

Chapter 13: Images, WAVs, and Sprites

This chapter will introduce the really advanced multimedia and graphical statements. Saving images to a file, loading them back, playing sounds from WAV files, and really cool animation using sprites.

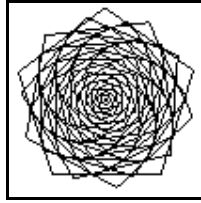
Saving Images to a File:

So far we have seen how to create shapes and graphics using the built in drawing statements. The **imgsave** statement allows you to save your images to one of many standard image formats.


Program 78 Draws a series of pentagons, each a little bigger and rotated to make a beautiful geometric flower. It would be nice to use that image somewhere else. This program creates a PNG (Portable Network Graphics) file that can be used on a Website, presentation, or anywhere else you may want to use it.

```
1 # 5pointed.kbs
2 #
3 graphsize 100,100
4 clg
5 color black,clear
6 for s = 1 to 50 step 2
7     stamp 50,50,s,s,{0,-1, .95,-.31, .59,.81,
8     -.59,.81, -.95,-.31}
9 next s
10 #
10 imgsave "5pointed.png", IMAGETYPE_PNG
```

Program 78: Save an Image



Sample Output 78: Save an Image



**New
Concept**

```
imgsave filename  
imgsave filename, type
```

Save the current graphics output to an image file. If the type is not specified the graphic will be saved as a Portable Network Graphic (PNG) file.

Type maybe specified with either a string extension or using a predefined constant.

String	Constant
"png"	IMAGETYPE_PNG
"jpg" or "jpeg"	IMAGETYPE_JPG
"gif"	IMAGETYPE_GIF

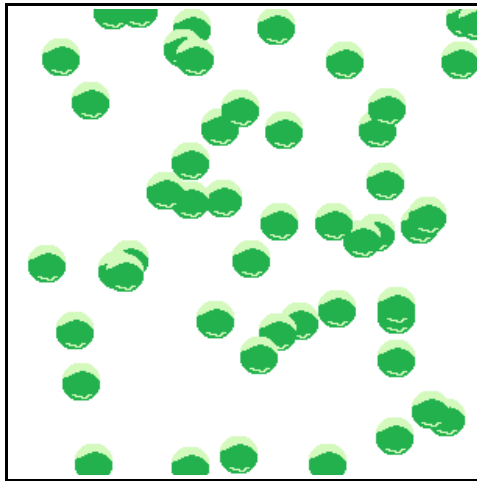
Images From a File:

The **imgload** statement allows you to load a picture from a file and display it in your BASIC-256 programs. These images can be ones you have saved yourself or pictures from other sources.

```
1 # imgloadball.kbs
```


```
2 # load an image from a file
3
4 clg
5 for i = 1 to 50
6     imgload rand * graphwidth, rand * graphheight,
7     "greenball.png"
8 next i
```


Program 79: Imgload a Graphic



Sample Output 79: Imgload a Graphic

Program 79 Shows an example of this statement in action. The last argument is the name of a file on your computer. It needs to be in the same folder as the program, unless you specify a full path to it. Also notice that the coordinates (x,y) represent the CENTER of the loaded image and not the top left corner.

 <p>Warning</p>	<p>Most of the time you will want to save the program into the same folder that the image or sound file is in BEFORE you run the program. This will set your current working directory so that BASIC-256 can find the file to load.</p>
---	---

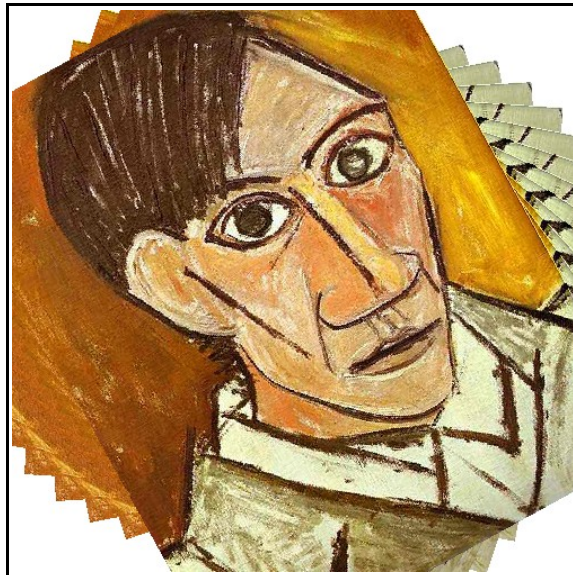
 <p>New Concept</p>	<pre>imgload x, y, filename imgload x, y, scale, filename imgload x, y, scale, rotation, filename</pre> <p>Read in the picture found in the file and display it on the graphics output area. The values of <i>x</i> and <i>y</i> represent the location to place the CENTER of the image.</p> <p>Images may be loaded from many different file formats, including: BMP, PNG, GIF, JPG, and JPEG.</p> <p>Optionally scale (re-size) it by the decimal scale where 1 is full size. Also you may also rotate the image clockwise around it's center by specifying how far to rotate as an angle expressed in radians (0 to 2π).</p>
---	---

The **imgload** statement also allows optional scaling and rotation like the **stamp** statement does. Look at Program 80 for an example.

```
1 # imgloadpicasso.kbs
2 # show img with rotation and scaling
3 # photo from
  http://i988.photobucket.com/albums/af3/fikarvista/pic
  asso_selfport1907.jpg
4
5 graphsize 500,500
```

```
6  clg
7  for i = 1 to 50
8      imgload graphwidth/2, graphheight/2, i/50,
      2*pi*i/50, "picasso_selfport1907.jpg"
9  next i
10 say "hello Picasso."
```

Program 80: Imgload a Graphic with Scaling and Rotation




Sample Output 80: Imgload a Graphic with Scaling and Rotation

Playing Sounds From a WAV file:

So far we have explored making sounds and music using the **sound** command and text to speech with the **say** statement. BASIC-256 will also play sounds stored in WAV files. The playback of a sound from a WAV file will happen in the background. Once the sound starts the program will continue to the next statement and the sound will continue to play.

```
1 # numberpopper.kbs
2 # mp3 files from
3 # http://www.grsites.com/archive/sounds/
4
5 fastgraphics
6 wavplay "cartoon002.mp3"
7
8 speed = .05
9 for t = 1 to 3
10     n = int(rand * 6 + 1)
11     for pt = 1 to 200 step 10
12         font "Tahoma",pt,100
13         clg
14         color black
15         text 10,10, n
16         refresh
17         pause speed
18     next pt
19     speed = speed / 2
20 next t
21 # wait for sound to complete
22 wavwait
23
24 wavplay "people055.mp3"
25 wavwait
26 end
```

Program 81: Popping Numbers with Sound Effects

 <p>New Concept</p>	<pre>wavplay filename wavplay (filename) wavwait wavstop</pre> <p>The wavplay statement loads a wave audio file (.wav) from the current working folder and plays it. The playback will be synchronous meaning that the next statement in the program will begin immediately as soon as the audio begins playing.</p> <p>Wavstop will cause the currently playing wave audio file to stop the synchronous playback and wavwait will cause the program to stop and wait for the currently playing sound to complete.</p>
---	---

Moving Images - Sprites:

Sprites are special graphical objects that can be moved around the screen without having to redraw the entire screen. In addition to being mobile you can detect when one sprite overlaps (collides) with another. Sprites make programming complex games and animations much easier.

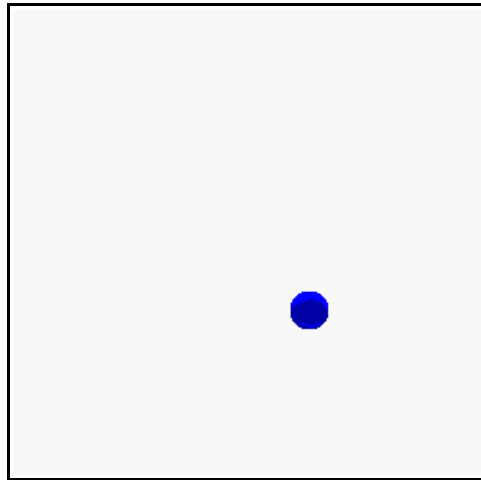
```

1  # spritelball.kbs
2  # sounds from
3  # http://www.freesound.org/people/NoiseCollector
4
5  clg
6
7  spritedim 1
8
9  spriteload 0, "blueball.png"
10 spriteplace 0, 100,100
11 spriteshow 0
12
13 dx = rand * 5 + 5
14 dy = rand * 5 + 5
15

```

```
16 while true
17     spritemove 0, dx, dy
18     if spritex(0) <= spritew(0)/2 or spritex(0) >=
graphwidth - spritew(0)/2 then
19         dx = dx * -1
20         wavplay
"4359__NoiseCollector__PongBlipF4.wav"
21     end if
22     if spritey(0) <= spriteh(0)/2 or spritey(0) >=
graphheight - spriteh(0)/2 then
23         dy = dy * -1
24         wavplay
"4361__NoiseCollector__pongblipA_3.wav"
25     endif
26     pause .05
27 end while
```


Program 82: Bounce a Ball with Sprite and Sound Effects





Sample Output 82: Bounce a Ball with Sprite and Sound Effects


As you can see in Program 82 the code to make a ball bounce around the screen, with sound effects, is much easier than earlier programs to do this

type of animation. When using sprites we must tell BASIC-256 how many there will be (**spritedim**), we need to set them up (**spriteload**, **spritepoly**, or **spriteplace**), make them visible (**spriteshow**), and then move them around (**spritemove**). In addition to these statements there are functions that will tell us where the sprite is on the screen (**spritex** and **spritey**), how big the sprite is (**spritew** and **spriteh**) and if the sprite is visible (**spritev**).

 <p>New Concept</p>	<pre>spritedim <i>numberofsprites</i> spritedim (<i>numberofsprites</i>)</pre> <p>The spritedim statement initializes, or allocates in memory, places to store the specified number of sprites. You may allocate as many sprites as your program may require but your program may slow down if you create too many sprites.</p>
---	--

 <p>New Concept</p>	<pre>spriteload <i>spritenummer, filename</i> spriteload (<i>spritenummer, filename</i>)</pre> <p>This statement reads an image file (GIF, BMP, PNG, JPG, or JPEG) from the specified path and creates a sprite.</p> <p>By default the sprite will be placed with its center at 0,0 and it will be hidden. You should move the sprite to the desired position on the screen (spritemove or spriteplace) and then show it (spriteshow).</p>
--	---

	<pre>spritehide <i>spritenum</i>ber spritehide (<i>spritenum</i>ber) spriteshow <i>spritenum</i>ber spriteshow (<i>spritenum</i>ber)</pre>
New Concept	<p>The spriteshow statement causes a loaded, created, or hidden sprite to be displayed on the graphics output area.</p> <p>Spritehide will cause the specified sprite to not be drawn on the screen. It will still exist and may be shown again later.</p>

	<pre>spriteplace <i>spritenum</i>ber, <i>x</i>, <i>y</i> spriteplace (<i>spritenum</i>ber, <i>x</i>, <i>y</i>)</pre>
New Concept	<p>The spriteplace statement allows you to place a sprite's center at a specific location on the graphics output area.</p>



New Concept

```
spritemove spritenumber, dx, dy  
spritemove ( spritenumber, dx, dy )
```

Move the specified sprite x pixels to the right and y pixels down. Negative numbers can also be specified to move the sprite left and up.

A sprite's center will not move beyond the edge of the current graphics output window (0,0) to (**graphwidth-1**, **graphheight-1**).


You may move a hidden sprite but it will not be displayed until you show the sprite using the **showsprite** statement.



New Concept

```
spritev (spritenumber)
```

This function returns a true value if a loaded sprite is currently displayed on the graphics output area. False will be returned if it is not visible.



New Concept

`spriteh(spriteNumber)`
`spritew(spriteNumber)`
`spritex(spriteNumber)`
`spritey(spriteNumber)`

These functions return various pieces of information about a loaded sprite.

<code>spriteh</code>	Returns the height of a sprite in pixels.
<code>spritew</code>	Returns the width of a sprite in pixels.
<code>spritex</code>	Returns the position on the x axis of the center of the sprite.
<code>spritey</code>	Returns the position on the y axis of the center of the sprite.

The second sprite example (Program 83) we now have two sprites. The first one (number zero) is stationary and the second one (number one) will bounce off of the walls and the stationary sprite.

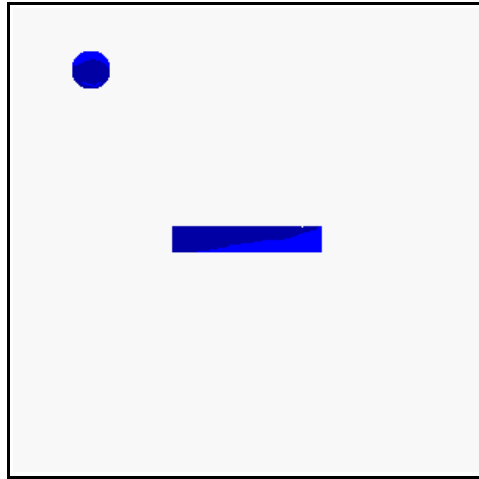
```

1  # spritebumper.kbs
2  # show two sprites with collision
3
4  color white
5  rect 0, 0, graphwidth, graphheight
6
7  spritedim 2
8
9  # stationary bumper
10 spriteload 0, "paddle.png"
11 spriteplace 0, graphwidth/2, graphheight/2
12 spriteshow 0
13


```

```
14 # moving ball
15 spriteload 1, "greenball.png"
16 spriteplace 1, 50, 50
17 spriteshow 1
18 dx = rand * 5 + 5
19 dy = rand * 5 + 5
20
21 while true
22     if spritex(1) <=0 or spritex(1) >= graphwidth -1
23     then
24         dx = dx * -1
25     end if
26     if spritey(1) <= 0 or spritey(1) >= graphheight -1
27     then
28         dy = dy * -1
29     end if
30     if spritecollide(0,1) then
31         dy = dy * -1
32         print "bump"
33     end if
34     spritemove 1, dx, dy
35     pause .05
36 end while
```

Program 83: Two Sprites with Collision



Sample Output 83: Two Sprites with Collision

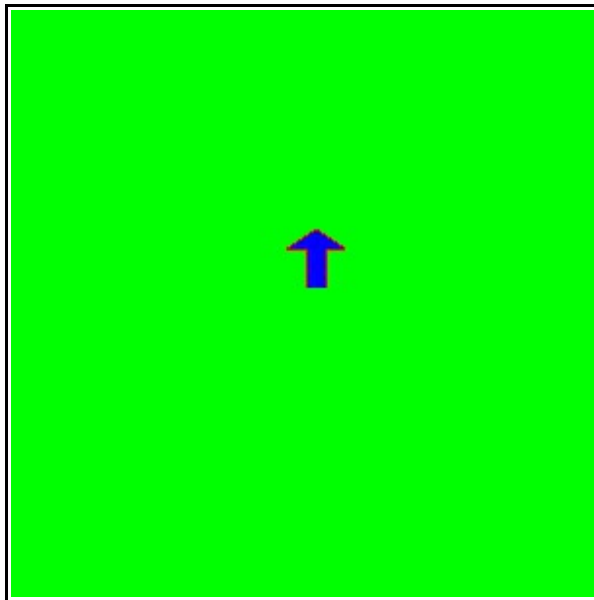
 <p>New Concept</p>	<pre>spritecollide(spriteNumber1, spriteNumber2)</pre> <p>This function returns true if the two sprites collide with or overlap each other.</p>
---	---

Sprites may also be created using a polygon as seen in Chapter 9: Custom Graphics – Creating Your Own Shapes. This is accomplished using the **spritepoly** statement.


```
1 # spritepoly.kbs
2 # create a sprite from a polygon
3 # that follows the mouse
4
5 spritedim 1
6 color red, blue
7 penwidth 1
```


```
8  spritepoly 0, {15,0, 30,10, 20,10, 20,30, 10,30,  
10,10, 0,10}  
9  
10 color green  
11 rect 0,0,graphwidth, graphheight  
12  
13 spriteshow 0  
14 while true  
15     spriteplace 0, mousex, mousey  
16     pause .01  
17 end while
```

Program 84: Creating a Sprite From a Polygon



Sample Output 84: Creating a Sprite From a Polygon

 <p>New Concept</p>	<pre>spritepoly spritenumber, { points } spritepoly (spritenumber, { points }) spritepoly spritenumber, array_variable spritepoly (spritenumber, array_variable)</pre> <p>Create a new sprite from the list of points defining a polygon. The top left corner of the polygon should be in the position 0,0 and the sprite's size will be automatically created.</p>
---	--

 <p>Big Program</p>	<p>The "Big Program" for this chapter uses sprites and sounds to create a paddle ball game.</p>
---	---

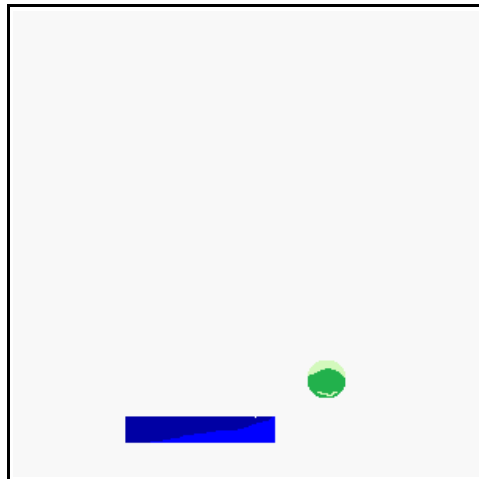
```
1 # sprite_paddleball.kbs
2 # paddleball game made with sprites
3 # sounds from
  http://www.freesound.org/people/NoiseCollector
4
5 print "paddleball game"
6 print "J and K keys move the paddle"
7 input "Press enter to start >", wait
8
9 color white
10 rect 0, 0, graphwidth, graphheight
11
12 spritedim 2
13 color blue, darkblue
14 spritepoly 0, {0,0, 80,0, 80,20, 70,20, 70,10, 10,10,
  10,20, 0,20}
```



```
15  spriteplace 0, 100,270
16  spriteshow 0
17  spriteload 1, "greenball.png"
18  spriteplace 1, 100,100
19  spriteshow 1
20  penwidth 2
21
22  dx = rand * .5 + .25
23  dy = rand * .5 + .25
24
25  bounces = 0
26
27  while spritey(1) + spriteh(1) - 5 < spritey(0)
28      k = key
29      if chr(k) = "K" then
30          spritemove 0, 20, 0
31      end if
32      if chr(k) = "J" then
33          spritemove 0, -20, 0
34      end if
35      if spritecollide(0,1) then
36          # bounce back ans speed up
37          dy = dy * -1
38          dx = dx * 1.1
39          bounces = bounces + 1
40          wavstop
41          wavplay "96633__CGEffex__Ricochet_metal5.wav"
42          # move sprite away from paddle
43          while spritecollide(0,1)
44              spritemove 1, dx, dy
45          end while
46      end if
47      if spritex(1) <=0 or spritex(1) >= graphwidth -1
48  then
49          dx = dx * -1
50          wavstop
51          wavplay "4359__NoiseCollector__PongBlipF4.wav"
52      end if
53      if spritey(1) <= 0 then
```


```
53     dy = dy * -1
54     wavstop
55     wavplay "4361__NoiseCollector__pongblipA_3.wav"
56     end if
57     spritemove 1, dx, dy
58     # adjust the speed here
59     pause .002
60 end while
61
62 print "You bounced the ball " + bounces + " times."
```




Program 85: Paddleball with Sprites



Sample Output 85: Paddleball with Sprites

Exercises:

 <p>Word Search</p>	<pre> i s d d i m e n s i o n o z u s e j i e s c a l e h e w d w k p v c i r z n r o y d a s o z j r p m a u o z l u i v p h a e m i t s t t o m e l w r s c f v f t a m p c c l l a i e q o h o t e e i a i g o i t t w j l i m t l l d w p c t e i q a o l i e p o a e f e w h r w n v r i e t v a i t t j i p q b p p t s s i m d h i s d s o s v i l t i a r m t r r e c u u r w o a g o y p s p r p z h p a p g e y a n d s s e f s s f t s b k i m g l o a d u o </pre> <p>collision, dimension, image, imgload, picture, rotation, scale, spritecollide, spritedim, spritehide, spriteload, spritemove, spriteplace, spritepoly, spriteshow, wavplay, wavstop, wavwait</p>
---	--

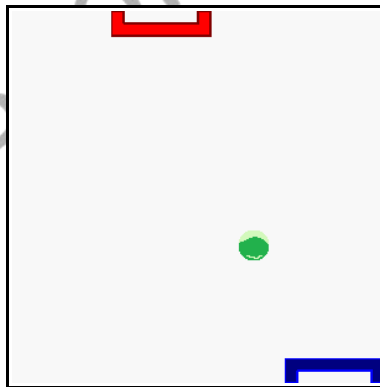
 <p>Problems</p>	<p>1. Write a program to draw a coin, on a graphics window that is 100x100 pixels with a face on it. Save the image as "head.png". Have the same program erase the screen, draw the back side of the coin, and save it as "tail.png". Make the coins your own design.</p> <div style="display: flex; justify-content: center; gap: 20px;">   </div>
--	--

2. Now write a simple coin toss program that displays the results of a coin toss using the images created in program 1. Generate a random number and test if the number is less than .5 then show the heads image otherwise show the tails image.

For an extra challenge make random heads and tails appear on the screen until the user presses a key.

3. Use a program like "Audacity" to record two WAV audio files, one with your voice saying "heads" and the other saying "tails". Add these audio files to the program you wrote in 2.

4. Type in and modify Program 85: Paddleball with Sprites to create a two player "ping-pong" type game. You will need to add a third sprite for the "top" player and assign two keys to move their paddle.



Free eBook Edition