ThinkingSketch Cookbook
Version 2.01E

ThinkingSketch Unit
www.sketch.jp

Oct. 4. 2002

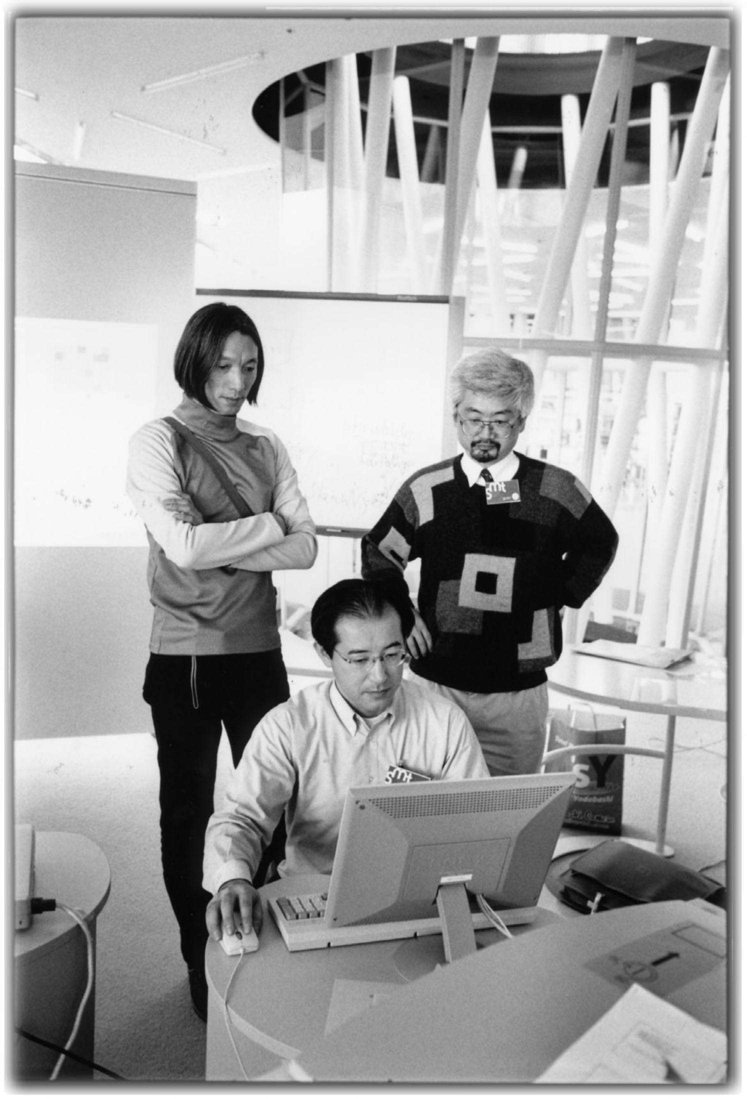


Figure 1: ThinkingSketch Unit Members

Introduction

ThinkingSketch is a software for drawing. One of the most charactristic aspect of this software is the automatic creation of pictures.

ThinkingSketch does not simply create many figures randomly and quickly, but enables user control generation process and make the output figures closer to what the user wants to be.

This "User's Guide" is written for the beginner of ThinkingSketch and introduce one of the fundamental function of ThinkingSketch.

Now, let us introduce you to the world of ThinikingSketch.

ThinkingSketch Unit

Chapter 1

Preparation

As ThinkingSketch is written in language Java, you need to enable your computer to run Java program. The easiest way to have runtime environment is to install JRE(Java Runtime Environment). [1] JDK is another choice for you. [2]

If your computing environment do not have Java runtime environment, please download the latest JRE or JDK from the website `java.sun.com` and install it.

As you need about 10M bytes to install Java environment in case you choose the smallest one, please be reminded that you need some amount of storage area.

Please confirm that you can run any application that is written in Java and then download ThinkingSketch form `http://www.sketch.jp/`.

Then follow the instraction in this cookbook and execute ThinkingSketch. [3]

[1]You can not make Java program using only JRE.

[2]You need more resources for this environment. This enables you to create Java program.

[3]Command `run` or `ts` will start ThinkingSketch execution.

Chapter 2

Let's try

2.1 The first step

Following the instruction of the software package, after starting up the ThinkingSketch, a window appears on the full screen area on your computer.

You can find a rectangular area at the bottom of the window. From here you can input text.

Enter command strings and hit enter key then immediately the command will be executed.

Figure 2.1: Initial Screen

2.2 Imitating Mondrian

Do you know the name of a Dutch panter, Piet Mondrian (1872-1944)?

As the first step of the ThinkingSketch, lets grenerate abstract patterns that imitates Mondrian.

Move your pointer to the rectangular area under the window [1] and click, then input the following text.

```
mondrian
```

then hit enter key. You will find a pattern that is consist of multiple rectangles with frame.

[1] We call this area as 'input area.'

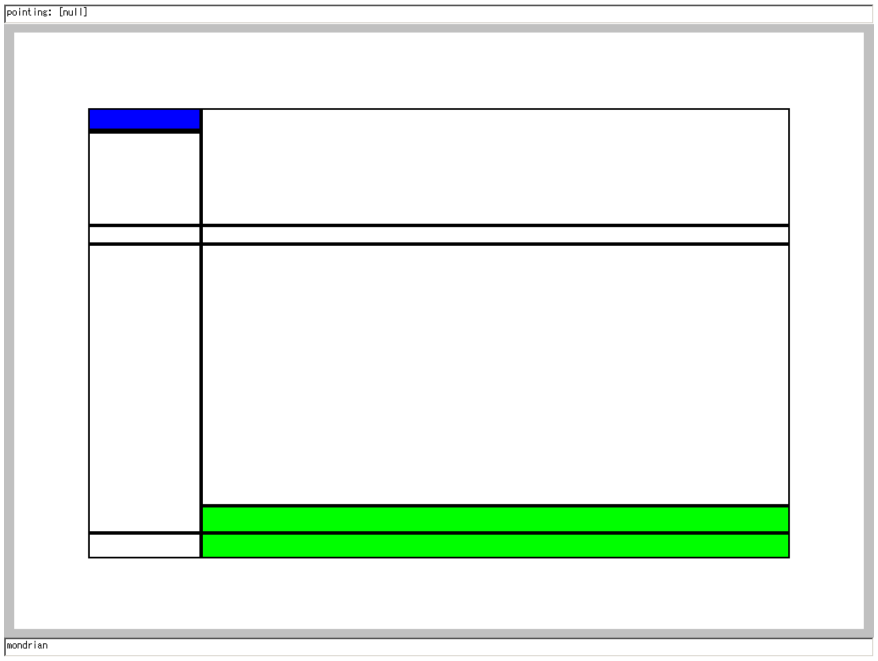


Figure 2.2: A figure which imtate Mondrian

2.3 'mondrian' command

If you get the first fugure, you can hit enter key more few times. (Keyboard focus must be on the input area.) [2]

Do you recognize the change of the figure on the screen?

Here the command 'mondrian' is executed and red, blue, and/or green rectangular will be added. [3]

We have another command 'mond' whose name is shorter. [4] When we use 'mond' command, mesh pattern will appear with rectangular. ThinkingSketch choose two of the vertical lines and horizontal lines to specify a rectangle then colors the rectangle.

[2]ThinkingSketch has a history (retrieve) function. If you are going to input the same command as you entered before, you can hit upward-arrow key or downword-arrow key to retrieve and reuse the command.

[3]The 'mondrian' command is a command that is extended as with plug-in function. We know the painter, Piet Mondrian, did not use green so much, however, as this function is for just for the trial use we currently do not care about the use of color.

[4]Command 'mond' is created to explain command 'mondrian'.

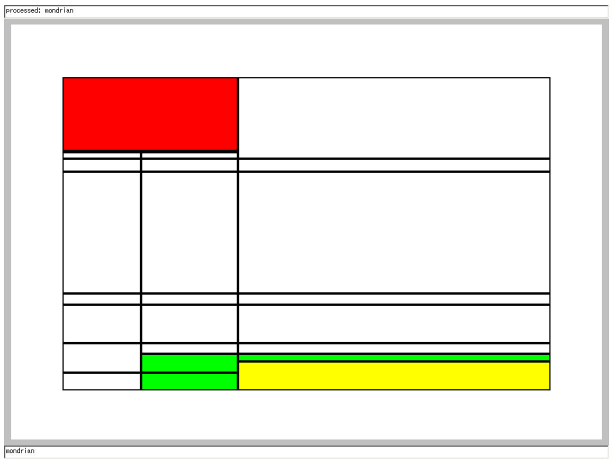


Figure 2.3: Add more

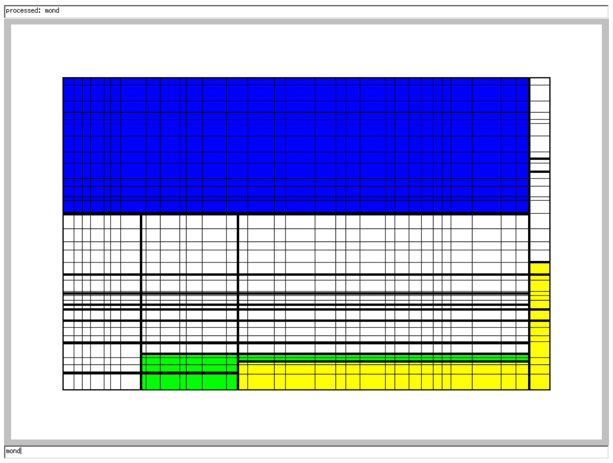


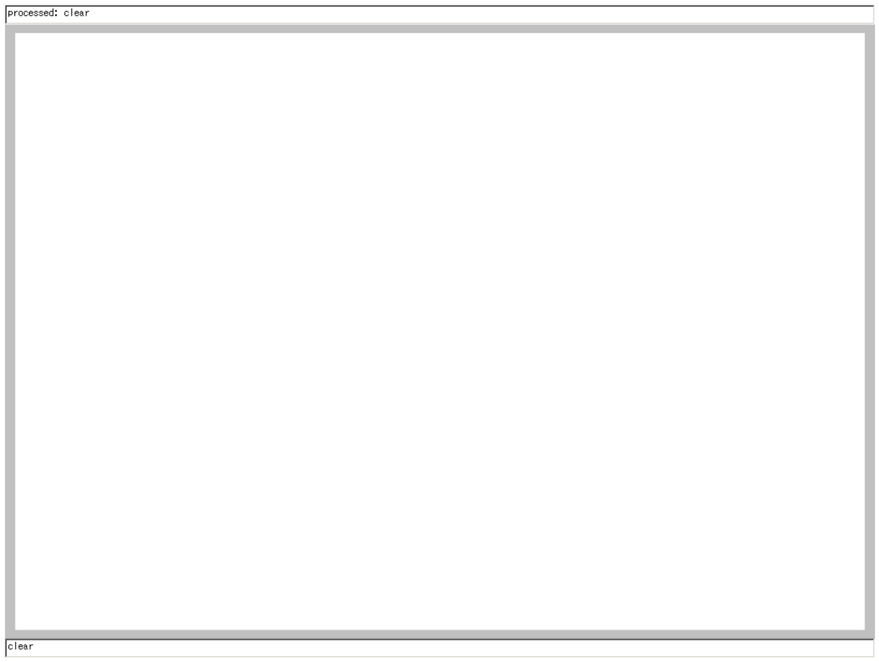
Figure 2.4: Show also supplemental lines

2.4 Clear Screen

Let's clean up the screen. Input

```
clear
```

command using keyboard. Screen will be reset to the initial status.



```
processed: clear
```

```
clear
```

Figure 2.5: After clearing up

2.5 Import Graphig Parts

On ThinkingSketch, we can store the graphical parts of drawing as indivisual graphic objects.

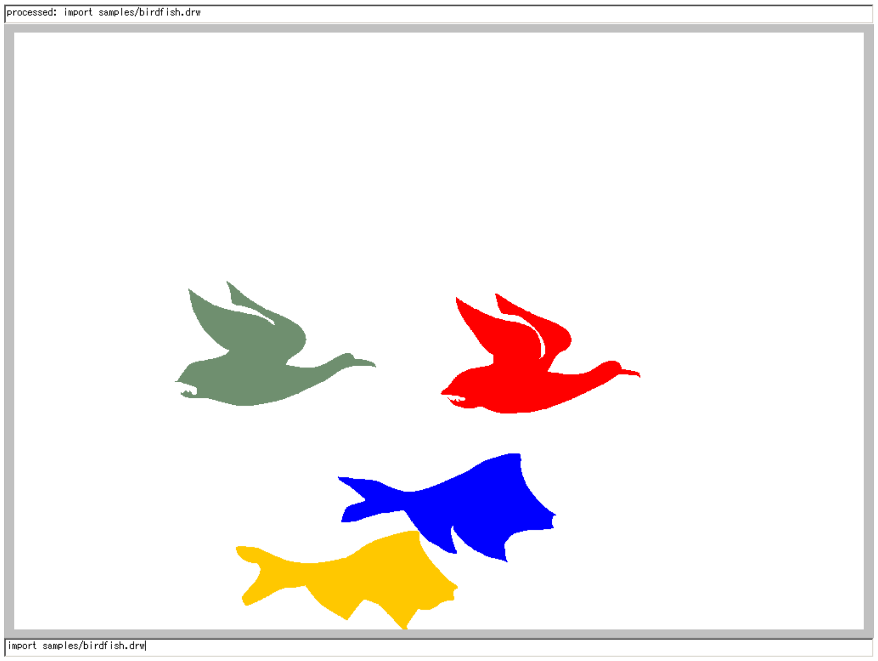
Specify command input area as

```
import samples/birdfish.drw
```

and execute it.

With this opearation, graphic parts that are already made and are stored before will be inport to ThinkingSketch. [5]

[5] If the specified file do not have ThinkingSketch readable format, ThinkpingSketch will just ignore it.


```
processed: import samples/birdfish.drw
```

```
import samples/birdfish.drw
```

Figure 2.6: samples/birdfish.drw is imported

2.6 Storing in 'parts box'

Before this operation we need to have one or more objects on the screen. After importing objects from file, you can see objects on the screen.

Execute

```
storage
```

command. Then graphic objects will disappear from the screen.

With this operation, graphic objects on the screen will be stored in 'parts' as individual parts. (If some objects are already stored they will appear to the front screen.)

pointing: [null]

storage

Figure 2.7: Move parts to the 'parts box'

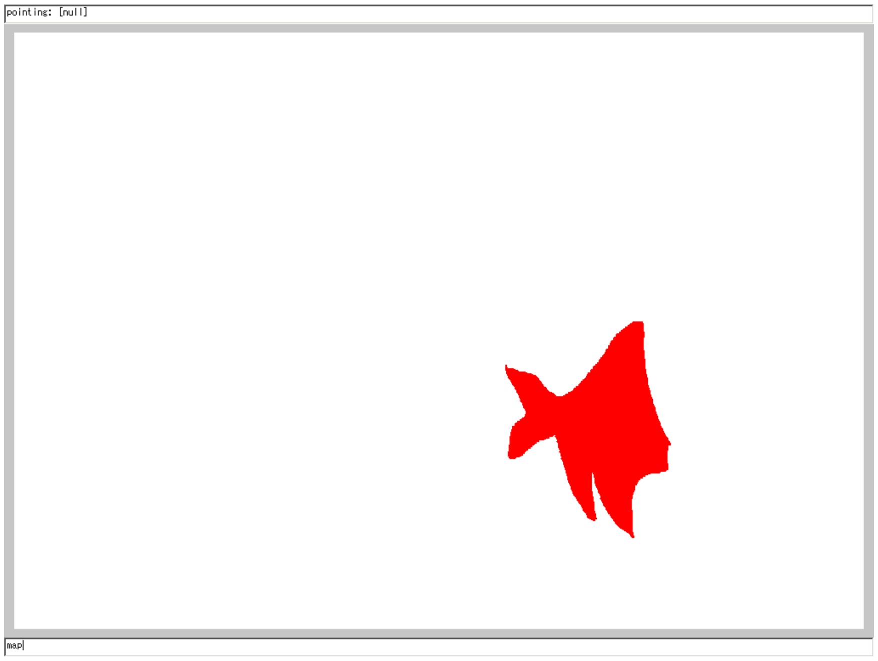
2.7 Use of command '*stella'

We can automatically place the graphic objects on the screen. Let's execute

```
*stella
```

command.

One object, we had stored in the 'parts box' will be appeared on the screen. '*stella' command choose one command from the parts box and change the size and position to fit into a rectangluar area that will be specified with randomly selected two vertical lines and two horizontal lines.


pointing: [null]

map

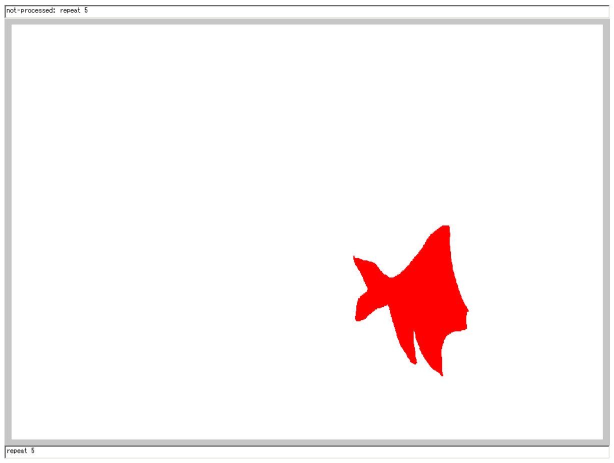
Figure 2.8: Move a part from parts box using '*stella' command

2.8 Copy Multiple Graphic Parts from the Parts Box

Those commands, whose name starts from `*` , can be used to move multiple objects with one operation. In order to specify the number ob object to be moved is specified with the 'repeat' command with an argument. For example, if we execute the following command,

```
repeat 5
```

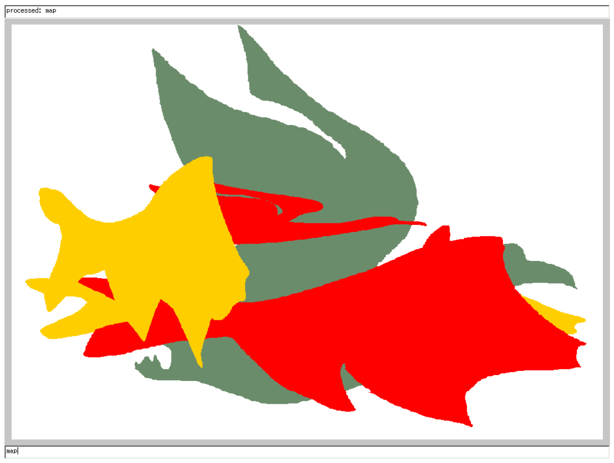
five objects will be generated on exch execution of '*stella' command.



```
not-processed: repeat 5
```

```
repeat 5
```

Figure 2.9: set repeat parameter



```
processed: map
```

```
map
```

Figure 2.10: execute *stella command again

2.9 Use of other commands '*pollock', etc.

We can place the graphic object as parts using other commands. Those commands are as follows.

```
*stella
*pollock
*imai
*mattise
```

Please try, these commands one by one with 'clear' command. There are clear differnce between the effects of each command.

The command we introduced first '*stella' changed the height and width of objects. In case of command '*pollock', it does not change the size but only redefine the position to be placed based on the roundomly generated patterns.

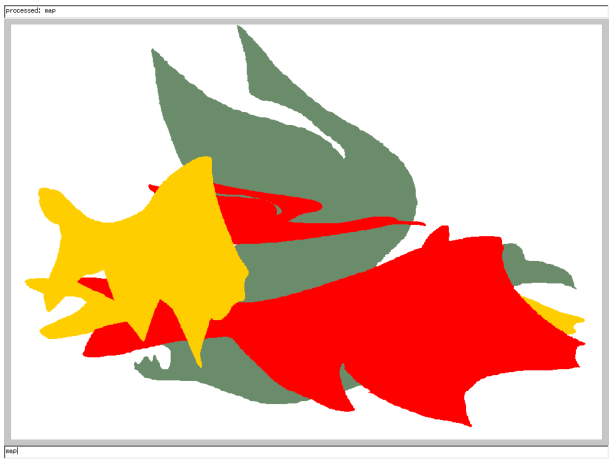


Figure 2.11: Example of *stella command execution

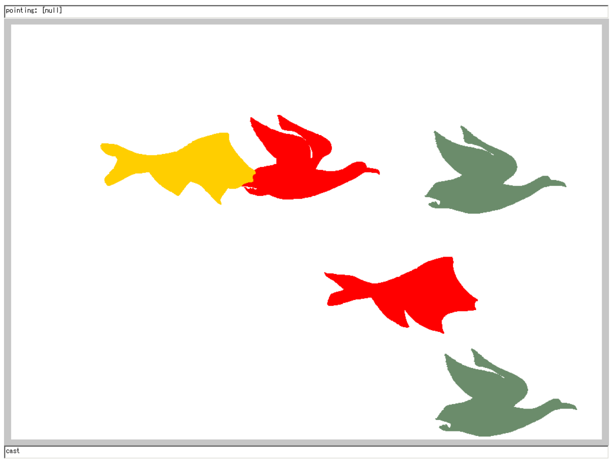


Figure 2.12: Example of *pollock command execution

2.10 Use of *imai and *matisse

We executed the following two commands, here.

```
*imai
*matisse
```

You can see the results from the figures (Fig. 2.13, 2.14) in the right page. The result of *imai command looks drastic than that of *stella. This is because, one of the side of new rectange is assigned to be a part of the rectangle. On the other hand, *mattise will limit the size of target rectangle. As the result of such limitation, we feel the output picture is a little bit more delicate.

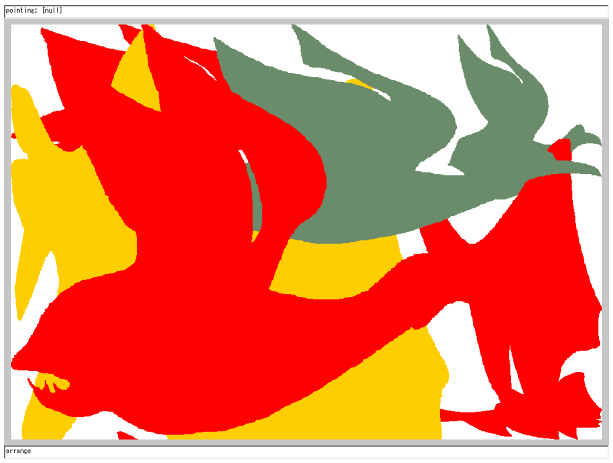


Figure 2.13: Example of *imai command execution

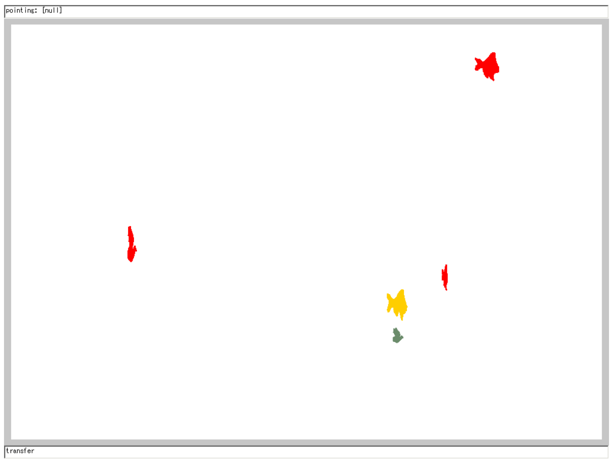


Figure 2.14: Example of *matisse command execution

2.11 Repeatedly used *matisse command

Try *matisse command.

```
*matisse
```

several times. The number of graphic objects will be filled and gradually graphical pattern will appeare.

The result if command execution is different because of the randomness in this command. But, the number of object is also an important factor of generated picture. Repeated use of the command or change of settings of repeat command, please appreciate the result.

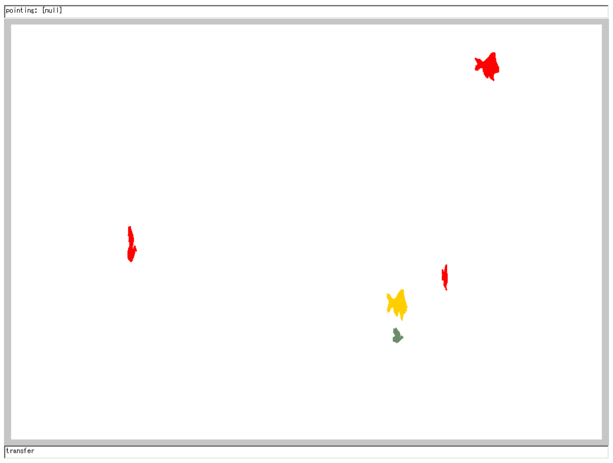


Figure 2.15: Example of execution of single *matisse command

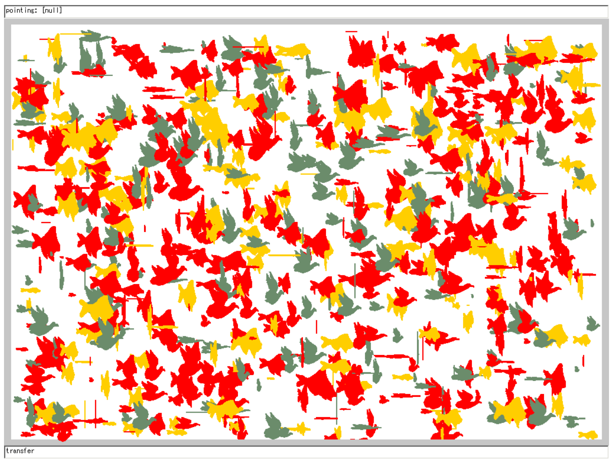


Figure 2.16: Example of *matisse command execution

2.12 mask commond

When we used commands introduced before, objects are placed to whole the screen.

However, it is possible to specify the target area for object placement can be limited to a rectanguler area.

Execute following command, before the execution of command *matisse.

```
mask 100 200 300 400
```

Then objects are placed in the rectangle that will be defind with two points (100, 200) and (300, 400).

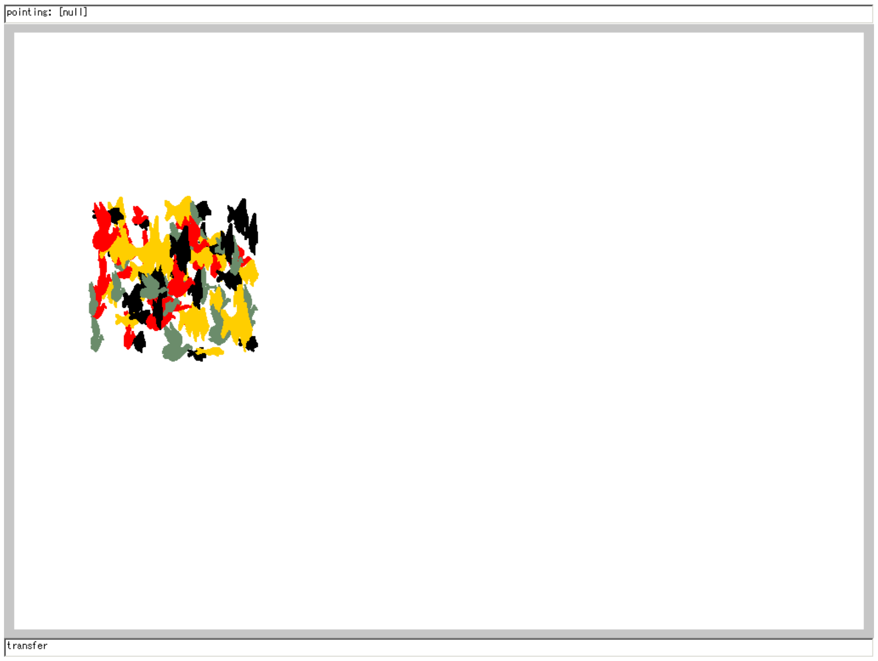


Figure 2.17: Example of execution commands mask and *matisse

2.13 Clearing up Screen and Parts Box

Clean up both screen and parts box at a time. Execute the folloing command.

```
reset
```

Screen and parts box will be initialized. [6]

[6]clear command initialize only the screen and does not make any change to the parts box.

Figure 2.18: Empty screen and parts box with `reset`

Chapter 3

Let's make graphic parts

3.1 Primitive

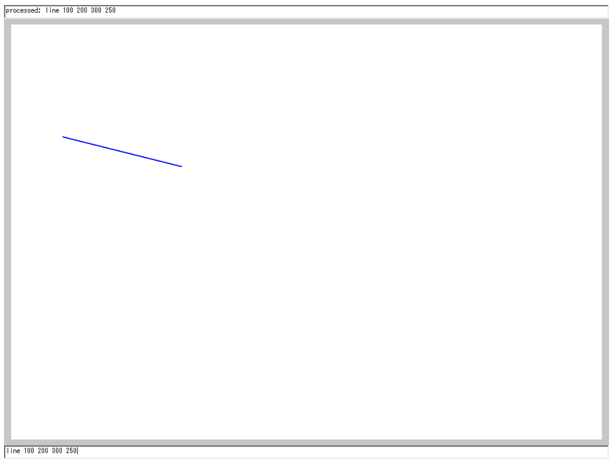
We explained how to place graphical parts that is already made.

In this chapter, we will explain how to create your original graphical part. (Here after let us simply call graphical part as part.)

As the first step, execute the following command.

```
line 100 200 300 250
```

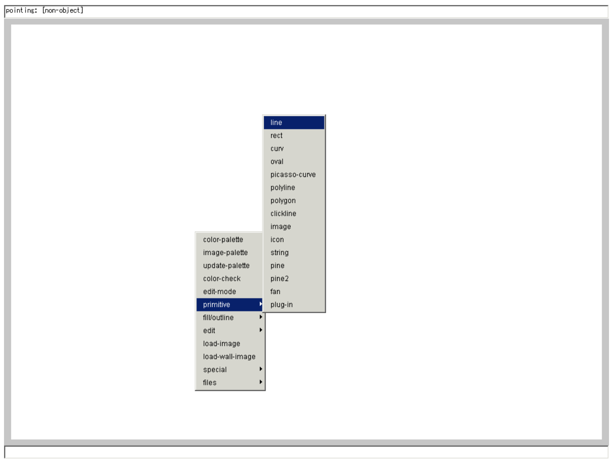
You will find a line on of whose terminate point is `(100, 200)` and another is `(300, 250)`. This line can be used as a part. Such a command execution is not the only way to generate parts. You can use the pointing device such as mouse. By clicking right buttun on canvas, a popup menu will appeare.



```
processed: line 100 200 300 250
```

```
line 100 200 300 250
```

Figure 3.1: Draw a line



```
pointing: [non-object]
```

Figure 3.2: Select a primitive from popup menu

3.2 Use Image as Wallpaper

A function for utilizing existing image (file) as a wallpaper of this editor. We can display one picture on canvas. You can trace the shapes of the wallpaper and create new part. [1] As the first step, execute the following command.

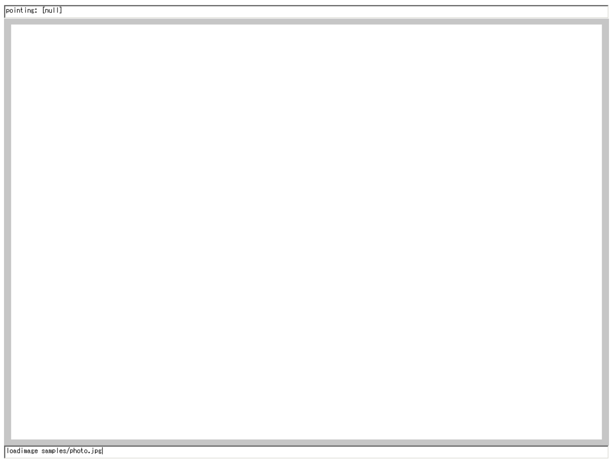
```
loadimage samples/photo.jpg
```

Nothing will be changed on canvas or screen. Then execute following command.

```
wall
```

Then, you will find a new image, which is read from the file.

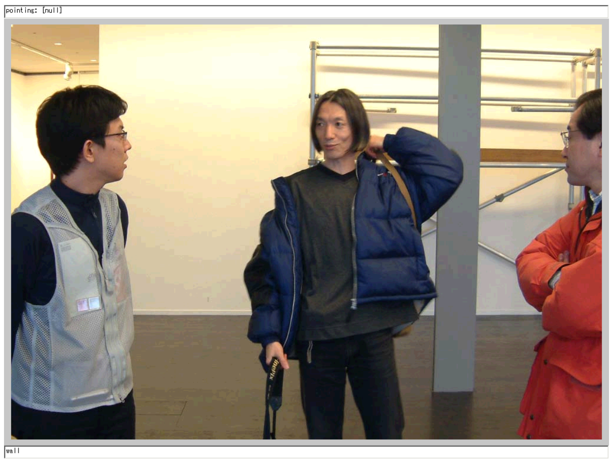
[1]As image data format JPG and GIF format is supported. In case you need to use diffrent type data, you need to convert them to either JPG or GIF.



```
pointing: [null]
```

```
loadimage samples/photo.jpg
```

Figure 3.3: Read JPG image into this system



```
pointing: [null]
```

```
wall
```

Figure 3.4: Display the image as the wallpaper

3.3 Trace Image

The primitive that is good for trace is `DrawCurv`. You can specify current primitive as `DrawCurv`, by command

```
curv
```

or by choosing 'curv' from popup menu. Then by tracing the shape on the wallpaper free shape premitive can be generated. If you add 'clear' parameter to the wall command,

```
wall clear
```

we can check the exact shape by clearing up the wallpaper.

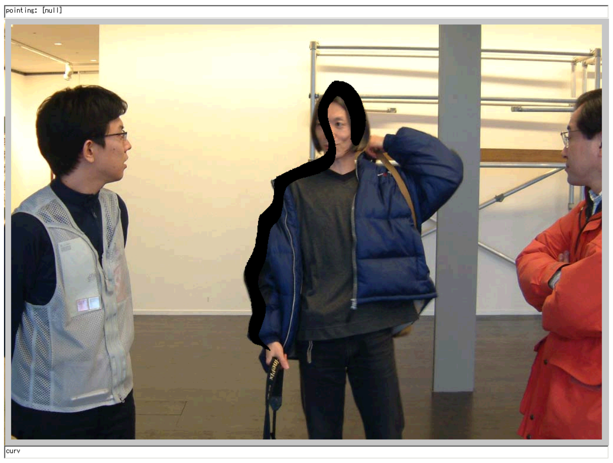


Figure 3.5: Execution of trace

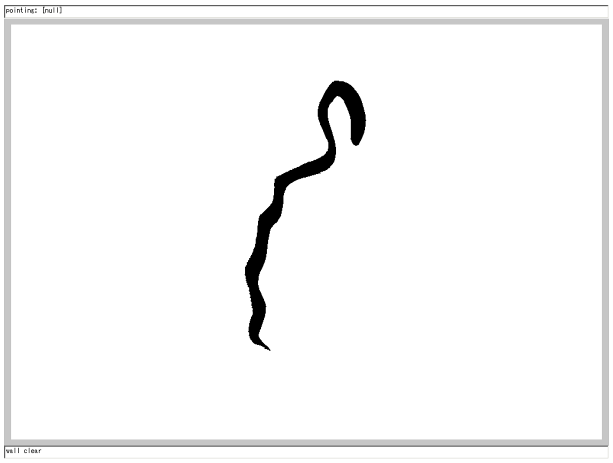


Figure 3.6: By removing wallpaper exact traced shape will appear

3.4 Coloring

We can color any object. By executing following command,

```
color red
```

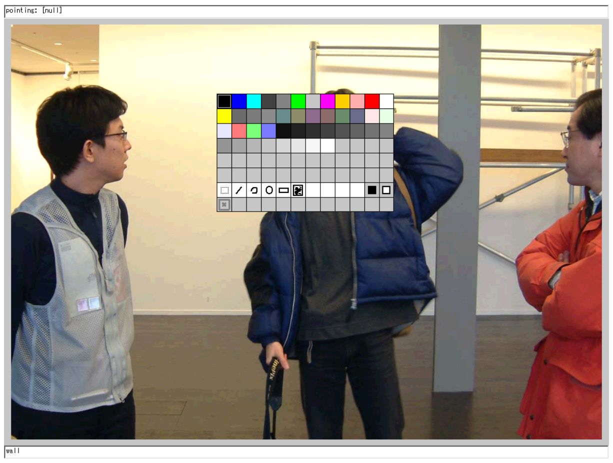
you can specify object color. [2] After specifying the current color, the specified color will be used as object color. Color palette is also available. By choosing 'color-palette' in popup menu, standard color palette will appear.

[2]Color name that used in Java, black, blue, cyan, darkGray, green, lightGray, magenta, orange, pink, white, red, yellow are used as names of colors. You can also specify the color by 256(0 - 255) level brightness. For example you can specify color red with command `color 255 0 0`.

pointing: [null]

color red

Figure 3.7: Specification of color


pointing: [null]

wall

Figure 3.8: Color Palette

3.5 Generate a palette from the Wallpaper

You can generate a new palette by extracting from the current wallpaper. Choose 'update-palette' from then popup menu then current color palette will be updated. In order to show the new palette use 'image-palette' menu item.

Using this function, you can easily take the same coloring strategy same as great master.

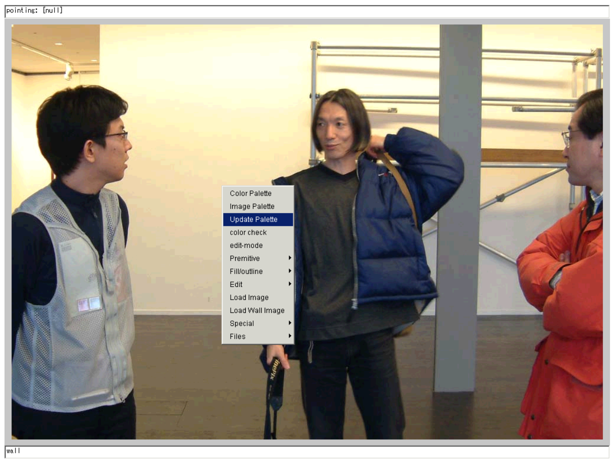


Figure 3.9: Update image-palette

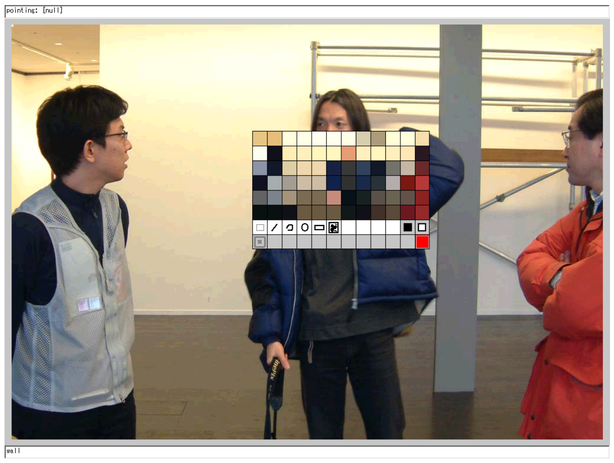


Figure 3.10: Palette that has Wallpaper's coloring strategy

3.6 Other attributes than color

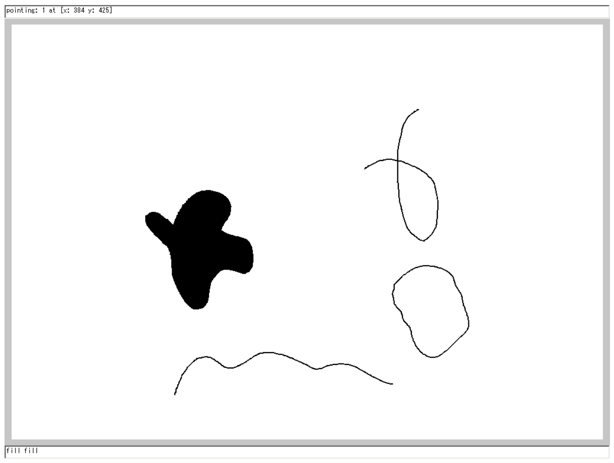
By specifying `fill` attribute, you can define whether you will fill the inside of the target object.

```
fill fill
```

and

```
fill outline
```

is the examples.



```
pointing: 1 at [x: 384 y: 425]
```

```
fill fill
```

Figure 3.11: Figures with fill attribute and outline attribute

Chapter 4

Use of Macro

4.1 Introduction to Macro

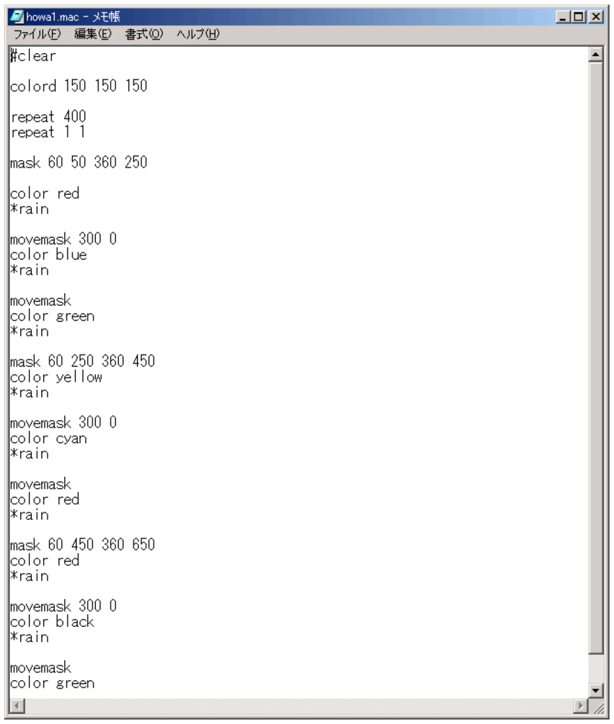
We call the function that sequentially execute the commands that is witten in text file as 'macro'.

By using this function, you can shortcut the process of retyping to get your favarable situation and output. If you have any set of input sequences of command inputs, you can make your own macro program.

Using any text edotor, edit the sequence of commands and save it as a file. One command should be written as oneline. [1]

You can use your favorate texteditor such as 'NotePad' on Windows. Example at the right page is a editing macro file 'howa1.mac'.

[1] Macro can be used with any extension. We are now recommending you to use the extension '.mac' for the macro.



```
#clear

colord 150 150 150

repeat 400
repeat 1 1

mask 60 50 360 250

color red
*rain

movemask 300 0
color blue
*rain

movemask
color green
*rain

mask 60 250 360 450
color yellow
*rain

movemask 300 0
color cyan
*rain

movemask
color red
*rain

mask 60 450 360 650
color red
*rain

movemask 300 0
color black
*rain

movemask
color green
```

Figure 4.1: Check the contents of howa1.mac with NotePad

4.2 Macro 'howa1.mac'

The text in the right page is the whole of the macro we define here. [2] The number of lines are 40 including empty line. We will give comments line by line.

[2]On each line, we added the line number that is not a part of original file. It will be used for the reference.

```
colord 150 150 150

repeat 400
mirror 1 1

mask 60 50 360 250
color red
*rain

movemask 300 0
color blue
*rain

movemask
color green
*rain

mask 60 250 360 450
color yellow
*rain

movemask 300 0
color cyan
*rain

movemask
color red
*rain

mask 60 450 360 650
color red
*rain

movemask 300 0
color black
*rain

movemask
color green
*rain
```

Macro "samples/howa1.mac"

`colord` on the first line is to define how to change the colors of object, when we make copies of an object from parts box to canvas. [3]

You can specify parameters by number from 0 to 255. Coloring factor for RGB will be specified. By specifying larger number, bigger change (color distribution) will be suggested.

```
colord <red> <green> <blue>
```

On line 3, the number of objects that is created with one command is specified as 400 with command `repeat`. [4]

Line 4 is a kind of initialization. [5]

[3]The color of new object will be defined by giving change of RGB parameters.

[4]The number of specified parameter is one.

[5]In most cases, you do not need this. For the stable execution, this statement is added. We do not make farther explanation, here.

```
1: colord 150 150 150
2:
3: repeat 400
4: mirror 1 1
5:
```

Part of macro "samples/howa1.mac line 1-5"

By using `mask` on line 6, you can specify the taget rectangular area to make copy (from the parts box) for the commands of object generation form the parts box to the canvas such as `*stella`, `*pollock`, `*imai`, `*matisse`,

```
mask <left> <top> <right> <bottom>
```

On line 7, `color` specifies the color of objects which are generated in line 8 with *rain command. By specifying parameter as `red` objects's color are closer to red.

On line 8, `*rain` is used. *rain is a command for generating lines directing from upper left corner to lower right corner.

On line 10, `movemask` is used to shift current masking area to specified direction.

```
movemask <L-R direction> <U-D direction>
```

On line 11 and 12, specify the generetion of lines with colors close to blue.

On line 14, `movemask` specifies the movement of rectanguler masking area. [6]

```
movemask
```

On line 15, 16, specify the generetion of lines with colors close to green.

[6]If you do not specify any parameters, the amount of shift specified before will be used, again.

```
 6: mask 60 50 360 250
 7: color red
 8: *rain
 9:
10: movemask 300 0
11: color blue
12: *rain
13:
14: movemask
15: color green
16: *rain
17:
```

Macro "samples/howa1.mac line 6-17"

Lines 18-28, simply repeat the description of line 6-17. Difference is the position of generated objects.

With these descriptions three rectangular area will be filled by lines.

```
18: mask 60 250 360 450
19: color yellow
20: *rain
21:
22: movemask 300 0
23: color cyan
24: *rain
25:
26: movemask
27: color red
28: *rain
```

Macro "samples/howa1.mac line 18-28"

In order to execute macro use command `exec` or `!`. After the command, specify file name.

```
! samples/howa1.mac
```

In the right page, example of execution is shown.

Specify rectangular area on the screen. Then specify typical color. Generating colors that is closed to specified typical colors. Draw 400 lines 9 times changing the color specifications and places.

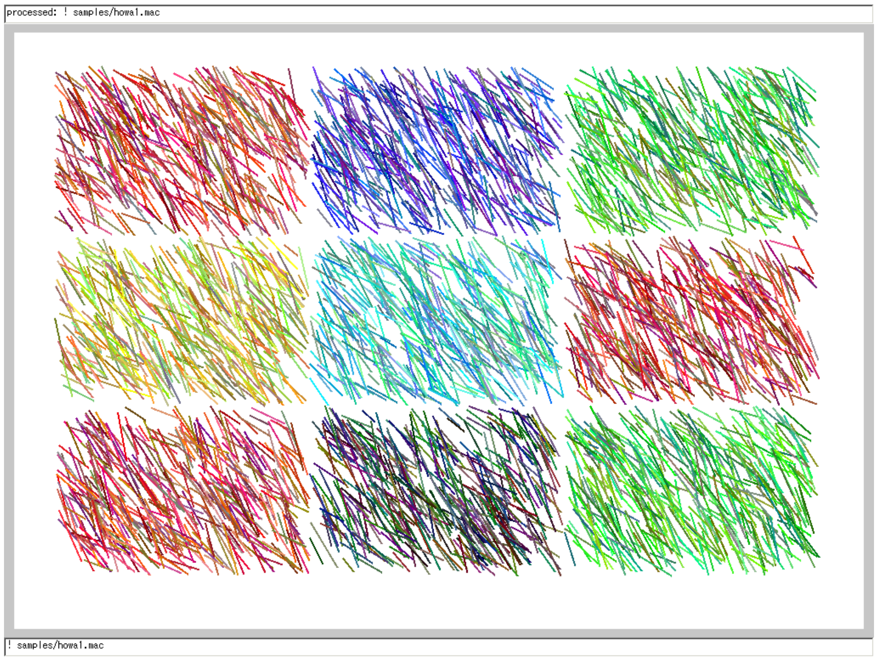
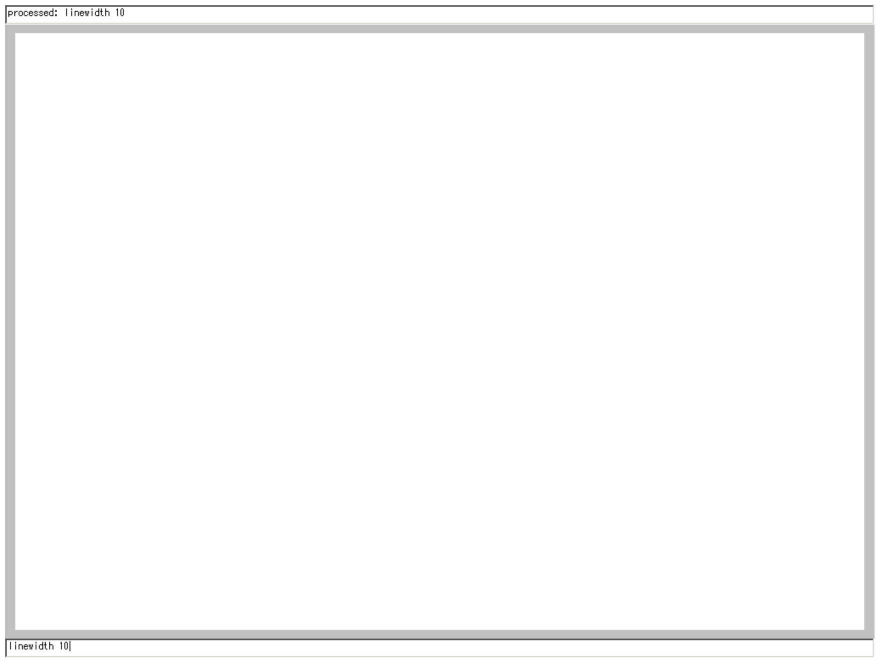


Figure 4.2: Execution of macro samples/howa1.mac

Before executing macro, change the width of lines, using command `linewidth`.

```
linewidth 10
```

By this command, the width of lines becomes 10.



```
processed: linewidth 10
```

```
linewidth 10
```

Figure 4.3: Change of line width

Try execute the same macro.

```
! samples/howa1.mac
```

We can see the change of results. The output resembles, but overall impression becomes different.

Thus, you can use command and macro, alternatively to chech what kind of figure would be generated.

If we change *rain command to *boxer or *bubble, diffrent figure will appeare.

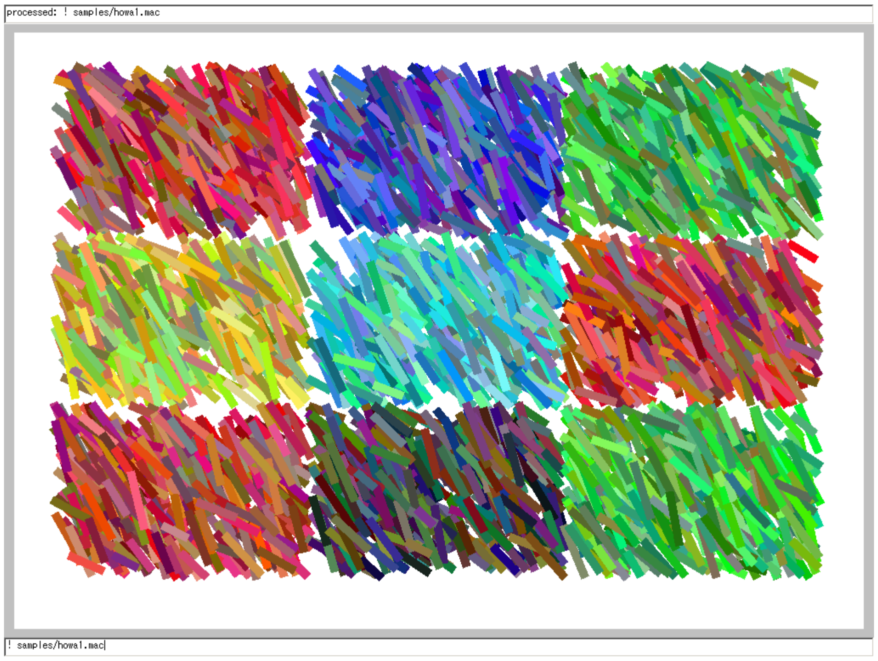


Figure 4.4: Executuion of samples/howa1.mac

4.3 Macro igo.mac

We are using random parameters to specify the position. But, we can genearate some regular patterns.

`samples/igo.mac` is one of the examples. This program consists of 16 lines.

```
color black

wmax 50
wmin 50
hmax 50
hmin 50

gridw 50
gridh 50

repeat 25

fill fill
*bubble
fill outline
*bubble
```

Macro "samples/igo.mac"

In line 1, specify color as black.

`wmax` and `wmin` are used to set the parameter for `*bubble`. [7]

We are specifying the maximun and minimum width of target objects as 50. The width of generated object will be 50.

By specifying with `hmax` and `hmin`,
We can specify the hight of target objects as 50 in a same manner.

`gridw` and `gridx`, specifies the grid size of object. object position will be quontified to be on the grid. On this macro the size of grids for x and y direction are specified as 50.

[7] *matisse、*rain、and *boxer commands are also under control of commands wmax and wmin. Commands such as hmax, hmin, gridw, gridh are the same.

```
color black

wmax 50
wmin 50
hmax 50
hmin 50

gridw 50
gridh 50

```

Macro "samples/igo.mac line 1-10"

`*bubble` creates ovals. The number of ovals is specified with `repeat` command. It will be executed on line 14 and line 16.

On line 11, specify the number of generated object as 25.

`fill` commands on line 13 and line 15 specifies fill or do not fill the target objects.

50 objects will be generated after executing this program.

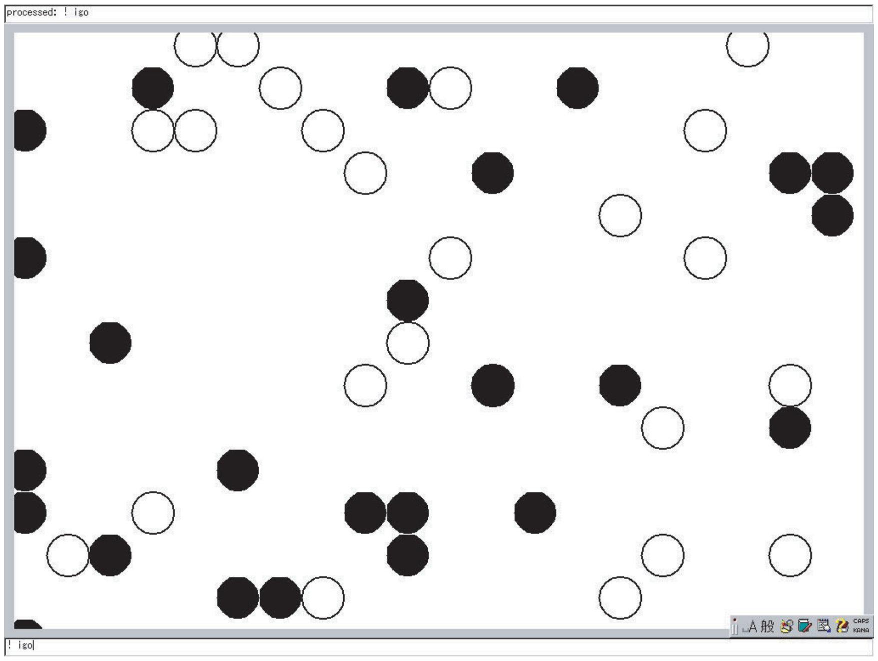
```
11: repeat 25
12:
13: fill fill
14: *bubble
15: fill outline
16: *bubble
```

Macro "samples/igo.mac" line 11-16

Let's execute following macro

```
! samples/igo.mac
```

We have some design patterns like the figure on the right page.



```
processed: ! igo
```

```
! igo
```

Figure 4.5: Execution of sampels/igo.mac

4.4 Macro yanagi_style.mac

Following macro is made to realize an existing image.

The art work in the right page is an artwork that is created by one of ThinkingSketch Unit member.

Figure 4.6: An artwork of ThinkingSketch Unit

To realize the image of the artwork in prefious page, we created a macro.

This macro can be executed as macro `samples/yanagi_style.mac`. Right hand figure is the result of the execution of `yanagi_style.mac`.

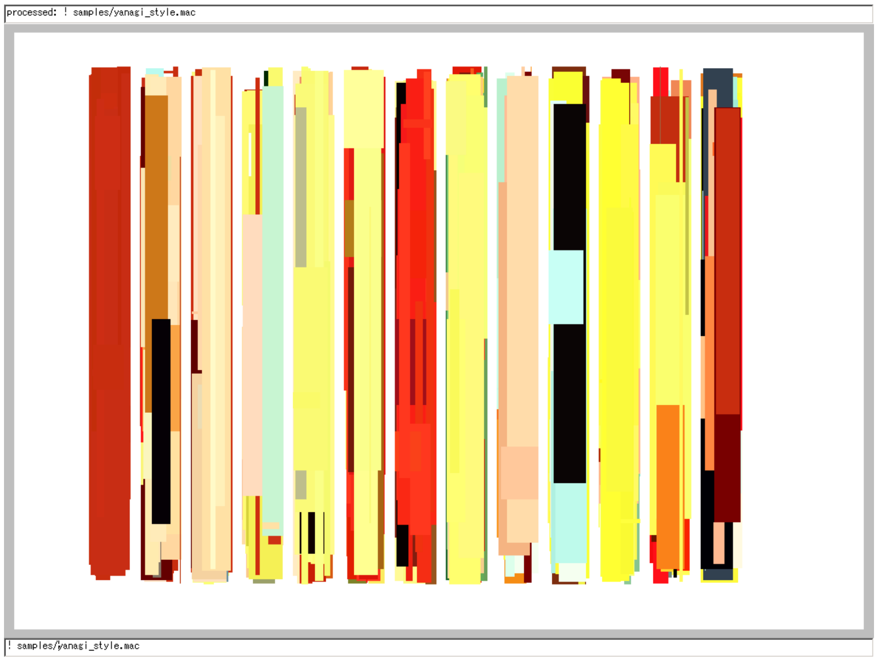

processed: ! samples/yanagi_style.mac
! samples/yanagi_style.mac

Figure 4.7: Execution of yanagi_style.mac

All the macro program is shown in right page.

Specify parts to be displayed from line 1 to 5.

What we are doing here is to get scanned image as JPG format file `yanagi_style.jpg` and generate a set of rectangles whose colors are used in the original image with command `colormap`. Those parts will be moved to parts box with `storage` command.

Line 7 specifies the number of parts to copy at a time as 50. Line 8 specifies the target (mask) area. After line 9, execute `*stella` command shifting mask area.

```
// select color
loadimage samples/yanagi_style.jpg
colormap

storage
//
repeat 50
mask 100  50 150 660
*stella
movemask 60 0
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
movemask
*stella
```

Macro "samples/yanagi_style.mac"

In this stage, the macro program is not completed, yet.

We need to update the macro until to have enough tastes and effects.

AS this is object based graphical editor, you can edit the final output.

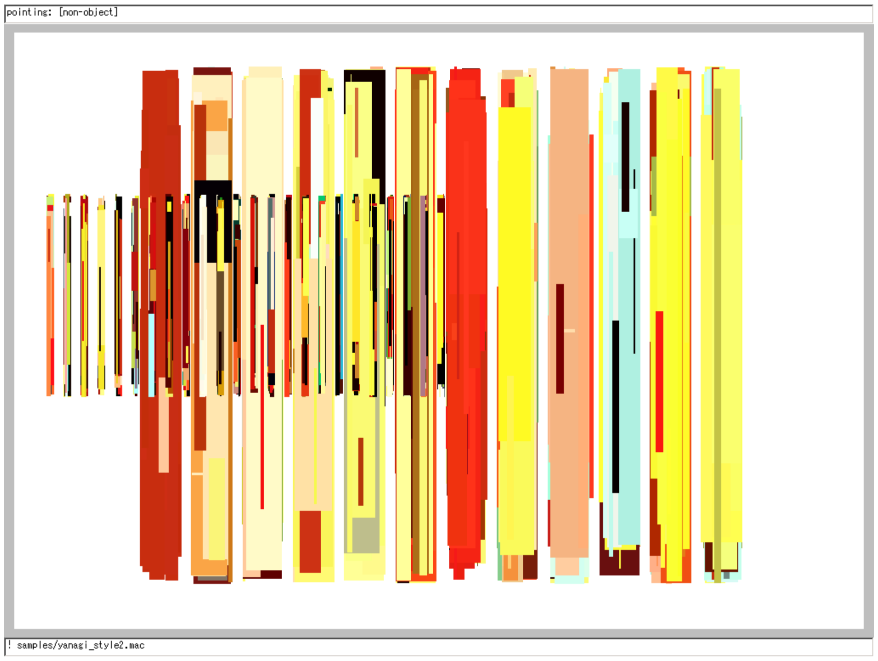


Figure 4.8: Macro has changed more

Chapter 5

Beginner's Works

Let us introduce some examples. Please try to make your own original figure.

The artwork on the leftside is the one that is drawn by a person T (who lives in Tokyo) as a beginner of ThinkingSketch.

This figure has simple but strong impression.

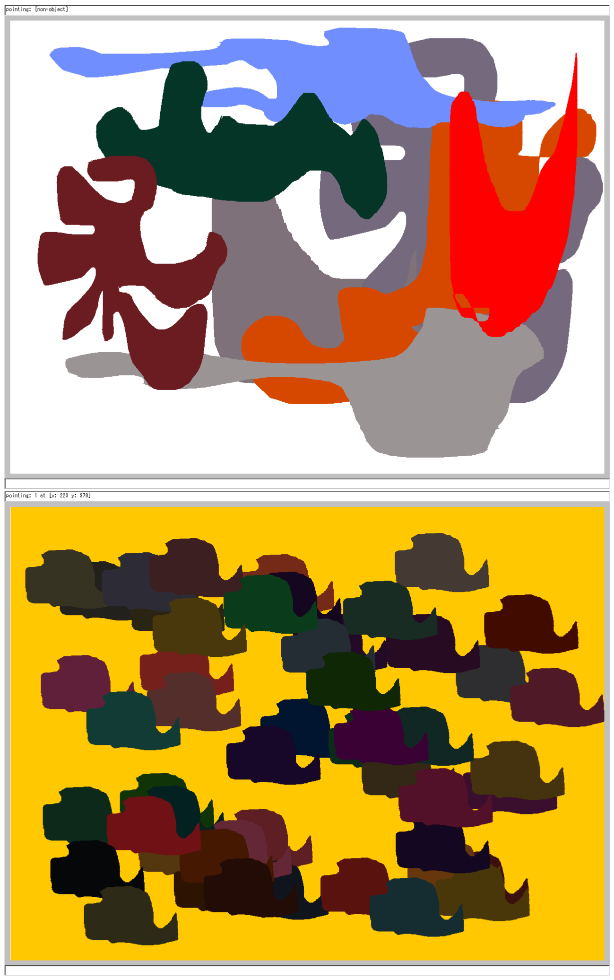

```
pointing: [non-object]
```
```
pointing: 1 at [x: 223 y: 970]
```

Figure 5.1: Works drawn by T

Next works are 'Red Forest.' Creator is the same person. He already had a motif of this picture. Using the existing motif this image was drawn.

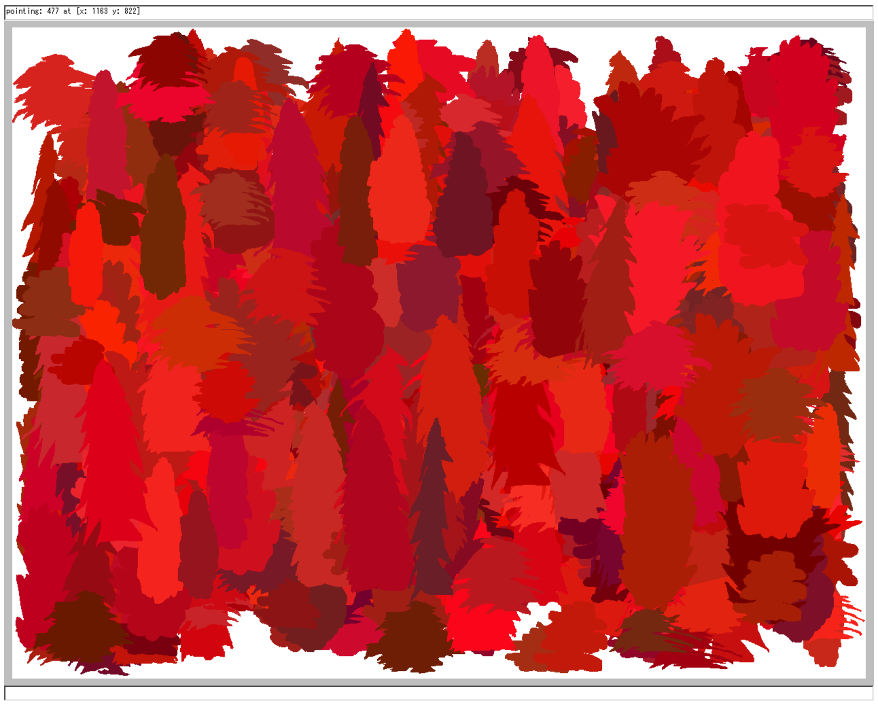


Figure 5.2: 'Red Forest' by T

This work is based on the motif of matisse. Creator is K. After making parts looks like part of Matisse's work. Place them with *pollock command. Slite adjustment was made after *pollock execution.

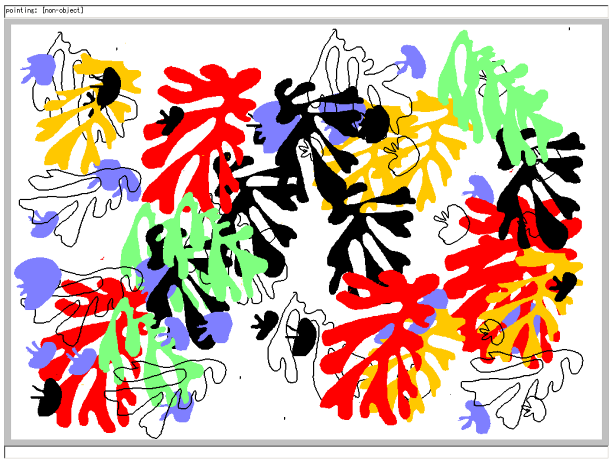


Figure 5.3: Matisse Arrangement by K

Creator is M. Image comes from spring landscape in Japan. Used `*imai` and `*pollock`. [1]

[1] He specified a parameter to change the distribution on canvas.

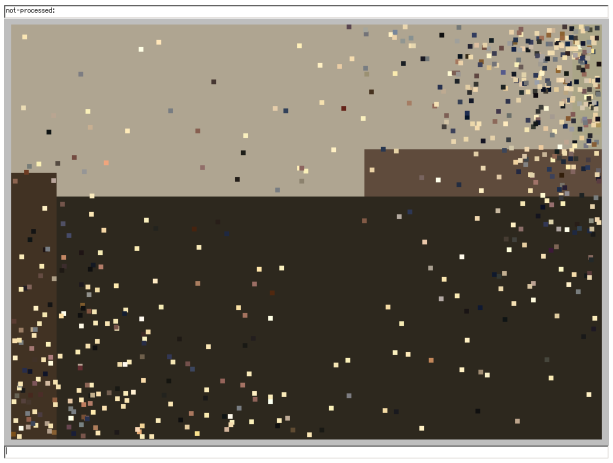


Figure 5.4: Spring Landscape by M

Update History

AUG 26/2002 : Update to Version 2.0 Alpha. The biggest change is (1) support of multiple pages. (2) Support of transparency of colors. Those changes are our size of focus in this cookbook.