Chapter 22
Anims and Animbrushes

by Martin Kees

(This facility was contributed by Martin Kees. We at Delta Research are very grateful to Martin for his generosity. Martin also contributed code for the ARexx interface and assisted us greatly with beta testing. Fred Fish disk 516 is all Martin's work. It includes a demo version of a CEL animation program called XL which allows "onion-skin" drawing of Animations. It also includes a fascinating puzzle called Enigma, a loom simulator for weaving design, and more. Watch for more great work from this talented programmer.)

Introduction

Amiga programs use a standard IFF file format for exchanging graphic images. These include still pictures, animations, and animbrushes. This allows you to move images between drawing programs like DeluxePaint, presentation programs like Amiga Vision, image processing programs like Art Department Professional, and other programs.

This toolbox provides routines that allow you to load animations and animbrushes, to play the animations and access the frames of an animbrush. Using JForth will allow you to control and coordinate an animated presentation at a very high level.

You are encouraged to study the anim source code files, modify, and expand the code to meet your own needs. If you wish to do a lot with animation you should consider upgrading your AMIGA in three ways:

1) Fast Memory Expansion

2) Chip Memory expansion to 1 meg with the Fatter Agnus

3) A big fast Hard Drive

All three will make your programming environment much more comfortable as well as decrease your development time.

ANIM Formats

Before we explore the details of the ANIM routines, a little lecture about animation file formats will help you to understand the different ANIM functions. There are many ways to achieve a particular effect. Some are more memory efficient but slower, others are fast but conserve strained memory resources. Some animation effects might best be done with just the JForth PICTURE routines.

There are two formats for storing the graphics information of an animation: an ANIMATION and an ANIMBRUSH. An ANIMATION is the more complicated of the two. Its advantages are that it works well with a double buffered display and it can be decompressed quickly. An ANIMBRUSH has the advantage of ease of changing directions in playback at the cost of slightly slower decompression times. Understanding the structure of the two formats can help you take advantage of their strong points in your code.

Both formats contain an initial ILBM graphic of the first frame of the animation. The rest of the animation is stored in DELTA chunks which are data chunks that just specify the changes that must be made to a previous frame to generate a new frame. The DELTA chunks for the two formats are interpreted differently for the two types of files.

An ANIMATION DELTA chunk contains the absolute value of new bytes in any changed area of the graphic. The DELTA chunks are setup so that each chunk modifies, not the current, but the previous frame of the animation. When double buffering is used the hidden previous frame is modified by a DELTA chunk to produce the next frame. Two extra DELTA chunks are added to the end of the animation sequence that allow regeneration of the first two frames of the animation so that looping effects can be performed.

An ANIMBRUSH DELTA chunk contains the exclusive OR of the old and new bytes in any changed areas of the display. The DELTA chunks act on the current frame to produce the next frame so that only one complete bitmap needs to be maintained. The last DELTA chunk modifies the last frame to produce the first frame anew for looping. Since the data is an exclusive OR of two consecutive frames, if the same delta is applied to the resulting frame, you get back the original. This makes it easy to generate the frames in forward, backward, or ping pong order.

Compiling the ANIM Toolbox

The ANIM and PICTURE toolboxes have so many routines that they will not fit in the normal JForth dictionary. Luckily it is a simple matter to expand the dictionary space. You can store this expanded Forth wherever you have room for it. Let's assume you want to place it in a directory called TMP:. You will need to substitute the name of your own directory.

Enter in the SHELL:

```
RUN COM:JFORTH
```

Enter in JFORTH:

```
200 #K !    \ allocate a 200K dictionary
SAVE-FORTH TMP:AnimForth
BYE
```

You now have a larger dictionary. Enter in the SHELL:

```
RUN TMP:AnimForth
```

Enter in JFORTH:

```
INCLUDE JANIM:LOAD_ANIM
```

After that compiles, let's save it so we don't have to recompile each time.

```
SAVE-FORTH TMP:AnimForth
```

From now on, you can just run that image and have everything ready for immediate use.

Tutorial 1 - Displaying an ANIM File

(Before proceeding with these tutorials, you should be familiar with the PICTURE IFF system described in chapter 21.)

Before performing any JForth graphics operations, we must initialize the graphics system and open the GRAPHICS and INTUITION library. To do this, enter:

```
GR.INIT
```

This tutorial assumes you have Deluxe Paint III as a source of an animation. If not use any ANIM-5 animation file that's available to you. Enter:

```
CD DPAINT:ANIM
DIR
```

You should see a file called CRY. If the above doesn't work, don't worry. You can use any ANIM for this tutorial, assuming that it fits in your memory.

Animations use a special structure to keep track of all the *cels* in the ANIM as well as other graphical

information. This structure is an extension to the PICTURE structure. Let's declare a structure for our ANIM. Enter:

```
ANIMATION CRYANIM
```

Now we can load the animation from disk. This will use the IFF parser to scan the IFF file. The deltas, which describe the differences between successive cels, will be loaded into memory. If the load fails, it will return an error flag on the stack. We print that to make sure the load worked. In an animation application, you should check this flag and respond appropriately. Enter:

```
CRYANIM ANIM.LOAD? CRY  .
```

Since this is the first ANIM loaded, it will open a screen of the proper size and display the first frame. To get back to the WorkBench screen, hold down the <LEFT-AMIGA> key and hit the 'N' key. To get back to the image, hit <LEFT-AMIGA>+M.

Now back in JForth, enter:

```
ANIM_LOOP CRYANIM ANIM.PLAY
```

ANIM.PLAY will start displaying the animation in a loop until you click in the top left corner on the hidden closebox or hit a key on the keyboard.

This Animation played as fast as the ANIM.PLAY routine could generate the next frame. This may be too fast for your particular purpose. The defered word ANIM.DELAY is called within the playback loop after each new frame is displayed. A simple way to slow down the playback would use the Vertical Blank delay:

```
: WAIT20  ( --- )
  20 WAIT.FRAMES
;
' WAIT20 is ANIM.DELAY
```

Note that the delay word must have a stack diagram that neither expects nor leaves anything on the stack.

To remove the animation from memory when finished. Enter:

```
CRYANIM ANIM.FREE
```

Tutorial 2 - ANIM Control and Disk Based ANIMS

Assume we have an animation called SPIN in the current directory. Let's do some explorations with it. First let's create an animation structure for it and load it into memory. Enter:

```
ANIMATION MYSPIN
MYSPIN ANIM.LOAD? SPIN   .
```

You should see the first frame appear. Now jump back to the JForth window using the keyboard combination <LEFT-AMIGA><N>. (Hold down the left amiga key, then press 'N'. You can get back to the Animation with <LEFT-AMIGA><M>.) Click in the JForth window and enter:

```
MYSPIN ANIM.STATS
```

You will see a list of information about the SPIN animation. The value displayed for CURRENT FRAME will be zero based for the first loop of the animation and 1 based on subsequent loops. Enter:

```
MYSPIN ANIM.DISPLAY.NEXT? .
```

You should see the next frame of the animation appear. If you have HISTORY ON you can step through the animation by pressing UP-ARROW RETURN combinations. Note that you will keep looping through the frames of the animation after you have passed the "last" fame. If garbage appears instead then the animation you are viewing does not have a two frame loop ending in the animation file. Flip back to JForth and do ANIM.STATS at different points in the animation. You should note that the sequence of frame numbers change between the first loop and the second. (Note that you can also use ANIM.VIEW.NEXT? which is faster but can have some surprising behavior. See the

references to PIC.VIEW for more explanation.)

The ANIMATION structure maintains two pictures. One is displayed and the other is hidden. You can see them by:

```
MYSPIN ..@ an_hiding     PIC.DISPLAY
MYSPIN ..@ an_displaying PIC.DISPLAY ( restores the original)
```

Let's play the SPIN animation from disk. First free the memory version:

```
MYSPIN ANIM.FREE
```

This also closes the screen. Enter:

```
MYSPIN ANIM.DISK.LOAD? SPIN  .  ANIM_ONETIME MYSPIN ANIM.PLAY
MYSPIN ANIM.FREE
```

You will see the animation play through once and stop on the last frame. Depending on the speed of your disk drive and the complexity of the animation results may range from good to poor (from floppy drive). Loading and playing from disk might allow very long animations to be shown or at least previewed.

Tutorial 3 - ANIMBRUSHES

AnimBrushes are like brushes in a paint program, except they have multiple frames or cels, like an Animation. An example might be a bird flapping its wings. For this next tutorial you will need a still picture and an AnimBrush. You could use the ones that came with JForth or you could make your own. Let's assume you now have a background picture that we will call BACKG and an AnimBrush brush that we will call BIRD. (It doesn't have to be a bird but we have to call it something!) It is best if they are made with the same pallette, otherwise the AnimBrush colors may look odd.

Although AnimBrushes could be loaded into their own screen, they are more useful when drawn on top of another image. Let's, therefore, load the background picture first so that it defines the screen. The filenames used are for the images on the JTools disk. You may substitute your own files. Enter:

```
PICTURE BACKG      \ declare a picture structure
" jpics:mountains.pic" BACKG $PIC.LOAD? . \ load image
```

Now let's define the AnimBrush stucture and load an AnimBrush file:

```
ANIMBRUSH BIRD
" jpics:bird.anbr" BIRD $ABR.LOAD?.
```

Since a picture was being displayed at the time of the load you saw no change in the display. If nothing was being displayed then a screen would open and display the first frame of the animbrush. You may want to drag the WorkBench screen down a bit so that you can see the graphics and the JForth window at the same time.

An AnimBrush can be blitted onto the screen just like brushes, enter:

```
10 20 BIRD ABR.BLIT
```

The special thing about AnimBrushes is that you can advance to the next frame and blit that. Enter:

```
BIRD ABR.ADVANCE
10 20 BIRD ABR.BLIT
```

If you don't want the full rectangle of the brush, you can blit transparently. Let's move to new x,y coordinates so we can see the difference. Enter:

```
BIRD ABR.ADVANCE
95 14 BIRD ABR.TRANS.BLIT    \ note .TRANS.
```

Use the <UP-ARROW> key to reenter the two previous commands several times. Notice that the new blits overlap the previous blits. Techniques to prevent this by saving the background and restoring are described in the PICTURE tutorial. The same techniques can be used with AnimBrushes. Actually,

almost any PIC. call can be used with AnimBrushes including wipes and other effects. Don't however, use PIC.FREE or PIC.LOAD with AnimBrushes!

Now let's define some words that will help us further explore AnimBrushes. Enter:

```
: SHOW.BIRD ( -- , display current frame of bird )
   10 20 BIRD ABR.BLIT
;
: NEXT.BIRD  ( -- , show next cel of Bird )
   BIRD ABR.ADVANCE
   SHOW.BIRD
   BIRD ABR.GET.FRAME . CR? \ print where we are now
;
```

ABR.GET.FRAME returns the frame, or cel, of the AnimBrush currently ready to blit. We can force an AnimBrush to a particular frame using ABR.GOTO.FRAME. Enter:

```
0 BIRD ABR.GOTO.FRAME  \ move to first frame
SHOW.BIRD   \ show it
NEXT.BIRD   \ show next frame
3 BIRD ABR.GOTO.FRAME
SHOW.BIRD
```

If you tell ABR.GOTO.FRAME to a frame beyond the end of your brush, it will just print a message to that effect and do nothing. You can tell how many cels you have by reading it out of the AnimBrush structure. Enter:

```
BIRD S@ ABR_CELS .  \ print total number of cels
```

Let's now define a word that will continuously advance the AnimBrush. Enter:

```
: PLAY.BIRD ( -- )
   BEGIN
      NEXT.BIRD
      6 WAIT.FRAMES \ wait 6 video frames, 6/60 seconds
      ?TERMINAL
   UNTIL
;
PLAY.BIRD
```

Notice that the bird is advancing and that the numbers go up to the highest frame then start over again at zero. Let's reverse the direction of play. Hit <RETURN> to stop PLAY.BIRD and enter:

```
BIRD ABR.REVERSE
PLAY.BIRD
```

If you would like the brush to PINGPONG, or go both directions, enter:

```
ABR_PINGPONG BIRD S! ABR_FLAGS
PLAY.BIRD
```

Notice the numbers go up to the maximum then back down to zero, etc. To get back to the normal mode, enter:

```
ABR_LOOP BIRD S! ABR_FLAGS
PLAY.BIRD
```

When you're done using these images don't forget to enter:

```
BIRD ABR.FREE
BACKG PIC.FREE
```

To further explore this toolbox, look in the directory, JANIM:TESTS. It contains test programs that

can also serve as simple examples. By studying these files, you will see how to create animbrushes from two pictures using ABR.BUILD?. You can then append other pictures to the end using ABR.APPEND.CEL?. You can also edit the internal structure of animbrushes using ABR.DUP.CEL? , ABR.DELETE.CEL? and ABR.REPLACE.CEL?.

Animation Tips

If you need to edit an animation, you can convert it to an animbrush using ANIM>ANIMBRUSH?, edit it, then convert it back using ANIMBRUSH>ANIM?.

In an application, you can bring the animation to the front using ScreenToFront(). The screen address is stored in the variable SIFF-SCREEN so enter:

```
SIFF-SCREEN @ ScreenToFront()
```

ANIM Support Glossary

ANIM IFF tools

These words are used in reading ANIM-5 IFF files.

$ANIM.LOAD? ($filename animation --- error?)

Load animation for playback. The main load routine. IF an_flags set to ANIM_DISKMODE then a disk mode load will occur.

$ANIM.DISK.LOAD? ($filename animation --- error?)

The ANIM_DISKMODE flag is set and $ANIM.LOAD is called. The animation will be kept on disk and read as it is played. This is handy for very large ANIMs.

$ANIM.PREP? ($filename --- error?)

Setup of variables and animation filename previous to an $ANIM.SCAN? .

$ANIM.SAVE? ($filename animation -- error?)

Saves the animation to the file. You cannot save a DISKMODE animation with this word.

$ANIM.SCAN? ($filename --- error?)

Scans the anim file for delta and anim-header chunks.

ANIM.BLIT (x y animation --)

Animations contain two pictures that are needed to reconstruct the images using double buffers. This word will blit the currently picture to the destination RastPort.

ANIM.DISK.HANDLER (size chkid --)

Reads DLTA chunks and collects the FSEEK and size data in lists to be able to access the chunk from disk.

ANIM.DISK.LOAD? (animation <filename> -- error?)

For disk based loading with filename in the input stream. This cannot be used in a colon definition. Use $ANIM.DISK.LOAD? instead.

ANIM.HANDLER (size chkid -- , handles ANIM specific chunks)

Reads DLTA chunks and allocates memory for them.

ANIM.LOAD? (animation <filename> -- error?)

For memory mode loading with filename in the input stream. This cannot be used in a colon

definition. Use $ANIM.LOAD? instead.

ANIM.PARSER (size chkid -- , recursively parse ANIM)

Used by $ANIM.SCAN to collect info about ANHD and DLTA chunks present in the file. Precollecting this data allows more efficient memory allocation for the ANIM file.

ANIM.READ.ANHD? (size --- error?)

Reads an Anim-Header chunk into ANIM-HEADER from current IFF file. Checks for correct size.

ANIM.SAVE? (animation <filename> -- error?)

This cannot be used in a colon definition. Use $ANIM.SAVE? instead.

Saves the animation to the filename from the input stream.

ANIMATION Words

These are used to display ANIMATIONs.

ANIM.ADVANCE? (animation -- error? , advance to next frame)

Calls ANIM.APPLYDELTA or ANIM.APPLYDISKDELTA as needed to generate the next frame. The HIDDEN and DISPLAYING buffer pointers are then switched. Will loop around at end.

ANIM.APPLYDELTA (anim --)

Using the hidden pic buffer applies the current memory based delta. Then bumps the delta pointer looping it to 1 if at the last delta. Does not swap the hidden and displayed buffer pointers or cause the new data to be displayed.

ANIM.APPLYDISKDELTA? (anim -- error?)

Same as ANIM.APPLYDELTA but obtains the data from disk.

WARNING: to use this word you need to previously open the disk based file with ANIM.DISK.OPEN. At the present time only ONE disk based anim can be open. Since the IFF routines are used to read the data you also must not do an IFF read or write operation while the file is open.

ANIM.CHECK (animation -- , abort if bad)

Looks for the correct value in the an_key field.

ANIM.DELAY (---)

A deferred word called by ANIM.PLAY between frames. It can be used to slow down the playback or add syncronization effects.

ANIM.DISK.OPEN? (animation -- error?)

Opens the original file read by the ANIM.DISK.LOAD routine for disk based animations. If you gave a directory relative filename at the original load, you must be in the same current directory to reopen it.

ANIM.DISK.CLOSE (---)

Closes the file.

ANIM.DISPLAY.NEXT? (animation -- error?)

Calls ANIM.ADVANCE? to generate the next frame then displays it using PIC.DISPLAY. This is slower then ANIM.VIEW.NEXT? but preserves the Intuition Screen order.

ANIM.FREE (animation -- , free all parts of animation)

Releases all the allocated memory for the animation and clears the animation struct.

ANIM.GET.DEPTH (animation -- depth)

Returns the number of bit planes in the animation.

ANIM.LAST.FRAME? (animation --- flag)

Returns a TRUE flag if at the last frame of an animation. This is valid ONLY IF the animation contains the typical two extra DELTA chunks for looping. Otherwise returns TRUE on the 3rd from last frame. Problem if the anim has less than 3 frames.

ANIM.PLAY (loopflag animation ---)

Loopflags are ANIM_LOOP (TRUE) which plays the animation in a loop, or ANIM_ONETIME (FALSE) which plays up to the "LAST" frame.

ANIM.STATS (animation ---)

Displays info about the current state of the animation.

ANIM.VIEW.NEXT? (animation -- error?)

Calls ANIM.ADVANCE? to generate the next frame then displays it using PIC.VIEW. This is much faster then ANIM.DISPLAY.NEXT? but can cause some confusion because it overlays the Intuition Screen display using low level code. See PIC.VIEW.

ANIMATION (<animname> ---)

Creates a structure in the dictionary for an animation.

ANIMBRUSH Words

Load and display AnimBrushes.

$ABR.LOAD? ($filename animbrush --- error?)

Since the first IFF load routine used opens a screen, you should load your backgroud picture first so that the display mode can be set. After loading the default direction is set to ABR_FORWARD and the mode flag is set to ABR_LOOP. See ABR.REVERSE.

$ABR.SAVE? ($filename animation -- error?)

Saves the animbrush to the file.

ABR.ADVANCE (animbr --)

Applies current DELTA to the animbrush then updates the delta pointer depending on the current state of the direction and mode flags.

ABR.BLIT (x y animbr ---)

Blits the animbrush into the current rastport then calls ABR.ADVANCE to generate the next frame. A simple PIC.BLIT will do as well but won't advance the frame. The PIC.BLIT call will work since the start of the animbrush structure is a PICTURE struct.

ABR.CHECK (animbr -- , abort if bad , used internally)

ABR.FREE (animbr -- , free all parts of animbrush)

ABR.GET.FRAME (animbr --- current-frame)

Returns the current frame number (0 1 2 .. N) of the bitmap state of the animbrush. Uses the current direction to deduce the current frame.

ABR.GOTO.FRAME (frame animbr ---)

Cycles in the current direction until the asked for frame is generated.

ABR.LAST.FRAME? (animation --- flag)

Returns a TRUE flag if at the last frame of an animbrush.

ABR.LOAD? (animbrush <filename> -- error?)

Reads filename from input then calls $ABR.LOAD. This cannot be used in a colon definition. Use $ABR.LOAD instead.

ABR.REVERSE (animbr ---)

Reverses the direction of the animbrush. That is the order in which the frames are generated by ABR.ADVANCE is reversed. ABR.REVERSE does NOT change the current bitmap. Since we allow forward and backward modes we must deduce the correct DELTA to use if we want to change directions.

ABR.SAVE? (animation <filename> -- error?)

Saves the animbrush to the file name from the input stream.

ABR.STATS (animbrush ---)

Displays info about the current state of the animbrush.

ABR.TRANS.BLIT (x y animbr ---)

The PIC.TRANS.BLIT word only casts the bitmap into its shadow on the first call. Since our bitmap is changing from frame to frame ABR.TRANS.BLIT recasts the shadow at each frame.

ANIMBRUSH (<animbrname> ---)

Creates a structure in the dictionary for an animbrush.

CONVERSION Words

Support for various conversion and animation editing code for EOR delta encoding used in Animbrushes. A true error? flag can be returned if there is insufficient memory for the operation. For examples of their use, see “JAnim:tests/test_conversion.f”.

ABR.APPEND.CEL? (picture animbr -- error?)

Appends a picture to the end of an AnimBrush. Must be same size as the animbrush.

ABR.BUILD? (pic0 pic1 animbr -- error?)

Builds a two frame animbrush from the two pictures. The width and height of the pictures should be set to the size of the entire bitmap of the pictures. You can use PIC.WHOLE to ensure this. The two pictures must be equal in size. The color map of pic0 is used in the generated animbrush.

ABR.CUR.DELTA (abr -- delta-address)

Returns the address of the current (next) delta

ABR.DELETE.CEL? (cel# animbr -- error?)

Deletes given cel from animbrush. An error can occur if memory cannot be allocated for the new deltas.

ABR.DUP.CEL? (cel# animbr -- error? , inserts a duplicate cel)

Adds a new cel to animbrush that is a duplicate of the given cel.

ABR.MAKE.DELTA (pic0 pic1 -- delta | 0)

Using the two given pictures makes an EOR type delta which is used in AnimBrushes. If the Delta cannot be allocated, this will return zero. You must free the delta when you are done using FREEBLOCK unless you add it to an AnimBrush. In that case it will get freed when you free the AnimBrush The pictures must be the same size.

ABR.NORM (animbr --)

Changes direction to FORWARD and mode to LOOPING, saving previous state.

ABR.REPLACE.CEL? (picture cel# animbr -- error?)

Modifies the cel to be the given picture. Picture must be the same size as the animbrush. Calculates the needed changes in the deltalist.

ANIM.MAKE.DELTA (pic0 pic1 -- delta | 0)

Using the two given pictures makes an ABS type delta which is used in Animations. If the Delta cannot be allocated, this will return zero. You must free the delta when you are done using FREEBLOCK unless you add it to an Animation. In that case it will get freed when you free the Animation. The pictures must be the same size.

ANIM>ANIMBRUSH.PARTIAL? (first_cel# last_cel# anim animbr -- error?)

Convert part of an Animation to an AnimBrush.

ANIM>ANIMBRUSH? (anim animbr -- error?)

Constructs an AnimBrush from all of an initalized Animation. This calls ANIM>ANIMBRUSH.PARTIAL? with a first cel# of 1, and a last cel# of 1. This will go through all cels and wrap around to the beginning.

ANIMBRUSH>ANIM.PARTIAL? (first_cel# last_cel# animbr anim -- error?)

Convert part of an AnimBrush to an Animation.

ANIMBRUSH>ANIM? (anim animbr -- error?)

Constructs an Anim from all of an initalized AnimBrush. This calls ANIMBRUSH>ANIM.PARTIAL? with a first cel# of 1, and a last cel# of 0. This will go through all cels and wrap around to the beginning.

ENCODECOL (blk ht --)

Encodes a column in VSkip format in Delta-work @.

EORplane (ABSpl0 ABSpl1 ABSdeltabuff w h --)

Calculates and rotates an EOR type deltaplane.

MAKE.ABSDELTA (pic0 pic1 --)

Creates an absolute style delta for use in ANIMs. Delta stored at DELTA-WORK. Used in ANIMBRUSH>ANIM.

LOW LEVEL support words

These words are used internally by the Animation system. You would not normally call these directly. They are documented here in case you need to make internal modifications or to extend the system.

ADD.YTABLE.USER (ytable --)

Increment the YTable user counter. See FREE.YTABLE

ALLOC.YTABLE (byteoffset linesize -- ytable | 0)

Creates or reuses a YTable. Fails if no memory available or no room is left in list. The constant MAX_Ytabs determines the memory reserved for the list.

ALLOC.YTABLE.TRACKER (-- tracker | 0)

Called automatically by the first request for a YTABLE. Won't hurt anything if done again. A YTABLE is a multiplication table used by the ANIM and ABR routines to optimize the decoding of deltas.

CountSames (buff length -- SameCount)

Low-level word to scan a delta plane for same runs.

CREATE.YTABLE (byteoffset linesize --- ytable | 0)

Allocates and calculates a multiplication table for the decode routines.

DECODE_VKPLANE (inDELTA outBITPlane ytable ---)

ASM code to decode a bitplane of info from a DELTA chunk. This routine expects the DELTA to consist of the absolute byte values in vertical byte skip form. Used by ANIMATION files.

DECODE_XORVKPLANE (inDELTA outBITPlane ytable ---)

ASM code to decode a bitplane of info from a DELTA chunk. This routine expects the DELTA to consist of EOR byte values in vertical byte skip form. Used by ANIMBRUSH files.

FINDOP (buffptr len --- cnt op)

Main low-level word for VSkip encoding of a rotated EOR encoded buffer.

PUSHBYTE (byte memblk --)

Pushes byte to allocated memoryblock

FIND.YTABLE (byteoffset linesize --- ytable | 0)

Checks if an appropriate YTable is available for shared use. If so returns its address, if not returns 0.

FREE.YTABLE (ytable ---)

Frees memory for a YTable if not marked by another user. Last free deallocates the ytable list as well.

FREELIST? (listaddr --)

Listaddr is the relative address of an allocated memory block obtained by a ALLOCBLOCK call. This block contains addresses of a series of memblock addresss. FREELIST? parses the list and does a FREEBLOCK on each memblock in the list, then does a FREEBLOCK for the list.

SCANFORZU (dataaddr length -- Z U)

Low-level word to scan a delta plane for zero and unique runs.