Using Screens

If you are familiar with the screen concepts set out in the last Chapter, this is where you make AOZ screens come alive. This Chapter explains how to manipulate your screens, and we have provided ready-made demonstrations of the techniques on disc, complete with guide notes in their listings.

Copying screens

Any rectangular part of a screen can be copied and moved on the current screen or to any other screen, time and time again, at great speed. Copying between the "physical" and "logical" screens is fully discussed in Chapter XX.X (7.2), along with detailed explanations of double buffering.

SCREEN COPY

instruction: Copy an area of a screen

```
Screen Copy source number To destination number
Screen Copy source number,x1,y1,x2,y2 To destination number,x3,y3
Screen Copy source number,x1 ,y1 ,x2,y2 To destination number,x3,y3,mode
```

SCREEN COPY is the most important screen command of all. It can be used to achieve classic screen techniques like "wiping" from one screen to another, as well as providing all sorts of special effects. At its simplest level, use this command to copy the whole contents of one screen to another screen.

Simply give the number of the source screen that holds the image to be copied, which can be a logical or physical screen. Then determine the number of the destination screen, which is where you want the image copied to. For example:

```
Screen Copy 1 To 2
```

Exact sections of screens can be copied by giving the coordinates of the top left-hand and bottom right-hand corners of the areas to be copied, followed by the number of the destination screen and the coordinates where the copy's top left-hand corner should be placed. If the destination screen number is omitted, the copied image will appear at the new coordinates on the current screen. For example:

```
Circle 50,50,10 : Wait 50
Screen Copy 0,20,20,70,70 To 0,100,100
```

Note that there are no limits to these coordinates, and any parts of the image that fall outside of the current visible screen area will be clipped automatically. There is also an optional parameter which selects one of 255 possible blitter modes for the copying operation. These modes affect how the source and destination areas are combined together on the screen, and are set using a bitpattern in the following format:

Mode Bit - Source Bit - Destination Bit

- 4 - 0 - 0

- 5 - 0 - 1
- 6 - 1 - 0
- 7 - 1 - 1

Please note that the bottom four bits in the pattern are not used by this instruction, and should always be set to zero.

Also, that SCREEN COPY combines the source and destination graphics using blitter areas B and C, but not area A.

To short-circuit the mass of all 255 options, here is a list of five of the most common modes, along with their binary bit-map patterns, followed by a ready-made working example to examine:

Mode - Bit-pattern - Effect

- REPLACE - %11000000 : Replace destination graphics with a copy of the source image. (Default mode.)
- INVERT - %00110000 : Replace destination graphics with an inverse video image of the source.
- AND - %10000000 : Combine together the source and destination images, with a logical AND operation.
- OR - %11100000 : Overlap the source image with the destination graphics.
- XOR - %011000000 : Combine together an inverse source image and destination graphics, with an Exclusive OR.

Examine that demonstration program, and use the mouse pointer to copy the image anywhere on screen, with a single click of the left mouse button. Keep the button held down and move the mouse pointer to see the full potential of SCREEN COPY, then press any key to call up the next mask and repeat the process.

Scrolling the screen

DEF SCROLL

instruction: Define a scrolling screen zone

```
Def Scroll number,x1,y1 To x2,y2,horizontal value, vertical value
```

Using the AOZ system, you are able to define up to 16 different scrolling screen zones, and each one
can have an individual pattern of movement. Simply follow your DEF SCROLL command with the number from 1
to 16 of the zone you are setting up. Then give the coordinates of the area of the zone to be scrolled, from the top left-hand corner to the diagonally opposite bottom right-hand corner. Finally, give this zone a scrolling pattern by setting the number of pixels to be shifted horizontally, and the number of pixels to be shifted vertically during each scrolling operation. Positive horizontal values will cause a shift to the right whereas negative values will shift the zone towards the left of the screen.

Similarly, positive vertical values will scroll downwards and negative values cause an upward scroll.

SCROLL

instruction: Scroll a screen zone

```
Scroll zone number
```

To scroll a screen zone already specified with a DEF SCROLL setting, use SCROLL followed by the zone number you require.

Enlarging and reducing the screen

ZOOM

instruction: Change size of part of screen

```
Zoom source number, x1 ,y1 ,x2,y2 To destination number,x3,y3,x4,y4
```

This one command allows you to produce a range of remarkable effects that change the size of the image in any rectangular area of the screen. Depending on the relative sizes of the source and destination areas, images can be magnified, shrunk, squashed and stretched as you wish. ZOOM is qualified by the number of the screen from where your source picture will be taken, followed by the coordinates of the top left-hand corner and bottom right-hand corner of the area to be grabbed, After the TO structure, give the number of the destination screen and the new coordinates of the area which is to hold the zoomed image. AOZ will automatically re-size the image.

The LOGIC function may be used to grab an image from the appropriate logical screen, instead of specifying a physical screen number. In the same way, you are allowed to deposit a zoomed image to a logical screen. This is explained below.

Physical and logical screens

Note: this section is only valid in Amiga hardware emulation mode, there is no notion of double buffer in a browser with Javascript.

When you watch the moving images shown at the cinema or on video, you are watching an illusion. Graphical animation in the movies is created by a fast sequence of still pictures known as frames. Television screens do not display moving images either. They fool the brain and the eye by updating still images on the screen, fifty images every second.

In order to create really smooth moving graphics, your computer has to complete all new drawing operations in less than one fiftieth of a second. So the AOZ programmer must achieve this speed, otherwise programs will suffer from an ugly flicker. The problem is solved by using a technique that switches between screens during drawing operations.

This is how it works:

Think of the actual area where images are displayed as the "physical" screen. Now imagine that there is a second screen which is completely invisible to the eye, where new drawing operations are executed. Call that the "logical" screen. Flicker-free movement is achieved by switching between the physical and logical screen.

The physical screen is displayed as usual, then once the new image has been drawn on the logical screen, they are swapped over. The old physical screen becomes the current logical screen, and is used to receive the drawing operations that will make up the next image. This process is completely automatic when using the DOUBLE BUFFER command, which is fully explained in

SCREEN SWAP

instruction: Swap over logical and physical screens in Amiga hardware emulation mode only. Has no effect in "PC" mode.

```
Screen Swap
Screen Swap number
```

This is the command that swaps over the physical and logical screens, so that the displays are instantly switched between the two of. them. If the DOUBLE BUFFER command has been engaged, this process is automatic.

LOGBASE

function: Return the address of logical screen bit-plane

```
address=Logbase(plane)
```

The LOGBASE function allows expert programmers to access the Amiga's screen memory directly. The current
screen is made up of six possible bit-planes, and after LOGBASE has been called, the address of the required plane is returned, or zero is given if it does not exist.

PHYBASE

function: Return the address of the current screen

```
address=Phybase(plane)
```

In hardware emulation mode, PHYBASE returns the address in memory of the specified bit-plane number for the current screen. AOZ emulates the bitplanes as on the Amiga, you should use the same address calculations. If this plane does not exist, a value of zero is given. For example:

```
Loke Phybase(0),0 : Rem Poke a thin line directly onto screen
```

PHYSIC

function: Return identification number for physical screen. In hardware emulation mode, PHYBASE returns the address in memory of the specified bit-plane number for the current screen. AOZ emulates the bitplanes as on the Amiga, you should use the same address calculations...

```
number=Physic
number=Physic(screen number)
```

The PHYSIC function returns an identification number for the current physical screen. This number allows you to access the physical image being displayed by the automatic DOUBLE BUFFER system, and the result of this function can be substituted for the screen number in ZOOM, APPEAR and SCREEN COPY commands. The PHYSIC identification number of the current screen will be returned, unless an optional screen number is specified.

LOGIC

function: Return identification number for logical screen

```
number=Logic
number=Logic(screen number)
```

Use the LOGIC function to get an identification number for the current logical screen, or use an optional screen number to specify a particular logical screen. The identification number that is returned can now be used with the ZOOM, APPEAR and SCREEN COPY commands, to change images off screen, without affecting the current display.

Screen synchronisation

It has already been explained that the image on your screen is updated fifty times every second. A single update consists of an image drawn by an electron beam scanning across every line of the screen until it reaches the bottom right-hand corner, at which point the beam switches off and starts scanning again at the top left-hand corner of the screen. The period between the completion of one screen
and the beginning of the next update is known as the "vertical blank period", or VBL for short. This is the period when AOZ jumps in to perform important tasks like moving Bobs and swapping screens.
AOZ is so efficient, it considers a 50th of a second to be a huge waste of time, and is eager to get on with any other tasks that need doing. This means that your programs could get out of synchronisation with what is actually happening on screen, so there are situations when AOZ must be instructed to wait for the next vertical blank period, in order keep in step.

WAIT VBL

instruction: Wait for next vertical blank period

```
Wait vbl
```

This simple command can be included to achieve perfect synchronisation, and is especially useful after a SCREEN SWAP.

Screen compaction

Naturally you will want to exploit the most spectacular images in your programs, but the idea becomes less attractive because of the large amounts of program memory that get used when a graphical screen is used. With a single, 64 colour, full-size screen consuming 60k of RAM, the AOZ programmer needs to crunch the data that makes up screen graphics, pack it into an acceptable amount of memory and then unpack it when necessary.

SPACK

picture compactor extension: Pack a screen

```
Spack screen number To bank number
Spack screen number To bank number xl,y1,x2,y2
```

This command stands for "screen pack", and it supports all standard graphic modes, including HAM.
The original AMOS on the Amiga was using its own format of picture compression, AOZ creates PNG images.

```
Spack 7 To 20
```

If the selected memory bank does not already exist, AOZ will reserve it automatically before packing in the screen data, which includes everything about the image including its mode, size and any offsets or display settings. This means that when the data is unpacked, the image will be re-created in its original state. Your new memory bank will be stored in fast memory if available, and will be saved along with your Basic program. After SPACK has been called, you can determine the size of your crunched screen with the LENGTH function.

If you only want to pack a part of any screen and not bother about the remaining area, simply add the coordinates of the top left and bottom right-hand corners of the section to be packed. Note that all x-coordinates will be automatically rounded to the nearest 8 pixel boundary.

To provide you with the maximum memory saving, AOZ will try and SPACK your images using several alternative strategies. It will then choose the method that consumes the least amount of memory. You are requested to be patient during the five or six seconds that this process takes, and are assured that unpacking takes less than a second, so your programs will run smoothly. If a one second delay is not acceptable to you, please see the alternative system that uses GET CBLOCK and PUT CBLOCK

PACK

picture compactor extension: Pack screen data

```
Pack screen number To bank number
Pack screen number To bank number xl,y1,x2,y2
```

The PACK command is slightly different from SPACK, because it only compresses the image data into a PNG. This means that the image must always be unpacked into an existing screen. Also there will be a slight flicker when the image is unpacked, unless the screens have been double buffered, so it is better to use single buffered screens here. Screen numbers, memory bank numbers and optional coordinates for smaller sections of the screen to be packed are used in exactly the same way as with the SPACK command. X-coordinates are rounded to the nearest 8 pixel boundary too in Amiga hardware emulation.

SPACK is fully compatible with the standard AUTOBACK system explained in Chapter XX.X (7.2), so it is easy to combine compacted images with moving screens. Images can even be unpacked behind existing Bobs, so it is possible to exploit this command together with SCREEN OFFSET to create superb scrolling backgrounds.

UNPACK

picture compactor extension: Unpack a compacted screen

```
Unpack bank number
Unpack bank number,x,y
Unpack bank number To screen number
```

As you might expect, this is used to unpack crunched images. Using double buffered screens will give smooth results, otherwise unpacking may get messy, and always make sure that the destination screen is in exactly the same format as the packed picture or an error will be generated.

To unpack screen data at its original position, state which memory bank is to be unpacked, like this:

```
Unpack 15
```

To re-draw the packed image starting from new top left-hand corner coordinates, include them after the bank number. If the new image does not fit into the current screen, the appropriate error message will
appear.

The other form of the UNPACK command is to open a screen and unpack the data held in the selected bank to that screen. For example:

```
Unpack 15 To 1
```

If the screen you select already exists, its image will be replaced by the newly unpacked picture within one second.