Pyromaniac II

The Sequel

Gerph, November 2021

0. Introduction

Introduction

How I'll do this talk

- Some talk about how RISC OS does things.
- There are 9 sections, with questions spread within them.
- Slides will be available at the end, together with some other resources.
- At the conclusion I'll answer any questions people have for as long as people want.

Introduction

What I'll talk about

1. Some background.
2. VNC and Sprites (and questions).
3. Font Manager and Screen Modes.
4. Documentation (and questions).
5. Testing.
6. Miscellaneous bits.
7. System demo (and questions).
8. Conclusions.

1. Background

Background

Who am I?

- A RISC OS architect and engineer.
- My day job is working with test and build systems.
- In previous times did a lot of things with RISC OS, which you can read about on my site if you're interested - gerph.org/riscos
- I know RISC OS inside and out, and I work on it because it's fun.

Background

Recap

What did I show last year?

- RISC OS Build system, and how it works - build.riscos.online
  - A cloud system for building and testing RISC OS software.
  - Available to all, for free.
- RISC OS Pyromaniac, the operating system that powers it.
  - A reimplementation of RISC OS from scratch in Python.
  - Intended for debugging and testing.
- This RISC OS presentation system which runs on it!
- An online service which demonstrates RISC OS Pyromaniac - shell.riscos.online
- A lot of open source software and resources - pyromaniac.riscos.online

Background

What I said I was going to look at

Here's what I said I wanted to have a look at:

- More APIs.
- Better handling of corner cases.
- Sprites (sigh).
- Back Trace Structures.
- Finish the pending branches - Windows, Zipper, EasySockets, Git, DCI4, ...
- Using it for actual testing - that was what it was for!
- So many other opportunities.

2. VNC and sprites

VNC server

A more accessible system

It would be cool...

- To have the shell server be more accessible.
- To allow you to have a graphical view on the system.
- Maybe pipe the graphics operations directly to a browser canvas.

But that's a lot of work.
How about using VNC instead? - let's write a library.

VNC server

Writing a new library (1)

The library ...

- Needed to be reusable
- Needed to work with the common VNC clients.
- Needed to be simple enough to be used without much boilerplate.
- Needed to handle multiple concurrent connections.
- Needed to allow some connections to be read only.
- Needed to be able to take input from the user.
- Needed to be able to change the pointer.
- Needed to be able to handle clipboard.

VNC server

Writing a new library (2)

What do we support?

- Password controls whether you can control the session, or only view.
- Display format can have most RGB formats.
- Only ever supplies data as raw RGB - never compressed...
- ... but it only delivers changing rows.
- Display size changes can be communicated to the client.
- Mouse and keyboard input is supported.
- Multiple simultaneous connections supported.

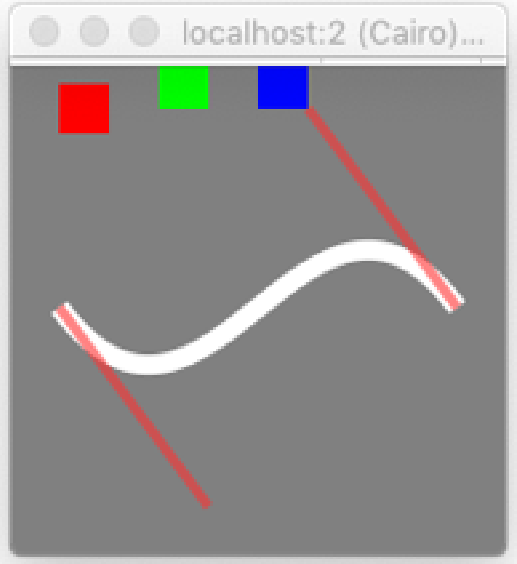
But ...

- It doesn't support changing the pointer.
- Or the clipboard.

VNC server

What does it look like?

```
# Create the animator
screen = Screen()
animate_thread = threading.Thread(target=screen.animate)
animate_thread.daemon = True
animate_thread.start()

# Create the server
server = cairovnc.CairoVNCServer(port=5902, surface=screen.surface)
server.serve_forever()
```



VNC server

Integrating with Pyromaniac

How easy was it to integrate with Pyromaniac?

- Not especially hard.
- Getting the mouse input right was surprisingly frustrating.
- Needed multi-threaded locking adding to the library.
- The entire implementation is 317 lines.
- 130 of them are the key mapping table.

Sprites

How do you draw sprites?

Let's look at what happens when you put a sprite on the screen in RISC OS.

- Work out what the sprite is (`OS_SpriteOp 18` or use the sprite name).
- Ask for a colour translation table (`ColourTrans_GenerateTable`).
- Request a plot of the sprite (`OS_SpriteOp` in one of its many forms).

Sprites

What do you need to draw sprites?

- Graphics primitives, like a graphics cursor and colour selection.
- Modes that are shallow (paletted), and deep (linear colour components).
- Mode information, like the depth and dimensions.
- Palette information for mapping colours in low modes.
- Colour translations, for finding the best colours.
- Sprite area management, like loading and finding the right sprites.
- Sprite area information, to know what sprites are.
- Sprite rendering, to get the sprite on to the screen.

Sprites

Drawing sprites in Pyromaniac (1)

Graphics primitives like being able to select colours to draw with, and then drawing lines and text, may not seem like they're necessary for sprite plotting, but they're needed for a few things...

- Colour selection is used for the rarely needed 'plot mask' operations.
- Graphics cursor positioning is needed for some 'plot at cursor' operations.
- Graphics windowing needs to bound any rendered windows.

Sprites

Drawing sprites in Pyromaniac (2a)

Two types of modes:

- Shallow modes are paletted and have 256 colours or fewer.
- Deep modes have linear colour components (15bpp and 24bpp modes).

How are they specified:

- The current mode, usually specified as -1 to interfaces.
- Numbered modes.
- Mode selectors, which give the basic screen parameters for a mode.
- Sprite mode numbers, which just contain the colour type and the density.
- Sprites, which can be treated like modes in some cases.

Sprites

Drawing sprites in Pyromaniac (2b)

Getting a `Mode` from a mode specifier:

```
def getmodedef(self, mode_or_address):
    """
    Convert from a mode specifier (number, selector, sprite mode word, etc) to Mode.
    """
    modesel = ModeSelector(self.ro, mode_or_address)
    return modesel.modedef
```

Sprites

Drawing sprites in Pyromaniac (3)

```
if mode in (-1, 0xFFFFFFFF):
    self.modedef = self.ro.kernel.vdu.getmodedef(-1)

elif mode >= 256 and (mode & 1) == 0:
    sprite_address = self.ro.kernel.api.os_spriteop_get_address(area=mode,
                                                                sprite_name=sprite_name)
    self.modedef = self.ro.kernel.vdu.getmodedef(sprite_address)

else:
    # Numbered mode, or a mode descriptor
    self.modedef = self.ro.kernel.vdu.getmodedef(mode)

self.colours = self.modedef.ncolour + 1
if self.colours == 64:
    self.colours = 256
```

Sprites

Drawing sprites in Pyromaniac (4a)

- To operate on a sprite we need to find it in a sprite area.
- Pyromaniac has an object for a `SpriteArea`, which can locate sprites by name.
- Within the area, we create objects for sprite itself - a `Sprite` object.
- This `Sprite` object knows how to extract information from it:
  - Width and height
  - Mode - resolution, depth and colour type.
  - Palette
  - Image data
  - Mask data

Sprites

Drawing sprites in Pyromaniac (4b)

```
@handlers.osspriteop.register(spriteop.SpriteReason_ReadSpriteSize)
def OS_SpriteOp_28(ro, reason, regs, area, sprite):
    regs[3] = sprite.width
    regs[4] = sprite.height
    regs[5] = 1 if sprite.mask_offset else 0
    regs[6] = sprite.mode

    if ro.kernel.sprites.debug_spriteop:
        print("Read sprite size {!r}, name {!r} => {}x{}, mask {}, mode {} (&{:08x})"
                .format(sprite, sprite.name,
                        sprite.width,
                        sprite.height,
                        bool(sprite.mask_offset),
                        sprite.mode,
                        sprite.mode))
```

Sprites

Drawing sprites in Pyromaniac (5)

- `ColourTrans_GenerateTable` turns a palette into a translation table for rendering sprites.
- Takes source and destinations, which can be any of the mode specifiers.
- Can change the colours as the table is generated with a transfer function.

A simple call in BASIC for translation from a sprite to the current mode might be:

```
SYS "ColourTrans_GenerateTable", 256, sprite%, -1, -1, 0, %01 TO ,,,,pixtrans_size%
DIM pixtrans% pixtrans_size%
SYS "ColourTrans_GenerateTable", 256, sprite%, -1, -1, pixtrans%, %01
```

Sprites

Drawing sprites in Pyromaniac (6a)

Plotting sprites: Source

Source palette

	Colour 0	&00000000
	Colour 1	&0000FF00
	Colour 2	&00FFFF00
	Colour 3	&FFFFFF00

Sprite

3	1	1	3
1	3	3	3
1	3	3	3
1	3	1	3
3	1	1	3
3	3	3	3

Sprites

Drawing sprites in Pyromaniac (6b)

Plotting sprites: ColourTrans_GenerateTable

Source palette

	Colour 0	&00000000
	Colour 1	&0000FF00
	Colour 2	&00FFFF00
	Colour 3	&FFFFFF00

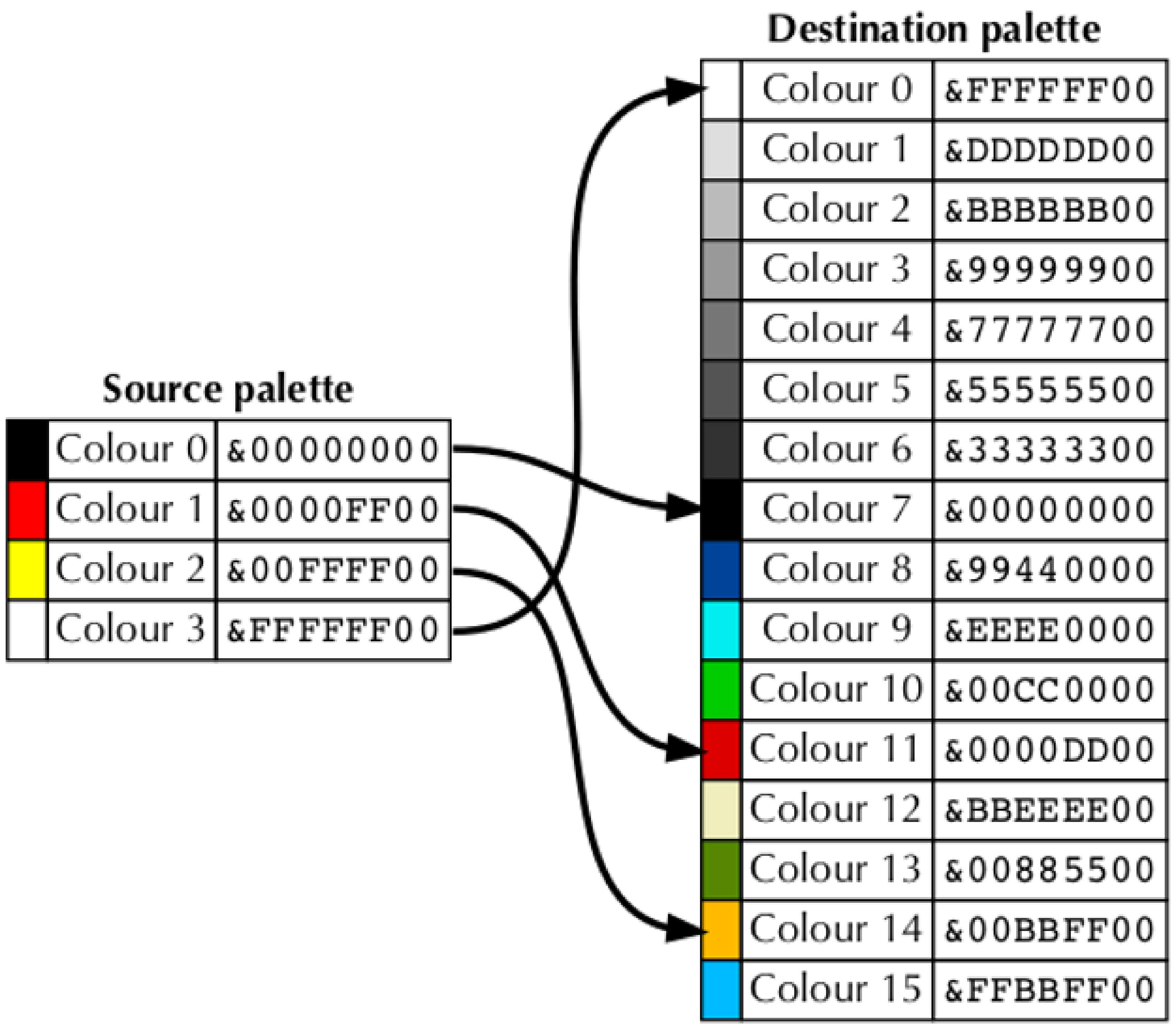
Destination palette

	Colour 0	&FFFFFF00
	Colour 1	&DDDDDD00
	Colour 2	&BBBBBB00
	Colour 3	&99999900
	Colour 4	&77777700
	Colour 5	&55555500
	Colour 6	&33333300
	Colour 7	&00000000
	Colour 8	&99440000
	Colour 9	&EEEE0000
	Colour 10	&00CC0000
	Colour 11	&0000DD00
	Colour 12	&BBEEEE00
	Colour 13	&00885500
	Colour 14	&00BBFF00
	Colour 15	&FFBBFF00

Sprites

Drawing sprites in Pyromaniac (6c)

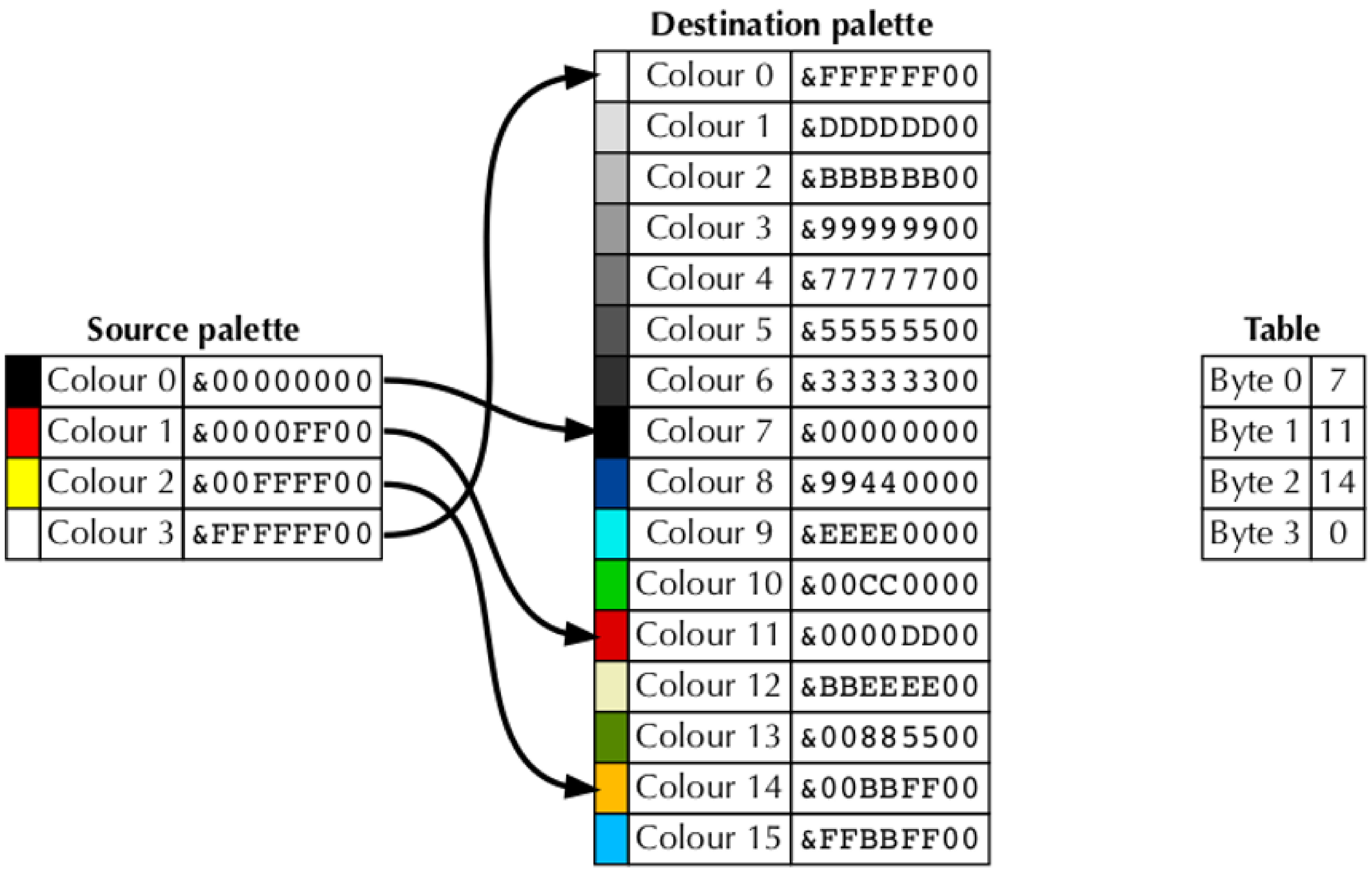
Plotting sprites: ColourTrans_GenerateTable



Sprites

Drawing sprites in Pyromaniac (6d)

Plotting sprites: ColourTrans_GenerateTable



Sprites

Drawing sprites in Pyromaniac (6e)

Plotting sprites: OS_SpriteOp

Sprite

3	1	1	3
1	3	3	3
1	3	3	3
1	3	1	3
3	1	1	3
3	3	3	3

Table

Byte 0	7
Byte 1	11
Byte 2	14
Byte 3	0

Sprites

Drawing sprites in Pyromaniac (6f)

Plotting sprites: OS_SpriteOp

Sprite

3	1	1	3
1	3	3	3
1	3	3	3
1	3	1	3
3	1	1	3
3	3	3	3

Byte data

11 01 01 11	= 215
11 11 11 01	= 253
11 11 11 01	= 253
11 01 11 01	= 221
11 01 01 11	= 215
11 11 11 11	= 255

Table

Byte 0	7
Byte 1	11
Byte 2	14
Byte 3	0

Sprites

Drawing sprites in Pyromaniac (6g)

Plotting sprites: OS_SpriteOp

Byte data

11 01 01 11	= 215
11 11 11 01	= 253
11 11 11 01	= 253
11 01 11 01	= 221
11 01 01 11	= 215
11 11 11 11	= 255

Table

Byte 0	7
Byte 1	11
Byte 2	14
Byte 3	0

Colour lookup

215	= (3, 1, 1, 3)
221	= (1, 3, 1, 3)
253	= (1, 3, 3, 3)
255	= (3, 3, 3, 3)

Sprites

Drawing sprites in Pyromaniac (6h)

Plotting sprites: OS_SpriteOp

Byte data

11 01 01 11	= 215
11 11 11 01	= 253
11 11 11 01	= 253
11 01 11 01	= 221
11 01 01 11	= 215
11 11 11 11	= 255

Table

Byte 0	7
Byte 1	11
Byte 2	14
Byte 3	0

Colour lookup

215	= (3, 1, 1, 3)
221	= (1, 3, 1, 3)
253	= (1, 3, 3, 3)
255	= (3, 3, 3, 3)

Translated lookup

215	= (0, 11, 11, 0)
221	= (11, 0, 11, 0)
253	= (11, 0, 0, 0)
255	= (0, 0, 0, 0)

Sprites

Drawing sprites in Pyromaniac (6i)

Plotting sprites: OS_SpriteOp

Byte data

11 01 01 11	= 215
11 11 11 01	= 253
11 11 11 01	= 253
11 01 11 01	= 221
11 01 01 11	= 215
11 11 11 11	= 255

Table

Byte 0	7
Byte 1	11
Byte 2	14
Byte 3	0

Destination palette

Colour 0	&FFFFFF00
Colour 1	&DDDDDD00
Colour 2	&BBBBBB00
Colour 3	&99999900
Colour 4	&77777700
Colour 5	&55555500
Colour 6	&33333300
Colour 7	&00000000
Colour 8	&99440000
Colour 9	&EEEE0000
Colour 10	&00CC0000
Colour 11	&0000DD00
Colour 12	&BBEEEE00
Colour 13	&00885500
Colour 14	&00BBFF00
Colour 15	&FFBBFF00

Colour lookup

215	= (3, 1, 1, 3)
221	= (1, 3, 1, 3)
253	= (1, 3, 3, 3)
255	= (3, 3, 3, 3)

→

Translated lookup

215	= (0, 11, 11, 0)
221	= (11, 0, 11, 0)
253	= (11, 0, 0, 0)
255	= (0, 0, 0, 0)

→

Paletted lookup

215	= (&FFFFFF00, &0000DD00, &0000DD00, &FFFFFF00)
221	= (&0000DD00, &FFFFFF00, &0000DD00, &FFFFFF00)
253	= (&0000DD00, &FFFFFF00, &FFFFFF00, &FFFFFF00)
255	= (&FFFFFF00, &FFFFFF00, &FFFFFF00, &FFFFFF00)

Sprites

Drawing sprites in Pyromaniac (6j)

Plotting sprites: OS_SpriteOp

Byte data

11 01 01 11	= 215
11 11 11 01	= 253
11 11 11 01	= 253
11 01 11 01	= 221
11 01 01 11	= 215
11 11 11 11	= 255

Paletted lookup

215	= (&FFFFFF00, &0000DD00, &0000DD00, &FFFFFF00)
221	= (&0000DD00, &FFFFFF00, &0000DD00, &FFFFFF00)
253	= (&0000DD00, &FFFFFF00, &FFFFFF00, &FFFFFF00)
255	= (&FFFFFF00, &FFFFFF00, &FFFFFF00, &FFFFFF00)

Output data

(&FFFFFF00, &0000DD00, &0000DD00, &FFFFFF00)
(&0000DD00, &FFFFFF00, &FFFFFF00, &FFFFFF00)
(&0000DD00, &FFFFFF00, &FFFFFF00, &FFFFFF00)
(&0000DD00, &FFFFFF00, &0000DD00, &FFFFFF00)
(&FFFFFF00, &0000DD00, &0000DD00, &FFFFFF00)
(&FFFFFF00, &FFFFFF00, &FFFFFF00, &FFFFFF00)

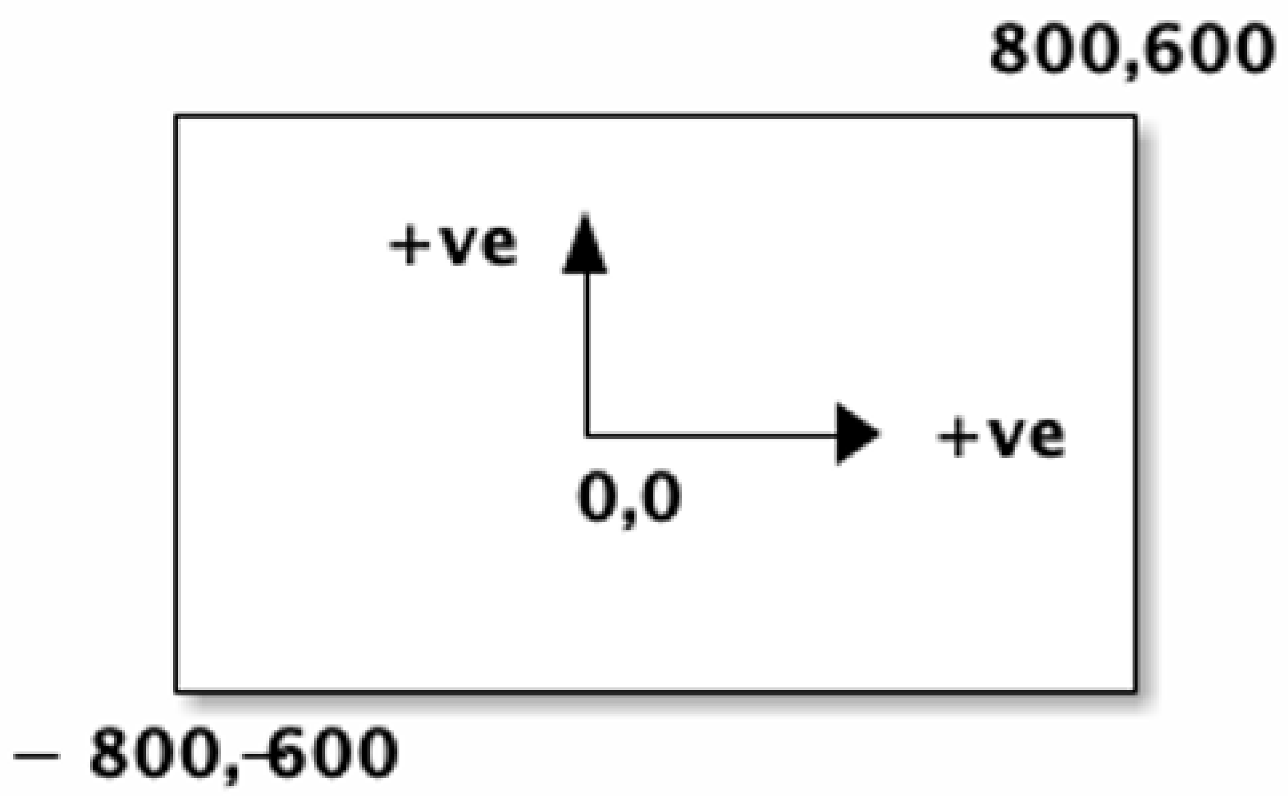
Sprites

Coordinate space (1)

- Coordinate spaces describe where you start drawing from and which direction is positive in each axis.
- RISC OS uses cartesian coordinates, just like the BBC.
- These are mapped to pixels on the screen.
- Pyromaniac has to then map them to the coordinates used by the Cairo graphics system.

Sprites

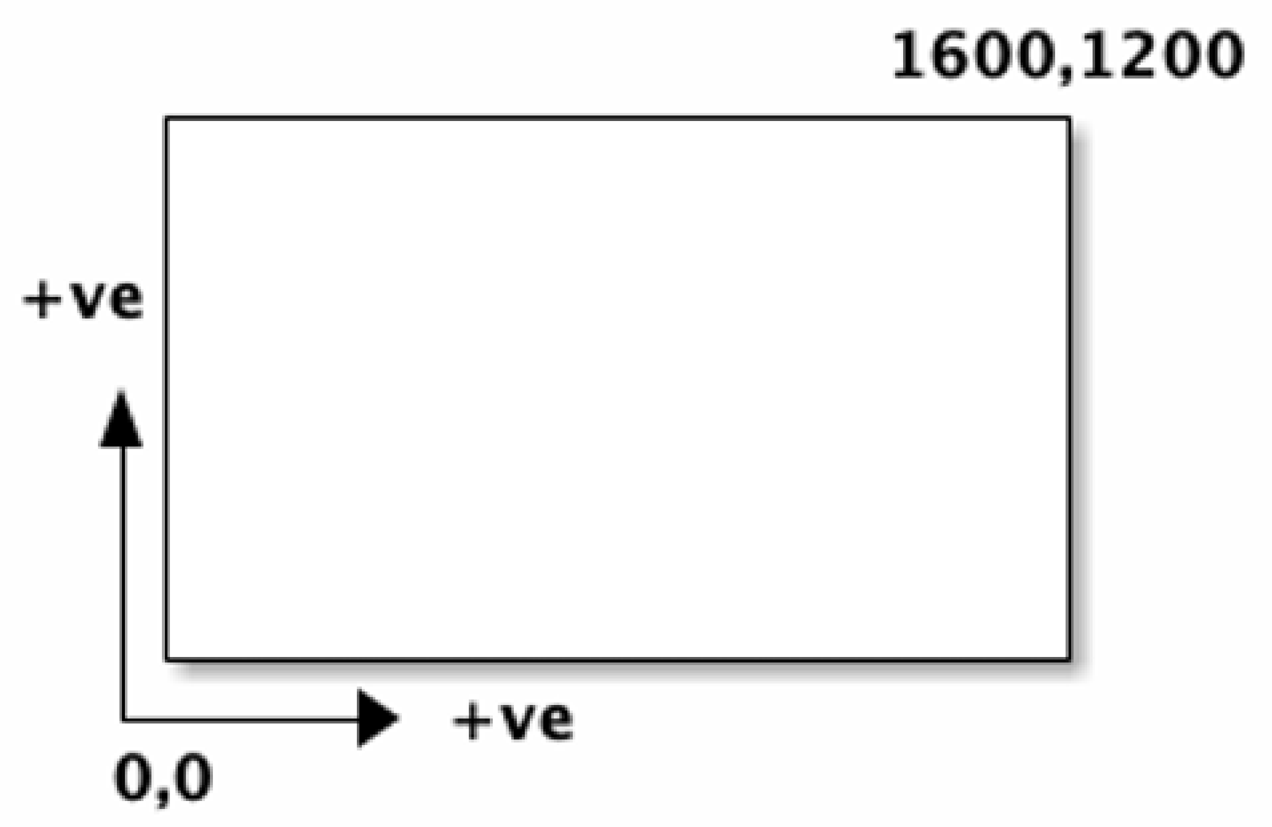
Coordinate space (2)



- Origin is specified by the user.
- X-coordinates increase to the right of the screen.
- Y-coordinates increase up the screen.
- User coordinate space has coordinates which are scaled by the eigenfactors, representing the shape and size of pixels.
- Bottom left coordinates are (-800, -600).
- Top right coordinates are (800, 600).

Sprites

Coordinate space (3)

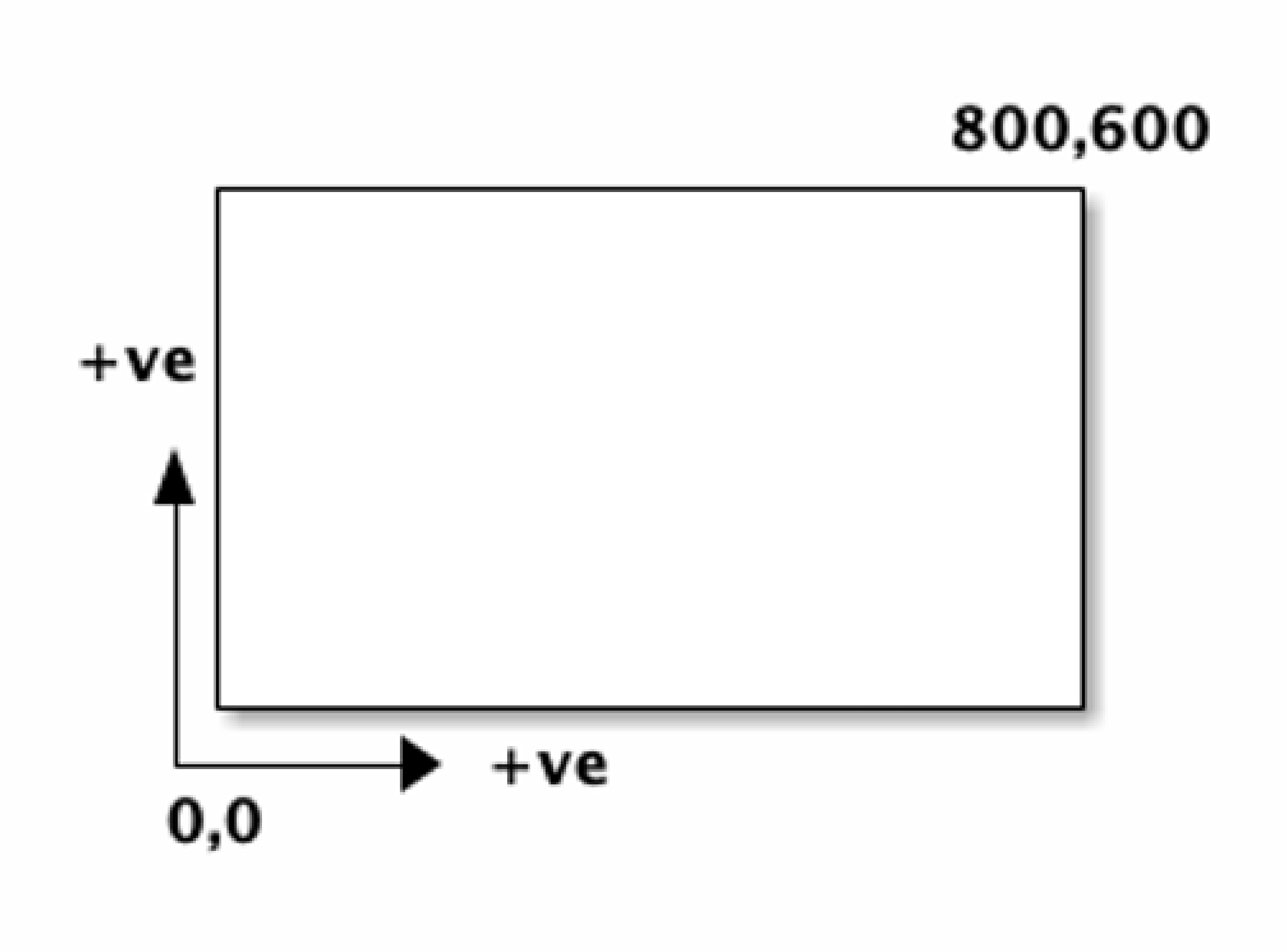


- Origin is specified **at the bottom left**.
- X-coordinates increase to the right of the screen.
- Y-coordinates increase up the screen.
- User coordinate space has coordinates which are scaled by the eigenfactors, representing the shape and size of pixels.
- Bottom left coordinates are **(0, 0)**.
- Top right coordinates are **(1600, 1200)**.

Sprites

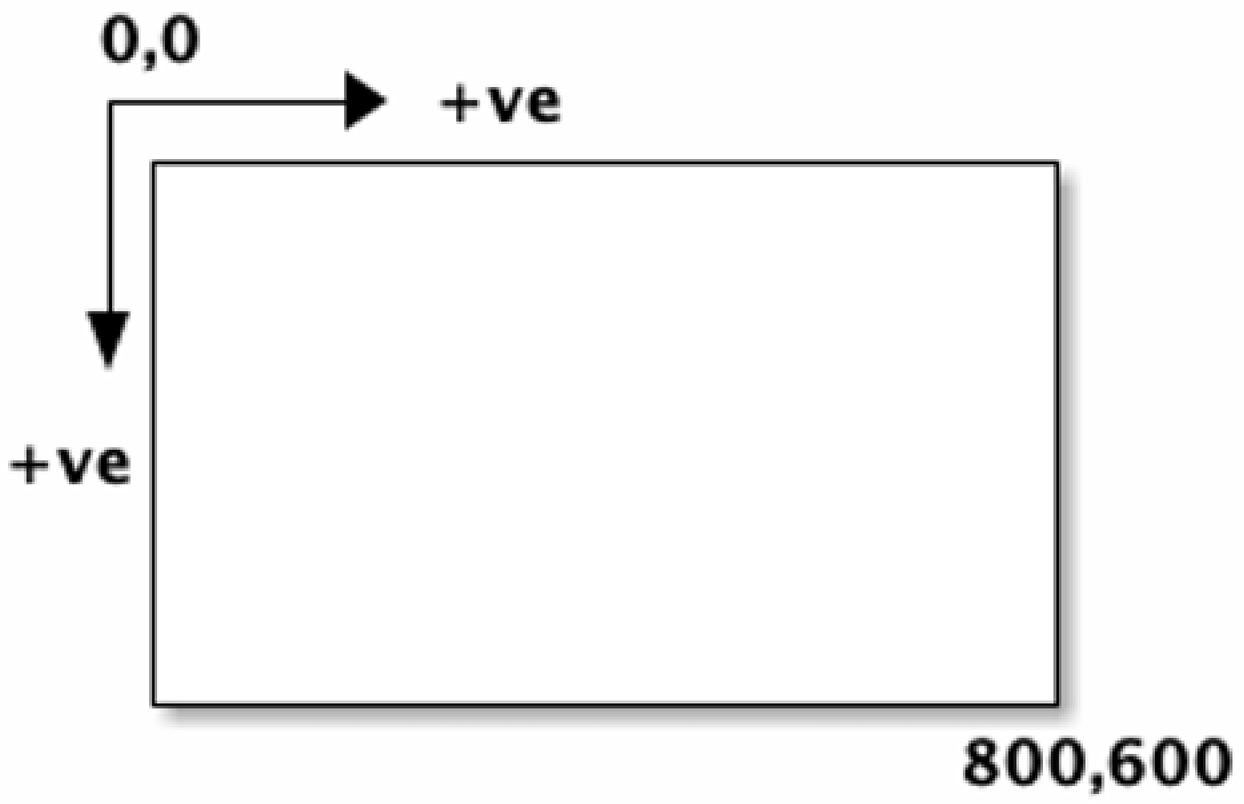
Coordinate space (4)

- Origin is specified at the bottom left.
- X-coordinates increase to the right of the screen.
- Y-coordinates increase up the screen.
- **Coordinates map directly to pixels.**
- Bottom left coordinates are (0, 0).
- Top right coordinates are **(800, 600)**.



Sprites

Coordinate space (5)



- Origin is specified at the **top** left.
- X-coordinates increase to the right of the screen.
- Y-coordinates increase **down** the screen.
- Coordinates map directly to pixels.
- Bottom left coordinates are **(0, 600)**.
- Top right coordinates are **(800, 0)**.

Sprites

Transformations (1)

To resize the sprites with `OS_SpriteOp`...

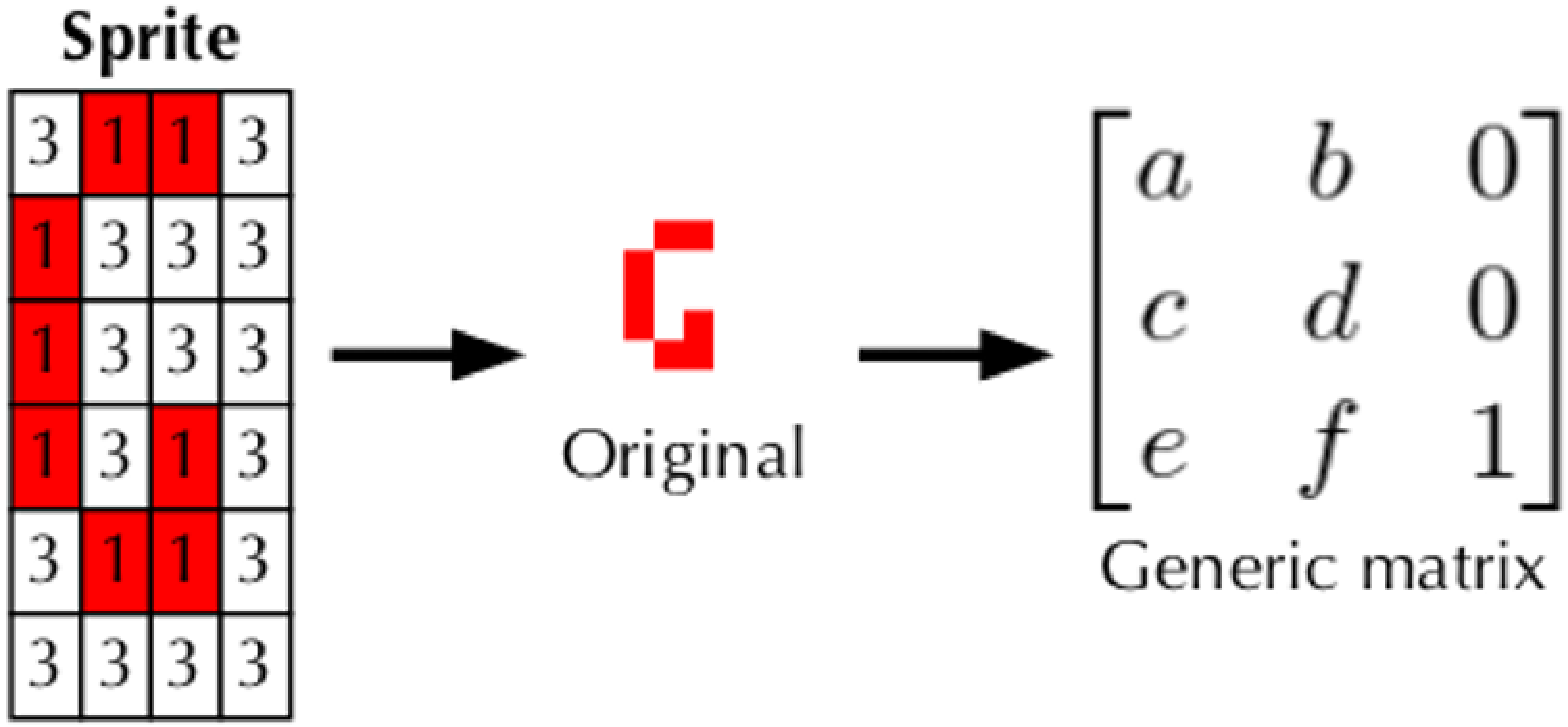
- Some calls always render 1:1 on the screen.
- Some calls take a transformation matrix.
- Some calls take a scale block.

Pyromaniac has `Scale` and `Matrix` objects to handle these operations.

Sprites

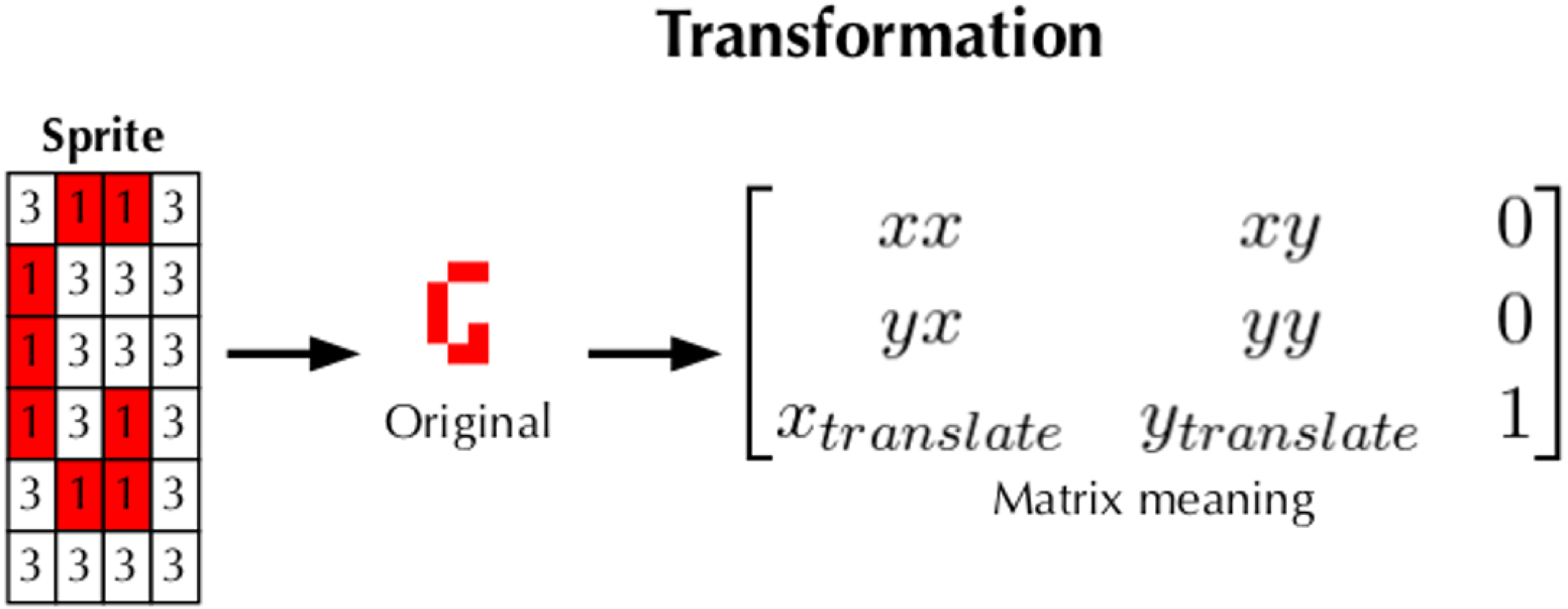
Transformations (2)

Transformation



Sprites

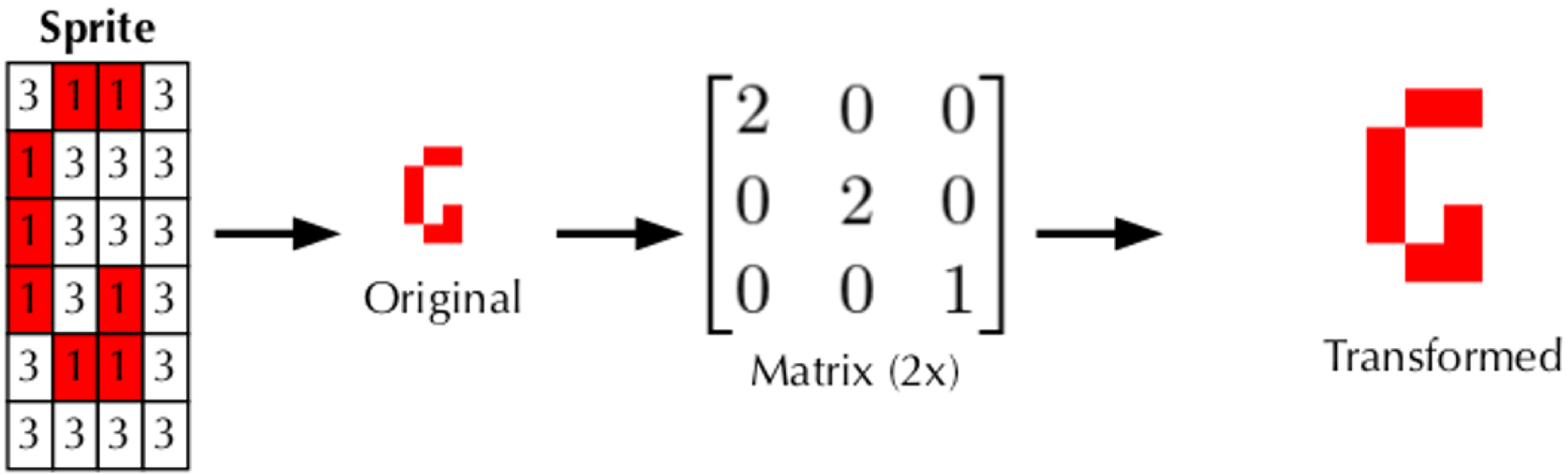
Transformations (3)



Sprites

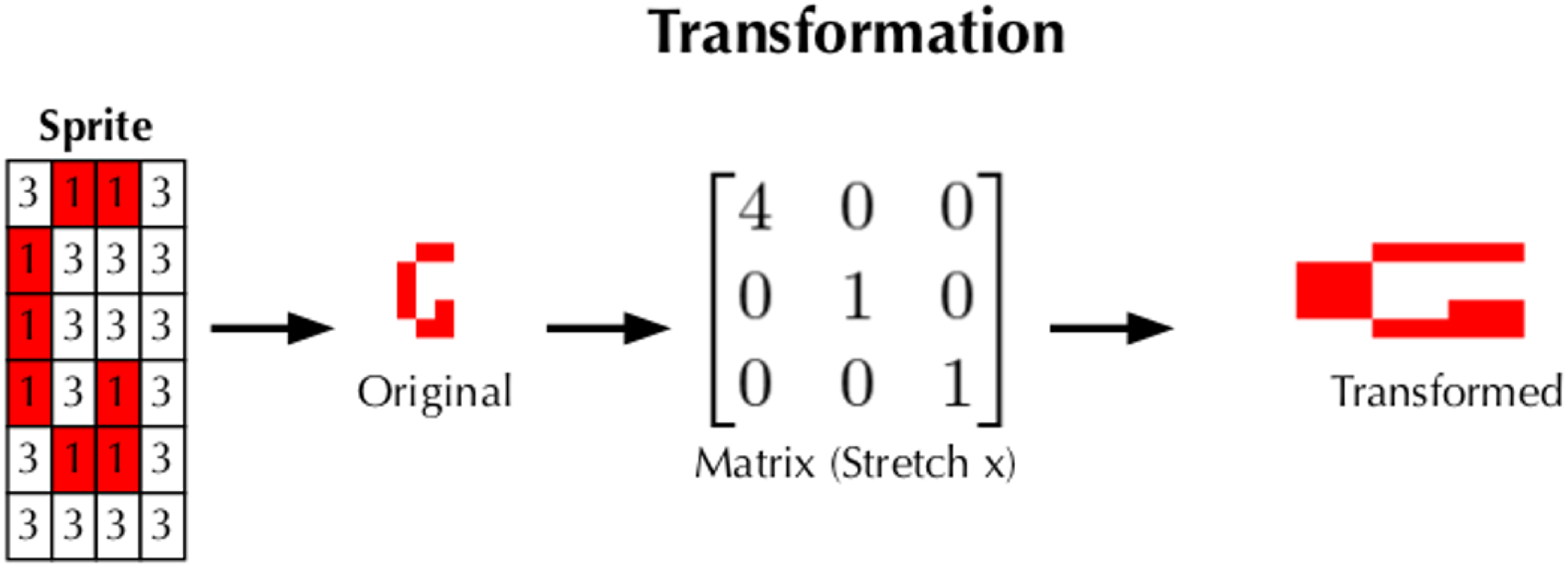
Transformations (4)

Transformation



Sprites

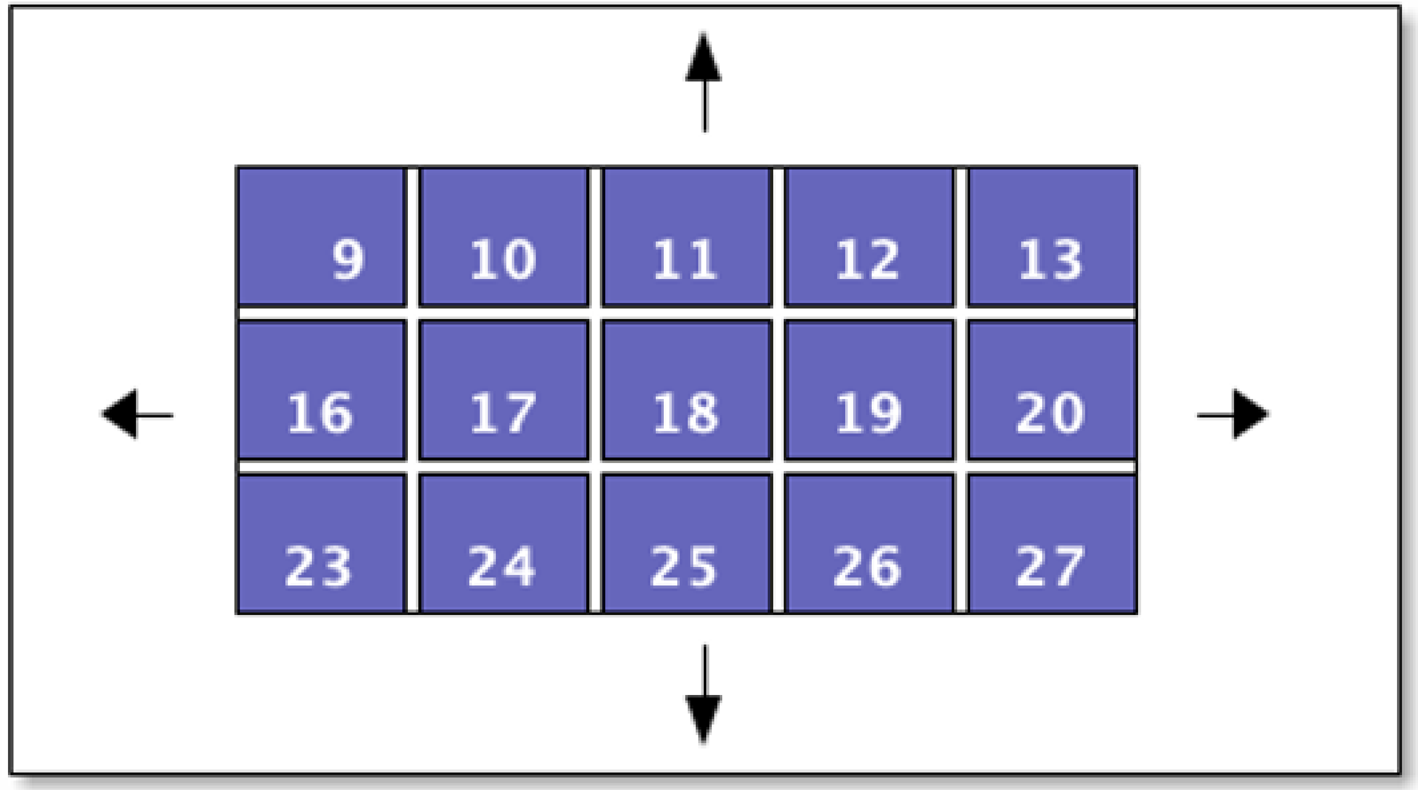
Transformations (5)



Sprites

Tiling (1)

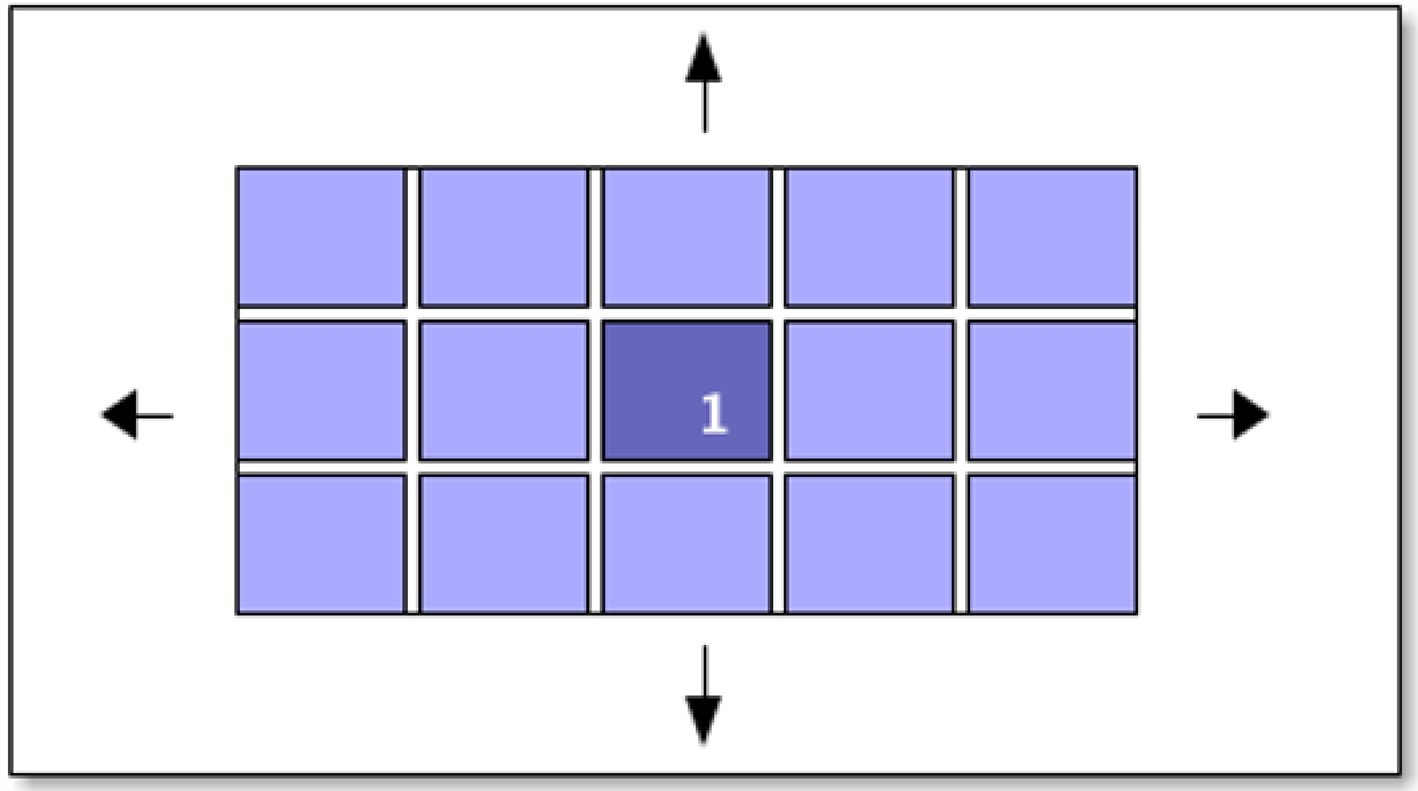
- Tiling is used for the desktop background tile.
- Traditionally this was done by repeatedly calling `OS_SpriteOp`, like this:



Sprites

Tiling (2)

Tiling a sprite with the interface is simple - plot the sprite at a single location and it fills the graphics window:



Sprites

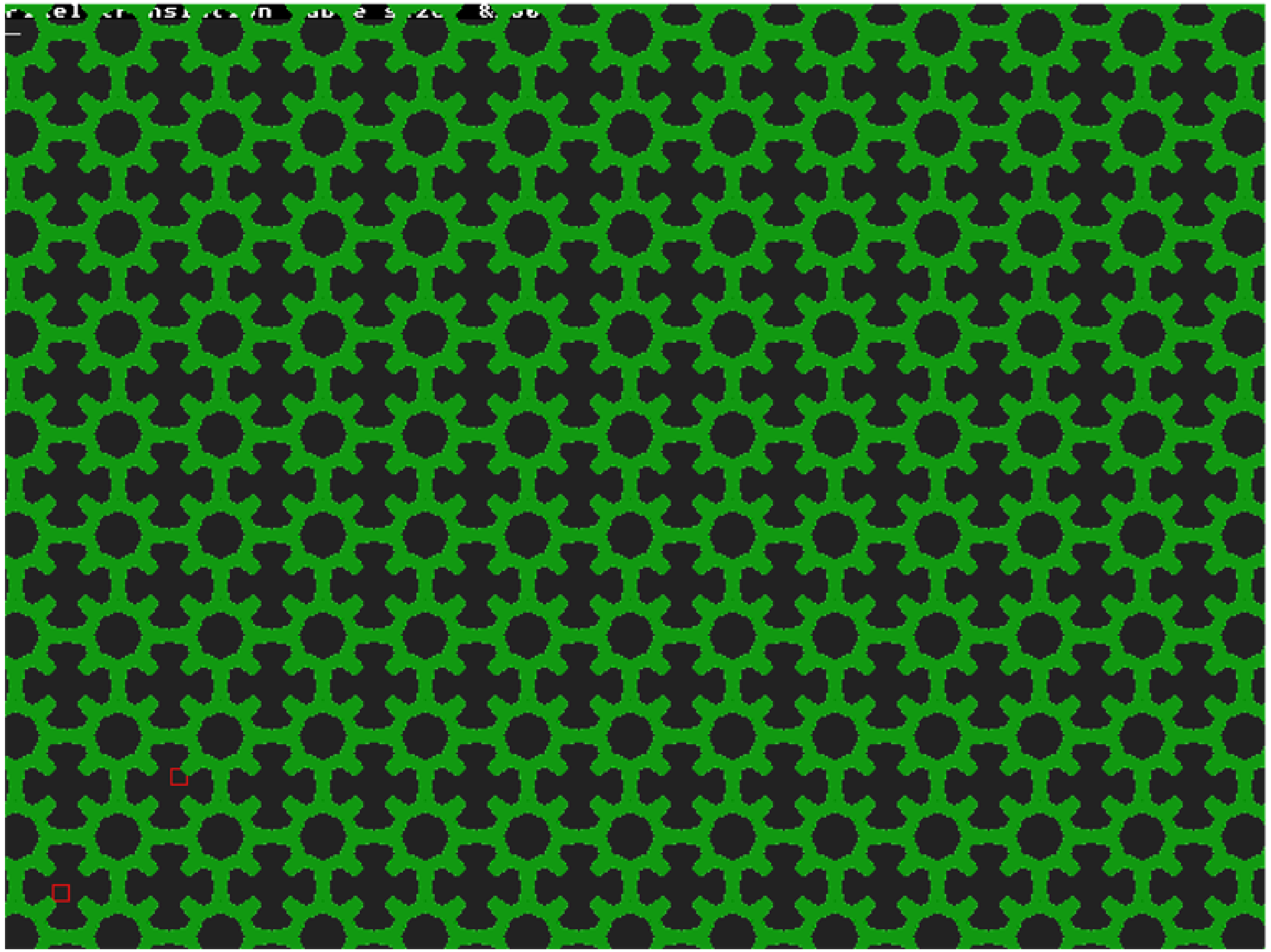
Tiling (3)

```
if surface:
    spattern = self.cairo.SurfacePattern(surface)
    spattern.set_filter(self.cairo.FILTER_NEAREST)
    spattern.set_matrix(invcmatrix)
    if tile:
        spattern.set_extend(self.cairo.Extend.REPEAT)
    context.set_source(spattern)
    graphics._set_action(plot_action)
else:
    graphics._set_colour(plot_colour, plot_action)

if tile:
    context.rectangle(graphics.windowx0, graphics.scrheight - graphics.windowy1 - 1,
                      graphics.windowx1 - graphics.windowx0 + 1,
                      graphics.windowy1 - graphics.windowy0 + 1)

context.set_matrix(cmatrix)
if not tile:
    context.rectangle(0, 0, sprite.width, sprite.height)

if translucency:
    alpha = (255 - translucency) / 255.0
    context.clip()
    context.paint_with_alpha(alpha)
else:
    context.fill()
```

Sprites

Tiling (4)

Tiling a cog sprite:



VNC game demo

To play your own game, instructions are at: https://railpro.riscos.online/

Connect to VNC at: vnc.railpro.riscos.online display 8 (port 5908)

Password: `password`

3. Fonts

Fonts

Where were things last year?

- FontManager kinda worked.
- But it was on a branch - I wasn't confident with it yet.
- Didn't handle control codes properly, or consistently.
- Required a lot of work for me to be happy with it.

Fonts

Simple text (1)

- Simple rendering was simple - stop on a 0 byte.
- Fine for the presentation, because it doesn't do anything fancy.

To render fonts, RISC OS Pyromaniac has two major parts:

- The graphics system's font interface.
  - For example selecting fonts, sizing simple text, drawing text with transformations.
- The RISC OS-facing SWI interface.
  - For example `Font_FindFont`, `Font_ScanString`, or `Font_Paint`.

Largely, Pyromaniac just turned the SWI calls into graphics system calls in this simple text system... but the WindowManager needs more.

Fonts

Simple text (2)

Fonts

Control codes

Font control codes the SWIs need to support:

- `0`, `10`, `13` - terminates the string
- `9`, `11` - moves the cursor horizontally and vertically by a specified amount
- `17` - changes the foreground colour
- `18` - changes the foreground and background like `Font_SetFontColours`, using GCOLs
- `19` - changes the foreground and background like `ColourTrans_SetFontColours`, using RGB values.
- `21` - hides text until the next control character
- `25` - sets the underline parameters
- `26` - selects the font to use
- `27`, `28` - changes the transformation matrix (with and without translation)

Fonts

Spacing parameters

Spacing in menus would usually be put in the menu text like this:

```
Save     ^F3
```

And then the WindowManager handles the alignment for you:

- It gets the size of the string without any extra spaces.
- Subtracts from the menu width.
- And then uses the remaining space as the inter-word spacing parameter.

Pyromaniac only supported this in a simple way by splitting on spaces. And it didn't support the inter-character spacing.

Fonts

Font encodings (1)

- `WIMPSymbol` provides the shift and other arrows which were in the VDU 4 only Wimp.
- The WindowManager switches the font string to the `WIMPSymbol` font when it sees these characters.
- When you use the character 139 (&8b) - the scroll up arrow - it gets converted to:
  - `26`, *WIMPSymbol font handle*, `139`, `26`, *desktop font handle*
- The FontManager uses font-specific encodings to handle this.
- So Pyromaniac has a version of this internally.
- The `python-codecs-riscos` module was updated to add the encodings for the symbol fonts `Sidney`, `Selwyn` and `WIMPSymbol`.

Fonts

Font encodings (2)

Fonts

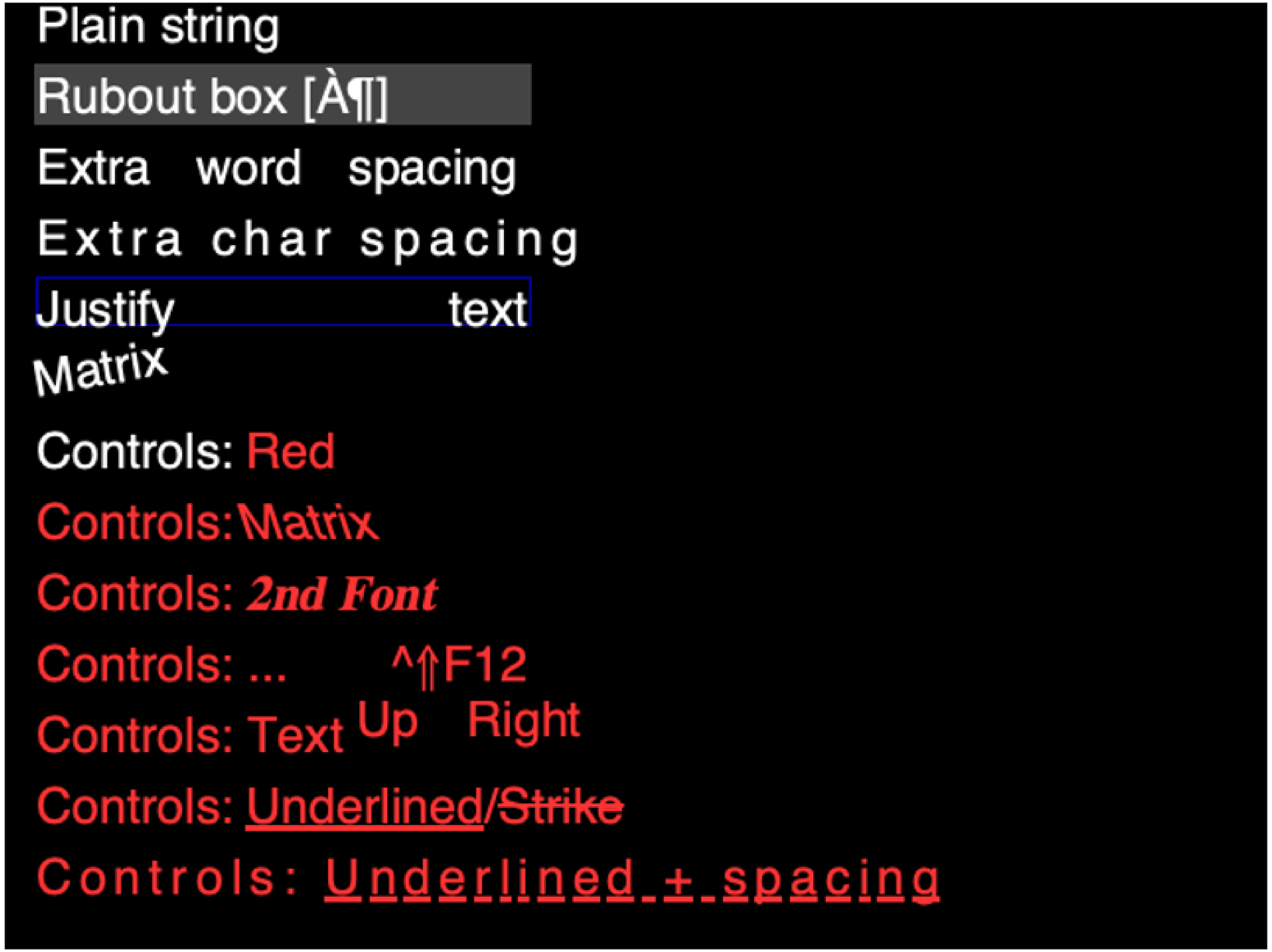
Improved control codes parsing (1)

How the WindowManager handles `Replace shift-F5` as a menu item:

- `[26, 1]` - *change font to desktop font*
- `"Replace "` - *regular string*
- `[26, 2]` - *change font to WIMPSymbol*
- `"\x8b"` - *regular string for the shift character*
- `[26, 1]` - *change font to desktop font*
- `"F5"` - *regular string*

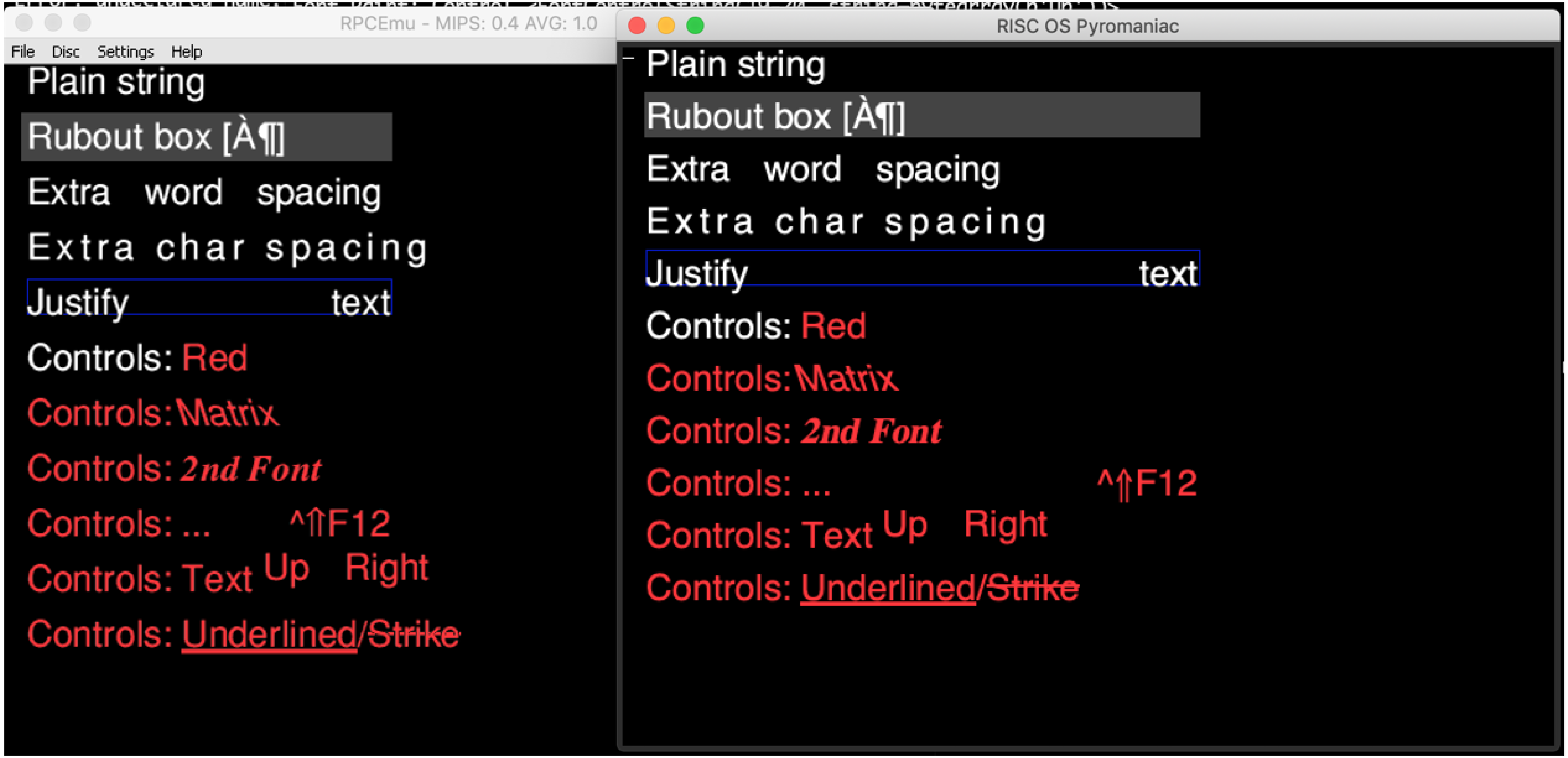
Fonts

Improved control codes parsing (2)



Fonts

Improved control codes parsing (3)



Fonts

Proper control code parsing (1)

- This is still a bit of a bodge.
- We only process things properly in `Font_Paint`.
- `Font_ScanString` and friends still use the control code stripping method.
- So I started again...
- ... But learnt from what I'd done before.
- The new parser was written using the principles of what I'd learnt about processing control code segments.
- It wasn't written inside Pyromaniac at all.

Fonts

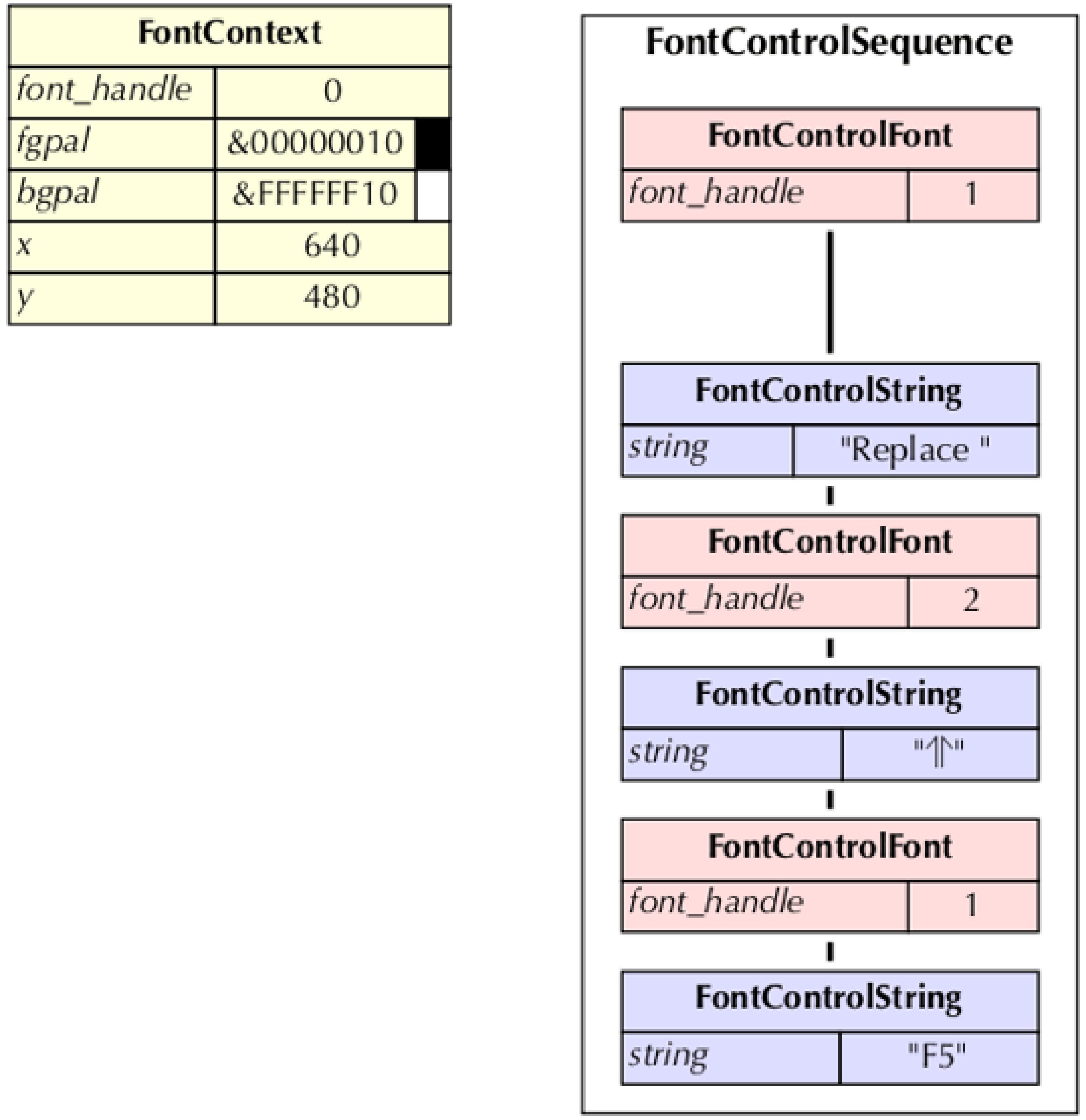
Proper control code parsing (2)

- `Matrix` and `Bounds` - define structures to manage transformation matrices and bounding boxes.
- `FontContext` - holds all the state for sizing or rendering, and controls processing.
- `FontControlParser` - performs reading the string and creating a list of operations.
- `FontControlSequence` - manages the list of operations described by the control sequence.
- `FontControl*` - classes which perform the operations for each control, eg `FontControlString`, `FontControlRGB`.

Fonts

Proper control code parsing (3a)

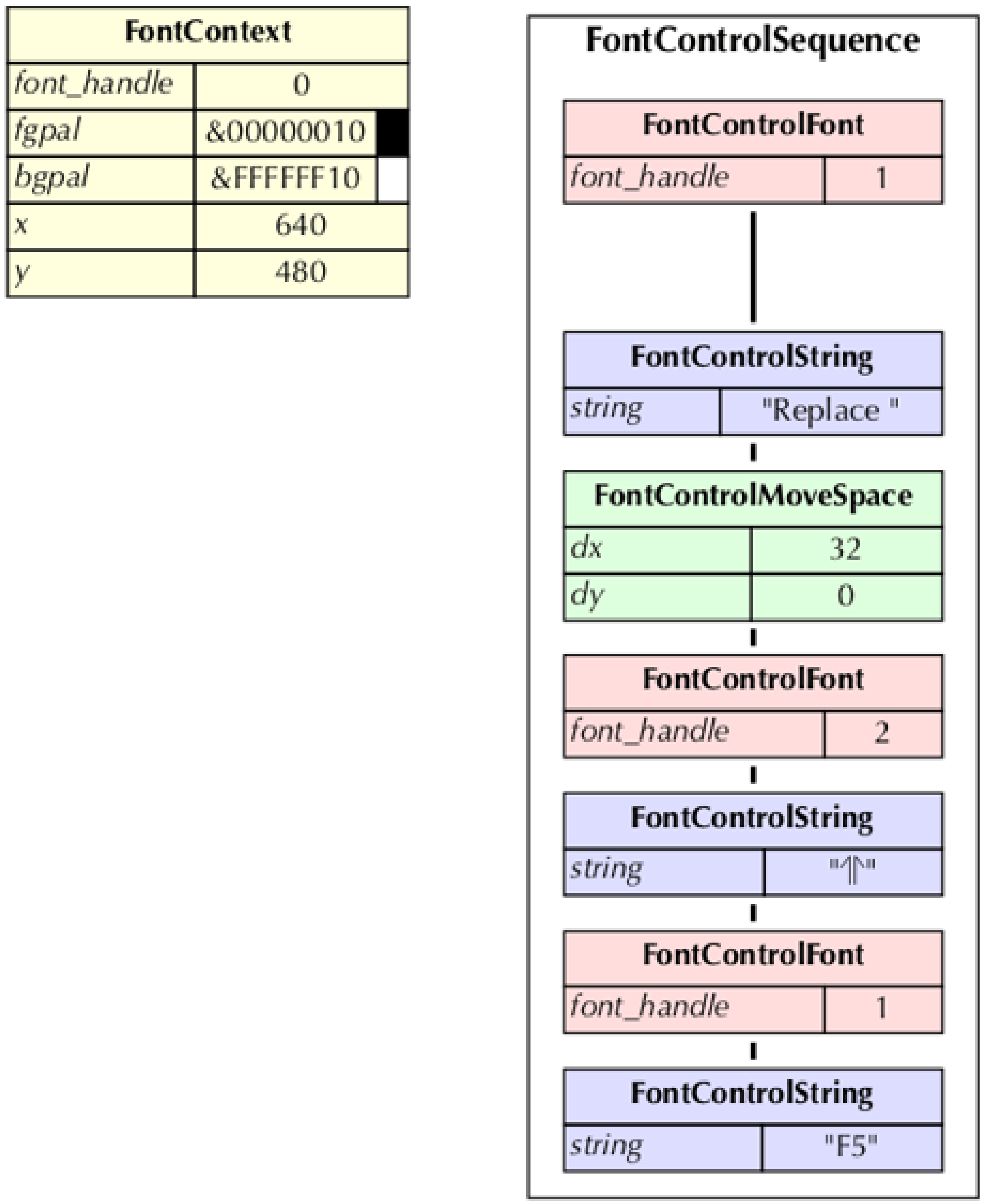
Font control sequence processing



Fonts

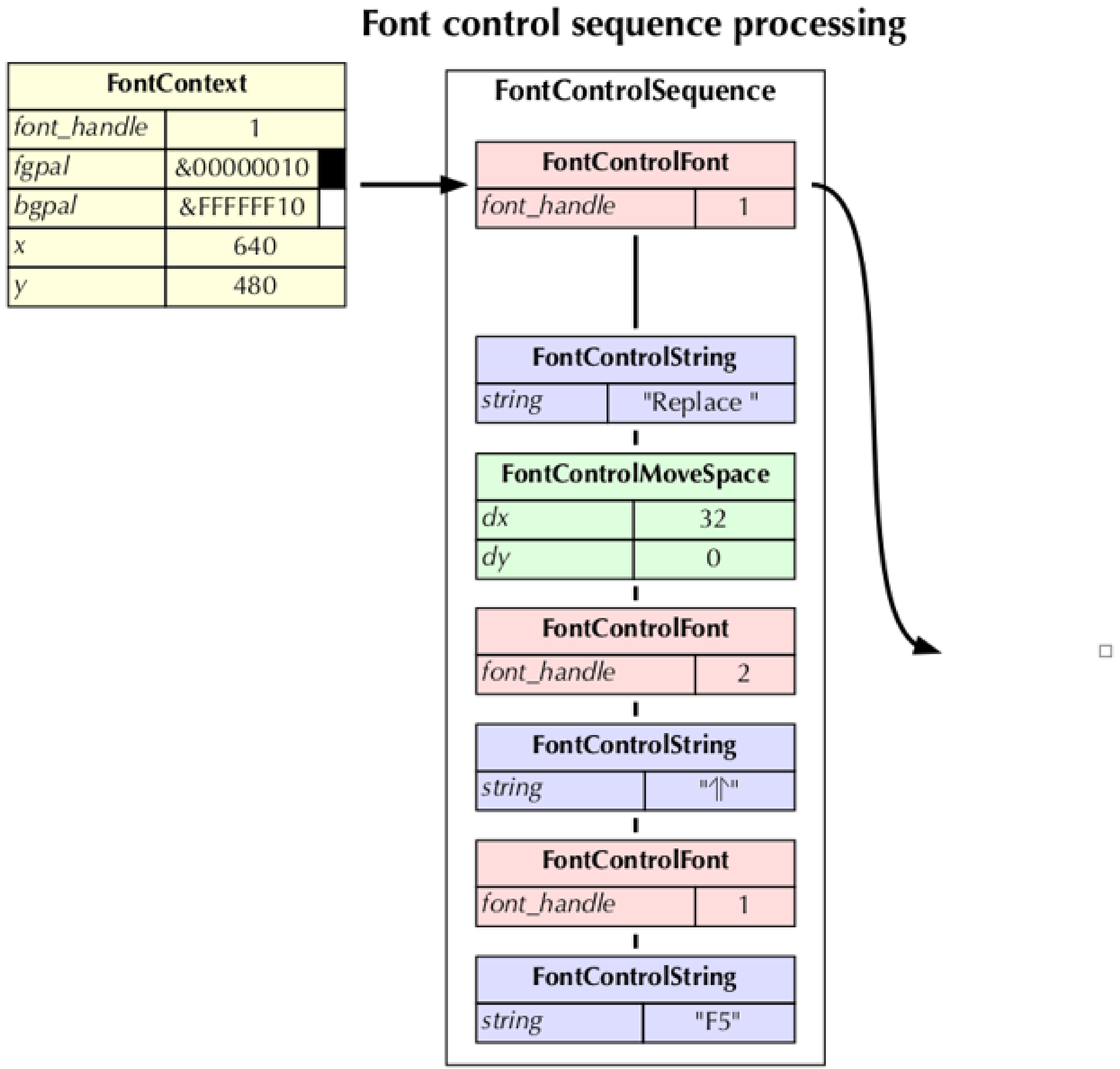
Proper control code parsing (3b)

Font control sequence processing



Fonts

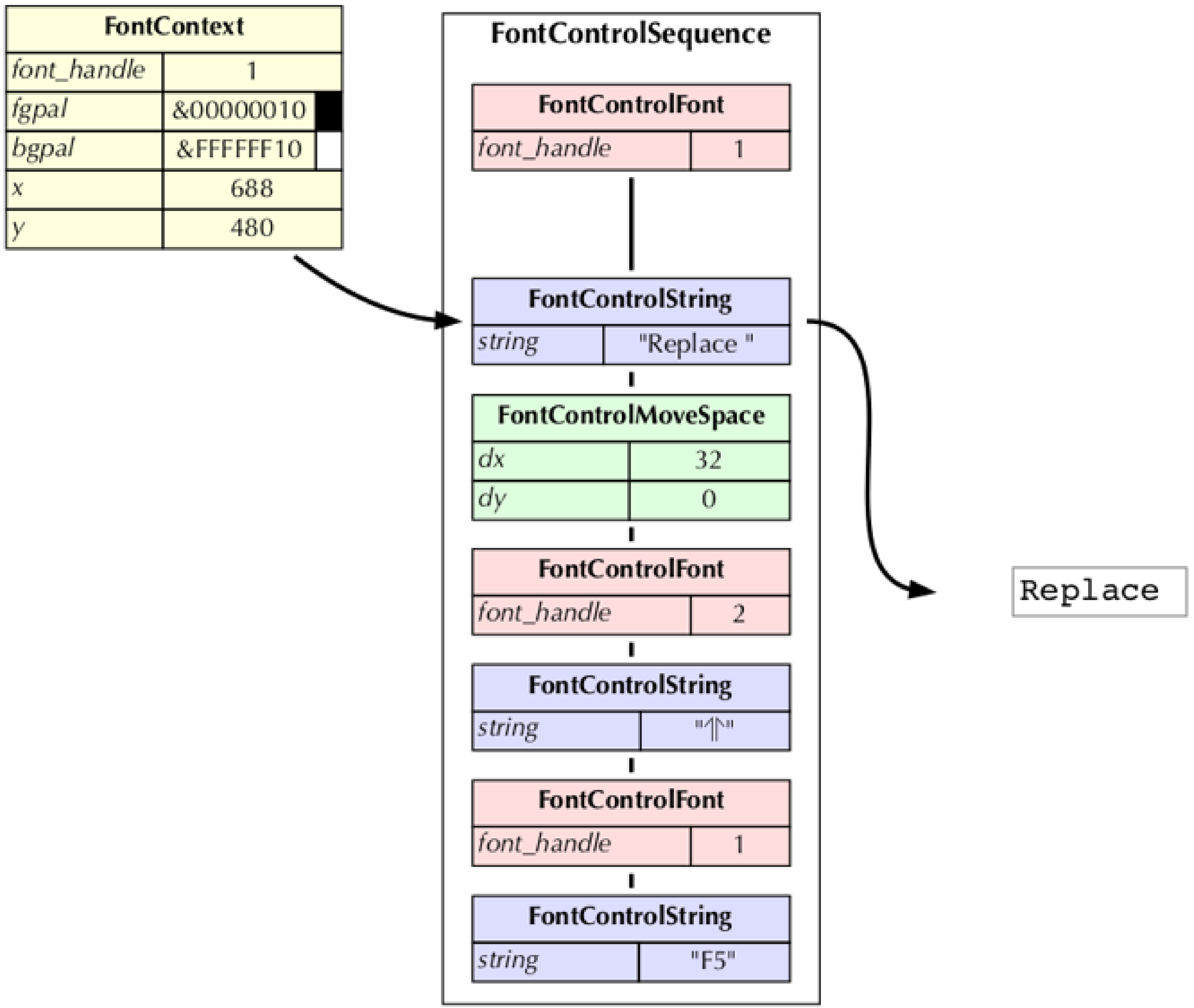
Proper control code parsing (3c)



Fonts

Proper control code parsing (3d)

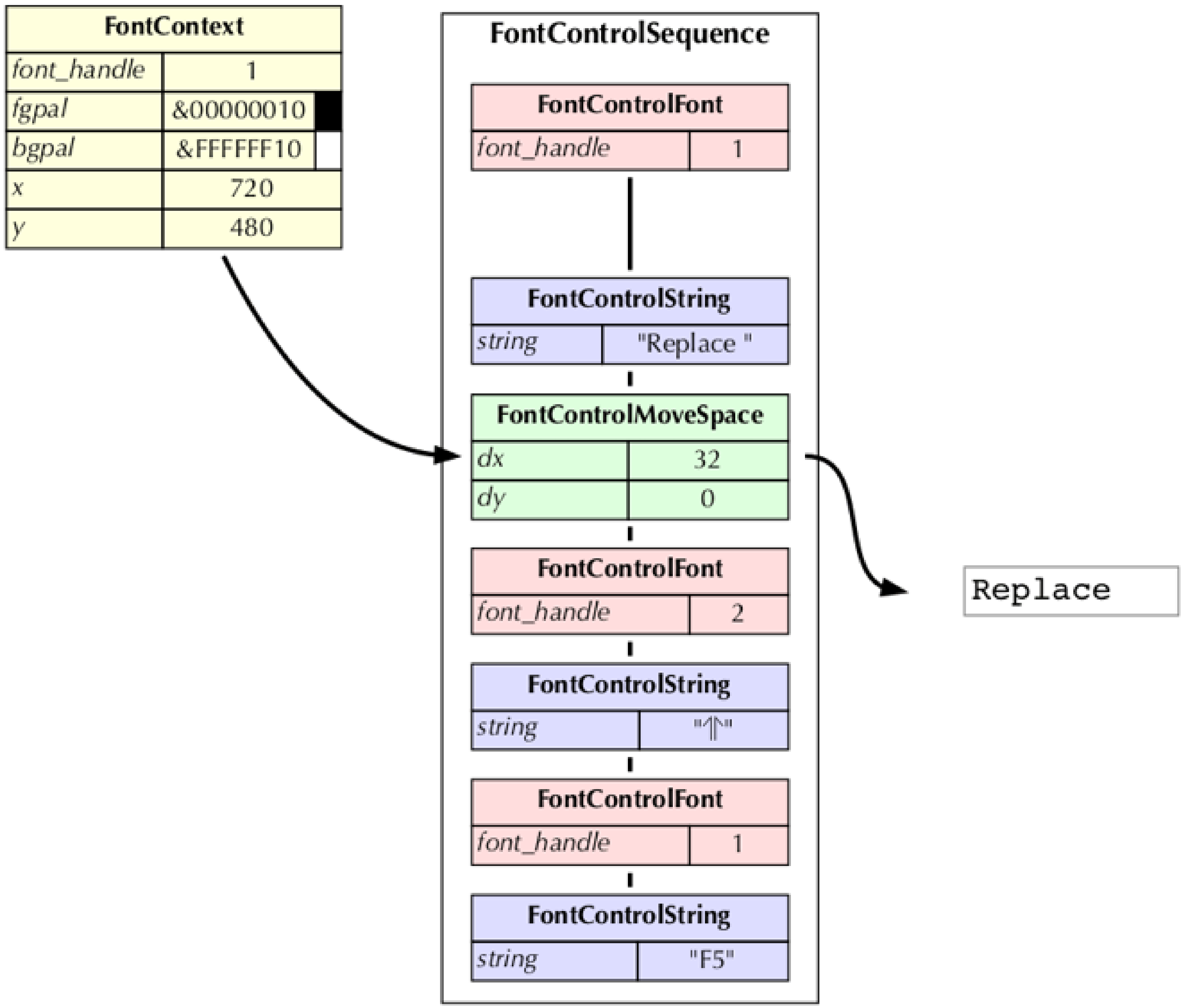
Font control sequence processing



Fonts

Proper control code parsing (3e)

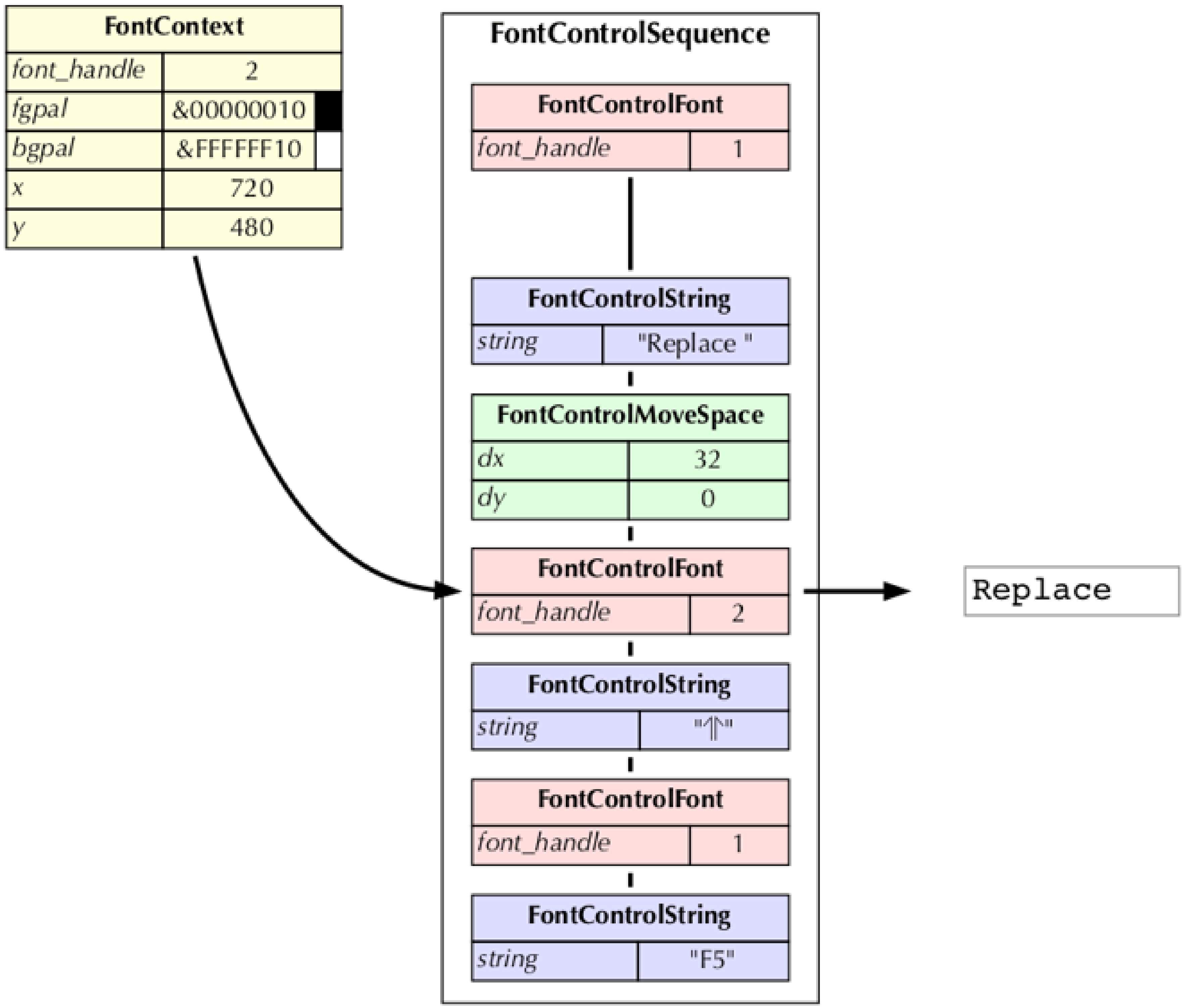
Font control sequence processing



Fonts

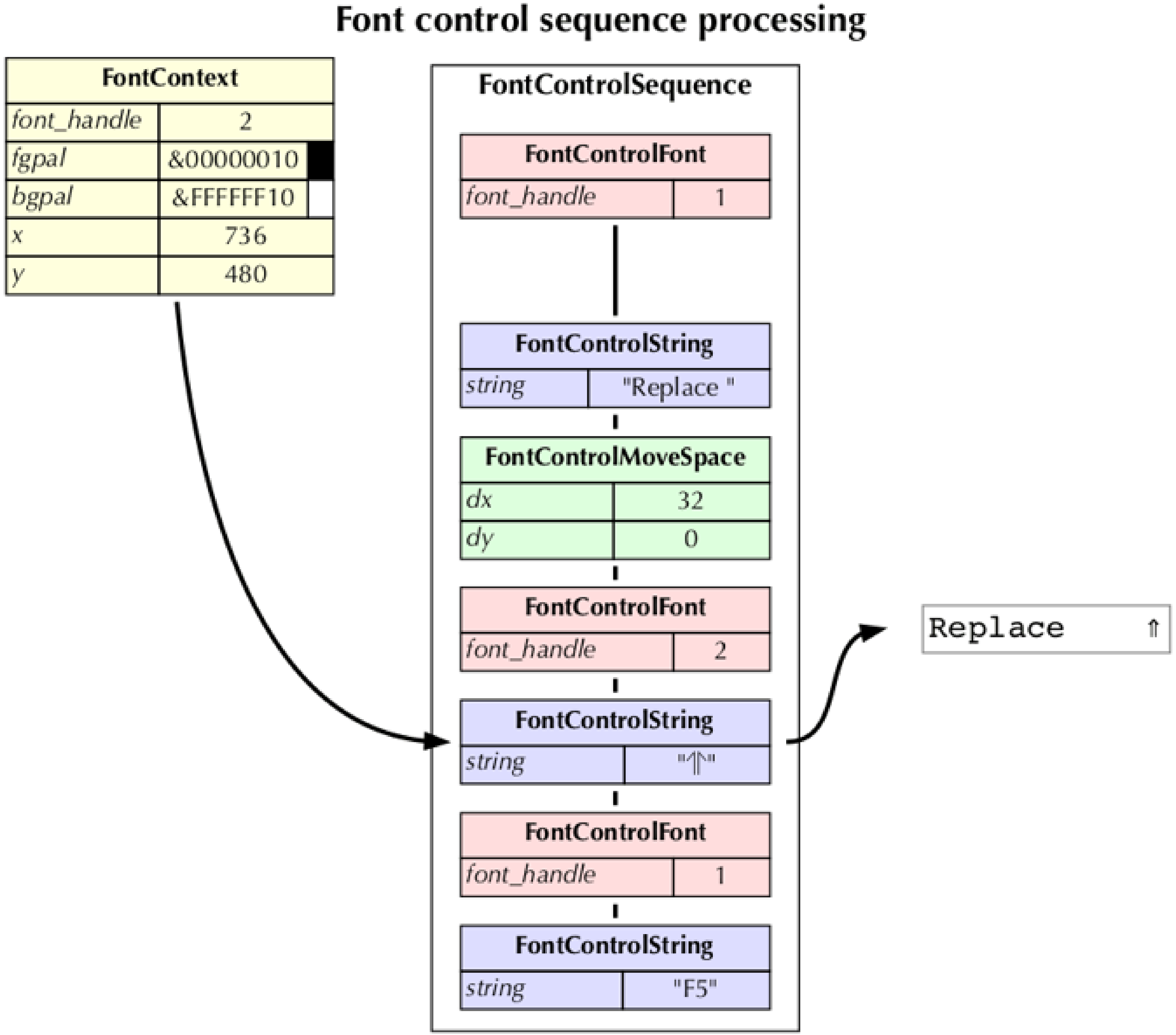
Proper control code parsing (3f)

Font control sequence processing



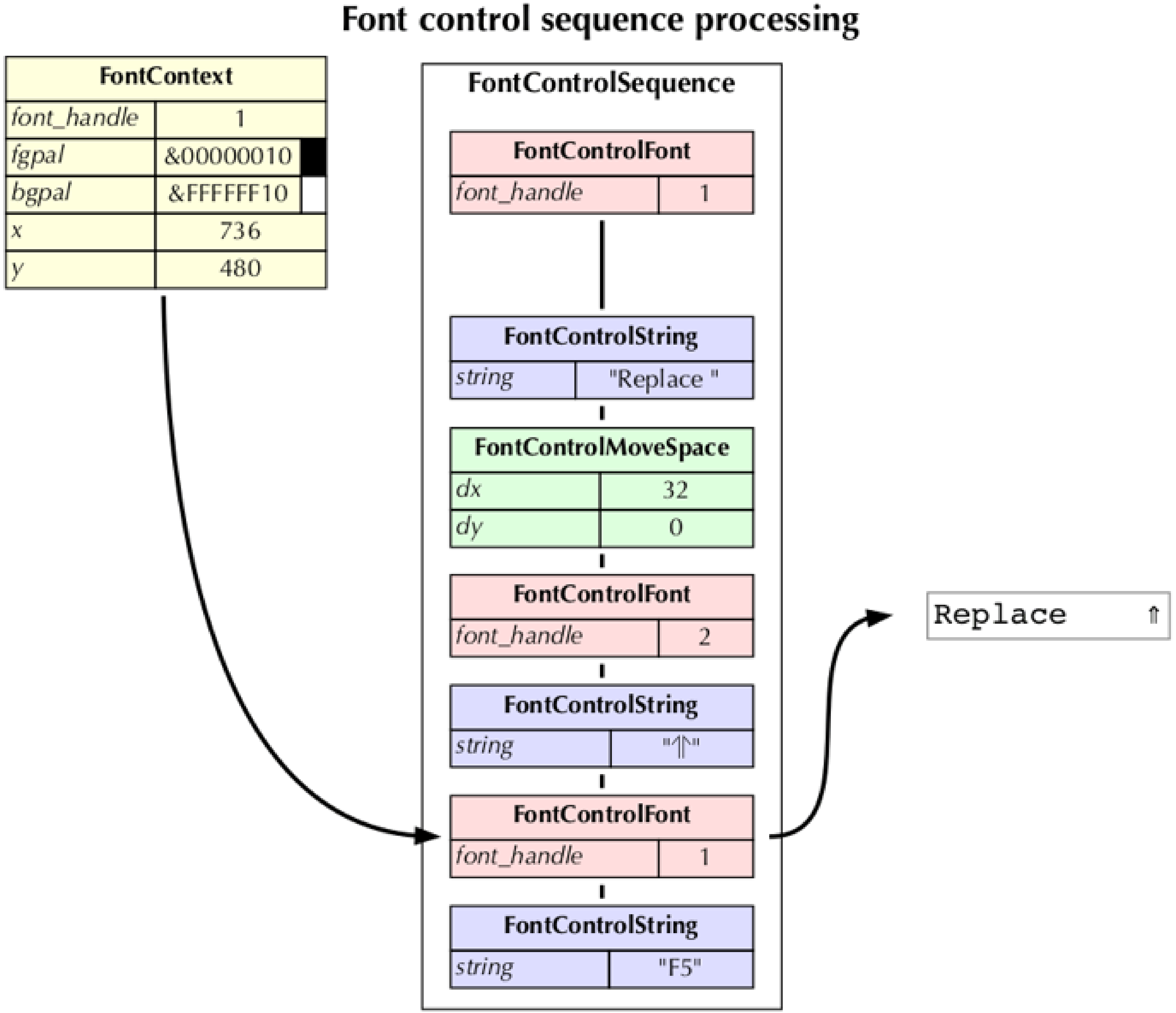
Fonts

Proper control code parsing (3g)



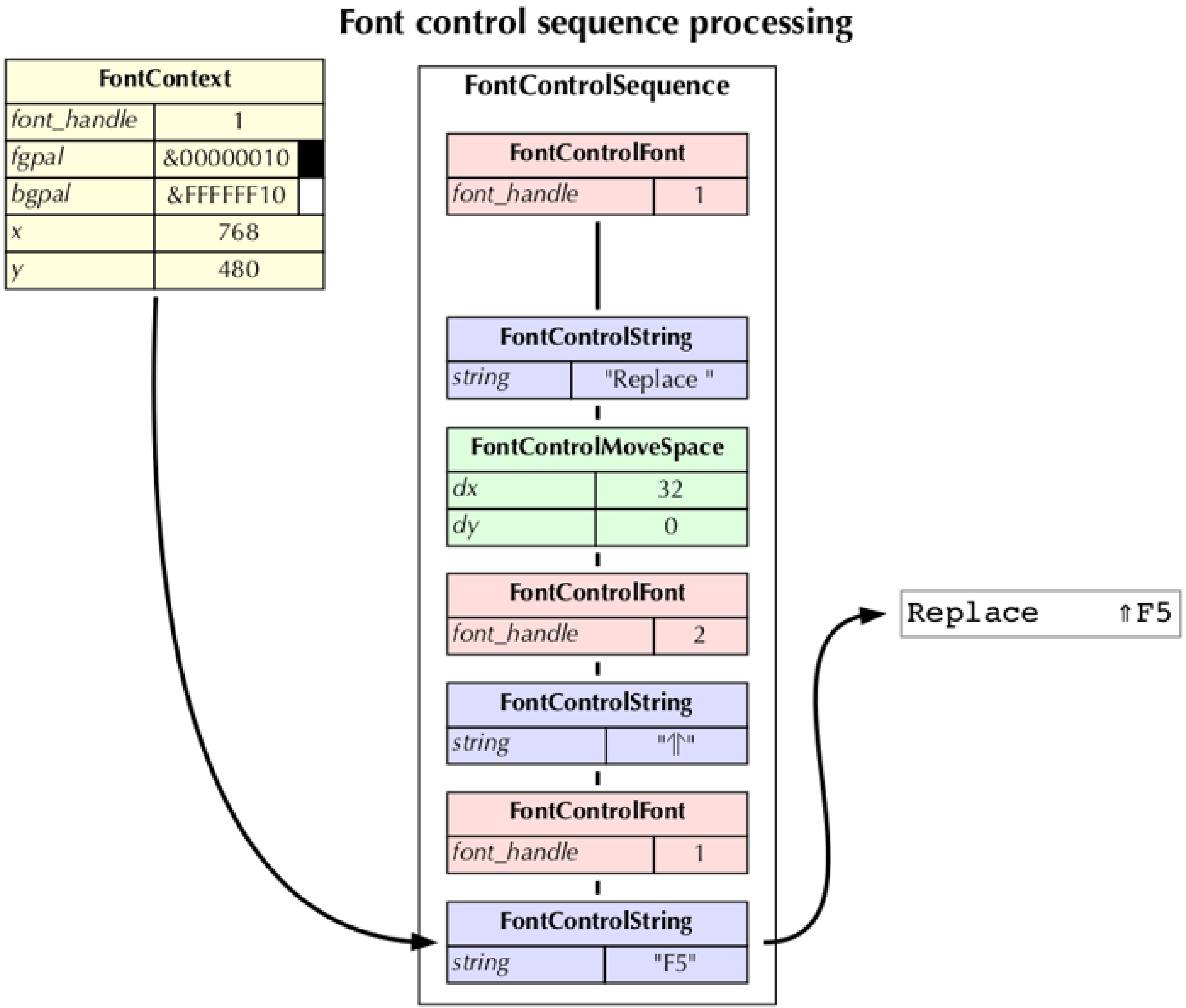
Fonts

Proper control code parsing (3h)



Fonts

Proper control code parsing (3i)



Fonts

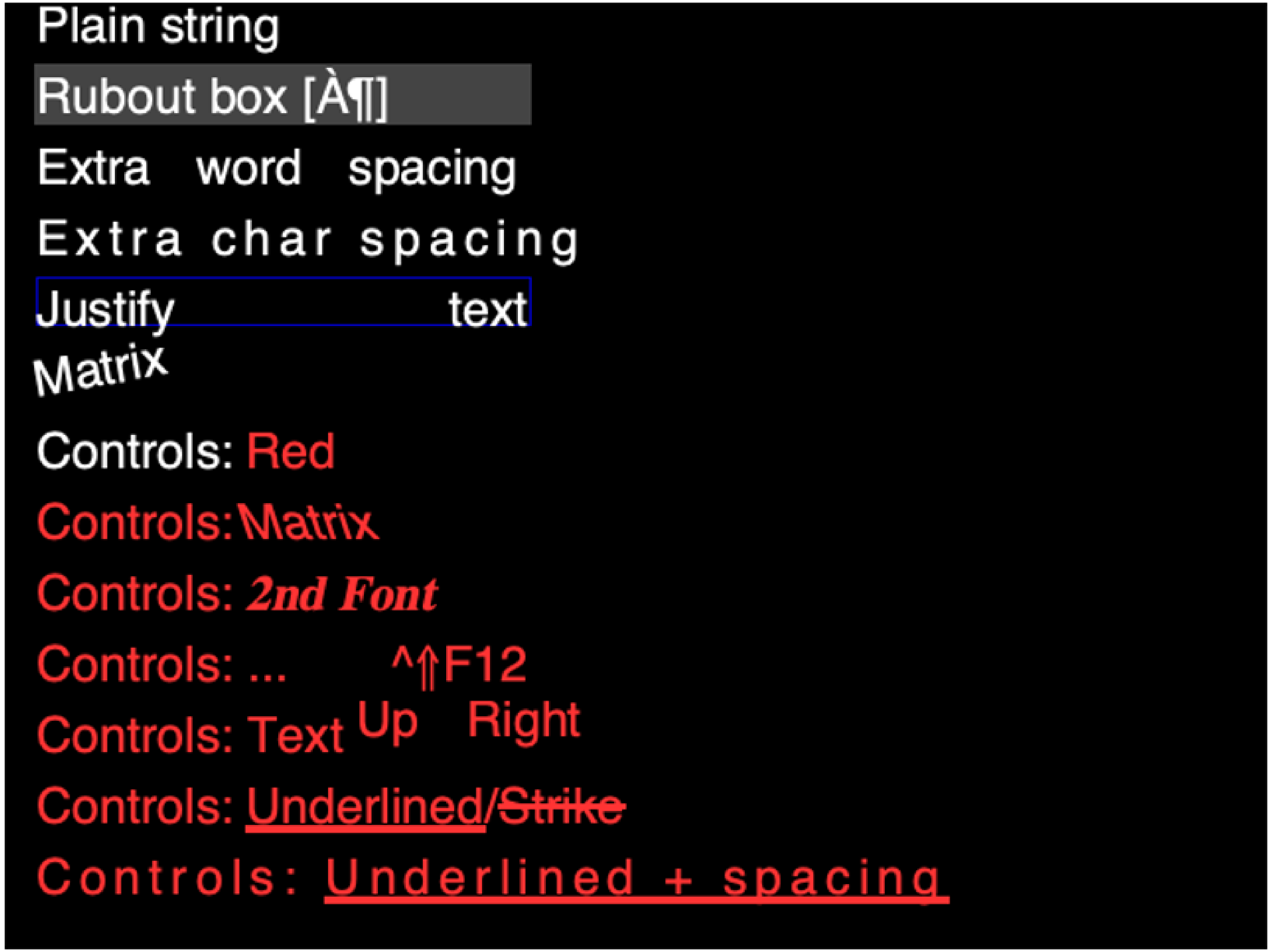
Proper control code parsing (4)

What about `Font_ScanString` and friends?

- They do the same thing, but call `size` instead of `paint`.
- At the end they can return other parameters such as the size and indexes.
- They update a `future_context` for `Font_FutureFont` or `Font_FutureRGB`.

Fonts

How's it compare?



Fonts

What does it look like?

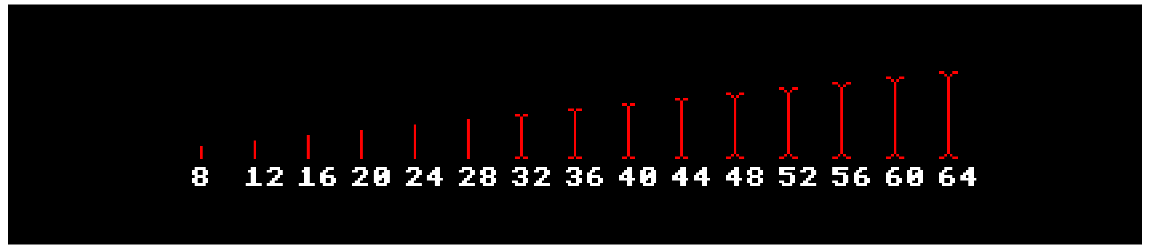
`Font_ScanString` has code that looks like this:

```
self.context.copy(to=self.future_context)
self.future_context.select_font(rohandle)

memstring = self.ro.memory[regs[1]]
fc = FontControlParserPyromaniac(self.ro)
fc.debug_enable = self.debug_fontparser
fc.parse(memstring, string_length)

self.future_context.transform = transform

split_char = chr(split_character) if split_character is not None else None
(split_offset, splits) = self.future_context.size(fc.sequence, spacing=spacing,
                                                 limits=(xmilli, ymilli),
                                                 split_char=split_char)
```

Fonts

Font caret

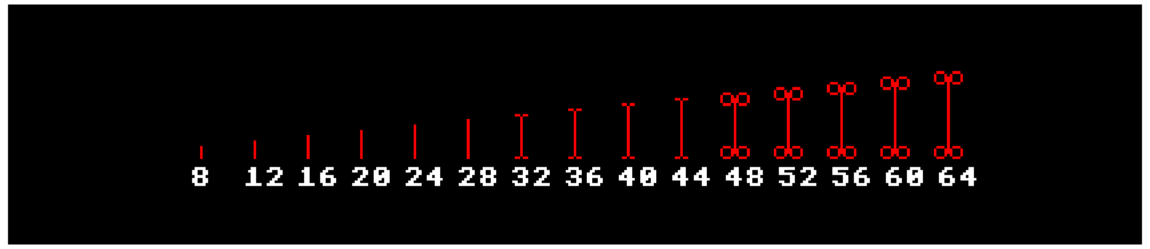
You can see here what the caret looks like at different heights:



Pyromaniac also offers you the option of putting loops on the ends; here's what the PRM says:

> *The height of the symbol, which is a vertical bar with 'loops' on the end, can be varied to suit the height of the text, or the line spacing.*

When enabled, the loops look like this:



Screen Modes

Introducing deep modes (1)

- Deep modes need to be handled like paletted modes.
- Palettes within the modes aren't indexed.

New `Palette` objects created with common interface:

- `palette.copy()`: Creates a copy of the palette.
- `palette.key()`: Return a hash value for this palette.
- `palette.lookup(rgb)`: Returns an exact colour number from an RGB.
- `palette.find_closest(rgb)`: Find the closest colour to the RGB value.
- `palette.find_furthest(rgb)`: Find the furthest colour from the RGB value.
- `palette.generate_32k_table()`: Return a 32K array of colour numbers.

Screen Modes

Introducing deep modes (2)

Buggy code:

```
def lookup(self, rgb):
    """
    Look up a colour from the palette by RGB value.

    @param rgb: The RGB value (&BBGGRRxx) to lookup

    @return: index of the colour, or -1 if not found
    """
    r = (rgb>>8) & 255
    if (r & 7) != (r >> 5):
        return -1   # Inexact colour
    g = (rgb>>8) & 255
    if (g & 7) != (g >> 5):
        return -1   # Inexact colour
    b = (rgb>>24) & 255
    if (b & 7) != (b >> 5):
        return -1   # Inexact colour

    return (r>>3) | ((g>>3)<<5) | ((b>>3)<<10)
```

Screen Modes

Mode strings (1)

Processing mode strings is needed so that BASIC for: `MODE "X800 Y600 C16M"`.

3 new SWI reasons were needed:

- `OS_ScreenMode 13`: Decode mode string to a mode specifier.
- `OS_ScreenMode 14`: Encode mode string from a mode specifier.
- `OS_ScreenMode 15`: Select mode by mode string.

Screen Modes

Mode strings (2)

Simple!

```
@handlers.osscreenmode.register(osscreenmode.ScreenModeReason_SelectModeString)
def OS_ScreenMode_15(ro, reason, regs):
    """
    OS_ScreenMode 15 (Select mode by mode string)
    => R0 = 15
       R1 = pointer to mode string

    This SWI is used to select a mode, given a mode string. Internally this is
    implemented as a conversion to a mode specifier (OS_ScreenMode 13) and
    mode selection (OS_ScreenMode 0), and is provided for convenience.
    """
    mode_string = ro.memory[regs[1]].string_ctrl

    with ro.kernel.da_sysheap.allocate(20 + 2 * 4 * 13 + 4) as modesel:
        spec = ModeSelector(ro, string=mode_string)
        buf = BufferData(ro, modesel)
        spec.write_selector(buf)

        ro.kernel.vdu.select_mode(modesel.address)

    return True
```

Screen Modes

Mode strings (3)

- The mode strings had originally been processed by the WindowManager.
- This presented a problem for greyscale modes - the `G` specifier.
- `Decode mode string` followed by `Select mode` would not know that it should be greyscale.
- New flag in the mode flags (mode variable 0) for greyscale modes (bit 9).
- The Mode objects spot this flag and report the default palette as greyscale.

Screen Modes

Mode enumeration

- `OS_ScreenMode 2` performs enumeration through `Service_EnumerateScreenModes`.
- Usually this would be handled by ScreenModes, using a loaded MDF.
- RISC OS Pyromaniac instead enumerates through the numbered modes.
- You can still select other modes manually.
- ScreenModes can be run and be used to load an MDF, if you really wanted.

Screen Modes

Multiple displays

- RISC OS Select allows multiple displays to be connected and switched between.
- RISC OS Pyromaniac doesn't yet allow that.
- But it can describe the display that is connected.
- It will be fun introducing multiple displays!

4. Documentation

Documentation

Pyromaniac's APIs

- The PRM-in-XML project was created in 2001 to make it possible to migrate documentation to a more maintainable format.
- Pyromaniac had documentation for some of the changed APIs in this format.
- In particular, `OS_AMBControl` is fully documented.
- These have been expanded from 3 to 8 documents.

Documentation

PRM-in-XML documentation project

- An article was written for Iconbar to explain why PRM-in-XML exists, and what it can do.
- Alan Robertson got involved, and tried out process of creating documents.
- He found many issues, which we addressed some of.
- I created a staging area to collect converted documents.
- Alan converted some functional specification documents from HTML to PRM-in-XML.

Documentation

Alan's conversions

Wimp_ForceRedraw (&400D1)

Wimp_ForceRedraw is changed so that it can be applied to windows owned by other tasks, because a child window may belong to another task.

In the past, redrawing the title bar of a window has been accomplished either by working out where the window's title bar is on the screen and calling Wimp_ForceRedraw with R0=-1 to invalidate that area, or alternatively by toggling the input focus in and out of the window to force its borders to be redrawn.

Neither of these methods is particularly satisfactory: the first could cause other windows on top of the one in question to be redrawn unnecessarily, and the second redraws the rest of the borders as well, and in the case of child windows, would also cause a redraw of the parent's title bar.

So Wimp_ForceRedraw is extended as follows:

On entry

R0 Window handle (as before)

R1 "TASK" (&4B534154)

This signals that the extended version of Wimp_ForceRedraw is being used, and R2-R4 are as stated below.

R2 +3

Redraw title bar.

Other values are reserved.

R3, R4 Ignored.

On exit
Unchanged

Interrupts
Unchanged

Re-entrancy
Unchanged

Notes
Since the value &4B534154 ("TASK") is far too big to be an minimum x coordinate, it is safe to use as described above.

Wimp_ForceRedraw
(SWI &400D1)

On entry

R0 = Window handle (as before)
R1 = "TASK" (&4B534154)

This signals that the extended version of Wimp_ForceRedraw is being used, and R2-R4 are as stated below.

R2 =

Value	Meaning
+3	Redraw title bar
	Other values are reserved

R3 - R4 = Ignored

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

Wimp_ForceRedraw is changed so that it can be applied to windows owned by other tasks, because a child window may belong to another task.

In the past, redrawing the title bar of a window has been accomplished either by working out where the window's title bar is on the screen and calling Wimp_ForceRedraw with R0=-1 to invalidate that area, or alternatively by toggling the input focus in and out of the window to force its borders to be redrawn.

Neither of these methods is particularly satisfactory: the first could cause other windows on top of the one in question to be redrawn unnecessarily, and the second redraws the rest of the borders as well, and in the case of child windows, would also cause a redraw of the parent's title bar.

So Wimp_ForceRedraw is extended as shown above.

Note: Since the value &4B534154 ("TASK") is far too big to be an minimum x coordinate, it is safe to use as described above.

Related APIs

None

Documentation

Updating the PRMs

- Tools now work on RISC OS and linux.
- They're more flexible about what they can do and how they report errors.
- The PRMs themselves have been updated to include more documents which had not been present previously.
- The results are looking good so far.

Documentation

Collaboration (1)

- Wanted a way to represent key and mouse input.
- Original method: `&shift;` and `&ctrl;` and the like are pretty inflexible.
- Try some elements to describe modifiers...

```
<key ctrl='yes' shift='yes' alt='yes' meta='yes'>X</key>
```

- Some other ideas...

```
<mouse shift='yes' button='select'/>
```

```
<input key='ctrl'/><input key='x'/>
```

```
<input shift='yes' mouse='select'/>
```

Documentation

Collaboration (2)

Eventual layout and example output:

```
<input><key name='W'></input>
<input><key name='ctrl'/><key name='X'/></input>
<input><mouse name='select' repeat='2'/></input>
<input><mouse name='select' action='drag'/></input>
<input><key name='shift'/><mouse name='select'/></input>
```

- W - the W key
- CTRL X - control and X (eg a cut)
- 2×SELECT - double click select (eg run a file)
- SELECT - drag select (eg making a selection)
- Press SELECT - press select (eg starting a selection)
- SHIFT SELECT - shift + select click (eg adding to a selection)
- ESC - press escape twice (eg some special operation to get out of a system?)
- SHIFT CTRL F12 - shift + control + F12 (eg shutdown)

Documentation

Making it look good

- The most recent version for the transformation is what you have seen here.
- This transformation supports HTML5 and CSS, with new elements planned to make it easier to structure certain elements.
- New styles can be applied to the CSS to change how it looks for a given purpose.
- A number of 'variants' of the styles are supplied and can generally be overlaid.
- This allows people to change what they don't like in the styling, and obviously the structured content itself is easier to manipulate.
- Acorn applied different styles over the years, with each manual looking subtly (or strikingly) different to the one that went before.

SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see page 1-78)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI cannot be re-entered as it disables IRQ

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see *List of software vectors (on page* 42))
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number
R1 = address of claiming routine
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI cannot be re-entered as it disables IRQ

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any earlier instances of the same routine are removed. Routines are defined to be the same if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

See below for a list of the vector numbers.

Example:

```
MOV R0, #ByteV
ADR R1, MyByteHandler
MOV R2, #0
SWI "OS_Claim"
```

Related SWIs

OS_Release (SWI &20), OS_CallAVector (SWI &34),
OS_AddToVector (SWI &47)

Related vectors

All

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see *List of software vectors (on page 39)*)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Documentation demo

5. Testing

Testing

How do you test these features

- Testing in most of RISC OS Pyromaniac is through expectation tests.
  - That means we write some output, and we compare it to what we expect.
  - Differences mean the tests fail.
- But that's not quite as easy for graphics.
  - Unless you make the graphics pixels into text too.
  - `PBM` files (plus `PGM`, `PPM` strictly) can be text forms of the graphics.
- When rendered into a small mode, the comparison becomes more manageable.

Testing

Testing sprites with text

```
P2
55 33
255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 1 1 1 1 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Testing

ColourTrans testing

- ColourTrans testing can be hard when there are so many combinations.
- It just deals with numbers, so we need to check they're the right numbers.
- Some interfaces like `ColourTrans_ReturnGCOL` can have some sampled tests.

```
Test ReturnGCOL conversion
Colour &0000ff00 => 1           Opposite  => 6
Colour &00ff0000 => 2           Opposite  => 5
Colour &ff000000 => 4           Opposite  => 3
Colour &ffffff00 => 7           Opposite  => 0
Colour &80808000 => 7           Opposite  => 0
Colour &81397900 => 4           Opposite  => 3
```

Testing

Testing user interfaces and platforms

- There are 3 user interfaces that you can use in the desktop:
  - WxWidgets
  - GTK
  - VNC
- None of them have any explicit tests.
- There are 2 applications that are produced:
  - Windows application
  - macOS application
- Neither have any explicit tests.
- Both work for me, and I've had some success with friends testing them. Eventually.

Testing

How much testing is there?

- There are now 1592 tests (up from 1022 last year).
- Code coverage is 67.19% (up from 65.8% last year).
- There are 82450 lines of python (up from 57997).
- There's a bunch more statistics up on the pyromaniac.riscos.online site.

Testing

Trace features (1)

- Better disassembly of some instructions.
- Can disassemble FPA instructions.
- More information about register and constant values in live debug:

```
3852384: LDR     r1, [r10, #&dc]              ; R10 = &05405738
3852388: TST     r1, #&1000000                ; R1 = &00000852, #16777216 = bit 24
```

- Decoding of dispatch tables and region names:

```
3841f78: CMP     r11, #&3e                    ; R11 = &00000032, #62  = '>'
3841f7c: ADDLO   lr, r11, #&2f                ; R11 = &00000032
3841f80: MOVLO   r11, r1                      ; R1 = &00000000
3841f84: ADDLO   pc, pc, lr, LSL #2           ; Table dispatch index #97
3842110: B       &0384331C                    ; -> Function: SWIWimp_ReadSysInfo
384331c: {DA 'ROM', module 'WindowManager': Function SWIWimp_ReadSysInfo}
```

Testing

Trace features (2)

- Improved MSR constant decoding:

```
3841e3c: MSRVS   apsr_nzcvq, #&20000000    ; #------ --- -- -- qvCzn
```

- `OS_WriteS` now reports the string that follows the instruction.
- A few more SWI interfaces report 'misuse' warnings.
- LegacyBBC can now report when its old interfaces are triggered.

Testing

Trace features (3)

- Locations in the trace now report region names.

```
Locations:
    r5  -> [&07065abc, &00000000, &00000000, &00000010] in DA 'Module area', module
'ColourPicker%Base' private word pointer
    r6  -> [&00000000, &0680141d, &06801426, &00000010] in DA 'Module area', module
'ColourPicker%Base' workspace
    r8  -> [&06801378, &068013c4, &00000000, &00000000] in DA 'Module area', module
'ColourPicker%Base' workspace
    r9  -> Function: resource_templates_free in DA 'Module area', module 'ColourPicker'
    r10 -> [&00000000, &00000000, &00000000, &00000000] in DA 'SVC Stack'
    r11 -> [&070528e4, &00000001, &07065aac, &00000000] in DA 'SVC Stack'
    pc is DA 'Module area', module 'ColourPicker': Function model_register+&a0
    lr is DA 'Module area', module 'ColourPicker': Function rgb_initialise+&104
```

Testing

Trace features (4)

- SWI traps allow a trace dump to be generated when a SWI is called.
- They now allow automatic transitions of the trace system:
  - `report`: Just reports the state as the SWI is entered and exited.
  - `trace`: Turns on code tracing whilst the SWI is executing.
  - `traceon`: Turns on code tracing when the SWI is entered.
  - `traceoff`: Turns off code tracing when the SWI is entered.
- This aids debugging if you know a SWI is called near where you're interested in.

Testing

UI Debug

- Command line debug options can be used to set the debug from the start.
- Inside RISC OS you can use `*PyromaniacDebug <options>` to change that.
- But sometimes it's easier to select things from the user interface... so...
- UI has a debug menu - which I'll show in a moment - so that you can change the debug information live.

6. Miscellaneous bits

Miscellaneous bits

Sound system (1)

- Original testing was using BBC program from *BBC Micro Music Masterclass*.
- This was a simple rendition of *Hall Of The Mountain King*.

```
 10 REM
 20 REM *** HALL ***
 30 REM
 40 REPEAT
 50 READ P
 60 IF P=0 THEN END
 70 SOUND 1,-10,P,5
 80 UNTIL FALSE
 90
100 DATA 61,69,73,81,89,73,89,89,85,69,85,85,81,65,81,81,61,69,
73,81,89,73,89,109,101,89,73,89,101,101,101,101,0
```

Miscellaneous bits

Sound system (2)

Converting RISC OS pitches to BBC pitches:

```
if pitch >= 0x100 and pitch <= 0x7FFF:
    # In BBC sound, 53 is middle C, with 4 steps per semitone;
    #    48 steps per octave
    # In RISC OS sound, &4000 is middle C, with &1000/12 steps per semitone;
    #    &1000 steps per octave
    riscos_pitch = pitch
    pitch = riscos_pitch / 4096.0 * 48
    pitch = int(pitch - 139 + 0.5)
    if self.debug_soundchannels:
        print("Converted RISC OS pitch &{:x} to BBC pitch {}".format(riscos_pitch,
pitch))
```

Miscellaneous bits

Sound system (3)

- For completeness I wanted to get the SoundScheduler working.
- This lets you play notes at later times using a beat schedule.
- Or you can call any SWI.
- Speeds were all wrong... but that fixed itself by not debugging it so much.
- SoundDMA is still on a branch and hasn't been updated this year.
- The sound system is complicated, but the implementation here works well enough.

Miscellaneous bits

Flashing cursor (1)

- Text cursor had only been implemented as a stub that didn't actually flash.
- Wanted to test that `OS_RemoveCursors` and `OS_RestoreCursors` were used properly.

All the code that needed to handle cursors has a context handler around them. The Font_Paint code looks like this:

```
with self.ro.kernel.graphics.vducursor.disable():
    self.context.paint(fc.sequence, spacing)
```

Miscellaneous bits

Flashing cursor (2)

The actual body of the cursor code looks like this:

```
(x0, y0, x1, y1) = self.ro.kernel.graphics.vdu4_coords(tx, ty)
y0 = y1 - self.ro.kernel.vdu.cursor_endline
y1 = y1 - self.ro.kernel.vdu.cursor_startline

fg = self.ro.kernel.vdu.fg
fg = 255 if self.ro.kernel.vdu.ncolour == 63 else self.ro.kernel.vdu.ncolour
for y in range(y0, y1 + 1):
    hline = self.ro.kernel.graphics.read_hline_internal(x0, x1, y)
    hline = [col ^ fg if col is not None else col for col in hline]
    hline = self.ro.kernel.graphics.write_hline_internal(x0, x1, y, hline)

# Notification that the bank was updated, so that the frame is rendered
self.ro.kernel.graphics.display_bank_updated()
```

Miscellaneous bits

Vectoring

- `WrchV`, which vectors all the VDU output (since BBC days) wasn't implemented.
  - Required for some `Wimp_CommandWindow` to work properly.
  - But it slows things down to do this for every character.
  - Now implemented, but bypassed if there are no claimants.
- `DrawV` is used by the `Draw` module to augment its interface.
- `FontV` had been intended to be vectored, but was disabled by Acorn.
  - Pyromaniac allows it to be enabled through configuration.

Miscellaneous bits

Hourglass

- The `riscos-hourglass-maker` repository has been updated.
- Support (on a branch) added for percentage digits, or a progress bar.
- The 'cog' hourglass within Pyromaniac uses these.

Miscellaneous bits

OS_Plot changes

Actions:

- `OS_Plot` only supported the simple 'set' operation.
- `EOR` is used quite often to animate shapes.
- Cairo provides a similar `DIFFERENCE` operator.
- Not quite right, but sufficient to work most of the time.

Dotted lines:

- `OS_Plot` can be used to draw lines using a dot pattern.
- The pattern is configured by `VDU 23, 6`, the length by `OS_Byte 163`.
- Cairo's dot pattern is defined by on-off lengths, so these needed converting.

Miscellaneous bits

Mouse input

- Mouse clicks worked, but...
- ... mouse double clicks never happened.
- Maybe the time wasn't reported properly?
- The mouse timestamp was wrong.
- ... but that wasn't the problem.
- The mouse buffer wasn't implemented, so maybe that was the reason.
- So I implemented the mouse buffer...

Miscellaneous bits

Mouse input

- Mouse clicks worked, but...
- ... mouse double clicks never happened.
- Maybe the time wasn't reported properly?
- The mouse timestamp was wrong.
- ... but that wasn't the problem.
- The mouse buffer wasn't implemented, so maybe that was the reason.
- So I implemented the mouse buffer...
- ... but then noticed that the events were never delivered to RISC OS.
- The WxWidgets interface weren't being delivered, because they were never requested!

Miscellaneous bits

New modules

- Zipper module
- TimerMod
- CryptRandom
- CDFSSoftPyromaniac driver
- Squash

System demo

8. Conclusion

Conclusion

What did I get done? (1)

- Graphics system improvements. Sprites, ColourTrans, deep modes, VNC server, hourglass.
- Filesystem improvements. More commands supported, FSControl, GBPB improvements, encoding improvements.
- Input improvements. Keyboard scans and mouse buffer handling.
- Debug improvements. Better trace reports, more info in disassembly. FPA instructions.
- Sound system improvements. SoundChannels. SoundScheduler.
- New modules. Zipper, TimerMod, Squash, CryptRandom, CDFSSoftPyromaniac.
- Many fixes across the system.
- PRM-in-XML documentation improvements.

Conclusion

What did I get done? (2)

- Shell server updated semi-regularly.
- RISC OS build server back end updated at the same time.
- The information site has been updated: https://pyromaniac.riscos.online/
- Lots of information about what's supported in the `Docs- Features` documentation.
- The full changelog in `Docs- Change Log` summarises many other things I couldn't cover here.

Conclusion

Is it still fun?

- Generally still fun!
- Collaborative working with Alan on the documentation system has been great.
- Taking the opportunity to talk about testing on RISC OS was a nice break in the middle of the year, and maybe I should do that more often.
- Sometimes investigating problems goes nowhere, but makes for interesting experiments.
- This presentation has been stressful to prepare - packing a year's things into a couple of hours is tricky.

Conclusion

What do I want to do next?

- Fixes for some of the known problems.
- Filesystem registration - allowing more than just the native filesystem to work.
- I have some ideas about sprite redirection.
- Multiple displays.
- Back Trace Structures.
- Port to Python 3.
- Experiment with changes to some of the internal interfaces.
- Use it to develop and test some things!

Questions

Info site: https://pyromaniac.riscos.online/
Shell: http://shell.riscos.online/