# The Basix of Blitz 3D

# COPYRIGHT NOTICE



# DISCLAIMER

The content is supplied "AS IS". The author and 2CWP disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The user must assume the entire risk of using the content. Any effect that this document, the information contained therein, or the use thereof is the sole responsibility of the user. Use it at your own risk.

# End User License Agreement
## (EULA)

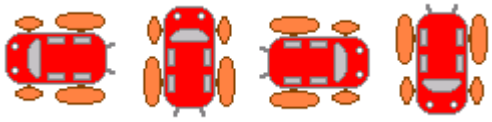
By using/downloading this book and/or support files, the user agrees to all the terms and conditions set out above as well as any included with this book/downloadable content, and all applicable copyright laws. If legally obtained, this document may be printed/stored for an individual's personal non-commercial use only. Any other use requires written authorization from the author, distributor listed above or copyright owner

# *Rotating Images*

When you are using graphics in your programs, sometimes you want to display an image at different angles. For instance, when using an overhead view of a racing car, you may want a picture of the car going left, right, up and down. Let's start with the car below.

car_left.bmp

Instead of using another program like Paint to create a new image of the car for each of the other directions in four separate pictures, we can simply load the original image into our program and let **RotateImage** create the other three pictures for us. The first picture is our original image that we had drawn of the car as a 64x64 pixel size image.



LEFT UP RIGHT DOWN

How did we do that? First draw your image with Paint or some other graphics drawing utility, save the file with a name you can remember and as a file type of BMP (bitmapped). Now write your program to load your original picture and create its rotated images into an array. To make this happen you will also use the **LoadImage** and **CopyImage** commands. Then use the **DrawImage** command to display your new images. Here is the source code for the car demo.

```
;Rotating_Images_1.bb
Graphics 800,600
AutoMidHandle True

Dim car (4)
Global rotation#

car(0)=LoadImage("car_left.bmp")

For i = 1 To 3
   temppic = LoadImage("car_left.bmp")
   rotation = i * 90
   RotateImage temppic,rotation
   car(i) = CopyImage(temppic)
Next

For j=0 To 3
   DrawImage(car(j),(j+1)*64+10,50)
Next

Text 80,100,"Press any key to exit."
WaitKey()
```

You don't have to start with your image pointing to the left. Take a look at this flying aircraft image shown moving in 8 directions using 45 degree increments. The original image points to the upper right. You can see that the programming changed very little to get more rotated images. We've taken the liberty to use compass directions for each direction (N-North, S-South, E-East, and W-West).

NE E SE S SW W NW N

```
;Rotating_Images_2.bb
Graphics 800,600
AutoMidHandle True

Dim aircraft (8)
Global rotation#

aircraft(0)=LoadImage("Jet_UpRight.bmp")

For i = 1 To 7
   temppic = LoadImage("Jet_UpRight.bmp")
   rotation = i * 45
   RotateImage temppic,rotation
   aircraft(i) = CopyImage(temppic)
Next

For j=0 To 7
   DrawImage(aircraft(j),(j+1)*64+10,50)
Next

Text 215,100,"Press any key to exit."
WaitKey()
```

***NOTE***: Because the **RotateImage** command is complex and timely on a large scale of usage, you should load and create all your rotated images at the beginning of your programs instead of during the action or events you are using the rotated images for in your program.

You can combine your images to create nicer looking and more functional programs. When you are drawing your images, make sure that you use the color ***BLACK*** for areas of your picture that will be transparent. That is, these are areas that need to let images drawn behind or drawn first on the screen to show through. A window for instance needs to let you see anything that was drawn behind it through its glass. Because of this use of the color Black for transparency, use a color like ***DARK GREY*** in place of Black for outlining and shading your images. For example, to create a clock we could draw a background clock faceplate that never rotates and use it with an hour-hand that rotates because the time changes every hour. Trust me this works.

Clock Faceplate

Hour-hand

Now we'll show you how to create the Hour-hands for a clock. You don't have to start with the clock hands pointing up, but it's just easier because that's what seems more natural for time. In this example we show a clock face with the hour-hand drawn over it. Take a close look at the source code below to see how we drew our rotating hour-hand over the stationary clock face. The same technique can be used for the Minute/Second hands of a clock.

```
;Rotating_Images3.bb
Graphics 800,600
AutoMidHandle True

Dim hour (12)
Global rotation#

back=LoadImage("clockface.bmp")
hour(0)=LoadImage("hourhand.bmp")

For i = 1 To 12
   temppic = LoadImage("hourhand.bmp")
   rotation = i * 30
   RotateImage temppic,rotation
   hour(i) = CopyImage(temppic)
Next

For j=1 To 12
   DrawImage(back,(j)*64,50)
   DrawImage(hour(j),(j)*64,50)
Next

Text 325,100,"Press any key to exit."
WaitKey()
```

You can rotate your images in any degree increment you choose, meaning that you can use 1-degree increments for 360 directions. The greater number of directions will mostly be used for intense things like navigational or astronomy programs; however you should experiment to find out exactly what you need. This quick chart can help you to determine what degree setting to use for your program.

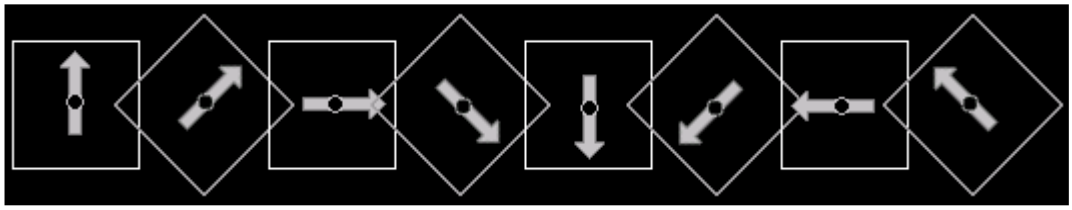
| **Number of Directions** | **Degrees Change for RotateImage** | **Example Uses** |
|---|---|---|
| 2 | 180 | Up and down on screen or left and right on screen |
| 4 | 90 | North, South, East and West |
| 8 | 45 | Low-level moving objects |
| 12 | 30 | Hour-hand on clock |
| 16 | 22.5 | Medium-level moving objects |
| 32 | 11.25 | High-level moving objects |
| 60 | 6 | Minute/Second-hand on clock |

Did you notice the **AutoMidHandle** function? This function allows you to control where the pivot point for rotating an image will be. TRUE sets rotation on the middle-most pixel of the image. FALSE sets rotation on the pixel at position 0,0 or the upper-left most pixel of the image. Look at these two examples showing both uses of the AutoMidHandle function for a simulated magnetic compass.

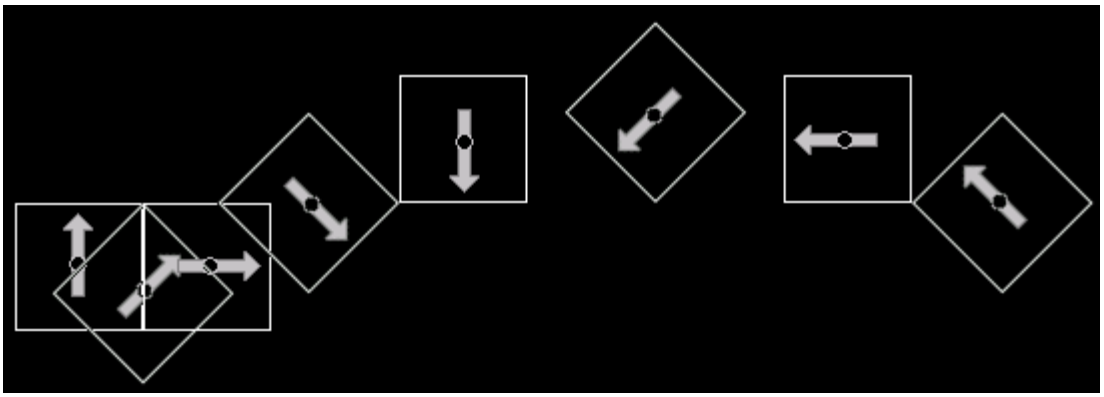
Background

Arrow

Here we rotate the arrow with AutoMidHandle set to TRUE.



Here we rotate the arrow with AutoMidHandle set to FALSE.



You can see how the arrow rotates the way we originally desired by setting AutoMidHandle to True as shown below.