

PPM Menu

The program `ppm_menu` can be used to edit and create images. The program keeps track of 3 `PPM` objects: input image 1, input image 2, and output image.

Quitting

The `quit` command terminates the program.

Reading and writing images

The commands `read1` and `read2` read a `PPM` file into the input images. The command `write` writes to a `PPM` file from the output image. `draw-ascii` is an additional output mode, displaying the output image to the terminal in a simple ASCII art format. The `copy` command allows us to copy from input image 1 into the output image.

Image editing

We also provide some primitive image editing commands. These commands generally work on the input image 1. The `size` command sets the image dimensions, and the `max-color-value` command assigns the maximum allowable color value for any color channel.

The `channel` and `pixel` commands allow the user to change the color of individual pixels.

Drawing to multiple pixels

The `clear` command sets all pixels to black. The `circle` and `box` commands allow the user to set a color for a group of pixels in a circular or rectangular shape.

Image filters

Image filters are commands that work on a group of pixels at the same time, in the same way. Some filters use only input image 1, while others use both input images. Some filters modify input image 1, and other filters modify the output image.

Filters that modify input image 1 using only input image 1

`*=` and `/=` modify the pixels of input image 1 using the pixels of input image 1 and a user specified number, with the mathematical operation specified in the command name on a channel by channel basis.

Filters that modify input image 1 using both input images

`+=` and `-=` modify the pixels of input image 1 using the pixels of input image 1 and input image 2, with the mathematical operation specified in the command name on a channel by channel basis.

Filters that modify the output image using only input image 1

`*` and `/` modify the pixels of the output image using the pixels of input image 1 and a user specified number, with the mathematical operation specified in the command name on a channel by channel basis.

`red-gray`, `green-gray`, `blue-gray`, and `linear-gray` modify the pixels of the output image using the pixels from input image 1. The output image only contains grayscale colors.

Filters that modify output image using both input images

`+` and `-` modify the pixels of the output image using the pixels of input image 1 and input image 2, with the mathematical operation specified in the command name on a channel by channel basis.

Image Creation via Paint by Numbers

In the paint by numbers mode, the user specifies the desired size of the image and a color pallet as a numbered table of colors. The user then chooses the color for each pixel in the image by putting the number of a color from the color table in a grid. Finally, the numbers in the grid are used to lookup colors in the

table, and those colors are used to assign to the output image.

The paint by numbers process:

- Select the desired pattern.
- Configure the number grid's size and maximum number.
- Store numbers in the number grid according to the desired geometric pattern.
- Configure the color table to contain the desired colors.
- Create the output image using the number grid and color table.
- Write the output image to a file.

Selecting the pattern

`complex-fractal`, `julia`, and `mandelbrot` allow the user to select the form. Some versions of the program may have additional patterns based on student choices during assignments, personal interest, or exam work.

Number Grid Configuration

The `grid` command allows the user to choose the size of the number grid, which is the ultimate size of the output image. It also sets the maximum numeric value that can be stored in a grid location.

Number Grid Manual Editing

`grid-set` allows the user to manually configure the number in a grid location.

Number Grid Computational Editing

Algorithms may be used to insert numbers into the number grid according to a mathematical pattern.

The process for computational editing is:

- Configure the pattern.
- Compute and store numbers in the grid.

Configuring the pattern

Most patterns are mappings between computations using real numbers in the number plane and the discrete row and column numbers of a grid location. `fractal-plane-size` allows the user to configure the region of the number plane that is mapped to the number grid rows and columns.

Some pattern forms have additional configuration options. `julia-parameters` allows the user to configure numeric parameters that control the Julia set pattern, if it is the currently selected choice.

Computing and storing numbers

`fractal-calculate` and `fractal-calculate-single-thread` are used to compute the grid numbers by the selected and configured pattern, and store the numbers in the grid.

Defining a Color Table

There is a built-in color table that the user can choose to use. But, there are additional commands to allow for a user created color table.

`set-color-table-size` sets the number of colors in the color table. `set-color` allows the user to configure the red/green/blue values for one color in the color table. `set-random-color` assigns a random set of red/green/blue values to the user selected color in the color table.

`set-color-gradient` sets a range of colors in the color table. The user selects the first and last color in the range, and the program linearly extrapolates between these two colors to fill in the spaces in the color table between them.

Translating Numbers and Colors to an Image

This is the last step in the paint by numbers image creation mode. It is assumed that the color table has been configured correctly, and the number grid has been configured correctly before this operation.

`grid-apply` uses the numbers in the grid and the colors in the built-in color table to assign pixel colors to the

output image. `grid-apply-color-table` uses the numbers in the grid and the colors in the color table to assign pixel colors to the output image.

Command Files

The user can create command files that with a sequence of commands to executed, then feed these commands to the program. This is convenient for creating an image by iterative updates to the commands, without having to retype all of the commands every iteration.

For example, if the commands are stored in a file named `pretty_picture.txt`, the commands can be used like this:

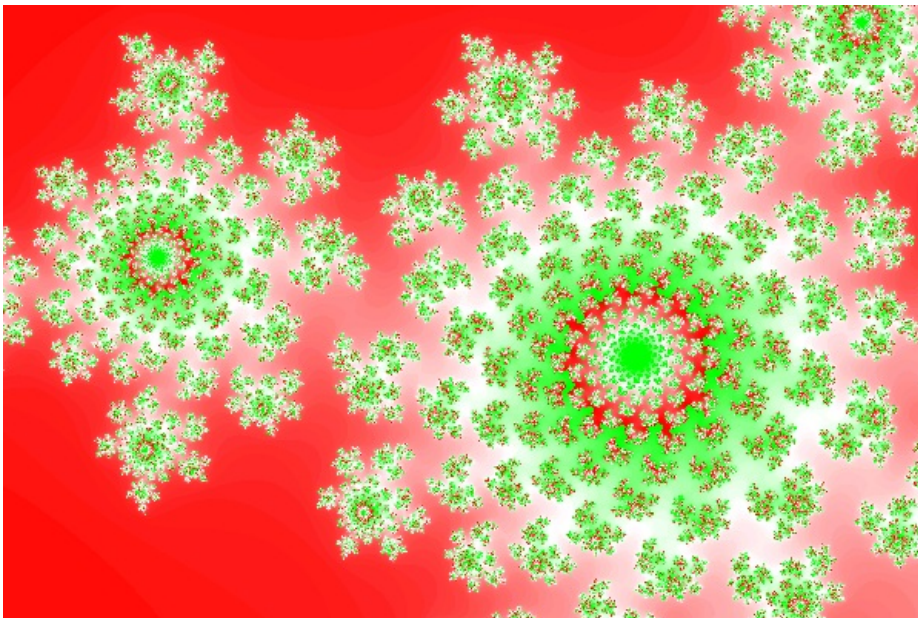
```
./ppm_menu < pretty_picture.txt
```

Sample `pretty_picture.txt`

```
# select the desired pattern
julia
# configure the grid size and maximum number
grid 400 600 200
# configure the pattern
fractal-plane-size -1.2 0.0 0.0 0.8
julia-parameters -0.4 0.6
# compute and store the numbers
fractal-calculate
# <<<<<<<< Start Color Table >>>>>>>>>
set-color-table-size 101
set-color-gradient 0 255 0 0 50 255 255 255
set-color-gradient 50 255 255 255 100 0 255 0
# <<<<<<<< End Color Table >>>>>>>>>
# create output image
grid-apply-color-table
# save image to file
write pretty_picture.ppm
# done
quit
```

Notice the `#` command is used to add comments to the command file that don't affect the picture created, but give the human user information about the commands.

[Download pretty_picture.txt](#)



[Download pretty_picture.ppm](#)

Table of Commands

Function Name	Description
---------------	-------------

Command Name		
quit	quit	"Quit."
#	commentLine	"Comment to end of line."
Read/Write		
read1	readUserImage1	"Read file into input image 1."
read2	readUserImage2	"Read file into input image 2."
write	writeUserImage	"Write output image to file."
copy	copyImage	"Copy input image 1 to output image."
draw-ascii	drawAsciiImage	"Write output image to terminal as ASCII art."
Editing		
size	setSize	"Set the size of input image 1."
max-color-value	setMaxColorValue	"Set the max color value of input image 1."
channel	setChannel	"Set a channel value in input image 1."
pixel	setPixel	"Set a pixel's 3 values in input image 1."
clear	clearAll	"Set all pixels to 0,0,0 in input image 1."
circle	drawCircle	Draw a circle shape in input image 1.
box	drawBox	Draw a box shape in input image 1.
Filters		
"+="	plusEquals	"Set input image 1 by adding in input image 2."
"-="	minusEquals	"Set input image 1 by subtracting input image 2."
"*="	timesEquals	"Set input image 1 by multiplying by a number."
"/="	divideEquals	"Set input image 1 by dividing by a number."
"+"	plus	"Set output image from sum of input image 1 and input image 2."
"-"	minus	"Set output image from difference of input image 1 and input image 2."
"*"	times	"Set output image from input image 1 multiplied by a number."
"/"	divide	"Set output image from input image 1 divided by a number."
red-gray	grayFromRed	Set output image by grayscale from red on input image 1.
green-gray	grayFromGreen	Set output image by grayscale from green on input image 1.
blue-gray	grayFromBlue	Set output image by grayscale from blue on input image 1.
linear-gray	grayFromLinearColorimetric	Set output image by linear colorimetric grayscale on input image 1.
Paint by Numbers		
grid	configureGrid	Configure the grid.
grid-set	setGrid	Set a single value in the grid.
grid-apply	applyGrid	Use the grid values to set colors in the output image.
grid-apply-color-table	applyGridColorTable	Use the grid values to set colors in the output image using the color table.
fractal-plane-size	setFractalPlaneSize	Set the dimensions of the grid in the complex plane.
julia-parameters	setJuliaParameters	Set the parameters of the Julia Set function.
complex-fractal	setComplexFractal	Choose to make a complex plane.
julia	setJuliaFractal	Choose to make a Julia set.
mandelbrot	setMandelbrotFractal	Choose to make a Mandelbrot set.
set-color-table-size	setColorTableSize	Change the number of slots in the color table.
set-color	setColor	Set the RGB values for one slot in the color table.
set-random-color	setRandomColor	Randomly set the RGB values for one slot in the color table.
set-color-gradient	setColorGradient	Smoothly set the RGB values for a range of slots in the color table.

fractal-calculate	calculateFractal	Calculate the escape values for the fractal.
fractal-calculate-single-thread	calculateFractalSingleThread	Calculate the escape values for the fractal, single-thread.