0. Introduction

MINIPAINT is an editor for high resolution pictures on the VIC-20.

MINIPAINT is pixel-oriented, i.e. you can freely define individual pixels. The size of the picture is fixed to 160 pixels width and 192 pixels height.

The main screen of the editor shows three windows. All edit operations are done in the first – largest – window, which shows a zoomed view into the picture. This window is overlaid by a dialog box, when you invoke the help screens or want to access storage media.

A second window shows the same view in original size. This allows you to judge the mixing of nearby colours in the analogue video signal.

The third window shows all currently available colours, the cursor co-ordinates, and other additional information. Pressing the SPACE key toggles between editor and a full-screen preview.

MINIPAINT and the stored picture files are compatible to both PAL and NTSC. The pictures can easily be used by your own programs in BASIC and machine language. For BASIC programmers, the MINIGRAFIK extension provides powerful commands for loading, saving, and manipulating pictures in this file format.

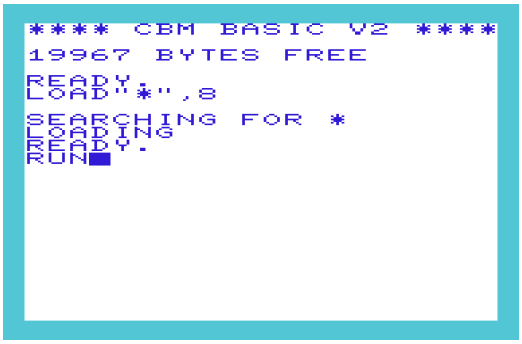
1. Starting MINIPAINT

If you are using the VICE emulator to run MINIPAINT, please look up Appendix A how to set up the correct configuration. The rest of this chapter assumes, that you are running MINIPAINT on a real VIC-20. MINIPAINT requires at least a +16K RAM expander, like the Commodore VIC-1111.

Insert the program disc into your disc drive, type:

LOAD“*”,8 and press RETURN to load the boot loader for MINIPAINT.

When the cursor reappears, type: RUN and press RETURN.



MINIPAINT shows a splash screen with a colour palette and paintbrush, while the main program is being loaded.

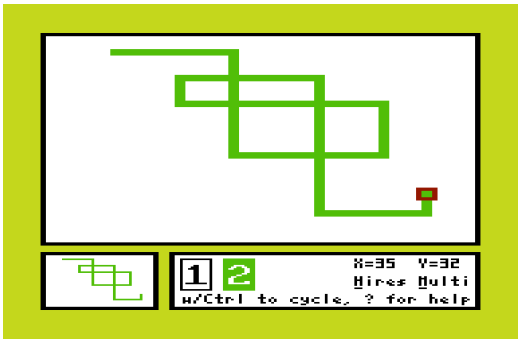
2. The Editor

MINIPAINT starts up with an empty canvas, showing a blinking cursor in the top-left corner of the screen, at the co-ordinates X=0 and Y=0. The cursor always marks where the next pixel is going to be set, and it is moved with the CRSR-keys. When the cursor reaches the screen border, the underlying picture is scrolled to put new parts of it to display.

The editor starts with the cursor in high resolution mode. The keys ‘1’ and ‘2’ define the pixel to background and foreground colour.

‘M’ switches to multi-colour mode, where two additional colours ‘3’ and ‘4’ become available: these set pixels in border and auxiliary colour, respectively. In multi-colour mode, the resolution is halved, and the cursor appears twice-wide at even x-coordinates. You switch back to high resolution mode with ‘H’.

After the pixel is set, the cursor advances in the same last direction, that was specified by moving the cursor with the CRSR-keys. This is the default setting, and it is selected by pressing either ‘f1’ or ‘f3’. While the CRSR-keys auto-repeat as usual, ‘f1’ also enables auto-repeat on the keys ‘1’ to ‘4’. This allows for easy drawing of long horizontal and vertical lines.



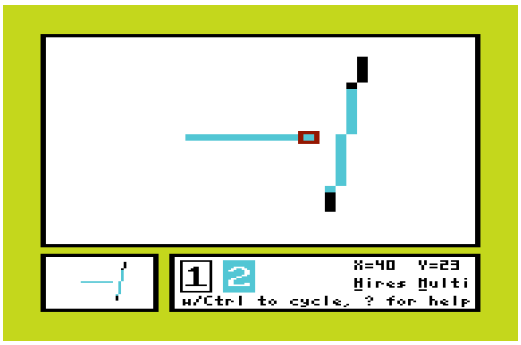
In case you want tighter control, ‘f5’ disables the automatic advancement of the cursor. Exclusive to that mode, the left arrow key ‘←’ toggles cursor draw, which draws pixels in the last used colour along with the cursor movement.

The assignment of logical colour number ‘1’ to ‘4’ to one of the 16 physical colours is changed with CTRL + the corresponding number key, in that order: Black, White, Red, Cyan, Purple, Green, Blue, and Yellow; with Orange, Light Orange, Light Red, Light Cyan, Light Purple, Light Green, Light Blue, and Light Yellow available only to background (1) and auxiliary colour (4).

The foreground colour (2) and multi-colour enable are set individually for each 8×16 pixel cell. Changes of the foreground colour affect the picture only as pixels are drawn. The background colour (1), border colour (3), and auxiliary colour (4) are global to the whole screen, and changes are immediately visible.

While foreground colour and multi-colour enable allow for a greater range of available colours in the whole picture, this can lead to undesirable effects of ‘colour clashing’: as the current pixel is set, parts of the surrounding 8×16 pixel cell might also be altered, if the attribute data must be changed. At least all other foreground pixels will also

assume the new colour, and it might lead to unexpected changes, when multi-colour enable is changed. This is due to video hardware restrictions, and cannot be avoided. MINIPAINT tries its best to incur only the least necessary changes to the attribute data, so the current pixel always is coloured as intended.



The tutorial in Chapter 4 shows one possible way to handle colour clashing. In general, some ahead planning is necessary. With MINIGRAFIK, you can prepare a picture with a grid of pixels in the bottom-right corner of each attribute cell. A small program, which does this is provided in Appendix E.

DEL (without SHIFT) cuts the current attribute cell with its attribute data into a clipboard, and then deletes the cell's contents. INST (with SHIFT) pastes the clipboard to any other cell of the picture. A copying effect can be had by immediately re-pasting the cell after the cut step. Large areas of the picture can thus easily be filled with a pre-defined pattern, or in a single colour.

Finally, HOME (without SHIFT) puts the cursor to the position X=0 and Y=0; CLR (with SHIFT) additionally erases the whole picture in memory, but keeps the current colour selection, and contents of the clipboard.

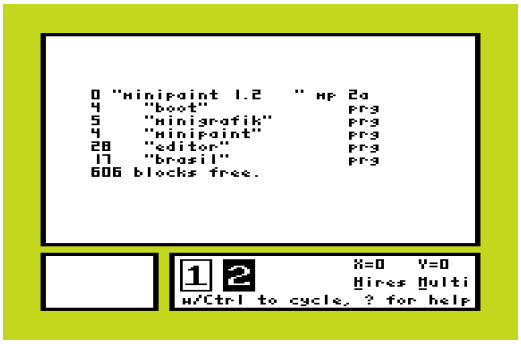
A quick help of all available editing operations is invoked by pressing '?' (SHIFT + /). SPACE toggles between editor and a full screen preview.

SHIFT + Q quits MINIPAINT.

3. Storage

As you are working on a picture, you can write it to a permanent storage medium, so you can reload it to continue editing at a later time, or use the picture within other programs.

With disc drives, the directory stored on disc is displayed with CBM-4. Holding the CBM key halts the listing, and CTRL scrolls the listing slowly. At the end, the number of free blocks is shown. MINIPAINT pictures need 17 blocks. Their file format is described in detail in Appendix D.



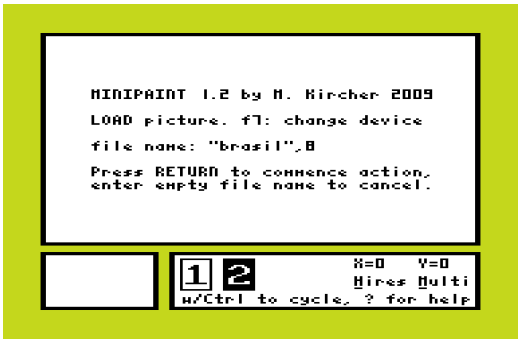
With SHIFT + S, you invoke a SAVE dialog box. Here, you enter a file name under which the picture is stored, between quotes. Behind the right quote, a number tells the current storage device, either 1 (tape), or 8..11 (disc drives on the serial bus). It defaults to the device MINIPAINT was loaded from. The function key 'f7' cycles the device number.

Pressing RETURN stores the picture to the device. On disc drives, a file with the same name is replaced, even if it is not a picture! A tape save notifies you with a border blinking in red/black to press RECORD & PLAY on the tape drive.

Not all characters are permitted within file names. In general, all lower- and upper-case letters a..z and A..Z, as well as all digits 0..9 are fine. Besides SPACE, the following 15 symbols are also allowed:

! % ' ( ) + - . / ; < > [ ] ^

The list comprises a common subset of PETSCII and 7-bit ASCII which cannot be mistaken by disc drives as commands, or parts thereof.



SHIFT + L invokes a similar dialog box for loading a picture. You specify the file name as well, but additionally '?' and '*' are allowed as wildcards. These wildcards only work as intended on disc drives. A tape load notifies you with a border blinking in green/black to press PLAY on the tape drive.

In both cases, the action is cancelled by entering an empty file name. This can be used to merely change the current device for the directory display.

4. Tutorial

When you draw a picture in MINIPAINT, at first you need to think about, which colours appear most often in the scene. For an example, consider a single apple tree against the sky, with the sun, some clouds, and a few birds.

The three most common colours are Light Blue for the sky, Green for grass and leaves, and Brown for the branches and trunk of the tree – where the darker version of Orange is the best approximation for. One of these three colours must be chosen from the lower 8 available colours, because it is assigned the border colour. In this case, there is only one choice, Green.

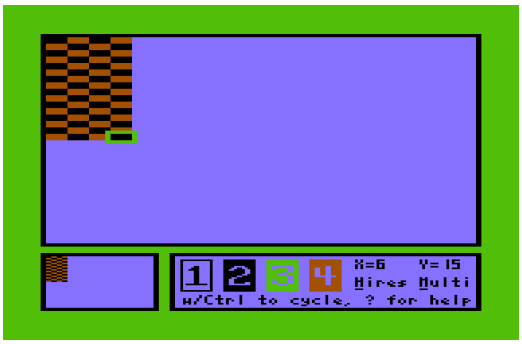
In case you do not want a visible border appearing around the scene, you might:

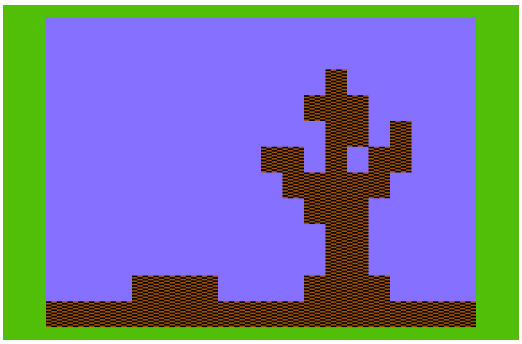
- choose background and border colour the same,
- in high resolution mode, set foreground the same as border colour, fill with foreground, and draw the scene as negative image in background colour. Even though this normally restricts the picture to two colours, the paint colour now can be any one of the 16 available colours, or
- in a similar fashion, in multi-colour mode, fill the picture with pixels in border colour beforehand.

Press 'M' to show the full palette display in the status window. Then change background (1) to Light Blue, border (3) to Green, and auxiliary colour (4) to Orange. The foreground colour (2) is used to draw everything else, for the moment you should set it to Black. Press CLR (with SHIFT) to clear the screen.

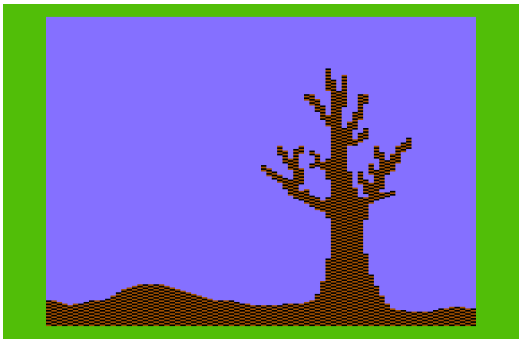
Orange is toned down to Brown for the branches and trunk of the tree. For this, mix Orange and Black in the top-left corner of the screen in a chequered pattern, up to X=6 and Y=15 –

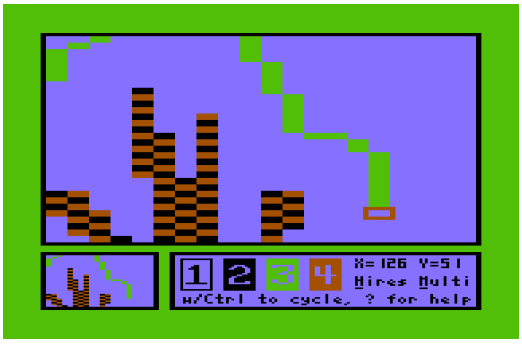
the whole top-left attribute cell. Cut the cell with the DEL key, and then sketch the bottom ground and tree with the INST key:



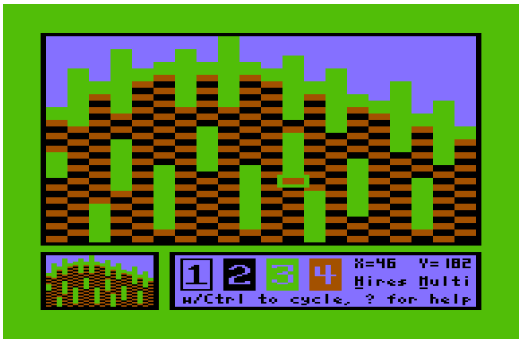


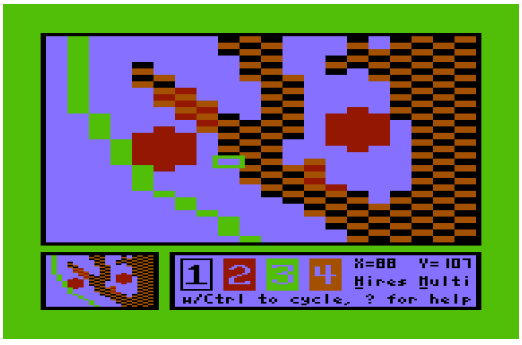
Refine the details by clearing pixels to background colour, and adding pixels in Black or Orange fitting to the pattern. Sketch the outline of the treetop in Green:



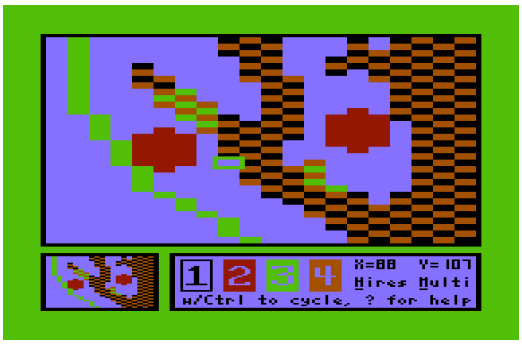


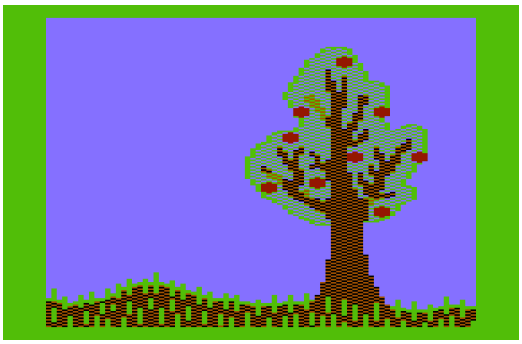
The grass at the bottom is painted in two steps: first put a two-pixel thick line in green between ground and sky. Then add blades of grass with vertical lines. To draw the apples, change the foreground colour (2) to Red. Place the apples within the treetop. While doing this, some parts of the branches might change colour from Black to Red. This is the colour clashing mentioned in chapter 2. Leave the affected parts alone for the moment.



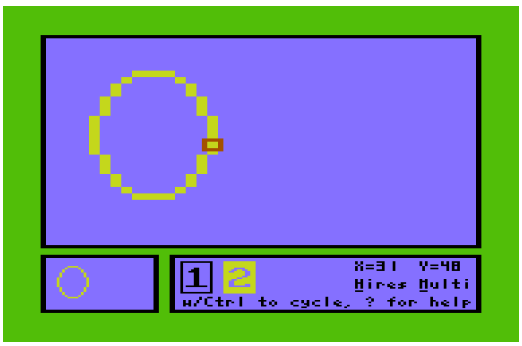


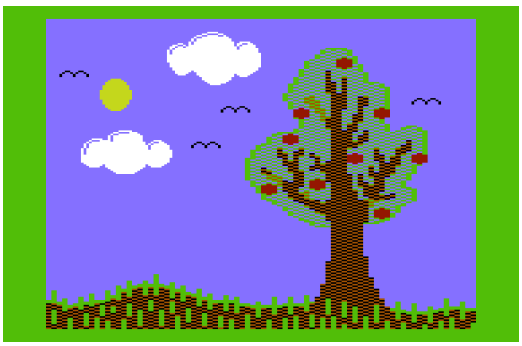
When you have placed all apples, take a look, where the branches have changed from Black to Red. There, replace the Red colour with Green. Since that colour is taken from the border colour (3) this does not incur yet another colour clash – and later will ‘hide’, that a colour clash had happened in those places. Then, within the outline of the treetop, apply another chequered pattern of Green over the background, so the Green pixels are aligned with those used to conceal the colour clashes. This step completes the tree.





To draw the sun, change the foreground colour (2) to Yellow and press ‘H’ to switch to high resolution mode. Place the sun in the top-left quarter of the picture. Draw the outline first, then fill the interior. Then, add some clouds. Change the foreground colour to White, and place the clouds in good distance to both sun and tree to avoid colour clashes. Add some structure along the top-left facing part of the clouds by clearing pixels to background colour. For the finishing touch, change the foreground colour back to Black again, and add some birds – the picture below to the right-hand side shows the result:





5. Credits

Shane Fell and Vanja Utne for inspiration and beta-testing,
Andrew Jacobs for his 16-bit BCD conversion routine, and
Wolfgang Wirth for the original MINIGRAFIK.

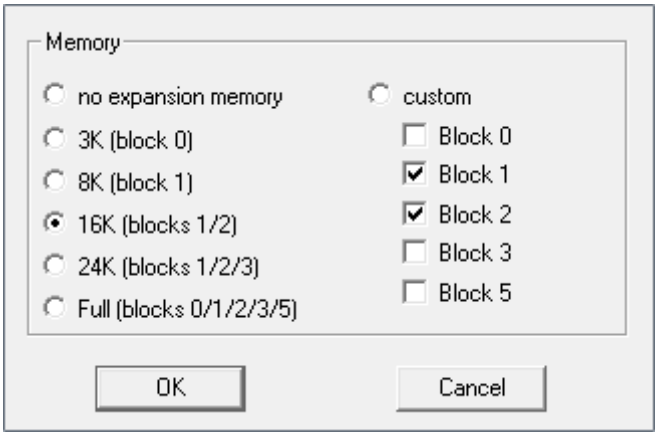
6. Licence

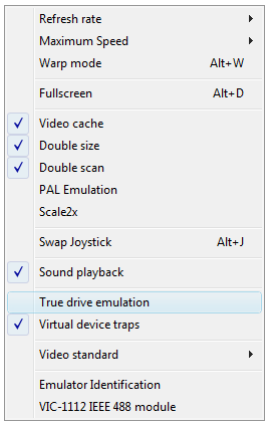
MINIPAINT and MINIGRAFIK are freeware. As long as I am credited, you are permitted to use MINIGRAFIK as support library for your own programs, regardless whether these programs are of commercial or non-commercial nature.

A. Using MINIPAINT within VICE

You need at least version 1.16 of VICE.

For the memory extension of 16K, select ‘Settings > VIC settings...’ in the menu of xvic. Choose the Memory option ‘16K (blocks 1/2)’, and press ‘OK’.





MINIPAINT does not require exact emulation of the 1541 disc drive, and will access disc images much faster without it. Tick off ‘True drive emulation’ in the Options menu.

To start MINIPAINT, select ‘Autostart disk/tape image...’ in the File menu. In the file requester, search for the file “minipaint.d64”, and press ‘Attach’.

B. MINIGRAFIK extension for CBM BASIC

MINIPAINT lets you to edit pictures in an interactive process. For automated picture processing, it is possible to script MINIPAINT. This is done with the supplied MINIGRAFIK extension for CBM BASIC. MINIPAINT itself consists of a part written in BASIC, which interfaces between MINIGRAFIK, responsible for providing the bitmap mode, and media access – and EDITOR, a machine language library designed to edit the bitmap in a zoomed view.

The extension is included on disc. From the VIC start-up screen, type:

LOAD“MINIGRAFIK”,8 + RETURN, and RUN + RETURN

You are presented again with the start-up message of CBM BASIC, but now there is slightly less memory available, that has been allocated for the bitmap, and the extension itself. MINIGRAFIK adds 6 custom commands and 1 function to the CBM BASIC interpreter. They are all prefaced with an ‘@’ character.

You should use the additional commands in program mode only, since ordinary text output (such as the READY prompt) interferes with the hires screen. Immediately after the THEN of an IF-clause, the commands of MINIGRAFIK must be prefaced with a colon ':'. Otherwise the running program is stopped with a '?SYNTAX ERROR'.

The new commands are: **@ON**, **@CLR**, **@RETURN**, **@SAVE**, **@LOAD**; '**@**' on its own specifies a pixel plot or line drawing command. The **@()** function takes two arguments.

Symbol explanations:

<...> denotes parameter expressions
[...] denotes optional parts of a parameter list

Commands:

@ON initialises the 160×192 pixel graphics. The screen is correctly recentered, regardless of MINIGRAFIK running on a NTSC or PAL VIC.

@CLR clears the hires screen. Furthermore the Colour RAM is initialised to the foreground colour.

@RETURN returns to text mode. If an error occurs during program execution, this command is executed on your behalf before outputting the error message.

@<colour>,<x>,<y> [TO <x2>,<y2>] – draw line in logical <colour> from (<x>,<y>) to (<x2>,<y2>). If the TO part is omitted, a single pixel at (<x>,<y>) is plotted.

<x> and <x2> may range from 0 to 159, <y> and <y2> from 0 to 191. The origin (x=0, y=0) is located in the top-left corner. All arguments can be written as numbers, variables, or arbitrary numeric expressions.

In high resolution mode (M=0) the allowed logical colours are:

- 0: clear pixel to background colour G
- 1: set pixel in foreground colour F

In multi-colour mode (M=1) two extra logical colours become available:

- 2: set pixel in border colour B
- 3: set pixel in auxiliary colour A

In multi-colour mode the resolution is halved, so that when <x> is even, <x> and <x>+1 address the same pixel.

The foreground colour (F=0..7) and multi-colour enable (M=0..1) are set individually for each 8×16 pixel cell. The background colour (G=0..15), border colour (B=0..7), and auxiliary colour (A=0..15) apply to the whole screen.

To assign a physical colour to the logical colours, issue the following POKEs:

POKE 36879,16*G+8+B (set background and border colour)
POKE 36878,16*A (set auxiliary colour)
POKE 646,8*M+F (set foreground colour and enable/disable MC mode)

Physical Colour Table:

0	Black	8	Orange
1	White	9	Light Orange
2	Red	10	Light Red
3	Cyan	11	Light Cyan
4	Purple	12	Light Purple
5	Green	13	Light Green
6	Blue	14	Light Blue
7	Yellow	15	Light Yellow

@SAVE [<filename> [,<device>]]

Save screen to device. A device-number greater than or equal 8 addresses the disc drive. In that case a filename must be given.

If the device-number is omitted, or equal 1, the screen is saved to tape. @SAVE without any arguments saves an unnamed screen to tape. A tape save notifies the user with a border flashing black/red to press RECORD & PLAY on the tape drive. When the save has completed, your program continues execution.

A description of the file format is given in Appendix D.

In case MINIGRAFIK is not active, or was not made available to the recipient of the image, the image still can be displayed by LOADing and RUNning the file. A BASIC SYS statement is provided at the begin of the file, which, when loaded to the normal BASIC start at $1201 (≥8K RAM expansion) calls a stand-alone display routine. The routine waits for a key press, and then resets the VIC.

The mechanism involved here does not work, when MINIGRAFIK already is loaded. In that case, use @LOAD instead, inside a program.

@LOAD [<filename> [,<device>]]

Load screen from device. A device-number greater than or equal 8 addresses the disc drive. In that case a filename must be given.

If the device-number is omitted, or equal 1, the screen is loaded from tape. @LOAD without any arguments loads the next screen from tape. A tape load notifies the user with a border flashing black/green to press PLAY on the tape drive.

When the load has completed, the image is displayed on screen, and your program continues execution after the @LOAD command. There is no wait for a key press.

DO NOT load anything with @LOAD, that has not been written by MINIPAINT or the @SAVE command!

Functions:

@(<x>,<y>)

This function returns the logical colour at (<x>,<y>). <x> may range from 0 to 159, <y> from 0 to 191. Both arguments can be written as numbers, variables, or arbitrary numeric expressions.

For a high resolution 8×16 pixel cell, the function returns 0 for background, and 1 for foreground colour. If the cell is multi-colour and <x> is even, then <x> and <x>+1 share the same logical colour, and you must be prepared for the function to return values from 0 to 3.

C. MINIGRAFIK memory map

This is mainly of interest to those who want to access the bitmap from machine language – memory is organised as follows, while MINIGRAFIK is active, and a bitmap is being displayed:

```
  $1000 .. $10EF:   "text" screen, address generator for VIC chip
  $10F0:            unused
* $10F1 .. $10FD:   "2008 SYS 8584" BASIC stub
* $10FE:            VIC register 36878 value, with bits 0..3 cleared
* $10FF:            VIC register 36879 value
  $1100 .. $1FFF:   160×192 pixel bitmap
* $2000 .. $2077:   compressed copy of colour RAM
  $2078 .. $246A:   MINIGRAFIK code
  $246B ..:         BASIC program storage

  and

  $9400 .. $94EF:   colour RAM
```

The areas flagged with '*' are in temporary use only, during an @SAVE or @LOAD operation. In text mode, the screen is located at 4096, and colour RAM at 37888, unchanged from the normal setup when an expansion of 8K or more is present. The following code is used to setup address generator and VIC registers for bitmap display:

```
.On
 CLC
 LDA #$10       ; first char is 'P', accessing $1100
 TAY
.On_00
 STA $0FF0,Y
 ADC #$0C
 BCC On_01
 SBC #$EF
.On_01
 INY
 BNE On_00
```

```
 LDY #$05       ; program VIC registers
.On_02
 CLC
 LDA $EDE4,Y
 ADC Offset,Y
 STA $9000,Y
 DEY
 BPL On_02
 RTS

.Offset
 EQUB $02
 EQUB $FE
 EQUB $FE
 EQUB $EB
 EQUB $00
 EQUB $0C
```

The address of a pixel is calculated thus: AD=4352+192*INT(X/8)+Y, and the definitions of SHIFT + M and reverse SHIFT + M in the character ROM at $8268 and $8668 provide convenient bitmasks for setting or clearing single pixels.

D. MINIPAINT MG picture file format

A MG picture format file is defined as follows:

- **CBM DOS file type**: **PRG**
- **length**: exactly 4097 bytes, including load address, comprised of:
- **load address**: **2 bytes**, $10F1, written in little-endian.
- **BASIC stub**: **13 bytes**, a BASIC program with one line: '2008 SYS 8584'. This is supposed to call the display routine, when the file is LOADed to the normal BASIC start at $1201, and then RUN.
- **VIC register values**: **2 bytes**, contents of $900E and $900F, stored in that order. The lower nibble of $900E, containing the volume, is stored as zero. MINIPAINT forces bit 3 of $900F (‘inverse video’) to 1 on load, i.e. normal, non-inverse video.
- **bitmap**: **3840 bytes**, contents of addresses $1100 to $1FFF, stored in column-major order: The first 192 bytes define the first 8 hires pixel columns. Within each byte, the left-most pixel is in the most significant bit.
- **compressed colour RAM**: **120 bytes**, compressed contents of addresses $9400 to $94EF in row-major order. The lower nibbles of two consecutive bytes in the colour RAM are combined into one byte. Of the byte pair, that one with the lower address (left) is stored in the lower nibble.
- **display routine**: **120 bytes**, saved from ‘addresses’ $2078 .. $20EF, but assumed to be run at address $2188 when the file is loaded to the normal BASIC start at $1201. Reconstructs the address generating text screen, reprograms all necessary VIC registers, and restores the bitmap and colour RAM at the correct place. After a key press, the VIC-20 then resets.

The contents of load address, BASIC stub, and display routine are assumed to remain **fixed**!

E. Example scripts using MINIGRAFIK

This small utility writes a picture file to disc, which has pixels set at the bottom-right corner of each attribute cell:

```
10 @ON:@CLR
11 FORY=1TO12:FORX=1TO20
12 @1,8*X-1,16*Y-1
13 NEXT:NEXT
14 @SAVE"GRID",8
15 @RETURN
```

The following program creates a boot loader, which first starts up MINIGRAFIK, and then a client program. The resulting file 'BOOT' should be placed first in the directory of the disc, so the main application can easily be run with 'LOAD"*",8/RUN' or autostarted in VICE. The year input appears as line number of a SYS statement, when the boot loader is LISTed.

```
10 DATA 1,18,12,18,-1,-2,158,32,52,54,50,50,0,0,0,169,1,166,186,160,0,32,186,255,169,10
11 DATA 162,88,160,18,32,189,255,169,0,162,61,160,32,32,213,255,134,43,132,44,32,240,32
12 DATA 169,1,166,186,160,0,32,186,255,169,-3,162,98,160,18,32,189,255,169,0,166,43,164
13 DATA 44,32,213,255,134,45,132,46,32,51,197,32,89,198,76,174,199,77,73,78,73,71,82,65
14 DATA 70,73,75:INPUT"NAME";N$:INPUT"YEAR";Y:DN=PEEK(186):S(3)=LEN(N$):S(2)=INT(Y/256)
15 S(1)=Y-256*S(2):OPEN2,DN,2,"BOOT,P,W":FORT=1TO99:READA:IFA<0THENA=S(-A)
16 PRINT#2,CHR$(A);:NEXT:PRINT#2,N$;:CLOSE2
```

Finally, here's a program to import *.pgm files as line-art:

```
10 DIMC%(1)
11 :
12 INPUT"PGM FILE";PF$
13 INPUT"TARGET";TF$
14 INPUT"FOREGROUND{2 SPACE}1{3 LEFT}";F
15 INPUT"BACKGROUND{2 SPACE}0{3 LEFT}";G
16 INPUT"BORDER{2 SPACE}0{3 LEFT}";B
17 INPUT"INVERT INPUT{2 SPACE}N{3 LEFT}";YN$:C%(1+(YN$="Y"))=1
18 :
19 LF$=CHR$(10):NU$=CHR$(0):OPEN2,8,2,PF$+",S,R"
20 GOSUB32:PRINTLI$:IFLI$<>"P5"THENCLOSE2:STOP
21 GOSUB32:PRINTLI$:IFLI$<>"160 192"THENCLOSE2:STOP
22 GOSUB32:PRINTLI$:IFLI$<>"255"THENCLOSE2:STOP
23 PRINT"HEADER OK. PRESS KEY."
24 GETA$:IFA$=""THEN24
25 POKE36879,16*G+8+B:POKE36878,16*5:POKE646,F:@ON:@CLR
26 FORY=0TO191:FORX=0TO159
27 GET#2,A$:@C%(-(ASC(A$+NU$)>=128)),X,Y
28 NEXT:NEXT:CLOSE2
29 GETA$:IFA$=""THEN29
30 @SAVETF$,8:@RETURN:END
31 :
32 LI$=""
33 GET#2,A$:IFA$<>LF$THENLI$=LI$+A$:GOTO33
34 IFLEFT$(LI$,1)="#"THEN32:REM IGNORE COMMENT LINES
35 RETURN
36 :
37 REM ** PGM IMPORT MONO, MG VERSION 2010-02-21 BY MICHAEL KIRCHER
```