How to GoDot #10

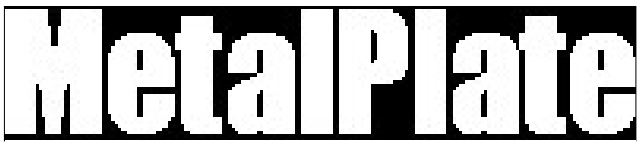
By Arndt Dettke

Scaling (up and) down

In many cases the size of the letters GoDot provides is not the size we need to have. For this purpose we must scale the writing to the wanted view before adding it to our graphics.

Scaling modifiers are **mod.ClipWorks** (options *Zoom* and *Shrink*), **mod.Squeeze2Clip**, **mod.StretchClip**, **mod.Double**, and **mod.Tile**.

If it doesn't matter whether the output of the scaling process is correct in its aspect ratio (like may be the case when dealing with letters), then it is quite easy to perform the process. Let's take the



Pic #1: "mptext.clp", 25 cards wide, 5 cards high

image from the last workshop as an example ("mptext.clp", see Pic #1), it contains the word "MetalPlate", written with GoDot. It's a GoDot clip of 25 cards width and 5 cards height. We want to stretch it to become 40 cards wide and 8 cards high.

Well, just load the clip image, render it (with button "*Display*"), and set the related clip

Pic #2: Both versions – original size and scaled up size

values for it (execute **mod.ClipWorks** and enter the values 0, 0, 25, 5). Install **mod.StretchClip** and execute it. This enlarges the letters to fill the whole screen area which you can quickly verify if you render the image again (or just click the preview window on main screen). It doesn't look too good but don't care, this is only an intermediate stage. Then you set the target clip values of 0, 0, 40, 8, install **mod.Squeeze2Clip** and execute this one. Render the resulting image, and you're done (see Pic #2, which additionally has been processed a bit after scaling the text by adding textures to both of the different sized writings – see H2G #8, step 3, in March/April 2004 issue of the Digest for directives). This way you can achieve any (card) width and height you want to have.

To scale an image leaving it undistorted (with a correct C64 screen aspect ratio of 8/5 or decimal 1.6) is another matter. GoDot's modifiers scale cardwise, not pixelwise, and we have not too many options to set values which lead to a correct ratio. ClipWorks values which divide to 1.6 are correct: width 32, height 20 (leads to a size of 80%), also 24 and 15 (60% size), 16 and 10 (40% size), and of course 8 and 5 (20% size) are applicable. To achieve exactly 50% size we use **mod.Tile**, and by repeated use we can scale down to 1/4 and even to 1/8.

The contrary of mod.Tile is **mod.Double**. You can choose one of the four quadrants of the screen (each 160 pixels wide and 100 pixels high) and scale it up to full screen. If another area on screen should be "doubled", you first have to move it to one of these quadrant positions with mod.Scroll.

It would be clever to combine two or more scaling processes to one undistorted result, and fortunately, this works. For instance scaling down to 75% (using mod.Squeeze2Clip) can be done by first scaling down to 60% and then scaling up (us-

ing mod.StretchClip) the intermediate image with the values needed for 80%.

Scaling table (values for correct C64 screen aspect ratio)

Desired Width	Related Height	Scaling Ratio	What to perform
32	20	80%	First set this clip, then do *Squeeze2Clip*
30	18 6/8	75%	(6/8 size) First 60%, then do *StretchClip* for 80%
25	15 5/8	62.5%	(5/8 size) First 50%, then do *StretchClip* for 80%
24	15	60%	First set this clip, then do *Squeeze2Clip*
20	12 4/8	50%	(4/8 size) Do *Tile*
16	10	40%	First set this clip, then do *Squeeze2Clip*
15	9 3/8	37.5%	(3/8 size) First 30%, then do *StretchClip* for 80%
12	7 4/8	30%	First 60%, then do *Tile*
10	6 2/8	25%	(2/8 size) Do *Tile* twice
8	5	20%	First set this clip, then do *Squeeze2Clip*
6	3 6/8	15%	First 60%, then do *Tile* twice
5	3 1/8	12.5%	(1/8 size) Do *Tile* three times
4	2 4/8	10%	First 20%, then do *Tile*
2	1 2/8	5%	First 20%, then do *Tile* twice

All other values aren't convenient or even impossibly to achieve.

I hope, the table explains clearly enough what this means. Here's an example where an image gets inserted into its original scaled down to a size of 3/8. It is another three step process. First load the image (here: "zone-aigles.4bt") and look in the scaling table where it reads: 3/8 is "*first 30%, then do StretchClip for 80%*". So, what have we to do for the 30%'s? Table: "*first 60%, then Tile*". Aha, what's the 60% process now: "*set clip values, then Squeeze2Clip*". Ok, so this is where we have to start from. In **mod.ClipWorks**, set clip values of 24 (width) and 15 (height), and apply **mod.Squeeze2Clip**. (If you like you can render the image after each step to see what happens to it.) Next step: 30%, just apply **mod.Tile**. Now the image is smaller than the wanted 3/8, we have to scale it up to achieve the desired result. Last step: 3/8, enter mod.ClipWorks and set values of 32 and 20 (the values for 80%), and apply **mod.StretchClip**. The resulting image now has a width of 15 cards and a height of 9 cards plus 3 pixel lines which is pretty much the exact ratio (15 divided by 9 3/8 is 1.6). Save the image for further processing (to "Undo 4Bit", recommended if you have an REU) and reload the original image.

Go to ClipWorks and enter these clip values as the target position of the scaled down image: *Wid 15*, *Hgt 9*, *Row 14*, and *Col 20*. We're abandoning the three pixel lines at the bottom of the scaled image here, but I think we can undergo this little loss. Load the "Undo 4Bit" by determining "*into Clip*" as the load target (and also "*Undo*" as the load *Source* if you used your REU).

At last, install **mod.FrameClip** (see H2G #1) and apply a frame of the color you like around the scaled down image. Render, and you're done (see Pic #3).

Pic #3: Scaled down to 3/8 of its size and re-inserted.

Command history

To scale down to 3/8 (Pic #3):

Load Replace "zone-aigles.4bt"
Display

(Inst: ClipWorks) – Execute
Wid: 24, Col: 15
Accept
Inst: Squeeze2Clip, Execute
Inst: Tile, Execute
Inst: ClipWorks, Execute
Wid: 32, Col: 20
Accept
Inst: StretchClip, Execute

Pic #4: Another example, the smaller trumpet is 75% (6/8) of the size of the bigger one, and inserted into the original image by masking.

I have uploaded the 4bit images and the new modifier (mod.Double) used in this Digest to my site (www.godot64.de), so you can easily duplicate the images shown here on your own.

Have fun using GoDot!