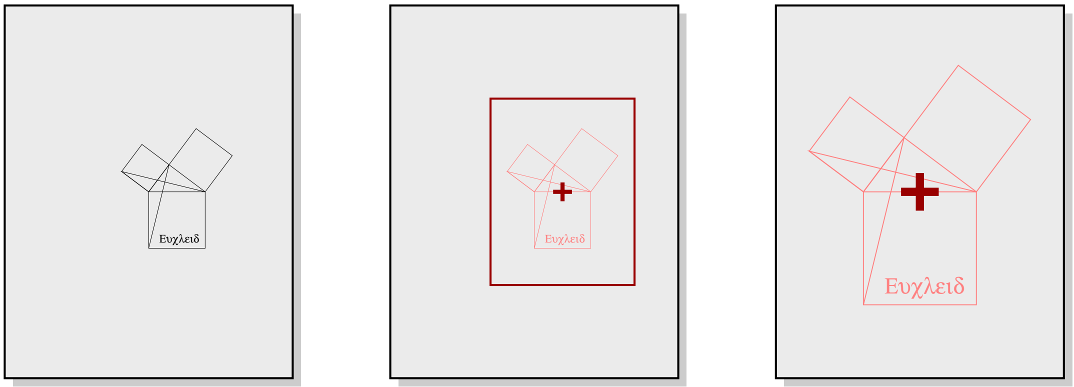
APPENDIX 5

Zooming

One of the greatest advantages of using PostScript for illustrations is that it is scalable—there are no artefacts in the illustration that show up when it is examined closely. This is in opposition to digital photographs, for example, when blow-up will start to show pixels. This appendix will explain how to take advantage of this.

1. Zooming

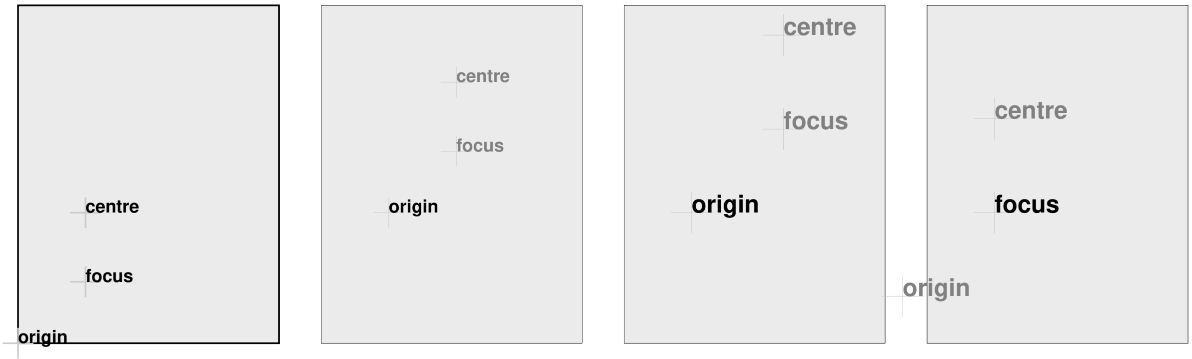
zoom:1 I shall explain here a procedure called `zoom` which has the effect of zooming in at a point by a given scale. The overall effect can be illustrated by these three figures, where the zoom factor is 2:



How can we do this? There are three arguments for this procedure. One is the scale factor c. If it's greater than one, the scale change is a magnification and we are zooming in. If it's less than 1, we are zooming out, not in. If it's exactly 1, there is no scale change, the zoom will amount to a translation of the origin. Another argument is a point (x, y) in the original figure. The last argument is the point (c_x, c_y) to which (x, y) is to be relocated. If we want to locate (x, y) at the centre of a page, for example, and if the current coordinate system is the page coordinate system. then $(c_x, c_y) = (306, 396)$. But if the origin of the current coordinate system is already at the centre of the page it is $(0, 0)$.

I call c the **zoom factor**, (x, y) the **focus** of the zoom, (c_x, c_y) its **centre**.

It is more or less clear that what we want is a succession of translation and scales, but in what order? And which ones? The simplest way to decide is to portray geometrically what has to be done:



This leads to the following code

```
cx cy translate
s dup scale
x neg y neg translate
```

which is to be inserted before the original drawing commands.

If you want to rotate the figure with the focus as the pivot, then add the correct line as here:

```
cx cy translate
20 rotate
s dup scale
x neg y neg translate
```

2. An explicit procedure

Call the following procedure before drawing.

```
 % On the stack when called are
 % [cx cy] [x y] s:  the place that is now (x, y) is located at [cx cy]
 % and lengths scaled by s
/zoom { 3 dict begin
  /s exch def
  aload pop
  /y exch def
  /x exch def
  aload pop
  translate
  s dup scale
  x neg y neg translate
  currentlinewidth s div setlinewidth
end } def
```

3. Playing around

Try this:

```
(zoom.inc) run
/draw {
  gsave
  1 0 0 setrgbcolor
  newpath
  x y moveto
  -100 0 rlineto
```

```
     200 0 rlineto
    x y moveto
    0 -100 rlineto
    0 200 rlineto
    stroke
    grestore
    x y moveto
    (Euclid) show
  } def

  /Helvetica-Bold findfont
  25 scalefont
  setfont

  /s 1 def
  /x 100 def
  /y 100 def
  {  % loop
    gsave
    [x y] [x y] s zoom
    draw
    grestore
    /s s 1.1 mul def
    showpage
  } loop
```

How would you get the text to rotate around the focus as the loop proceeds?

4. Code

See `zoom.inc`. There is a variant in there of the procedure `zoom`, called `Zoom`. The third argument for this procedure is an array of four numbers, specifying a linear transformation to be applied at the focus of the zoom. Used with `[s 0 0 s]`, for example, it is equivalent to a `zoom` with scale factor s.