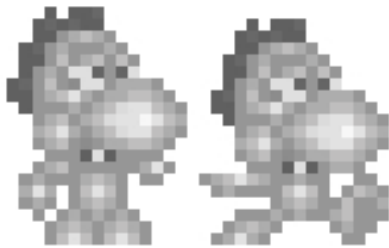


FIGURE 9-28: Basic Walking Example #1



FIGURE 9-29: Basic Walking Example #2

Complex walking needs many more key-frames to produce a convincing animation sequence, however. Figure 9-30 provides an example of the complex walking sequence. As you can see, a realistic walking sequences requires as many as 11 frames to complete the illusion of movement.

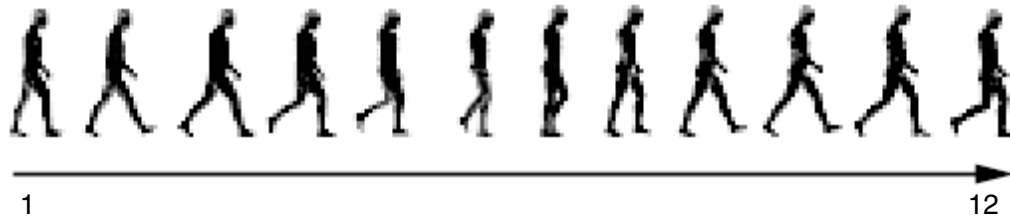


FIGURE 9-30: Complex Walking Example

Here's a breakdown of how a complex walking animation is constructed:

- **Frame 1:** This frame shows the character leading with the right leg. The right leg is actually in the process of moving backward but this won't become evident until later on in the animation sequence. This leg moves forward with the heel of the foot touching the ground and the toe and sole of the foot off the ground. The right arm is hanging straight down and the left arm is positioned forward while bent slightly at the elbow. As the animation progresses, the left arm will eventually move backward while the right arm moves forward.
- **Frame 2:** In this frame, the right leg extends forward. Meanwhile, the left leg begins to move backward in order to maintain support and provide the body with a stable platform. The right arm begins to move slowly backward to help counterbalance the general motion of the body while the left arm moves forward. The position of the character's head drops slightly as the weight of the body shifts and the back bends due to the change in the positioning of the legs.
- **Frame 3:** This frame has the right leg slightly bent at the knee. The foot is now largely flat with both the heel and toes nearly level. This is the point where the legs first begin to grip the ground. The rear leg now bends at the knee and its heel is lifted off the ground with the toes flattened. The right arm continues to move backward as the left arm moves slowly forward. The head now returns to its original position due to the new shift of body weight.
- **Frame 4:** In this frame, the right leg is now fully bent with the toes and heel now completely flat against the ground. The rear leg is still bent and moves off the ground, touching only at the toes. Both arms continue their movements as per frame 3. The position of the head is largely unchanged.
- **Frame 5:** This frame is a slight variation of frame 4 except that the right leg now begins to reverse its movement and starts to move backward. At the same time, the left leg is completely off the ground but moves forward. The right arm now moves forward in a counterbalancing action. The left arm draws back. The position of the head is unchanged, however.
- **Frame 6:** In this frame, the right leg is largely straight but continues to move backward. The left leg, still bent at the knee, moves forward to the point where it almost lines up with and begins to pass the right leg. The right arm

moves forward slightly while the left arm moves back slightly. The position of the head drops once again due to the shift in body weight.

- **Frame 7:** This frame demonstrates what is known as the *passing position* of the walk sequence. This is when one leg passes the position of the other leg. In this frame, the right leg is almost completely straight. Because of this, the character's back straightens and the head returns to its normal positioning. At the same time, the left leg passes in front of the right leg, although it's still bent and slightly off the ground. The right arm continues forward while the left arm continues backward.
- **Frame 8:** This frame shows the right leg starting to move backward slightly while the left leg continues to move forward. The right arm is now fully in front of the character's torso while the left arm is moving backward in small increments. The position of the head is unchanged from the previous frame.
- **Frame 9:** In this frame, the right leg is moving backward with the heel off the ground and is only stabilized at the foot by the toes. The left leg is nearing the completion of its forward movement. The right arm is nearing completion of its forward motion while the left arm continues its slow backward movement. The head is slightly dropped from the previous frame.
- **Frame 10:** This frame demonstrates the heel to toe part of the walk stride. In this frame, both legs are at their maximum distance from each other. In addition, only the toe of the foot stabilizes the right leg. The left leg has completed its forward movement. The right arm has completed its forward motion while the left arm has completed its movement backward. The head's position remains stable and largely unchanged.
- **Frame 11:** This frame shows the right foot positioned back. The left leg has locked out and the left foot lies flat against the ground. The right arm begins to reverse its movement and move backward. The left arm starts to reverse its motion and move forward. The head's position dips slightly due to the shifting in weight across the body.
- **Frame 12:** This is the final segment of the animation sequence. The right foot now begins its sweep forward while the left leg begins its sweep backward. The right arm continues to move backward while the left arm moves forward. The head returns to its normal position. You can create a very fluid walking sequence by cycling back to the first frame of the animation and then looping it.

Of all of the actions described in this section of the chapter, it's the legs that are the most difficult to visualize and animate. Therefore, use Figures 9-31 and 9-32 as "cheats" to help you break down this complex movement.



FIGURE 9-31: Walking Leg Movements (Part 1)



FIGURE 9-32: Walking Leg Movements (Part 2)

NOTE: There is a direct relationship between the number of frames present in a walking animation sequence and the speed at which it moves. Slowly moving objects will require more frames than faster moving objects due to the way our eyes discern movement.

TABLE 9-29: Walking Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Always have motion lines that are slightly wavy. Motion lines that are too wavy, particularly for complex animations, are probably too exaggerated and won't look convincing.
Motion angles	Have largely straight motion angles.
Key-frames and in-betweens	Requires at least two key-frames in simple walking sequences. As always, more frames will result in smoother walking sequences. In the complex walking sequence in Figure 9-30, the key-frames are frames 1, 3, 5, 7, 9, and 11.
Weight and gravity	Both of these can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will require heavier steps and result in slower overall walking movements and shorter exaggerations between frames.
Flexibility	Flexibility should be emphasized around the arms and legs. Depending on the size and the complexity of the object being animated, you might want to emphasize flexibility in the feet as well.
Secondary actions	Secondary actions occur extremely frequently during walking animations. Pay especially close attention to the head of the character, as it will bounce up and down as the position of the legs and feet change. In addition, hair and clothing will often exhibit secondary actions. For example, hair might flow or bounce (depending on length and style) and clothing might trail behind the character as it picks up speed. Incidentally, adding secondary actions to your walking sequences is an excellent way of tricking the observer into thinking the character is moving in a more realistic fashion than it really is.

<i>Animation Property</i>	<i>Comments</i>
Cycles and loops	Cycles are used extensively in all walking animation sequences, particularly for complex ones. All walking animations also make use of loops in order to provide the illusion of continuous and realistic movement.
Tempo	Used extensively to adjust the speed at which the walking movement occurs. Walking is usually done at a relatively slow pace. Therefore, keep the tempo moderate and constant.

The Running Primitive (Humans)

Running actions are commonly found in platform-style arcade games. They serve two purposes: first, they can heighten the sense of realism by simulating the act of acceleration. As the player applies more force on their controls, the on-screen character increases speed. Second, running actions allow the on-screen character to traverse more ground within the game area than when merely walking.

From an animator's standpoint, most of what applies to creating walking sequences applies to running as well. The main difference between the two movements is that the element of exaggeration between frames and the positioning of body parts to represent the various secondary actions is increased for running objects.

There are other differences between walking and running that should be mentioned. This section highlights the most important ones.

- **The arms**—Running actions emphasize the arms and their movement much more than walking. Arm movements in walking tend to be gradual, while arm movements in running tend to be highly exaggerated as the arm acts as a lever to help the overall motion of the object. This means that you should draw the arms of any running object with greater separation from the body. As such, take advantage of the element of flexibility as much as possible.
- **The body**—Running actions cause an object to have highly visible motion angles. This is because the body bends forward as it picks up momentum and speed while it moves. You will need to reflect this change in your running animations and tilt the body when creating such sequences.
- **The legs**—Running actions cause the object to cover more area than it would when walking. This means that the strides the legs take during a run should be extended to accurately depict this effect.
- **The head**—Running actions cause more violent up and down motions due to the increase in speed and the more extensive and rapid shift in body weight. Therefore, the up and down motion of the head should be more frequent and exaggerated. This means that running actions use more pronounced secondary actions than walking sequences.

Figure 9-33 provides an example of a complex running sequence. As you can see, a realistic running sequence requires as many as 12 frames to complete the illusion of movement.



FIGURE 9-33: Running Primitive Example (Humans)

Here's a frame-by-frame synopsis of a typical, complex running action:

- **Frame 1:** The animation sequence starts with both legs off the ground with the right leg back and the left leg forward. Both legs are bent at the knees with the right leg partially bent and left leg fully bent. The right arm is forward with a 45-degree upward bend. The left arm has a 45-degree downward bend. The character's back is slightly arched forward and head is bent slightly forward.
- **Frame 2:** This frame shows both legs off the ground, although much closer to the ground than in frame 1. Both legs are still bent at the knees. This time, however, the right leg is bent back at a 90-degree angle. The right arm is now bent at an 85-degree angle as it travels downward and back. Meanwhile, the left arm lunges slowly forward. The character's back is still arched and the head has dropped down a bit more.
- **Frame 3:** In this frame, the right leg, now fully extended back begins the process of moving forward. The left leg is now very close to the ground. The right arm moves downward and the left arm moves forward. The character's head and back are hunched over as the character picks up speed.
- **Frame 4:** This frame demonstrates the *contact position* of the running movement, or the first point at which the runner's feet touch the ground. In this frame, the left foot touches the ground while the right leg, although still bent, is rapidly moving forward. The right arm is bent and quickly moving backward while the left arm is bent and moving forward. The position of the head and back are largely unchanged from the previous frame.
- **Frame 5:** This part of the animation sequence demonstrates the *push-off position* of the running movement or the point at which the runner's feet leave the ground in order to gain momentum in the stride. In this frame, the left leg prepares to leave the ground once again as it starts to push off the ground with the left foot. Meanwhile, the right leg enters the passing position with the left leg and lunges forward. The right arm is now at an almost complete 180-degree bend, as it swings backward. The left arm pushes forward and has a 45-degree bend. Both the head and back begin to straighten.

- **Frame 6:** This frame shows the character near full stride and once again leaving the ground. The right leg is extended almost completely forward. The right leg is almost fully bent and completely off the ground. The left leg is almost straight as it extends backward toward the ground. Both arms appear to pump furiously. The right arm is completely back and bent at a perfect 90-degree angle. The left arm is bent upward at a near perfect 45-degree angle. The head and back are almost completely straight.
- **Frame 7:** This frame is very similar to frame 6 as both legs remain completely off the ground. The right leg is now fully extended forward while the left leg continues to stretch backward. The right arm is still bent at a 90-degree angle and begins to move forward. The left arm is still at a 45-degree angle and starts to move back and downward. The head and back continue to straighten up.
- **Frame 8:** In this frame, the right leg is still off the ground but rapidly moving backward and down. The left leg is almost fully bent upward and is quickly moving forward and down. The right arm is moving forward and the left arm continues to move backward. In this frame, the head and neck are slowly moving forward again as the running motion again produces momentum.
- **Frame 9:** This frame has the right leg slightly bent and now touching the ground. The left leg is fully extended back. It won't move any farther back at this point, only forward. The right arm continues to move forward while the left arm moves back. It's interesting to note that both arms are almost even with each other. This is the only time during the entire sequence that such an event will occur. The character's head and back are once again starting to straighten.
- **Frame 10:** This frame is almost the direct opposite of frame 4. Here, the right leg is still on the ground but is about to spring off the ground. The left leg is also in the process of swinging forward. The right arm is now almost at a 90-degree angle as it moves forward. The left arm continues to move backward. Both the head and neck continue to straighten.
- **Frame 11:** This frame shows the beginning of the springing action of the running movement. The right leg extends as it moves backward. The left leg moves forward in a wide, exaggerated stride. Both arms are now at a 45-degree angle with the right arm moving up and forward and the left moving backward and upward. The head and back are now straight.
- **Frame 12:** This is the final frame of the animation sequence and it shows the character in a full running stride. The actions here are very similar to those in frame 1. You can create a very fluid running sequence by cycling back to the first frame of the animation and then looping it.

Use Figures 9-34 through 9-37 as “cheats” to help you break down this complex movement. Unlike walking, it's important that you capture all of the motion associated with this primitive, including the exaggeration of the arms and legs.



FIGURE 9-34: Running Movement—Arms (Part 1)



FIGURE 9-35: Running Movement—Arms (Part 2)



FIGURE 9-36: Running Movement—Legs (Part 1)

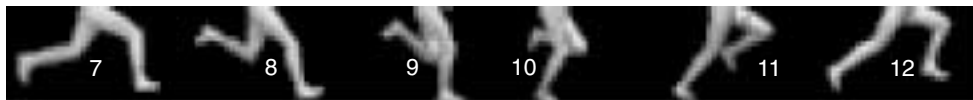


FIGURE 9-37: Running Movement—Legs (Part 2)

NOTE: Running actions are enhanced when you incorporate the element of anticipation. Anticipation helps to make runs look more realistic since it's not very convincing for full-speed running to occur from a walking or standard start. You can create the impression of anticipation by adding some additional transitional frames at the start of the running sequence. If this is too difficult or time-consuming, a simple alternative is to slow the tempo of the action during the first cycle of the animation.

TABLE 9-30: Running Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Running actions always have very wavy motion lines. Motion lines that are too straight will appear too stiff and rigid and won't look natural when animated.
Motion angles	Running sequences have very acute motion angles.
Key-frames and in-betweens	Requires at least two key-frames for simple running sequences. In complex running sequences, frames 2, 4, 6, 8, 10, and 12 are key-frames.
Weight and gravity	Both of these can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will require heavier steps and result in slower overall walking movements and shorter exaggerations between frames.

Animation Property	Comments
Flexibility	Flexibility should be emphasized around the arms and legs, particularly at the joints such as the knees and elbows. Depending on the size and the complexity of the object being animated, you might want to emphasize flexibility in the feet as well.
Secondary actions	Secondary actions occur very frequently in running animations. Pay close attention to the head of the character, as it will bounce up and down as the position of the legs and feet change. In addition, hair and clothing will often exhibit secondary actions. For example, hair might flow or bounce (depending on length and style) and clothing might trail behind the character as it picks up speed. For example, imagine a flowing cape or scarf. Incidentally, adding secondary actions to your running sequences is an excellent way of tricking the observer into thinking the character is moving in a more realistic fashion than it really is.
Cycles and loops	Cycles are used extensively in all running animation sequences, particularly for complex ones. All running animations also make use of loops in order to provide the illusion of continuous and realistic movement.
Tempo	Used extensively to adjust the speed at which the running movement occurs. Runs occur faster than walking or jumping sequences. Make sure that they have a faster tempo than the other primitives described in this chapter.

NOTE: You can add personality to your walking and running sequences by embellishing and exaggerating the various frames that make up the sequence. For example, you can simulate heavier characters (i.e., the effect of gravity) or double up frames to mimic the effect of a limp.

The Running Primitive (Animals)

Much of what has been discussed also applies to four-legged animals as well. Once you master the general theory behind creating walking and running sequences for humans, you should be able to handle most types of four-legged animals without too much trouble.

However, understand this: creating convincing movement for most four-legged animal characters is quite a bit more complex than doing so for human characters. Instead of having two legs to deal with, you now have four to manipulate. Thus, the best way to approach creating such animations is to create individual animation sequences for each set of legs. Although this works well for relatively simple four-legged actions, it will fail to produce convincing movement as the animation sequences get more complex. Therefore, I strongly recommend getting a copy of Eadweard Muybridge's excellent *Animals in Motion* as studying this book will give you important insights into how different four-legged animals move.

In any event, Figure 9-38 shows the basic action of a four-legged animal running. I decided to focus on the running action over walking because it is less complex and generic enough to represent a variety of four-legged animals including tigers, wolves, and even horses, albeit with somewhat less accuracy and realism.



FIGURE 9-38: Running Primitive Example (Animals)

Here's a breakdown of this action:

- **Frame 1:** The first frame has the animal's back largely straight. The right forward leg is arched back while its right rear leg is positioned forward and the left rear leg is moving backward.
- **Frame 2:** This shows the animal's body beginning to stretch outward. Both front legs are bent and extending outward while the rear legs are stretching out and back. The rear legs are positioned near the ground while the front legs are moving away from it. The head and back are angled as a result of this position.
- **Frame 3:** This frame shows the animal's body completely level and outstretched. At this point, all four legs are off the ground. This is the full-stride position of the movement.
- **Frame 4:** This frame is very close in appearance to frame 1, the main exception being that the right rear leg is closer to the ground and the left rear leg is extended farther back than its position in the first frame.
- **Frame 5:** Frame 5 is almost identical to frame 2. The primary difference between the two frames is the order of the front legs. Here, the right front leg is extended farther out and the left front leg is positioned slightly farther back. The positioning of the rear legs is so close that you probably don't need to make any changes to their position.
- **Frame 6:** This is the final frame of the animation sequence. It shows the body of the animal outstretched with all four legs off the ground. Although also similar to the contents of frame 3, there is a major difference in the position of the head and back. In this frame, the head of the animal is pointing down while its back is arched sharply. All four legs are also positioned at more extreme positions than what is shown in frame 3.

TABLE 9-31: Running Primitive Animation Property Summary

Animation Property	Comments
Motion lines	Running actions always have very wavy motion lines. Motion lines that are too straight will appear too stiff and rigid and won't look natural when animated.
Motion angles	Running sequences have very acute motion angles.
Key-frames and in-betweens	Requires at least two key-frames for simple running sequences. In complex running sequences, frames 1, 3, and 5 are key-frames.
Weight and gravity	Both of these can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will require heavier steps and result in slower overall walking movements and shorter exaggerations between frames.
Flexibility	Flexibility should be emphasized around the joints. Depending on the size and the complexity of the object being animated, you might want to emphasize flexibility in the feet as well.
Secondary actions	Secondary actions occur very frequently in animal running animations. Pay especially close attention to the head and tail of the animal. Animating these can greatly enhance and improve the realism of the sequence.
Cycles and loops	Cycles are used extensively in all running animation sequences, particularly for complex ones. All animal running animations also make use of loops in order to provide the illusion of continuous and realistic movement.
Tempo	Animals move at very fast speeds when running. Therefore, make sure you use an appropriately paced tempo to accurately represent this action.

The Jumping Primitive

Jumps are used in platform-style arcade games to help the on-screen character cover large sections of the game area in a single bound. They are also used to jump over obstacles and dodge other objects such as missiles and bullets.

There are many types of jumps but the most popular one is known as the *standing jump*. In this type of jump, the character jumps from a standing position. The jumping process has four parts: the *stand*, the *bend*, the *leap*, and the *landing*.

The stand is where the character begins the jumping sequence. The bend is where the character gains the strength and the position to make the jump. The leap is the actual jump itself. Finally, the landing is where the jump ends.

NOTE: Generally speaking, standing jumps have a shorter and more constant length than running jumps.

Figure 9-39 shows a simple running jump sequence. Here, there are only two frames needed: a standing frame (frame 1) and jumping frame (frame 2). Simple jumps work quite well for most purposes, particularly for unrealistic or cartoon-style characters. This is due to the exaggeration of movement that occurs between the two frames of the action.

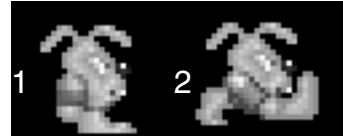


FIGURE 9-39: Simple Jumping Primitive Example

Figure 9-40 shows a complete, complex running jump sequence.



FIGURE 9-40: Complex Jumping Primitive Example

Here's a breakdown of each frame for this movement:

- **Frame 1:** In this first frame of the sequence, the character is standing. Both knees are bent and both arms are raised in anticipation of the jump.
- **Frame 2:** This frame shows the character starting to lean forward. Both legs are still bent and have moved since the last frame. The back and head are now bent forward and both arms are now straight down at the character's side.
- **Frame 3:** This frame shows the character at the start of the bend phase of the jump. Here, both legs are completely bent to support the back and head as they bend forward. Both arms make an exaggerated swing up and back to provide the jumper with additional pushing leverage.
- **Frame 4:** Frame 4 represents the start of the leap. Both legs are leaving the ground in a spring-like action while the arms rapidly swing forward in support. The head and body are bent completely forward at an 80-degree angle.
- **Frame 5:** This frame shows the character nearing the peak of the jump. The entire body of the character is at a 45-degree angle as it leaves the ground.
- **Frame 6:** This is the peak of the jump. Here, both legs push the character's body up and forward as they swing up and back. Both arms are straight and positioned above the character's head in support of this movement.
- **Frame 7:** Here we see the start of the landing. Although still off the ground, the character's legs swing forward. At the same time, the character's body, head, and arms lean sharply forward in anticipation of the landing.
- **Frame 8:** The character is nearing the point of landing. The entire body of the character curls in unison to absorb the impact of the landing. The arms, head, and back are bent forward while the legs start to bend backward.

- **Frame 9:** The character lands. The feet of both legs are now firmly on the ground while bent forward. The rest of the body is arched forward in order to minimize the effect of the impact.
- **Frame 10:** The jump sequence is completed. The character's body begins to rise and straighten as the force of the jump is absorbed and dissipated. Here, both knees remain bent, the back remains arched, and both arms continue their lunge forward. Unlike walking or running sequences, jumping animations are rarely cycled due to the nature of the action.



FIGURE 9-41: Complex Jump Sequence (Part 1)

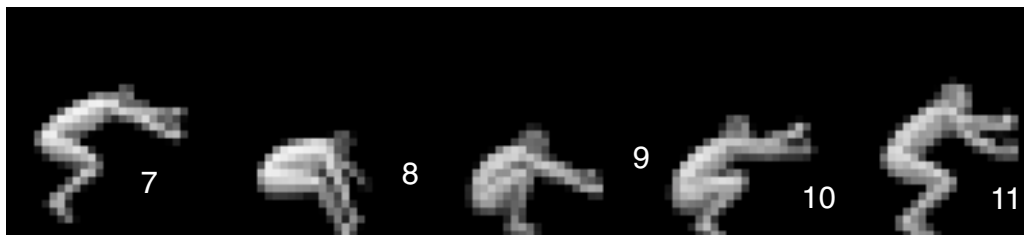


FIGURE 9-42: Complex Jump Sequence (Part 2)

TABLE 9-32: Jumping Primitive Animation Property Summary

Animation Property	Comments
Motion lines	Jumping animations always have very wavy motion lines. Motion lines that are too straight will appear too stiff and rigid and won't look natural when animated.
Motion angles	Jumping animations have very acute motion angles.
Key-frames and in-betweens	Requires at least two key-frames in simple jumping sequences. As always, more frames will result in smoother sequences. In complex jumping sequences, the key-frames are frames 1, 3, 5, 7, 9, and 11.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will require heavier steps and result in slower overall jumping movements and shorter exaggerations between frames.
Flexibility	Flexibility should be emphasized around the arms and legs, particularly at the joints such as the knees and elbows. Depending on the size and the complexity of the object being animated, you might want to emphasize flexibility in the feet as well.

<i>Animation Property</i>	<i>Comments</i>
Secondary actions	Secondary actions occur very frequently in jumping animations. Pay close attention to the head of the character, as it will bounce up and down as the position of the legs and feet change. In addition, hair and clothing will often exhibit secondary actions. For example, hair might flow or bounce (depending on length and style) and clothing might trail behind the character as it picks up speed. For example, imagine a flowing cape or scarf. Incidentally, adding secondary actions to your jumping sequences is an excellent way of tricking the observer into thinking the character is moving in a more realistic fashion than it really is.
Cycles and loops	Since jumps are largely “one-off” actions, cycles and loops have little or no application for these actions.
Tempo	Used extensively to adjust the speed at which the jumping movement occurs. Jumps happen more slowly than most other character actions. Adjust the tempo of the animation accordingly.

The Crawling Primitive

Crawling is an action used by many platform-style games to help the on-screen characters fit into tight spaces and to avoid dangerous obstacles and objects while moving.

The basic act of crawling involves three steps: *bending*, *crouching*, and *moving*.

Bend is where the character begins the crawling sequence. Crouch is where the character assumes the proper position to begin crawling. Moving is the crawling process itself.

All crawling actions involve the *opposite action* of the limbs. In other words, the arms and legs move in opposite directions from each other. For example as the left arm moves backward, the left leg moves forward and as the right arm moves forward, the right leg moves backward. This action uses pushing and pulling to propel the body forward. During a crawling action, it is the legs that push and the arms that pull.

Figures 9-43 and 9-44 represent a typical complex crawling sequence.

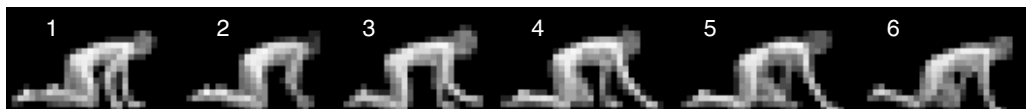


FIGURE 9-43: Crawling Primitive Example (Part 1)



FIGURE 9-44: Crawling Primitive Example (Part 2)

As you can see, a complex crawling sequence can require as many as 11 frames of animation. Here's a frame-by-frame breakdown of the action:


- **Frame 1:** The first frame shows the character fully bent on all fours. The right leg is positioned forward; the right arm is positioned back. The left leg is shifted back and the left arm leads forward. The head is nearly level and facing the ground.
- **Frame 2:** This frame has the right arm moving forward and right leg moving backward. The left arm moves backward and the left leg moves forward. It's interesting to note that both legs are touching the ground at the knees with the calves suspended in the air. This is done in order to propel the body forward and gain momentum.
- **Frame 3:** This shows the progression of the right arm moving forward and up. It moves up in order to gain leverage for the forward motion of the crawl. The right leg is now at the midpoint of its motion and is closely aligned with the position of the left leg. The left leg continues to move forward while the left arm moves backward.
- **Frame 4:** In frame 4, the right arm stretches outward and forward as it attempts to pull the body of the character forward. The right leg shifts backward to aid in stabilizing the body. The left arm pushes back and the left leg moves forward.
- **Frame 5:** Here the right arm is fully extended forward while the right leg continues its backward motion. The left arm reaches back while the left leg aggressively shifts forward.
- **Frame 6:** Frame 6 is similar to frame 5, as all of the limbs continue their direction and force of movement.
- **Frame 7:** In this frame, the right arm is now beginning to move backward and the right leg is beginning to move forward. The left arm and leg start to move forward and backward, respectively.
- **Frame 8:** This frame shows the right arm continuing its backward motion and the right leg its forward movement. The left arm and leg continue their supporting motion as well.
- **Frame 9:** Here the left arm is now outstretched and duplicating the action that the right arm made in frame 4. The right arm and both legs help stabilize the body while this occurs.

- **Frame 10:** This frame has the right arm slowly moving back while the right leg thrusts forward. The left arm is now nearly fully extended, as it approaches the ground. The left leg continues to shift backward.
- **Frame 11:** This frame shows the right arm moving back and coming close to the right leg as it moves forward. The left arm is now completely extended and touching the ground. The left leg continues to shift backward. At this point, the animation would cycle back to frame 1 to complete the sequence and the continuous flow of motion.

NOTE: In most situations, you can also create simple, two-framed crawling actions. However, such animations require heavily exaggerated movements between frames and ample delineation between different parts of the figure in order to convince the observer that movement is actually occurring.

TABLE 9-33: Crawling Primitive Animation Property Summary

<i>Animation Property</i>	<i>Comments</i>
Motion lines	Crawling animations have an almost constant motion line. This is due in large part to the fact that the head and back rarely change position during the course of the crawling motion.
Motion angles	Crawling actions have relatively mild motion angles.
Key-frames and in-betweens	Requires at least two key-frames in simple crawling sequences as shown in Figure 9-21. As always, the presence of more frames will result in smoother crawling sequences. In complex crawling sequences, the key-frames are frames 1, 3, 5, 7, 9, and 11.
Weight and gravity	Can influence the speed at which the animation is displayed. For example, larger objects or denser gravity will require heavier steps and result in a slower overall crawling motion and a shorter exaggeration between frames.
Flexibility	Flexibility should be emphasized around the arms and legs, particularly at the joints such as the knees and elbows. Flexibility should be added to the hands and feet as they tend to bend slightly as they grip the ground in support of the crawling movement.
Secondary actions	Secondary actions can occur during crawling actions but they tend to be very subtle in nature and more often than not can be ignored.
Cycles and loops	Crawling actions make extensive use of cycles and loops to complete the illusion of movement.
Tempo	Used extensively to adjust the speed at which the crawling action occurs. Crawling occurs very slowly in comparison to most other forms of character movement. Adjust the tempo of the animation accordingly.



NOTE: Undoubtedly, you can probably identify a number of other animation primitives that are found in arcade games and are not described here. However, most are probably variations of the ones identified in this chapter. Master these primitives and the rest will come easily to you.

Creating Your Animation Sequences

As you can see, creating animations, even simple ones, involves a lot of time, planning, and attention to detail. However, the process can be simplified somewhat if you follow these nine steps:

1. **Conceptualize**—Before you start designing, always have a clear idea of what you want to animate and how it will look. This will save you time and effort since you can only really start the animation process once you have a clear idea of what needs to be done.
2. **Decide on object behavior**—After determining the object’s look, you need to determine whether or not the object will be animated continuously (using cycles) or is a “one-off” and only animated once (no looping). Objects can have elements of both, however, so you need to decide on this before you start creating animation. Changing this behavior can complicate matters later on in the project.
3. **Choose a grid size**—The next step in the process is to choose a predefined grid size to contain and constrain the object that you are designing. Be sure to copy a number of grid squares to give yourself plenty of room to test and experiment with the animation sequences you create.
4. **Design the key-frames**—When you’re finally ready to start designing, begin by drawing the motion extremes or key-frames needed for the object sequence. To save time and effort, just use simple shapes to represent the main actions of the object. For example, use stick figures if creating character animation or basic geometric shapes (i.e., circles, squares, triangles, etc.) if it’s something else.
5. **Estimate the in-betweens**—Make an estimate of the number of in-betweens you think you will need to complete the sequence. Before you do this, remember that slower moving animations require more frames than faster moving animations. Be conservative when doing this. It’s actually easier to add additional transition frames to a sequence than it is to remove them.
6. **Create object motion lines**—When done, trace the motion line and motion angles for the sequence. Use your painting program’s Line tool to do this. It can always be erased later on. Before continuing, make sure that the properties of the motion line and motion angles are consistent with the type of object being animated. If not, make the appropriate adjustments to the sequence and

position the individual frames until the object starts to conform to these norms.

7. **Apply secondary actions and animation enhancements**—Add any secondary actions that might be needed, and if applicable check to see if the object exhibits sufficient flexibility. This is also a good time to add any additional “character” to your object(s). In other words, feel free to embellish the movement so it looks both convincing and enticing at the same time.
8. **Test each movement**—Don’t forget to test every animation you create. The easiest way to do this is to use your painting program’s Copy tool and copy one grid square to another and then apply an undoable undo. This temporarily combines two frames and then flips between them to produce a quick animated effect. For longer, more complex animations, see if your painting program offers an animation tool so that you can see the entire animation in context and at different tempos. A few of the ones mentioned in Chapter 6 actually do. In any case, be on the lookout for flaws in how the object moves and how the object appears. Often, you will catch minor mistakes such as discolored pixels or missing pixels, or even major mistakes such as improper movement (i.e., not enough exaggeration, too fast, etc.) at this stage of the process. Make corrections to the sequence as needed and do them before handing off your work to the programmer. It’s recommended that you do at least two such tests for complex animations, while one test can suffice for less sophisticated objects.
9. **Repeat**—Repeat the previous eight steps to create all of the animations required by your game.

It’s important to point out that different people have different opinions on how to go about this process. Some, like myself, prefer to create highly detailed figures and then animate them, while others prefer to work from rough items first.

How you choose to proceed really boils down to a matter of time, efficiency, and personal taste. When in doubt, particularly when you’re first starting out, I recommend that you work from rough shapes. Later, when you’re more experienced and comfortable with creating animation, you can work from more detailed object images.

General Animation Tips

- **Remember the relationship between frames and animation smoothness**—This relationship is one of the most important aspects of animation. In order to achieve the illusion of smooth motion, you need to use many frames. If design time or file size limits ever become an issue, you can always compensate by using fewer animation frames. Doing this produces a higher FPS, which can have the effect of making objects appear to move smoother than they actually are.

- **Always account for color**—Color can affect animations the same way it does other types of graphics. Always make sure that you choose suitable colors when creating your animations. Primary actions and secondary actions should be rendered in colors that make them easy to see. Otherwise, the effectiveness of the animation can be compromised. For more information on proper arcade game color usage and selection, refer to Chapters 7 and 8.
- **Use tempo wisely**—Tempo, if used properly, is your friend. If used incorrectly, it is your enemy. Use tempo to pace your animations. Your animations should never appear to move too fast or too slowly, as this will be perceived as unrealistic and distracting. Try to mimic nature. Study the speed at which different types of objects move in different situations. After a while, applying the correct tempo to your animated sequences will become second nature.
- **Try to individualize your objects**—Adding unique and individualized touches to your objects has the effect of making them seem real to the observer. Therefore, every distinct object you create and animate should have some sort of unique “personality” that distinguishes it from the other objects on the screen. One of the easiest ways to do this is to apply different degrees of exaggeration and embellishment (i.e., secondary actions) to each object.
- **Keep it simple**—Adding unnecessary complexity can ruin an animation sequence as it opens up the possibility of introducing flaws and other types of errors into it. To avoid doing this, keep your animations simple as much as possible. Therefore, stick with using established animation primitives and minimal frames unless the object requires more subtle effects. In other words, don’t do any more work than you have to!
- **Use exaggerated elements**—As an animation device, exaggeration adds realism and depth to animations. Therefore, use it as much as possible. Exaggeration is especially important when working with short animation sequences as they have fewer frames available to them in order to create effective and convincing motion.
- **Constantly observe**—Successful animation requires the careful study of the objects around you. Study how different things move. Study books on animation such as those by Eadweard Muybridge in addition to studying how the animation featured in your favorite arcade games was created. Studying these sources will give you useful insights into animation techniques and will enable you to create better and more accurate animations.

In any case, don’t get discouraged if the animation process doesn’t go smoothly for you the first few times you try. Like most things, creating effective animations takes time and experience. Start off by working on simple animation sequences first. After you’re comfortable with the basic animation process and are satisfied with the results, move on to animating more sophisticated subjects.

Remember, when it comes to creating arcade game animation, take baby steps. Never bite off more than you can handle. Take it slow. As the adage says, “Rome wasn’t built in a day,” and neither will your animations. Be patient and keep practicing. The more you do, the better you will eventually get at creating arcade game animation.

Animation Usage in Arcade Games

Although virtually every animation primitive described in this chapter has an application in every arcade game genre, some primitives are more commonly found in some genres than in others. For example, some games, such as shooters, have more complex graphic requirements and therefore can support a wider array of animation primitives. However, other genres such as Pong games tend to be less sophisticated and therefore cannot support the majority of animation primitives.

Table 9-34 summarizes the usage of the common animation primitives in the major arcade game genres.

TABLE 9-34: Summary of Animation Primitive Usage in Arcade Game Genres

<i>Animation Primitive</i>	<i>Maze/Chase</i>	<i>Pong</i>	<i>Puzzler</i>	<i>Shooter</i>	<i>Platformer</i>
Cylindrical	✓	✓	✓	✓	
Rotational	✓	✓	✓	✓	
Disintegration	✓	✓	✓	✓	✓
Color flash	✓	✓	✓	✓	✓
Scissors	✓	✓		✓	
Growing	✓	✓	✓	✓	✓
Shrinking	✓	✓	✓	✓	✓
Piston	✓		✓	✓	
Squeeze	✓		✓	✓	
Swing	✓		✓	✓	✓
Slide					✓
Open/close	✓	✓	✓	✓	✓
Bounce			✓	✓	
Stomp	✓		✓	✓	
Slinking				✓	✓
Flying				✓	✓
Walking				✓	✓

<i>Animation Primitive</i>	<i>Maze/Chase</i>	<i>Pong</i>	<i>Puzzler</i>	<i>Shooter</i>	<i>Platformer</i>
Running				✓	✓
Jumping			✓	✓	✓
Crawling					✓

Due to limitations imposed by different design styles, not all animation primitives can or should be used. For example, the more complex animation primitives really are not suitable for use in a retro-style arcade game. It simply doesn't look right. Conversely, games with realistic design styles should not overuse simple animation primitives as they undermine the element of realism present in these games.

Table 9-35 summarizes the suitability of animation primitives in the most common design styles.

TABLE 9-35: Summary of Animation Primitive Usage in Arcade Game Design Styles

<i>Animation Primitive</i>	<i>Cartoon</i>	<i>Retro</i>	<i>Realistic</i>
Cylindrical			✓
Rotational	✓	✓	✓
Disintegration	✓	✓	✓
Color flash	✓	✓	✓
Scissors	✓	✓	
Growing	✓	✓	✓
Shrinking	✓	✓	✓
Piston	✓	✓	
Squeeze	✓	✓	
Swing	✓	✓	
Slide	✓	✓	✓
Open/close	✓	✓	✓
Bounce		✓	
Stomp	✓	✓	
Slinking	✓		✓
Flying	✓	✓	✓
Walking	✓	✓	✓
Running			✓
Jumping	✓	✓	✓
Crawling			✓



NOTE: The information presented in Tables 9-34 and 9-35 are general suggestions only. Every arcade game project is unique and can define its own rules for the use of animation primitives.