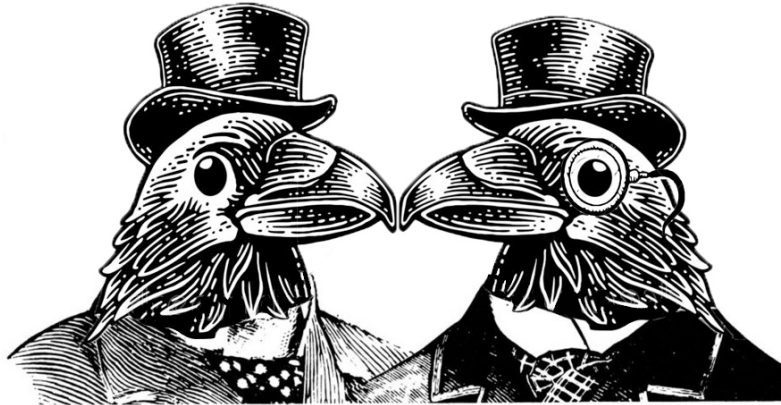


## UNPROTECTED

Backups may be made using standard copying procedures.

## COMPATIBLE

Any version of Apple II  
DOS 3.3 and ProDOS



# BLITTERANG

**HIGH-PERFORMANCE GRAPHICS ENGINE  
FOR APPLESOFT AND ASSEMBLER  
by ROBY SHERMAN**

### BlitLib

Add high performance image drawing to your Applesoft BASIC or assembly language programs in overwrite or XOR (inverted/xdraw) type modes

Blitlib directly uses the graphics you create in a standard image format, using your favorite hi-res picture editing software, allowing for easy and instant access to your graphics for ongoing image revisions and last minute tweaks

### GetCoords

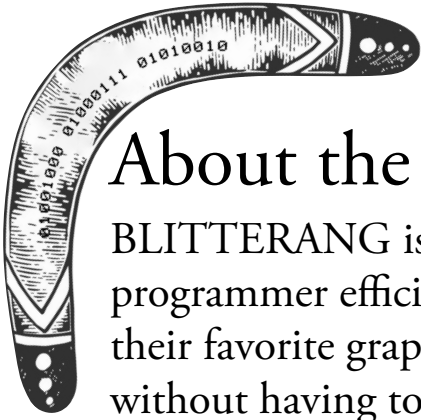
Quickly validate and identify the parameter values needed by Blitlib to display your images on the screen.

### ScreenShift

Invert the hi-res page, force it to shift between green/purple and blue/orange palettes, with this small library

### Demo Programs

Easy examples in BASIC and assembly show you how to put BLITTERANG to work



## About the Disk(s)

BLITTERANG is a machine language library designed to help the programmer efficiently add custom "draw and go" images (created in their favorite graphics programs) to their own software creations without having to assemble the graphics into a proprietary (and tough to edit) format and display them on the hi-res screen far more efficiently than with shape tables or other display options.

## WELCOME, BEGINNERS

BLITTERANG is for the semi-experienced Applesoft or assembly programmer. If you can BLOAD files, POKE memory addresses, and perform CALLs, you know enough to start using BLITTERANG to add graphics to your programs.

## DISTRIBUTION AND SUPPORT

The BLITTERANG software and its accompanying manual may be distributed non-commercially. Give a copy to your Apple II-loving friends, neighbors, and future in-laws.



You are free to incorporate BLITTERANG into your own personal software creations but you must BUY A COPY (gasp!) in order to be able to legally distribute the library as part of any sellable software product! For large endeavors, one copy must be purchased for every 1,000 sales of your software product.

Licensing BLITTERANG has a number of other benefits as well including, access to source code listings, priority support, recompilation / relocation assistance, etc. You also let the author know that you care and support Apple II software development! Please check the Crow Cousins web site for more details!

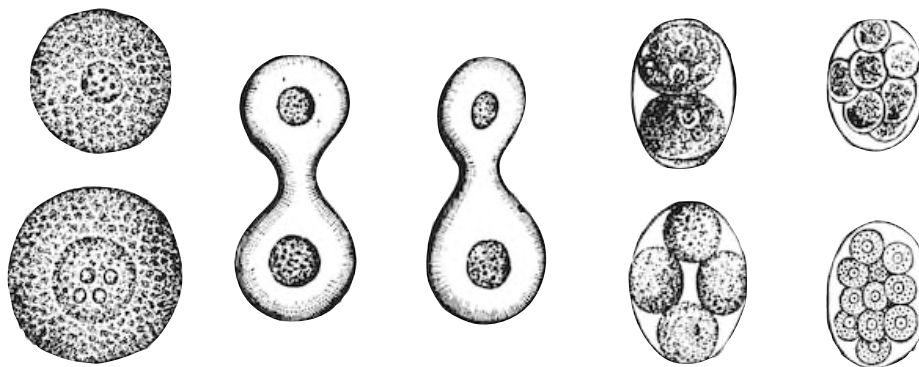
# BLITTERANG FACTS

Before we get too far into what Blitterang can do, it might be good to explain what blitting actually means! In a computer system, a blitter is a circuit or piece of software that is dedicated to the rapid movement and modification of data within that computer's memory. A blitter is capable of copying large quantities of data from one memory area to another relatively quickly.

In this case, BLITTERANG specializes in copying specified regions of your pre-drawn graphical images (what we call "[cells](#)") to the Apple II hi-res screen (\$2000-\$3FFF) for display or animation purposes with minimal overhead and maximum speed.

You can draw/edit/refine your cells, using normal graphics drawing programs and have instant access to them via BLITTERANG without having to constantly embed them back into your software code or encode them into some other proprietary (and less accessible) format.

Can't fit all of your cells into a single image? Please refer to [OPTIONS FOR USING MULTIPLE SOURCE IMAGES](#) for some ideas.



## OTHER FEATURES

Besides being "wicked fast" BLITTERANG offers a number of wonderful built-in capabilities:

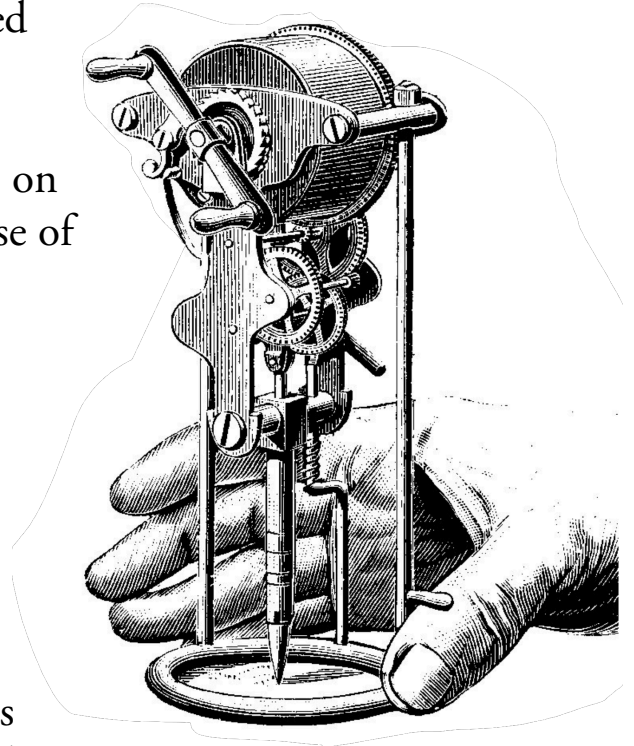
- Support for both bit or byte level image blitting (see the next section for more details)
- Overwrite and XOR/XDRAW type drawing modes
- Compact 1.5K library size for situations where bit-level blitting is not required
- Support for X coordinates beyond 255 (\$FF), up to the full 279 (\$117) value
- Automatic memory bleed / wrap-around / corruption prevention on reads and write from source images and hi-res screen memory

## BLIT MODES

BLITTERANG can copy multiple cells from a single image onto your hi-res screen using its two main blit modes:

Pixel (or bit)-level blitting - Cells are copied to the hi-res display based on the exact X and Y pixel coordinates you specify. This allows for more precise positioning of cells on the hi-res screen but it comes at the expense of speed (and 3k of memory). (Default)

Byte-level blitting - Cells are copied to the hi-res display, aligned at the whole byte level. This eliminates a vast amount of overhead (providing a 2-3x performance boost in the process) but comes at the expense of less granularity in positioning the cells on the hi-res screen (Y coordinates are at the pixel level, but X coordinates and width are rounded to the nearest bytes for drawing purposes).



Essentially this means that moving cells along the X-axis in the byte-level blit mode will cause them to jump 7 pixels at a time or, (if you care about the color not flipping, 14 pixels at a time).

This may or may not be a big deal, depending on what you're trying to do with a given blitted cell but, luckily, there is a simple enough technique to get around this limitation... **CREATE MULTIPLE CELLS!** That's right, you can create multiple cells that contains more granulized movement and display them sequentially before you jump to the next byte position along the X-axis!

## APPLY MODES

BLITTERANG also supports two different techniques for applying the copied cell data onto the hi-res display:

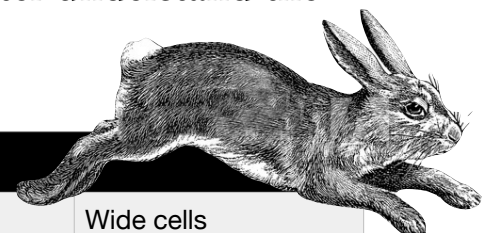

Overwrite - All cell pixels will overlay the corresponding target area on the hi-res screen. (Default). This produces the least amount of flickering but it should only be used in circumstances where cells will not intersect with each other (or the cells are square / rectangular shaped)

Exclusive Or (XOR) - Creates an "inverse" effect for all of the cell's lit pixels. This is similar to using the XDRAW command in Applesoft BASIC (although certainly much faster) to draw (and erase) a shape on the screen while preserving the background.

## HOW FAST IS IT?

As with all things in life, it depends... Are you calling BLITTERANG from Applesoft BASIC, compiled BASIC, or assembly? How large is the image you wish to display? Is the image larger vertically or horizontally? This simple table will help you to better understand the speed tradeoffs for various uses:

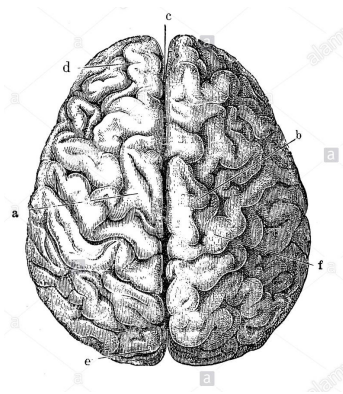
FASTER / BETTER PERFORMANCE		
Assembly Language	Byte-Level Blitting (with Overwrite)	Wide cells
Compiled BASIC	Byte-Level Blitting (with XOR)	Tall cells
Applesoft BASIC	Pixel-level Blitting (with Overwrite)	
	Pixel-level Blitting (with XOR)	
SLOWER / LOWER PERFORMANCE		



```
49 27 4d 20
46 4c 59 49
4e 47 21
```

## MEMORY USAGE

BLITTERANG typically loads into \$6000 and uses 1.6K - 5K depending on whether you want to use byte-level or pixel-level, blitting, respectively (or both).



49152	ROM	\$C000
38400	BASIC.SYSTEM	
	HIMEM MEMORY GROWS DOWNWARD FROM HERE	\$9600
32000	FREE MEMORY OR COMPILER RUNTIMES (E.G. BEAGLE COMPILER)	\$7D00
29696	FREE MEMORY	\$7400
26612	BLITTERANG: PRESIFT  (REQUIRED FOR PIXEL-LEVEL BLITTING)	\$6700
24576	BLITTERANG: BLITLIB  (REQUIRED)	\$6000
16384	HI-RES PAGE 2  (DEFAULT SOURCE IMAGE CELL SPACE USED BY BLITTERANG)	\$4000
8192	HI-RES PAGE 1  (DEFAULT DRAWING SPACE USED BY BLITTERANG)	\$2000
2048	APPLESOFT PROGRAMS GROW UPWARD FROM HERE	\$800
1024	TEXT PAGE	\$400
0	(RESERVED)	\$0

# HOW TO USE BLITTERANG

## STEP #1: LOAD IT UP

To use BLITTERANG, you must BLOAD, at the very least, the BLITLIB binary file into \$6000. On the ProDOS version of BLITTERANG, the binary lives in the /LIB directory.



```
BLOAD LIB/BLITLIB,A$6000
```

If you wish to also use the bit/pixel-level blit capabilities (which are enabled by default), you must also BLOAD the PRESIFT file into \$6700, which can also be found in the /LIB directory:

```
BLOAD LIB/PRESIFT,A$6700
```

## STEP #2: LOAD A SOURCE IMAGE FILE

Before BLITTERANG can draw images for you, you will need to load an image file containing the various "cells" you created. You can use your favorite graphics programs as long as the image is saved in an uncompressed format (34 sectors in DOS 3.3 or 17 blocks in ProDOS). Source image files must meet certain standards in order to work well with BLITTERANG. Refer to CREATING SOURCE IMAGE FILES for more information.

By default BLITTERANG expects source image files to reside in hi-res page 2 (A\$4000). We'll discuss customizing this later. For now, let's load our demo image at this address.

```
BLOAD PICS/PI.BIRDS,A$4000
```



## STEP #3: CONFIGURE BLITTERANG

Using POKEs from BASIC, let's tell BLITTERANG about the cell in the source image we want to display, where we want it to appear, and how we want it displayed.

We'll cover these parameters in more detail, later in the manual so they'll all make sense!

```
⌈POKE 24594,1: REM BYTE-LEVEL BLIT MODE
⌈POKE 24595,0: REM OVERWRITE TARGET PIXELS
⌈POKE 24579,1: REM SOURCE X BYTE OFFSET
⌈POKE 24580,30: REM SOURCE Y BYTE/PIX OFFSET
⌈POKE 24588,86: REM IMAGE WIDTH IN PIXELS
⌈POKE 24582,55: REM HEIGHT IN PIXELS
⌈POKE 24587,0: POKE 24586,105: REM DEST X
⌈POKE 24585,70: REM DEST Y
```

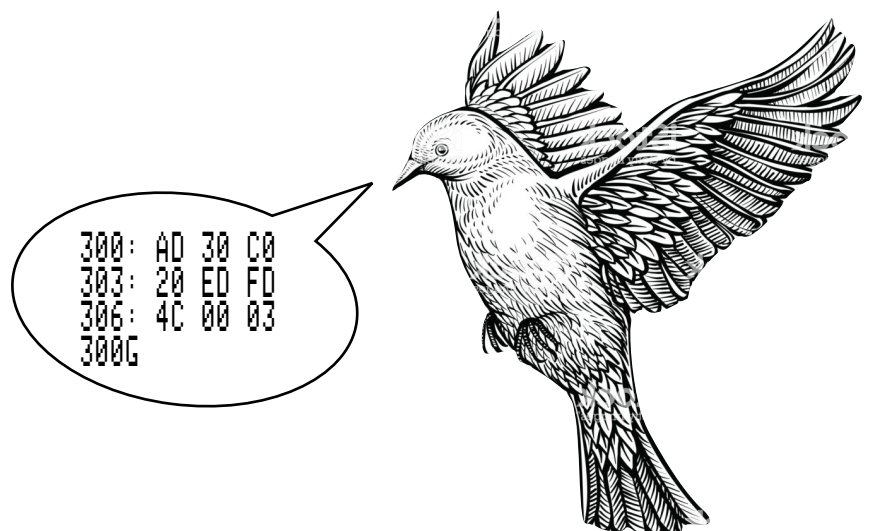
## STEP #4: CALL

Everything is ready to display your first image! If you're not already in HGR mode, let's start it and then call BLITTERANG to display the image!

```
⌈HGR
⌈CALL 24576
```

You can also call it from assembly language:

```
JSR $6000
```



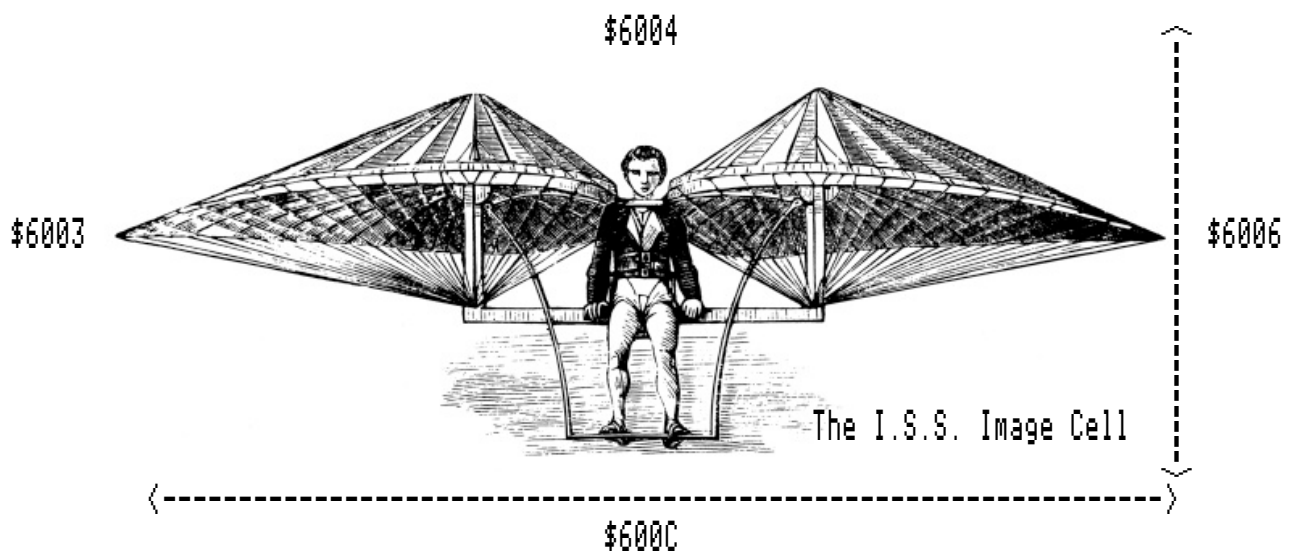
# CONFIGURABLE PARAMETERS

## OPERATION / DRAWING MODES


PARAMETER NAME	ADDRESS	VALUE RANGE	DESCRIPTION
TMODE	24594 (\$6012)	00 (Default), 01	<p>Sets the blit mode used when transferring a cell into the target hi-res page's memory:</p> <p><b><u>Pixel / bit level blitting (00)</u></b> is used for precise image placement on the screen at the specified X,Y target coordinates (at the sacrifice of speed and slight image quality).</p> <p><b><u>Byte level blitting (01)</u></b> is used for less accurate placement along the X-axis (rounded down by up to 7 pixels to the nearest byte), but with tremendously faster performance.</p>
AMODE	24595 (\$6013)	00 (Default), 01	<p>Sets the apply mode used when drawing copied pixels onto the target hi-res page:</p> <p><b><u>Overwrite (00)</u></b> is used to replace all target pixels overlapping with the cell</p> <p><b><u>Exclusive Or (XOR) (01)</u></b> is used to invert any target pixels with defined colored pixels (bits) in the cell. Redrawing the same image again, in the same target location, will cause the overlaid image to disappear.</p>

# SOURCE IMAGE / CELL DEFINITION

PARAMETER NAME	ADDRESS	VALUE RANGE	DESCRIPTION
SRCLEFT	24579 (\$6003)	00-39 (\$00-\$27)	Starting left byte position for the cell within the source image. This can be found by using the GETCOORDS program or by simply taking the X pixel location within the image and dividing it 7.  We <u>could</u> perform this calculation for you, but we won't, simply because we want to ensure that you're thinking about the left side alignment of your cell at the byte level. Please see <i>CREATING SOURCE IMAGE FILES</i> for more information.
SRCTOP	24580 (\$6004)	00-191 (\$00-\$BF)	Starting top position for the cell within the source image, in bytes/pixels
SRCWPIX	24588 (\$600C)	01-255 (\$01-\$FF)	The width of the cell in pixels. When byte-level blitting is used, the width is bumped to align with the next largest byte.  For performance purposes, the cell may not be wider than 255 pixels.
SRCHPIX	24582 (\$6006)	01-190 (\$01-\$BE)	The height of the cell in pixels (which also happens to be bytes)



## TARGET DEFINITION



PARAMETER NAME	ADDRESS	VALUE RANGE	DESCRIPTION
X2 / TARLPIX (Low Order Byte)	24586 (\$600A)	If TARLXPIXH =0: 00-255 (\$00-\$FF)  If TARLXPIXH=1: 00-24 (\$00-\$18)	The X coordinate on the hi-res screen where you wish to start drawing the cell.  This is used in conjunction with TARLPIXH. See next entry.
X1 / TARLPIXH (High Order Byte)	24587 (\$600B)	00-01 (\$00-\$01)	The high order byte used to specify target X coordinates >255. If X<255 then set this to 00.  For example: If X=270 then TARLPIXH=1 and TARLPIX=X-256 (14 in this case)  Put another way (in BASIC): X1=INT(X/256):X2=X-(X1*256)
Y / TARTPIX (TARTOP)	24585 (\$6009)	00-192 (\$00-\$C0)	The Y coordinate on the hi-res screen where you wish to start drawing the cell

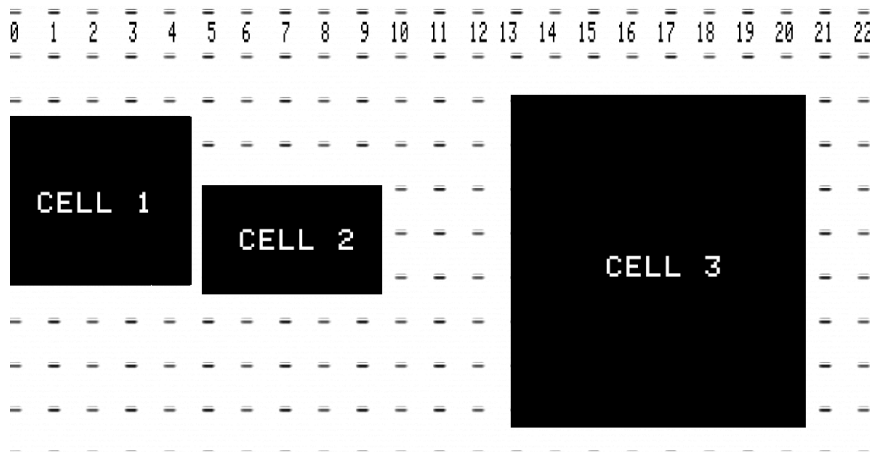
## ADVANCED SETTINGS (PLAY AT YOUR OWN RISK!)

ADDRESS	DESCRIPTION
24932, 24997 (\$6164), (\$61A5)	The high order byte pointing to the area of memory where BLITTERANG will draw. By default this value is 32 / \$20 corresponding to hi-res page 1 (\$2000)
24675, 24976 (\$6063), (\$6190)	The high order byte pointing to the area of memory where BLITTERANG reads source image / cell data from. By default this value is 64 / \$40 corresponding to hi-res page 2 (\$4000)



# CREATING SOURCE IMAGE FILES

BLITTERANG can access any standard hi-res image created with your favorite paint programs. These images are broken down into "cells" with each cell containing the shape(s) you wish to have drawn on the hi-res screen at one time:



You can have as many cells as you can squeeze into the image 280x192 pixels (40 bytes x192 bytes), as long as you abide by one simple rule:

The left-most boundary of any cell  
must begin within its own unique byte

This means that in standard hi-res, since a single byte contains 7 on-screen horizontal pixels, you must take care not to let two cells in a share pixels within that same byte.

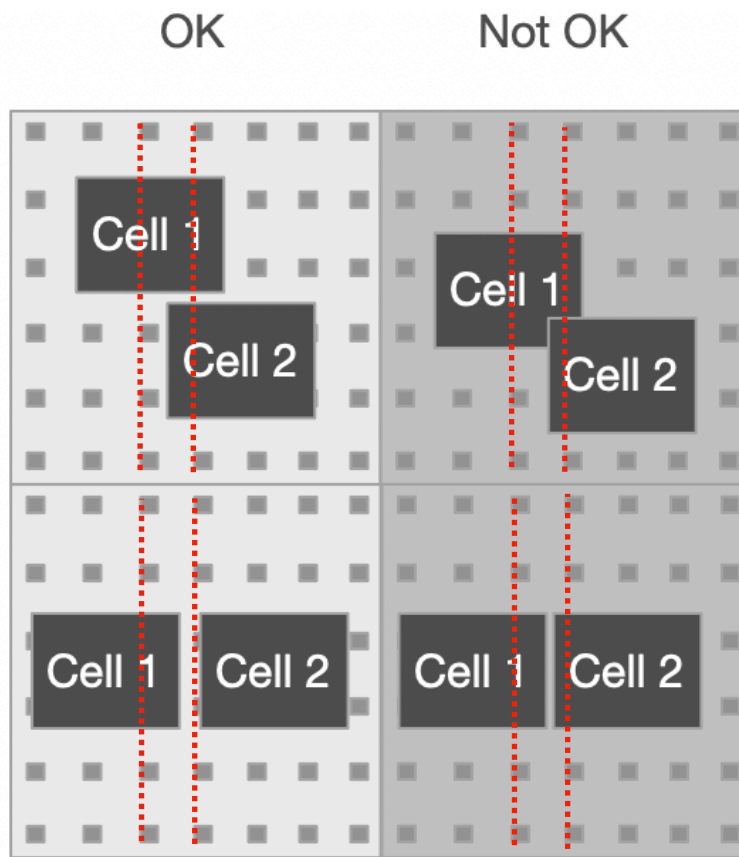


The easiest away to guarantee that you are following this rule is to make sure there is a minimum of 7 horizontal pixels between the left and right edges adjacent cells.

Can you make this spacing any tighter? Absolutely! On the next couple of pages we'll discuss some other tools and techniques to help!

## USE OUR TEMPLATE!

We've created a sample image for you, called PI.TEMPLATE in the PICS subdirectory. It contains byte markers (lit pixels) showing where each column of bytes begins on the source image. The 6 horizontal pixels following these markers are all part of the same on-screen byte.

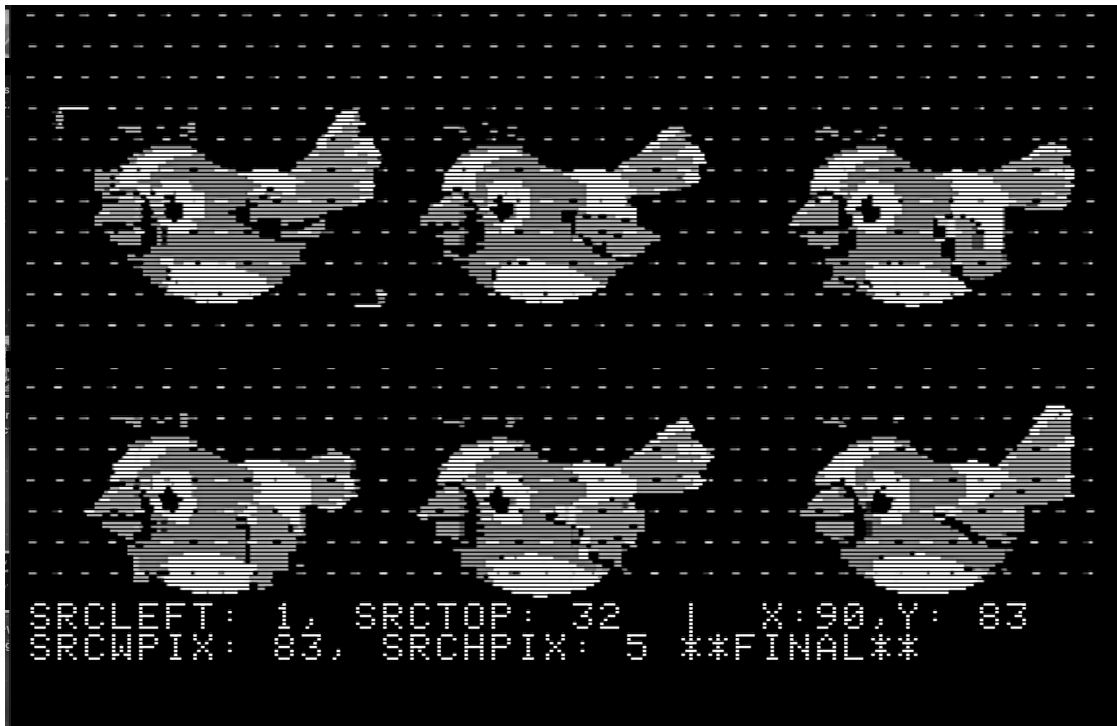


Simply make a copy of this file and open it in your favorite paint program, and start drawing your various object cells, making sure each cell's leftmost edge doesn't overlap with any other cell within the same space between the columns of lit pixels.

When you're done, save your picture and then GETCOORDS to help you determine the parameter values used by BLITTERANG for each cell you wish to draw.

## GETCOORDS

Are you tired of counting pixels to figure out the dimensions of each cell? Did you want to triple check that your cells are spaced properly? Just RUN our GETCOORDS helper program and enter the path to the image you want to check to get the answers you're looking for quickly!

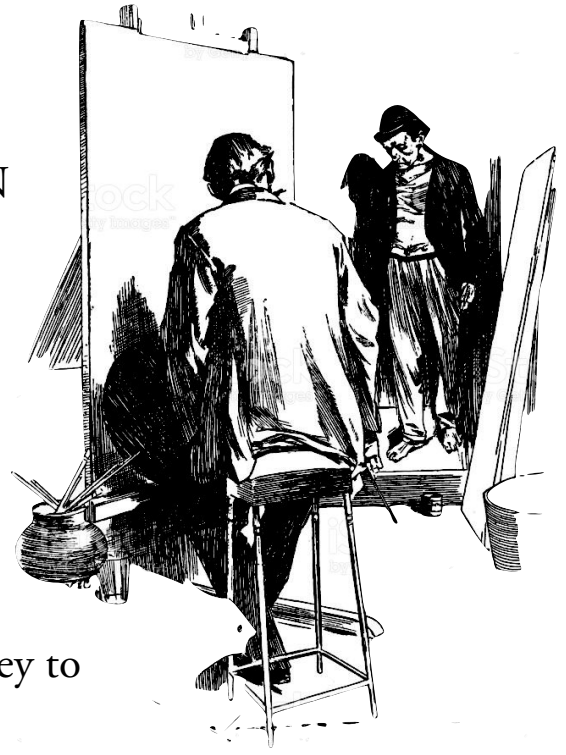


GETCOORDS will display the image containing all of the shapes (cells) available for drawing. The image will be superimposed with the byte markers (lit pixels) to help you identify the start and stop of the cells and to verify that nothing is overlapping within the space in-between.

Using the W/A/D/X keys, move the blinking cursor around the screen mark each cell's top/left and bottom/right locations and GETCOORDS will provide you with the appropriate location and size parameters used by BLITTERANG (e.g. SRCLEFT, SRCWPIX, etc.)

When you get to the top/leftmost location of a given cell, press the RETURN key to mark this position. Next, move the cursor to the cell's bottom/rightmost position and press RETURN again.

You will then be presented with a list of BLITTERANG parameters with their decimal values to use in your programs for drawing the cell (positional X and Y bytes on the source image, width, height, etc.). Write down all of the **\*\*final\*\*** parameter values. If you need to repeat the process, simply press a key to clear the results and start over.



To move the cursor along faster, simply hold down the CTRL key while pressing the movement keys. Need to see below the text window? No problem! Simply press the SPACE BAR to show / hide the rest of the hi-res page!

## Some Other Important Notes About GETCOORDS

- If the left side of a top/left mark is not aligned to the beginning of a whole byte, GETCOORDS will round SRCLEFT down to this position.
- The width (SRCWPIX) assumes that you will be using bit (pixel)-level blit alignment. If you plan on using byte-level blitting, this value will be rounded up to the next whole byte at drawing time.



# DRAWING YOUR FIRST IMAGE

## Try It In BASIC!

It's time for a pop quiz! Run GETCOORDS and load the example image PICS/PI.MISC. Using the keys described on the previous page, scroll the cursor around on the screen to identify the SRCLEFT (byte offset), SRCTOP (Y pixel value), SRCWPIX (image width in pixels), and SRCHPIX (image height in pixels) for the image of the snail! We'll use this information to define a new cell.



Do you think you found the right values? Compare your answers to ours below:

```

]POKE 24579,0 : REM SRCLEFT
]POKE 24580,66: REM SRCTOP
]POKE 24588,23: REM SRCWPIX
]POKE 24582,16: REM SRCHPIX

]X=100:X1=INT(X/256):X2=X-(X1*256):
POKE 24586,X2:POKE 24587,X1: REM TARGET X

*** OR ***

]POKE 24586,100:POKE 24587,0: REM TARGET X (RUNS
FASTER IN BASIC DUE TO NO MATH)

]POKE 24585,100: REM TARGET Y
```

Let's try to draw the snail! Use the following commands to set up the environment once Blitlib has been loaded:

```

]BLOAD PICS/PI.MISC,A$4000
]POKE 24594,0: REM TMODE 0:BIT-LEVEL BLIT
]POKE 24595,0: REM AMODE 0:OVERWRITE MODE
]HGR
]CALL 24576
```

If everything went as planned, you should have a wonderful little snail now drawn on your screen! If not, check your values and try again!

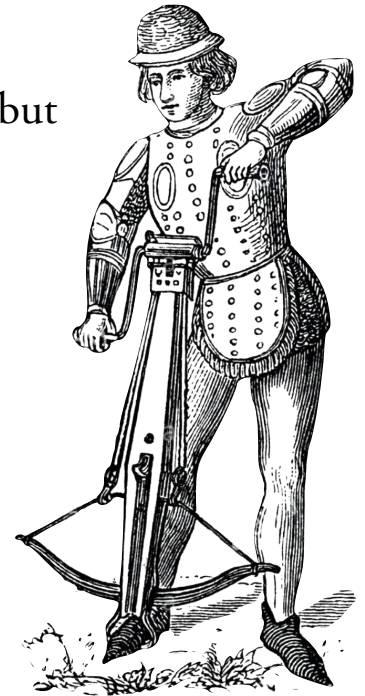
## Try It In (Mini) Assembler

Here's a similar example as the one on the previous page but written in assembler.

```
JBLOAD PICS/PI.MISC,A$4000
JHGR
JCALL-151
*!
!300: LDA  #$00          ; SRCLEFT
!302: STA  $6003
!305: LDA  #$42          ; SRCTOP
!307: STA  $6004
!30A: LDA  #$17          ; SRCWPIX
!30C: STA  $600C
!30F: LDA  #$10          ; SRCHPIX
!311: STA  $6006

!314: LDA  #$00          ; TARGET X1
!316: STA  $600B
!319: LDA  #$64          ; TARGET X2
!31B: STA  $600A
!31E: LDA  #$64          ; TARGET Y
!320: STA  $6009
!323: LDA  #$00          ; TMODE 0
!325: STA  $6012
!328: LDA  #$00          ; AMODE 0
!32A: STA  $6013
!32D: JSR  $6000          ; CALL BLITLIB
!330: RTS                ; END

!(RETURN)
*300G
```



(Yes, the redundant LDA #\$00's are there for illustration purposes only.)

# OTHER PROGRAMS

## The SCREENSHIFT library

LIB/SCREENSHIFT is a compact library used for on screen special effects such as inverting the hi-res screen, forcing on the hi-order bit on all bytes (making all green/purple dots orange/blue), or forcing off the hi-order bit on all bytes (making all orange/blue dots green/purple).

To use the library, simply load it into \$300 and set the parameters:

```
LIBLOAD LIB/SCREENSHIFT,A$0300

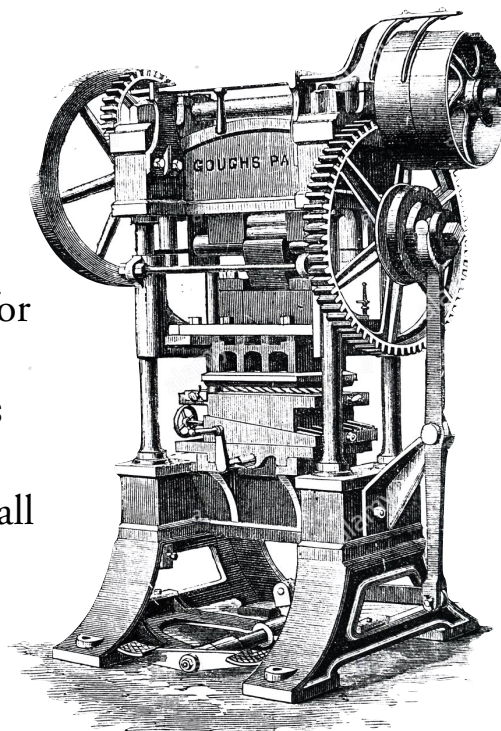
REM ## INVERT
POKE 779,255:POKE 780,89:CALL 768

REM ## SET HIGH ORDER BIT (ORANGE/BLUE)
POKE 779,128:POKE 780,25:CALL 768

REM ## UNSET HIGH ORDER BIT (GREEN/PURPLE)
POKE 779,127:POKE 780,57:CALL 768
```

By default, SCREENSHIFT uses hi-res page 1 (the same as blitlib), but it can be easily modified to use page 2 by performing the following:

```
POKE 771,64:POKE 802,96
```



Original  
Screen Shifter

## Demo Programs

The BLITTERANG disk contains a few fun and interesting programs demonstrating various capabilities while providing several opportunities to pick things apart for learning purposes.

Most demos showing off simple capabilities are coded in Applesoft BASIC so they are easy to view. Any assembler source code used for performance demonstrations in FLAPPY, SNOW, and HEARTS may be found in the LIB directory for use with Merlin and other tools.



PROGRAM	DESCRIPTION
GREETING	Our homage to the classic Beagle Bros graphics greetings programs
FLAPPY	Demonstrates bit- and byte-level blit performance rates for overwrite apply mode drawing from BASIC and assembler
SNOW	Demonstrates blitting performance and behaviors of XOR/XDRAW based apply mode from assembler
HEARTS	Similar to SNOW but using the XOR/XDRAW apply mode to draw larger, colored cells
TROPI	Demonstrates automatic offscreen trimming for cells skirting the right and bottom edges of the screen, along with left side cell trimming techniques that may be used as a cell "enters" the screen
RADIOACTIVE	Demonstrates the full-screen hi-res capabilities of the SCREENSHIFT library (inverse, force high-bit on, force high-bit off)
XOR.DEMO	A simple example of how XOR/XDRAW drawing interacts with a variety of background colors
AUTO.ROULETTE	A slightly morbid example game written in Applesoft BASIC. How much distance can you cover without hitting the mystery pedestrian?

# Advanced Techniques

## Options For Using Multiple Source Images

In some circumstances, you may have more images to display than can easily fit on a single source image. Depending on your performance needs there are several ways you can handle this which are briefly discussed below.

- Read a different source image from disk

This is the slowest option but it allows you to load as many source images as you need / can fit from disk, directly into \$4000

- Keep multiple images in AUX memory

If you are only interested in supporting enhanced //e's or higher, another option might be to copy multiple source images to AUX memory and then copy it back down to \$4000 in main memory when you need it.

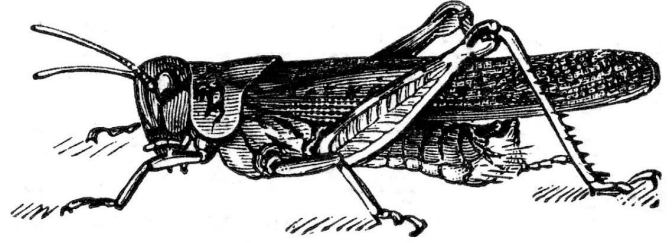


For more information on how this may be implemented, we highly recommend *Inside The Apple //c* By Gary Little which may be found on Asimov and other Apple II mirrors.



Regardless of which technique you choose it is highly recommended that you group the cells on your source images in such a way to minimize the number of loads / memory moves needed during a single drawn display or animation cycles, to reduce painful waiting for the user!

# FAQs



Q> Do I have to change every single parameter each time I call BLITTERANG to draw?

A> No. As long as the values for a given parameter (e.g. width or height) haven't changed, you don't have to set them again. This will actually save you some CPU cycles in drawing a large number of images.

Q> Do I need to align the top, bottom, or right sides of my cells to byte markers?

A> No. The top and bottom sides of a cell can start and stop at any Y coordinate.

The right-hand side of the cell can stop anywhere as long as it doesn't end up sharing a byte with another cell's left-hand side.

As stated earlier in the manual, this can be easily achieved by ensuring that there is a minimum of 7 pixels of horizontal space between each cell, or by using the template image, etc. Please refer back to CREATING SOURCE IMAGES for more information!



Q> Why do I see color clashing around images in the background when I use XOR to draw a cell image on top of them?

A> The XOR may be transferring a high order bit and then taking the opposite value in the underlying image. To prevent this from happening, ensure that the colors in your cell do not have the high bit set. In other words, make sure they are colored with black 0, green, purple, or white 0.

Q> GETCOORDS shows me that the left side of one of my cells doesn't perfectly align to a byte offset, is this a big deal?

A> Not really. If, for example, your cell can fit between two byte markers but it actually spans across three byte markers instead, that's fine as long as there are no conflicting pixels from other cells within the same area. It just means that it will take longer for each line of object to be drawn, because it is processing more data.

Q> When I draw a cell, the beginning of an adjacent cell is also drawn, why?

A> There may be several reasons for this:

- SRCWPIX is set too wide
- You are using byte-level blitting and the right side of the cell you're drawing overlaps the same two byte marker region as the beginning of a different cell.

