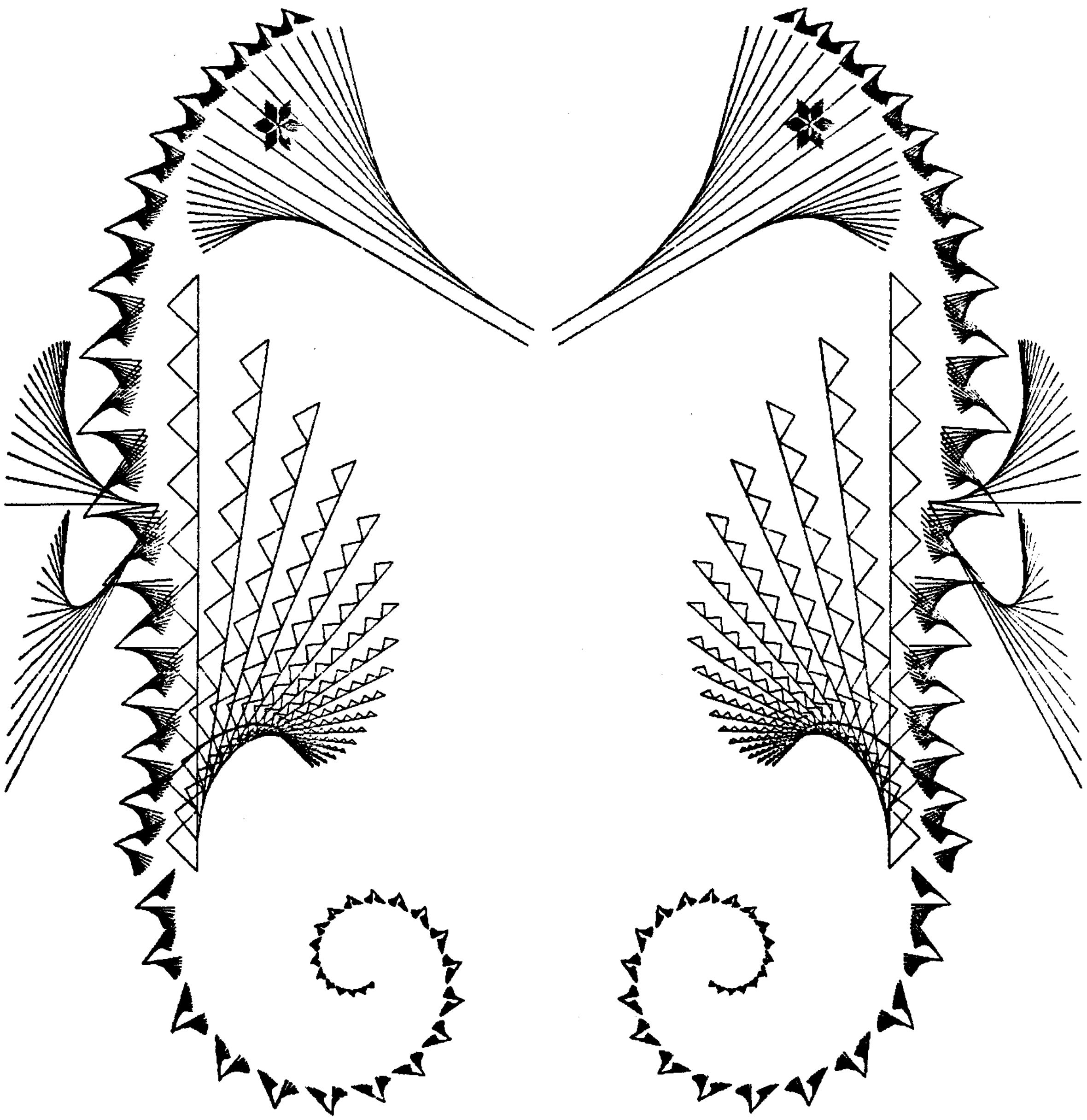
A TUTORIAL GUIDE TO GROPER



Groper Seahorses

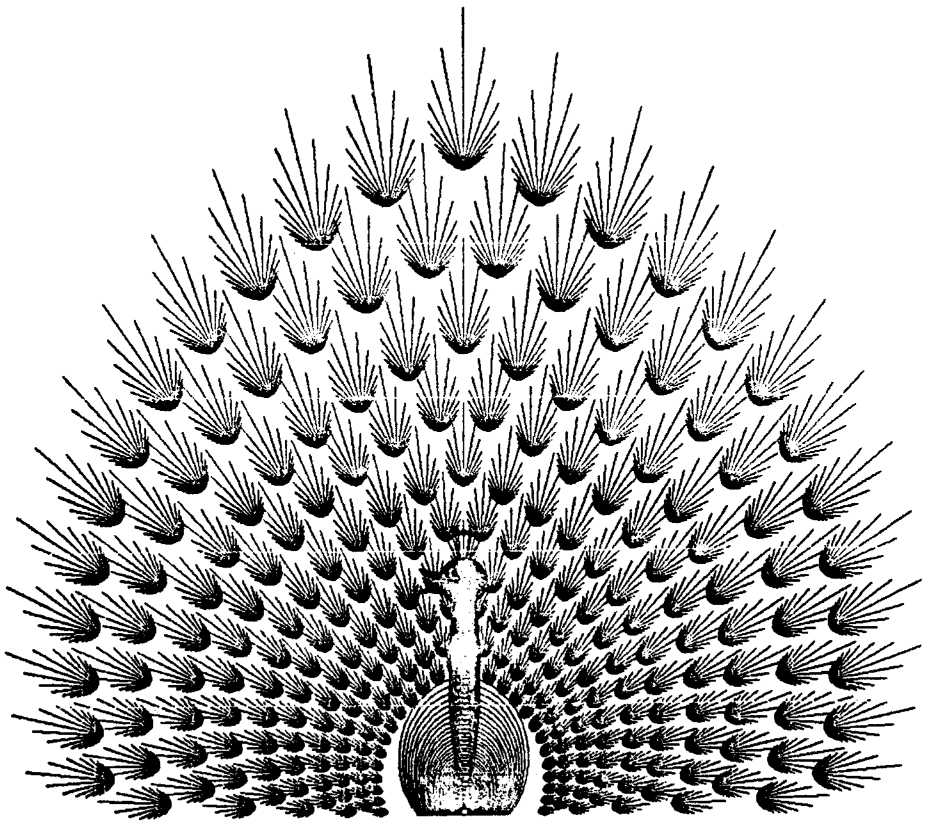
Greenberg, S. and Wyvill, B. (1983). A GROPER tutorial. Research Report 83/131/20, Department of Computer Science, University of Calgary, Calgary, Canada. October.

A Tutorial Guide To Groper

Saul Greenberg & Brian Wyvill
Department of Computer Science, University of Calgary.

NOTE: The following **Tutorial Guide to Groper** does not purport to be a full Groper manual. It is a guide intended to give the the novice user an idea of what Groper can do and how to do it.

Written September 1982 Saul Greenberg
Updated September 1983 Brian Wyvill



Groper Peacock

TABLE OF CONTENTS

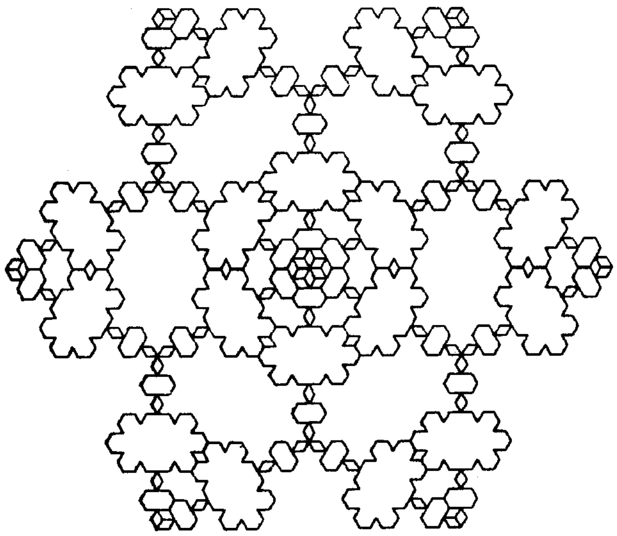
INTRODUCTION

Groper is a computer graphics language that allows an artist to define pictures as a collection of straight lines and other pictures. A picture can be used as a subpicture of other pictures. For example, a picture called 'buggy' can be assembled into the Groper Buggies figure by asking Groper to draw 6 buggies in a circle. (See picture on next page). In addition, a picture can be a subpicture of itself! This idea is called **recursion**, and it allows for many unusual pictures to be easily drawn.

Groper is an **interactive language**, which means that you, the artist, are in control of picture creation during each and every step of the drawing session. This allows for many benefits, such as being able to view and change your picture while you're constructing it, and the ability for Groper to detect any errors immediately. In addition, it implies that it is a good teaching tool, for what better way is there to learn than to do something and see your result immediately? If your picture isn't what you wanted, you can change it or start afresh. If its right, you can add to it even further.

This tutorial was designed with this **Learn By Doing** philosophy in mind. Thus, although it is important for you to grasp the message of each lesson, it is even more important for you to actually get on the computer and draw! It is recommended that you try the examples in each lesson, and then the **Learn by Doing** questions. When you feel comfortable with the lesson content, design your own pictures. Be creative and don't worry about errors. In Groper, some of the best results occur by accident!

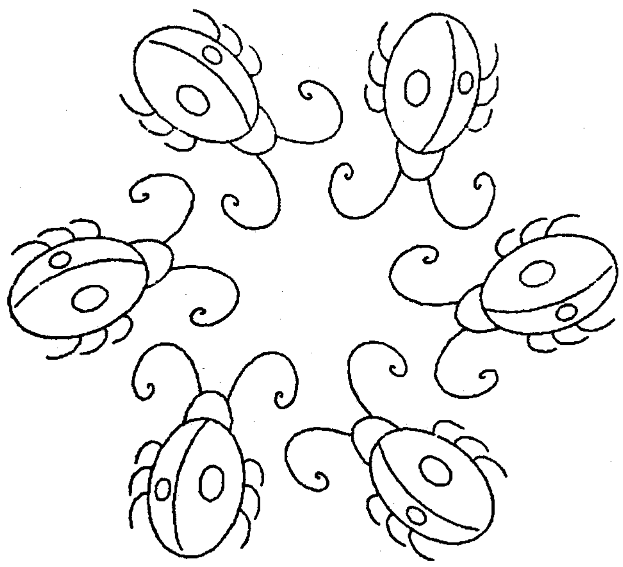
A basic understanding of co-ordinate or Cartesian geometry is helpful for the Groper artist. All 2-d pictures can be described in a simple **X Y** coordinate system. Thus you can create a picture or picture component at whatever coordinates you wish. It may be helpful for the novice to keep some graph paper close by so that he can trace out his drawing instructions by hand. Later on in the lessons, you will be learning how



Groper Rose

to make an actual **X - Y** axis upon which you can superimpose your pictures: This is important if the artist loses track as to exactly where a picture exists in the co-ordinate system.

The only picture that exists upon starting up Groper is a **horizontal line** connecting the points (0,0) and (1,0). Thus all your pictures are composed of this single building block. For example, the seahorses on the front cover are clearly made up of lines of different sizes and at different angles. Yet the image one first gets from that picture is one of flowing curves! The artist will soon learn the many techniques to produce real curves, such as those used to make the Groper 'buggies' below.



Groper Buggies

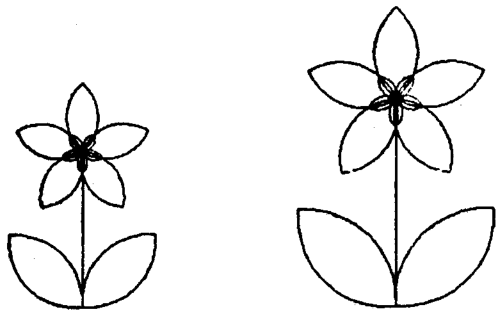
LESSON 1 - GETTING STARTED

There are certain sequence of events that must be followed by the user in starting up and quitting Groper. These events are shown below, the column on the right being annotations of the Groper commands printed on the left.

Note: All user-typed commands appear in italics throughout the tutorial.

INSTRUCTIONS	EFFECT
groper	This calls the GROPER program. A message will appear on the screen saying you have entered Groper.
device tk	This tells Groper that pictures should be drawn on the tektronix or visual 550 terminal that you are currently typing on.
bye	This terminates the Groper session. A copy of your session is automatically saved in a file called groper.grp. Use of this file will be explained later. It is also possible to do a panic halt of the session by hitting the <del> key. This will 'kill' your session at any point. However, no record of your session will be saved.

You are now ready to create your first picture!



Groper Flowers

LESSON 2 - CONSTRUCTION OF A SQUARE

A complete Groper session for constructing a square is illustrated in this lesson. The graphics device is assumed to be a Visual 550. The demonstration will use the commands shown in lesson 1 and will introduce four new commands: ***add*, *rotate*, *origin*** and ***plot***.

The ***add*** command is used for combining different pictures. For example, ***add line to pic1*** will create a picture called pic1 which consists of a single line (figure 1a). The reader is reminded that the **line** is the only pre-existing picture in Groper (see introduction). **Line** has its origin at 0,0 it is one unit long and connects the points 0,0 and 1,0.

The ***rotate*** command is used for rotating a figure. Thus we can create a vertical line by rotating it 90 degrees by the following commands: ***add line to pic2 rotate by 90***.

The ***origin*** command is used with the ***add*** command to designate where the picture should be added. For example, ***add pic2 to pic1 origin at 1,0*** will create a right angle similar to that shown in figure 1b. The ***origin*** command will place the origin of pic2 at coordinates 1,0 in pic1. A word of warning: If the coordinates are not given, Groper will assume they are 0,0. In addition, if the coordinates used are in fractions, then the fraction must have a leading zero.

The ***plot*** command instructs Groper to draw the picture on the graphical device connected to your terminal. A ***device*** command as described in lesson 1 must have been done, otherwise you will see nonsensical characters printed on your terminal.

The following complete Groper session will produce a square. User-typed instructions are shown in italics. The user should note that the '$','*' and '.' are **prompts** printed by the computer. A **prompt** indicates that the computer is ready for your next command.

```
$groper


Groper Mk 3.2  (Unix Version)          Update: 26 Sep 83  Time: 16:09:31
 good morning


.add line to square
        *plot square                          ;figure 1a

.add line to square rotate by 90 origin 1,0
        *plot square                          ;figure 1b

.add line to square origin 0,1
        *plot square                          ;figure 1c

.add line to square rotate by 90
        *plot square                          ;figure 1d

.bye
end of run.
$
```

LEARN BY DOING

1) Draw an equilateral triangle with sides of length 1 (angles of 60 degrees).

2) Draw a rectangle with a base of 1 unit and a height of 2 units.

3) Create a 'house' by adding the triangle to the top of the rectangle. You should note that Groper will take your created picture and expand or shrink it automatically to give the largest picture possible on the graphical device. This is called ***automatic scaling***.

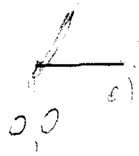


figure 1a

figure 1b

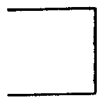


figure 1c

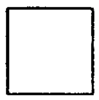


figure 1d

LESSON 3 - SIMPLE RECURSION

This lesson introduces the *limit* command and the idea of **simple recursion** as a drawing tool.

Suppose you want to draw a picture called *rowsq*, where *rowsq* is a row of 5 of the squares defined in the last example. I want each square placed 2 units away from the previous square. It would be very inconvenient to have to add each square individually to *rowsq* at the right place. Instead I can add one square to *rowsq* and a copy of *rowsq* like this:

```
add square to rowsq
   add rowsq to rowsq origin 2,0
```

At first glance this definition looks crazy since I have used *rowsq* to define *rowsq*! What Groper does is to say that *rowsq* is a square followed 2 units later by a *rowsq*, which is a square followed 2 units later by a *rowsq* .... and so on. But where does it all stop? This is where the *limit* command is used. Without it Groper will simply define *rowsq* as just one square. With the *limit* command Groper will define *rowsq* with as many squares as you choose.

```
limit rowsq 5 plot rowsq   ; see figure 2a
```

The *limit* command describes the depth of recursion, that is, the number of times the operation should be carried out. The idea that a picture may be defined in terms of itself is called **recursion**. This idea is extremely useful and very powerful.

Groper is at its most powerful in describing a picture in terms of itself. For example, the square in lesson 2 can be more concisely defined as:

```
add line to square
   add square to square rotate by 90 origin 1,0 limit of 4
```

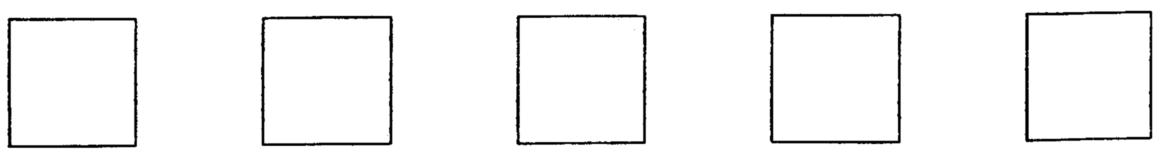


Figure 2a

LEARN BY DOING

1) Draw a square using the definition given above. Now create 4 different squares of different names, each with a different *limit* number. Try limits of 1,2,3 and 4. What figures are produced? What happens if you use a limit of 8?

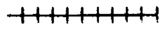
2) Draw the triangle from the previous lesson's **Learn by Doing** using the *limit* command.

3) Draw a hexagon (angle = 60 degrees).

4) Define a figure called *mystery*. Add a line and a *mystery* rotated by 10 degrees, an origin at 1,0 and a limit of 36. What kind of figure do you get? Why?

5) Draw an X and Y axis. Your basic unit should be a T lying on its side, such that the coordinates are 0,0 to 1,0 and 1,-0.5 to 1,0.5 (figure 2b). A single axis should be composed of 10 of these units (figure 2c). The complete axes should be a single axis rotated about the origin 4 times. The centre of the axes should have the co-ordinates 0,0 (figure 2d). Keep your axes description, as you will use it later.

6) Create a picture, called comp which is a composite of your circle, hexagon, square and axes. Plot comp. Why are the sizes of the picture elements so different? Describe this difference in terms of co-ordinate geometry.



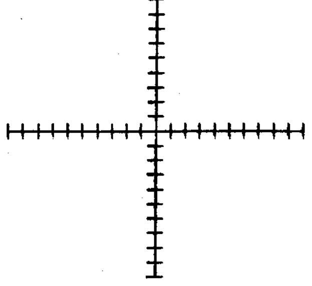


figure 2b   figure 2c   figure 2d

LESSON 4 - MAGNIFICATION

It soon becomes apparent to an artist that he must be able to manipulate a picture by changing its size, squashing or stretching it, and / or reflecting its image across different axes. This lesson introduces the tool used for these manipulations - the *magnification* command.

The *magnification* command works by multiplying the **X** and **Y** coordinates by a designated amount. This produces three basic effects:

1) **Changing the size of a picture**: It is possible to increase or decrease the size of pictures being created as shown by the following example (The square definition is as in lesson 3):

```
add square to pic1
   add square to pic1 origin 0.25,0.25 magnify 0.5
   plot pic1                                  ;figure 3a
```

pic1 consists of two squares, one without any change of size and the second has been halved in size and moved up and right by a quarter of a unit so that the large square surrounds the smaller.

2) **Reflecting a picture**: It is possible to reflect a picture across an axis by negating appropriate coordinates. For example, a diagonal reflection of the picture produced above can be done by:

```
add pic1 to pic2
   add pic1 to pic2 magnify -1,-1
   plot pic2                                  ;figure 3b
```

The original picture had its starting coordinates at 0,0 and existed in the (+,+) quadrant of a Cartesian graph. By negating these coordinates we are in effect flipping the picture into the (-,-) quadrant.

3) **Squashing and stretching a picture:** It is possible to squash or stretch a picture by magnifying the **X** and **Y** coordinates by different values. A square can be stretched into a rectangle by:

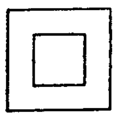


figure 3a

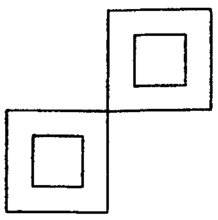


figure 3b

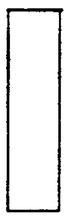


figure 3c

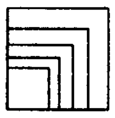


figure 3d

```
add square to rect magnify 0.5, 2
   plot rect                                  ;figure 3c
```

The rectangle produced will have a width of 0.5 units and a height of 2.0 units.

1) **Changing the size of a picture recursively**: Changing the size of a picture in a recursive manner can have some interesting and useful effects as shown by the following example:

```
add square to pic3
   add pic3 to pic3 magnify 0.8 limit 5
   plot pic3                                  ;figure 3d
```

The recursive figure produced consists of five squares each with the same origin but of different sizes. Each square is 8/10ths the size of its predecessor. Note that *magnify 0.8* has the same effects as *magnify 0.8, 0.8*.

Use of the *magnification* command in combination with other commands can produce many interesting effects. Try the examples in Learn by Doing and then experiment. Remember, some of the best pictures can happen accidently!

LEARN BY DOING

1) Make a row of 4 squares where each square is three quarters the size of the previous square.

2) Make a circle. Squash it by magnifying the X coordinate by 2 and the Y coordinate by 0.5. Make a picture which combines both the circle and the ellipse. Combine this picture with the axes to see if the ellipse was squashed correctly.

3) Create a tunnel of 6 squares, where each square is centered within the other and is half the size of the previous square. Reflect this picture into the (-,+) region, increasing its size by 2 in the process.

LESSON 5 - EDITING COMMANDS

It quickly becomes apparent that editing pictures is highly desirable. Some primitive editing facilities are available in Groper, through use of the ***list***, ***erase***, ***print***, ***remove*** and ***label*** commands.

Construction of a number of pictures often leads to a multitude of picture names that are quite easy to lose track of. The *list* command will print out all the pictures made in the current Groper session.

It is often desirable to remove all traces of a picture, erase it from the computers memory. This is achieved with the ***erase*** command. For example if you make a mistake in defining a picture called square, one simply types ***erase square*** to get rid it. If you then type *list*, you will see that square will no longer be in your list of pictures. You can now redefine square. Note: the ***erase*** command should be used carefully, for if the picture you erase is a building block of another picture, then that picture will be changed.

All pictures are composed of subpictures. The ***print*** command will print a picture's subpictures and the corresponding rotation angle, origin and scaling. In addition, it will print the label attached to the subpictures (discussed in the following section). For example, one can print the subpictures of a picture called square by typing ***print square***.

The ***label*** command attaches a label to your subpictures. Thus if you were putting a triangle on top of a rectangle you could say *add trian to rect orig 0,2 label tr*. This label may be used to edit your subpictures, as discussed in the following paragraph. However, Groper will automatically label your pictures for you, as seen in the ***print*** command. Thus one normally does not label a picture explicitly except that labels are also used to give a picture a colour. (See Lesson 10)

The ***remove*** command allows the removal of a picture's subpicture. For example, one can remove the triangle from the house picture (see lesson 2) by doing a ***print house***, seeing which label is attached to the triangle (let's say label B) and then typing ***remove*** *B*.

In this section I will explain a little about how Groper handles *errors*. It is a foregone conclusion that the user has already experienced some error messages. If you type something which groper cannot understand a fairly sensible message will be printed indicating what you have done wrong. The command line that you typed in will be printed and an arrow will point to the command that Groper couldn't understand. If you have typed several commands on the line then Groper will ignore everything after the error but all previous commands you did before the error will have been carried out! For example, Groper will produce the following error messages:

```
add line to triangle
   add line to triangle origin 1,0 roFate 120

      4: Command needed here.

      <ADD LINE TRIANGLE ORIGIN 1,0 ROFATE 120>
                                    ^
```

The arrow points to the end of the command sequence Groper couldn't understand. Thus the last command acted on is ***add line to triangle origin*** *1,0*. A new unrotated line will have been added to the triangle.

You may have noticed that there are two different prompts that Groper prints out, the dot and the asterisk. The asterisk prompt indicates that Groper is in the middle of defining a picture. It is useful to know this especially if you make an error. I have printed the previous example again with the prompts:

```
.add line triangle

        *add line triangle origin 1,0 roFate 120  print triangle

        4: Command needed here.

        <ADD LINE TRIANGLE ORIGIN 1,0 ROFATE 120 PRINT TRIANGLE>
                                      ^

        *rotate 120  print triangle

###### printing picture TRIANG ######
Picture Type  Vectors.  Recursion Limit     1

Label : A     r,g,b colour  -1   0   0 Table  -1
Transform of LINE
Angle                       0.000
Origin Ox,Oy                0.000     0.000
Magnification Mx,My         1.000     1.000

Label : B     r,g,b colour  -1   0   0 Table  -1
Transform of LINE
Angle                       120.000
Origin Ox,Oy                1.000     0.000
Magnification Mx,My         1.000     1.000
######
```

As soon as I type the first *add* command Groper prints the asterisk to indicate that triangle is being defined. Groper is now expecting any number of commands like rotate, origin and magnify, which actually affect the transformation of line in triangle. The roFate command is an error. An error message is printed and the rest of that command line is ignored. The *print* command should take Groper to dot level thus ending the sequence of asterisk level commands, however the print command has been ignored due to the preceeding error. Thus you can now simply retype the command line starting with the command that was in error as indicated above.

LEARN BY DOING

1) Use the three pictures previously created: square, rectangle and triangle. If you no longer have a copy of them recreate them. Do the following operations:
   - Create a picture called pict which is composed of the rectangle sitting on top of the square.
   - Put the triangle on top of the rectangle
2) List the components of pict using the *print* command.
3) Remove the rectangle from pict using the *remove* command. Plot Pict

4) Erase the picture altogether using the *erase* command. List all your pictures to see if it was successfully erased.

LESSON 6 - ADVANCED TECHNIQUES

This lesson will illustrate the building of certain pictures, such as spirals, stars and trees. The idea of **multiple recursion** is introduced. It is important for the student to analyze how the described pictures are made, so that he can apply these techniques to his own designs.

Construction of a **spiral** is similiar to that of a circle. The only difference is that a magnify command is used to shrink the lines, which causes the circle to rotate into itself.

```
add line to spiral
   add spiral to spiral rotate 10 magnify 0.95 origin 1,0 limit 100
   plo spiral                                      ;figure 4
```

The above example also demonstrates how one can abbreviate in Groper. Groper can recognize any command by its first three letters. Thus, *origin*, *origi*, *orig* and *ori* all have the same meaning. In addition, **readability** commands may be ignored. These are words which include: *to*, *and*, *at*, *from*, *by*, *is*, *for*, *with*, and *in*. These commands are totally optional, and may or may not be used by the Groper user to make his picture descriptions more readable. Notice also that anything you type on a line after a semicolon (;) is regarded as comment by Groper. E.g. *;figure 4* above.

It is now possible to construct a spiral which is composed of pictures, rather than lines. The following example will build a five cornered star (figure 5a), a circle of these stars (figure 5b), a spiral of these circles, and finally a picture composed of a reflected image of these spirals (figure 5c)!

```
add line to star
  add star to star orig 1,0 rotate 144 limit 5
  plot star                                ;figure 5a

add star to circ
  add circ to circ orig 2,0 rotate 60 limit 6
  plot circ                                ;figure 5b
```

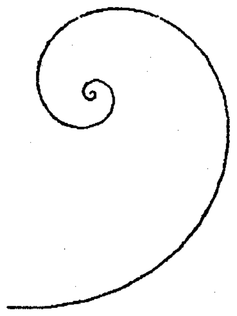


figure 4 - Spiral

```
add circ to spirl
   add spirl to spirl magnify 0.95 rotate -20 origin 0,5 limit 40
   add spirl to pair                                 ;Reflect the spiral
   add spirl to pair magnify -1,1
   plot pair                                         ;figure 5c
```

The concept of recursion was briefly mentioned in lesson 3. Most pictures drawn so far have used **simple recursion**, that is, most pictures have been defined in terms of themselves only once. The idea of recursion is not easy to grasp. Even expert computer artists will find multiple recursion difficult to follow beyond a few steps. Thus some analogies will be drawn.

1) A dictionary definition of recursion could be given as follows:

   Recursion: See recursion.

2) Imagine you are holding a photograph of someone holding a photograph. The inner photograph is the same as the one you are looking at. The same effect may be had by a T.V. camera focused on a T.V. screen which is also displaying what the T.V. camera is aimed at. Thus you have a picture of a T.V. in a T.V. in a T.V. in a T.V and so on ad nauseum.

3) Mirrors in some stores and barber shops may be placed directly opposite each other. Thus when you look at it, you see an infinite number of images of yourself.

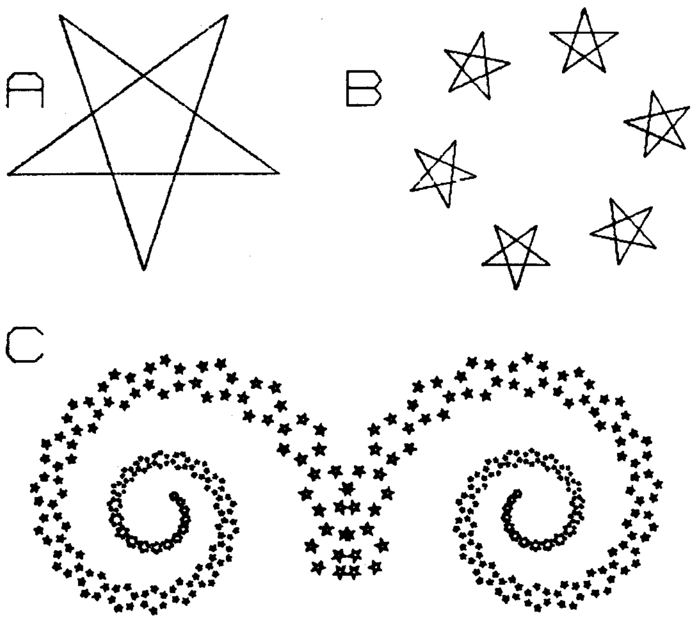


figure 5

Multiple recursion in Groper will now be introduced, that is a picture will be defined that uses itself more than once, for example, a tree.

```
add line to tree rotate 90                          ;vertical line
 add tree to tree origin 0,1 magnify 0.5 rotate 45 ;add a branch at the end

 add tree to tree origin 0,0.666 magnify 0.6 rotate -30 ;second branch
 limit tree 2 plot tree      ;figure 6a
 limit tree 3 plot tree      ;figure 6b
 limit tree 4 plot tree      ;figure 6c
```

At level 2 the tree (Fig. 6a) is just a line with two branches (trees). At level three there are two branches on the end of each branch of the level two tree (Fig 6b) at level four, two more branches are added to the end of each of the level three tree branches.

```
 add tree to tree origin 0,0.333 magnify 0.7 rotate 30
    plot tree                                ;figure 6d-f
```

Things get a bit trickier when the third branch is added. Each branch is a recursive reference to the whole tree so a complete tree as defined in figure 6a is now added on 2/3rds of the way up every line that figure a is made up of. However, each one of these 'trees' is in itself a tree. In Figure 6d, a level 4 tree, each branch or sub-tree is a level 3 tree, in the level 3 tree each branch is a level two tree. A level one tree is just a straight line.

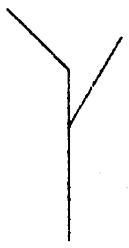


Fig. 6a

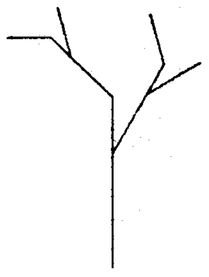


Fig. 6b

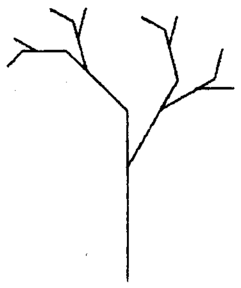


Fig. 6c

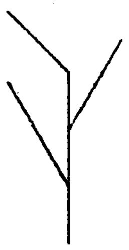


Fig. 6d

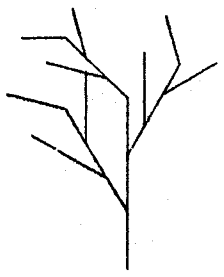


Fig. 6e

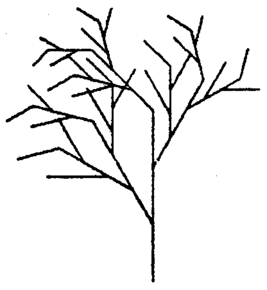


Fig. 6f

figure 6 - 3 trees of two and three branches, recursion levels 2,3 and 4

The best way to understand multiple recursion is to draw a few simple pictures using it. But be careful! A complex picture with many recursion levels takes a long time for the computer to figure out and to draw. Increasing the recursion limit to 10 the picture of Figure 6g is produced.

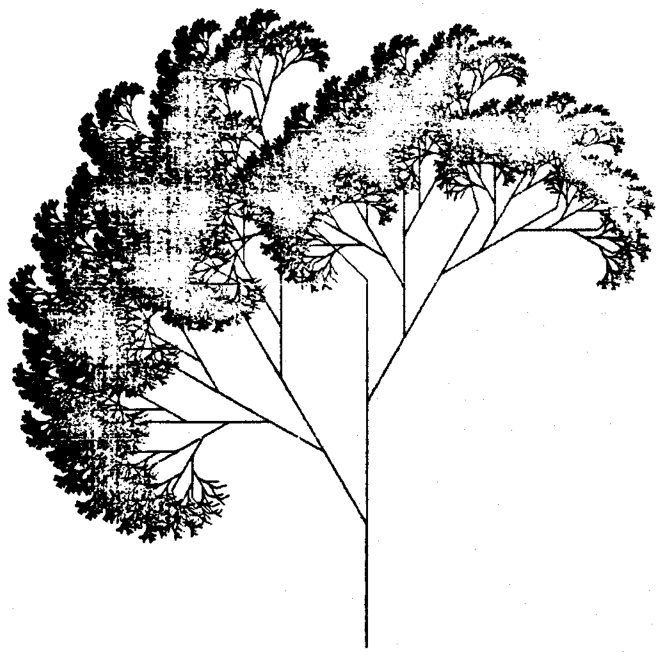


Figure 6g - Level 10 Tree of three branches

LEARN BY DOING

1) Create a picture composed of a single horizontal line attached at the right end to the middle of a small square of side length 0.2. Call this picture pic1.

2) Recursively create a picture composed of 3 images of pic1, where each succeeding image is 1 relative unit away from the previous and three quarters the size.

3) Add pic1 to itself rotated 90 degrees.

4) Hopefully, your picture has been correctly created. You will notice that it is a multiply recursive as it has been defined in terms of itself three times. Analyze exactly what has happened at each recursion level. Explain your result in terms of recursion levels.

LESSON 7 - CREATING A PICTURE LIBRARY

Pictures made within a Groper Session may be saved in a library of files for future reference. The Groper commands *log*, *restore* and the system commands *ls* and *rm* are introduce for library creation and maintanance.

A few **operating system** and **file** concepts will be explained to the non-expert computer user. When you log on to the computer, you are talking to the operating system. The operating system is a large computer program which allows you to interface with the computer. Thus you can give it commands to do certain things. One of the commands you have already used has been to tell it to run the Groper program. You can tell that you're in the operating system by the prompt it gives you (a '$').

A **file** in a computer is similiar to a file in a filing cabinet. It is merely a place with a name containing information which you can read, write or act upon. The Groper library is made by storing a copy of the Groper session in a file with a name of your choice.

The following sequence of commands allows you to keep a copy of your Groper session. The aim will be to put a picture of a square in your library.
-Enter Groper
-type *log square*. This tells Groper that you wish to keep a copy of your session in a file called square. NOTE: If a file called square has previously been made, Groper will not let you write to it.
-create your square and quit Groper.

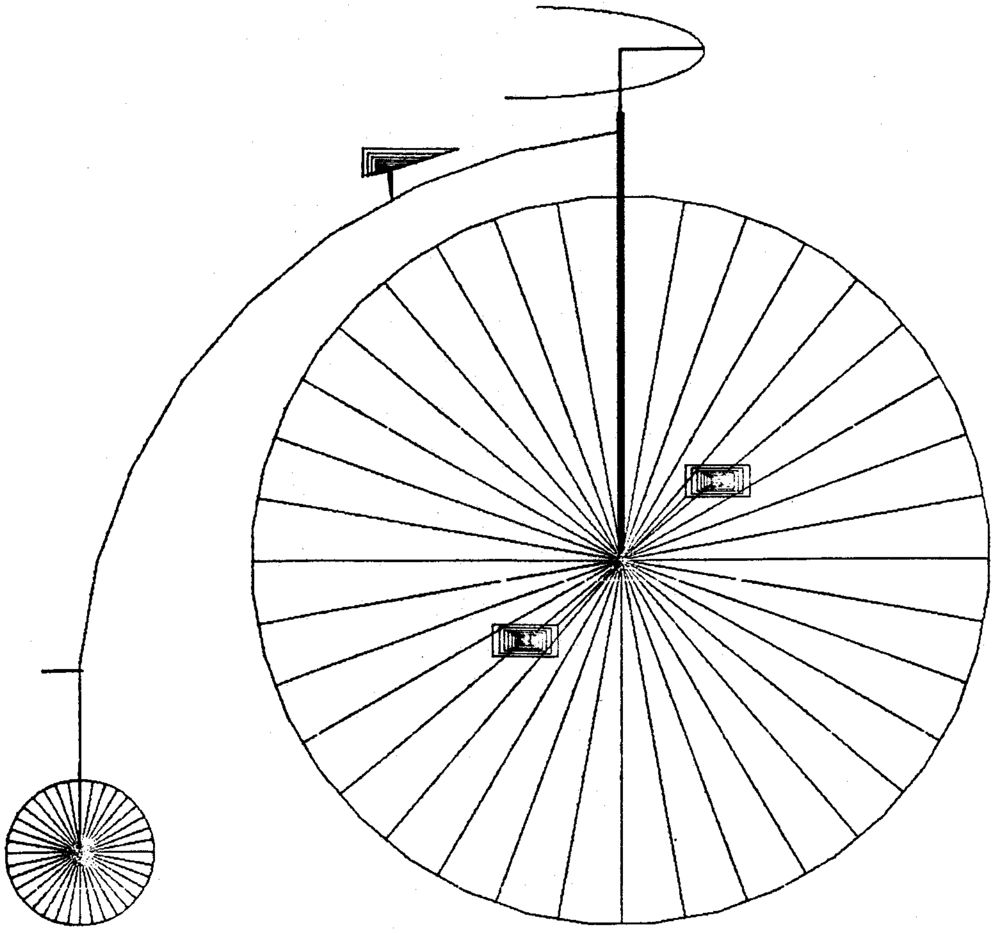
You may wish to see if your session was saved. To do this, type *ls* for list. This tells the operating system to list all your files. You will probably see two files, called groper.grp and square.grp. The grp is a suffix added on by Groper which simply says that these are Groper files. Groper.grp is a copy of every session which Groper automatically creates. Square.grp is the file that you explicitly created.

You may also wish to delete a picture file. For example, to remove the file groper.grp, type *rm groper.grp*. This removes the file groper.grp. Do an *ls* and see if that file is still listed.

The following sequence of events will show how a library picture is restored.
-Enter groper.
-Type *restore square*. Groper will display a message saying if it has succeeded in restoring the file.
-Type *list* to list your picture names. Were all the square's sub-pictures listed?
-Now try plotting your square. Was it restored correctly?

One should be cautioned when saving pictures. If you had created a picture during the Groper session with the same name as one of the pictures that you restored, Groper will try to combine them. This often leads to unpredictable and erronous pictures. It is thus highly recommended that you use unique and descriptive picture names for your library images.

Expert computer users may use the operating system editor to edit the actual saved file. Thus it is possible to remove all the undesirable commands and/or rubbish in your saved file.



Groper Penny-Farthing

LEARN BY DOING

1) Follow the above sequence of steps to create a library containing the axes picture from lesson 3. This is a handy picture to retain for it will allow you to plot other pictures on it to find unknown points.

2) Restore the axes picture in a new Groper session. Make any picture and combine it with the axes.

LESSON 8 - YET MORE COMMANDS

Three more commands are introduced in this lesson - ***between***, ***if*** and ***local***. The first is used to aid picture transformation while the last two are used to control recursion even further.

The ***between*** command allows the artist to pick up any picture by two points and place it on two points in another picture. Groper will automatically rotate and magnify the picture to line up those points. For example, an H can be plotted by:

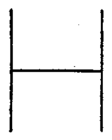
```
add line H rot 90
    add line H rot 90 ori 0.75,0
    add line H between 0,0 1,0 and 0,0.5 0.75,0.5      ;figure 8A
```

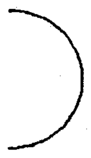
The first two commands will create the vertical lines of the H. The last command picks up LINE by the points (0,0) (1,0) and places LINE on H such that (0,0) falls on (0,0.5) and (1,0) falls on (0.75,0.5). An automatic magnification occurs to 'fit' the picture.

If a picture has a recursive sub-picture the ***local*** command allows the artist to give the sub-picture a different recursion limit from the original. For example, the following commands will create a half circle out of a circle:

```
add line circle
    add circle circle rot 5 origin 1,0 limit 72        ;circle created
    add circle halfcirc local 36                       ;figure 8B
```

The picture circle is not altered. However, the picture halfcirc is now defined as a circle with a recursion level of 36, rather than 72. Thus the limit of circle is temporarily altered to make the half circle. The ***local*** command produces interesting pictures





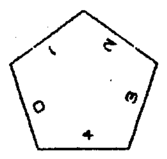


figure 8a figure 8b figure 8c

when used with a multiply recursive picture, for example different branches of a tree can be given different limits.

The *if* command allows the artist to add a picture to another only at a particular depth of recursion. Thus we can go to certain points of a figure without really knowing their exact coordinates. To demonstrate this, we will assume that there are five pictures representing numbers previously created in Groper, called numb1, numb2, numb3 numb4 and numb5. A pentagon will be created and each side will be individually numbered.

```
add line pentagon
    add pentagon pentagon ori 1,0 rot 72 limit 5       ;pentagon created
    add numb0 pentagon ori 0.5,0.1 if 0
    add numb1 pentagon ori 0.5,0.1 if 1
    add numb2 pentagon ori 0.5,0.1 if 2
    add numb3 pentagon ori 0.5,0.1 if 3
    add numb4 pentagon ori 0.5,0.1 if 4                ;figure 8C
```

This picture description also illustrates that Groper numbers the recursion levels from high to low. A recursion level of 5 is actually numbered by Groper as going from 0 through 4.

LEARN BY DOING

1) Create an apple composed of a circle and a stem. The top of the stem should be at the origin. Use the between command if possible.

2) Create the tree as in Lesson 6 using a recursion level of 5 rather than 10. This will create a sparsely branched tree. Add the apple to the end of all the branches by using the *if* command.

3) Create a recursive picture. Copy it into another picture except using different local commands. For example, create a spiral and experiment with varying the spiral tightness with the local command.

4) Create a multiply recursive picture such as the tree. Alter the tree by adding local commands at the end of each step. Try to achieve the effect of an Autumn tree - one that had different densities of branches.

LESSON 9 - TEXT

Practical applications of computer graphics often includes text as part of a picture, for example, annotations of drawings and poster preparations. Groper has a facility which allows the artist to add text to his picture, by defining a new building block called a **text picture**. The text picture allows you to manipulate text like a normal picture, that is, you can use a text picture the same way you use the **line** building block. In addition, you can define up to 12 **fonts** of the text itself (a font is the type of character set used, such as italics, roman, etc.)

One creates a **text picture** in a similar manner to a normal picture. Let's say you wish to make a text picture which contains the text string: This is a text string. This is done by:

add "This is a text string" to pic 1

The first thing to notice is that the string is surrounded by double quotes. When groper sees this, it realizes that you are creating a text picture. Try typing this in and plotting the picture. The first thing you will see is that there is a box which surrounds the text string. The box is just an indication of the space the string will take up in a picture. To see what it actually looks like, you must type **Softfonts**, a switch which turns software fonts on and off. Try it and plot your picture again.

You will see that it takes a longer time to plot the picture, due to the difference between **software and hardware** fonts. Harware fonts are those that come with a terminal, usually one size and shape and are thus very quick for the terminal to draw. Software fonts are characters that are graphically generated, that is, each character is in itself a picture - and thus takes a longer time to draw.

There are 12 software fonts available. These are:

simplexroman (default font)
complexroman
gothicenglish
gothicgerman
blockroman
triplexroman
complexscript
gothicitalian
duplexroman
complexitalic
triplexitalic
simplexscript

You can describe the font type of the text picture when it is created.
For example, we can make a text picture called "This is gothic italian"
by typing the following:

add "This is gothic italian" to pic2 font gothicitalian ; fig 9a

Try making 12 pictures, each of a different font to get an idea of what the fonts look like. (At the end of Appendix 3 some font examples are given).

The next thing that the artist would like to do is add the text picture to a normal picture. There are two ways to do this:

- Treat the text picture as an ordinary picture - that is, look at the size of the text picture and try to figure out the mess, or
- Use the special text picture features described below (a much better way).

Groper automatically knows where certain points of your text picture are. These points are:

a - top left
b - top centre
c - top right
d - left centre
e - centre centre
f - right centre
g - bottom left
h - bottom centre
i - bottom right

You can add the text picture to a new picture moving the origin to a point on the text e.g:

```
add "Hello" to pic3 font complexscript
add pic3 to pic4 origin right,centre
add line cross add cross cross rot 90 limit 4
add cross to tpik add pic4 tpik plot tpik          ;(figure 9b)
```

The origin of pic3 is at left,bottom (by default) the origin of pic4 is at centre bottom. The origin of the cross is at 0,0 and has been added to show the position of the origin of pic4. The **between** command can also be used to position text pictures, for example to add pic3 to the centre of a square:

```
add line to square                                  ;create a square
add square to square origin 1,0 rotate 90 limit 4
add square to pic5
add pic3 to pic5 between left,centre and right,centre and 0.1,0.5 0.9,0.5 ; fig9c
```

The text string is picked up by the left and right centre points and placed in the square at the indicated position.

LEARN BY DOING

1) Create a rectangle 2 units wide by 1 unit high and put your name in the centre of it. Font should be complex script.
2) Draw an envelope with a stamp in the upper right corner and your address centered on the envelope. Use different fonts (your choice).

This is gothic italian

Fig. 9a

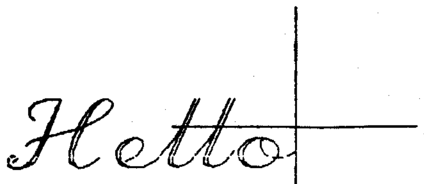


Fig. 9b

Fig. 9c

Text Examples

APPENDIX 1 : LEARN BY DOING - ANSWERS

Most answers presented here are not by any means the only solution. However, they have been tested and will create the correct result. The solutions presented here will keep pace with the lessons, that is, no techniques or commands will be used that haven't been presented at that lesson level.

Lesson 2 - Answers

```
1) add line to triang                             ;base
   add line to triang rotate 60                   ;left side
   add line to triang rotate 120 origin 1,0       ;right side
   plot triang                                    ;figure A1

2) add line to base                               ;base 1 unit long
   add line to height rotate 90
   add line to height rotate 90 origin 0,1        ;side 2 units long
   add base to rect                               ;assemble picture
   add base to rect origin 0,2
   add height to rect
   add height to rect origin 1,0
   plot rect                                      ;figure A2

3) add rect to house
```

```
3) add rect to house
     add triang to house orig 0,2
     plot house                                ;figure A3
```

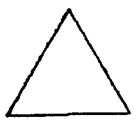


figure A1

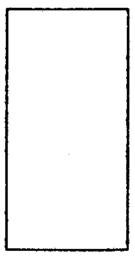


figure A2

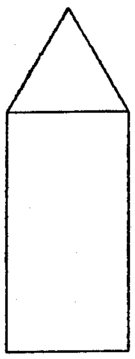


figure A3

Lesson 3 - Answers

1) Using limits of 1,2,3 and 4 will produce figures similiar to figure 1a through 1d in lesson 2. A limit of 8 will produce an identical looking square to that of a limit of 4. The difference is that the square is drawn twice on the same coordinates!

2)
```
add line to triang
        add triang to triang origin 1,0 rotate 120 limit 3
```

3)
```
add line to hexag
        add hexag to hexag origin 1,0 rotate 60 limit 6
        plot hexag                                  ;figure B1
```

4) A circle is obtained. Actually, it is a 36-sided polygon. However, the resolution limits of the graphical equipment and the human eye give the optical illusion of a circle. See figure B2.

5)
```
add line to lazyT
        add line to lazyT rotate 90 origin 1,-0.5   ;figure 2b
        add lazyT ax
        add ax to ax origin 1,0 limit 10            ;figure 2c
        add ax axes
        add axes axes rotate 90 limit 4             ;figure 2d
        plot axes
```

6) One must remember that the basic unit of all the figures made is a line of length 1. Thus the square is a 1 by 1 polygon with origins at 0,0. It thus takes up only 1 square unit of space on the axis. The circle on the other hand has a much larger size as it is composed of 36 lines going round. See figure B3.

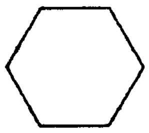


figure B1

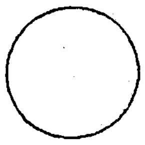


figure B2

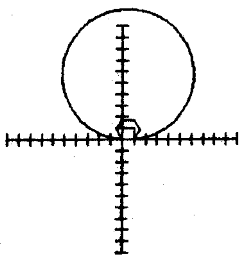


figure B3

Lesson 4 - Answers

```
1) add square to row                                    ;Assume square already made
     add row to row origin 2,0 magnify 0.75 limit 4
     plot row                                           ;figure C1

2) add circle to ellips magnify 2,0.5                   ;Assume circle already made
     plot ellips                                        ;figure C2

3) add square tunnel
     add tunnel tunnel origin 0.25,0.25 magnify 0.5 limit 6
     add tunnel pic1
     add tunnel pic1 magnify -2,2 limit 2
     plot pic1                                          ;figure C3
```

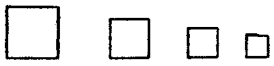


figure C1

figure C2

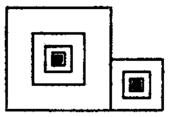


figure C3

Lesson 5 - Answers

1) Assume that the square, rectangle and triangle have been made similiar to those in lesson 2.

```
add square pict
   add rectangle pict ori 0,1
   add triangle pict ori 0,3
   plot pict                                        ;figure D1
```

2) Pict is shown to be composed of three picture elements: a square, rectan and triang. Origin and labels are described. The color description is not applicable at this point.

```
pri pict

###### printing picture PICT  ######
Picture Type  Vectors.  Recursion Limit    1

Label : A     r,g,b colour  -1   0   0 Table  -1
Transform of SQUARE
Angle                                           0.000
Origin Ox,Oy                                    0.000     0.000
Magnification Mx,My                             1.000     1.000

Label : B     r,g,b colour  -1   0   0 Table  -1
Transform of RECTAN
Angle                                           0.000
Origin Ox,Oy                                    0.000     1.000
Magnification Mx,My                             1.000     1.000
```

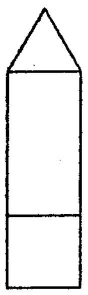
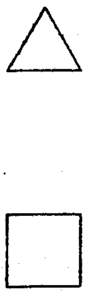



Figure D1   Figure D2

```
Label : C     r,g,b colour  -1  0  0 Table  -1
Transform of TRIANG
Angle                                        0.000
Origin Ox,Oy                                 0.000    3.000
Magnification Mx,My                          1.000    1.000
######
```

3) By the information given in the *print* command, the rectangle has the label **B**. Removal of this label deletes the rectangle from the picture.

remove b pict

```
erasing label B     from PICT
```

plot pict ;figure D2

4)

```
erase pict
 picture PICT  erased.
```

list ;Picture pict no longer exists

```
list of picture names.
    LINE
    SQUARE
    RECTAN
    TRIANG
```

Lesson 6 - Answers

1) ***add line square***
   add square square orig 1,0 rot 90 lim 4 ;make a square

   add line pic 1
   add square pic 1 mag 0.2 orig 1,-0.1 ;figure E1

2) ***add pic 1 pic 1 orig 2,0 mag 0.75 limit 3*** ;figure E2

3) ***add pic 1 pic 1 rot 90*** ;figure E3

4)

-Figure E1 is a non-recursive picture composed of a line and as square. The line is exactly one unit long.

-Figure E2 is a simple recursion picture of Figure E1. A new, smaller image of E1 is replicated 1 unit away from the previous image to a level of 3.

-Figure E3 is a multiply recursive picture of Figure E1. Study it closely and analyze how it was made:

-The picture in figure E2 is found in E3. They are the lowest horizontal lines.

-The first horizontal line has attached to it at a right angle a copy of figure E2 with the exception that *only two levels of recursion are found in that picture*.

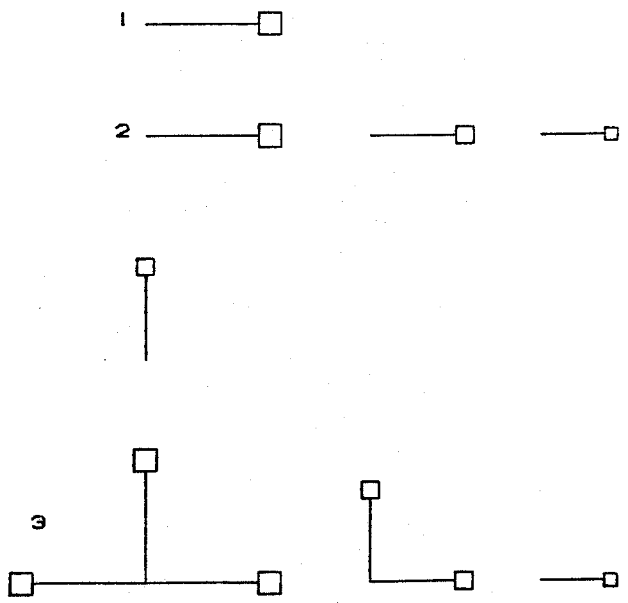


figure F.

This is due to the fact that we are now at the second level of recursion. In addition, another instance of the picture is found rotated 180 degrees from the origin. There is only one level of recursion found in this instance.

-The above image was thus formed of figure E2 being rotated around itself 3 times. The first instance had 3 levels of recursion (horizontal), The second instance had 2 recursion levels and the third had three.

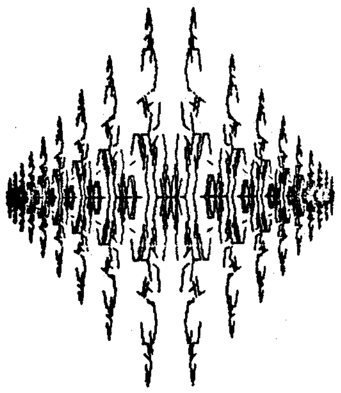
-The second picture along the horizontal line starts at a recursion level of two, thus we see the right angle image with a recursion level of only one.

-The last picture along the horizontal line starts at a recursion level of one, thus there are no rotated images of it.

Lesson 7 - Answers

1) *log axes*
   (Create your axes picture and exit)

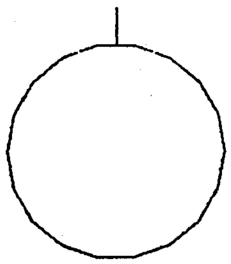
2) *restore axes* ;axes picture restored
   (Create other pictures and exit)


Fourfade

Lesson 8 - Answers

```
1) add line circle
   add circle circle ori 1,0 rot 20 lim 18
   add line apple between 0,0 1,0 and 0,0 -1,0        ;stem
   add circle apple mag 1,-1 ori -0.5,-1               ;figure 9a

2) (Assume tree with recursion level 5 made (see lesson 6)
   add apple tree ori 0,1 mag 0.05 rot -90 if 0
   plo tree                                            ;figure 9b
```



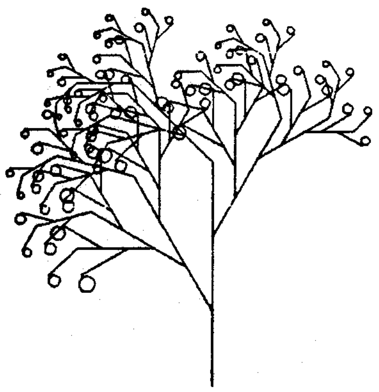


figure 9a figure 9b

Lesson 9 - Answers

1)

```
 add line square
add square square ori 1,0 rot 90 lim 4                 ;make square
softfonts                                              ;turn on software fonts
add square rect mag 2,1                                ;rectangle 2 x 1
add "Joe Bloggs" to pic1 font complexscript
add rect pic2
add pic1 pic2 bet left centre, right centre, 0.1,0.5 1.9,0.5
plot pic2                                              ;figure G-1
```

2)

```
soft                                                   ;Switch on software fonts
add rect envelope                                      ;make the envelope
add square stamp                                       ;make the stamp
add "10 cents" cost font triplexroman
add cost stamp between left centre right centre and 0.1,0.5 0.9,0.5
add stamp envelope between 0,0 1,1 1.5,0.7 1.75,0.95 ;stamp cost

add "Joe Bloggs" to name font gothicenglish            ;make the address
```

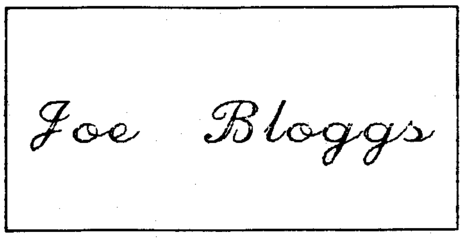


Fig. G-1

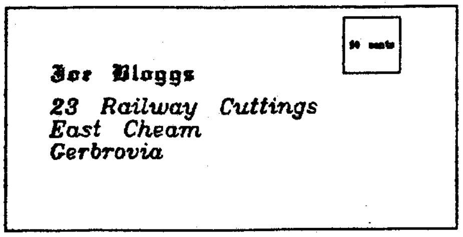


Fig. G-2

The Envelope

```
add "23 Railway Cuttings" to street font complexitalic
add "East Cheam" to town font complexitalic
add "Gerbrovia" to country font complexitalic

add name envelope origin 0.2,0.65 mag 0.1          ;Make the lettering smaller
add street envelope  origin 0.2,0.5  mag 0.1
add town envelope    origin 0.2,0.4  mag 0.1
add country envelope origin 0.2,0.3  mag 0.1
clear
plot envelope                                      ;figure G-2
bye
```

APPENDIX II - SUMMARY OF COMMANDS

Groper commands are grouped into topic headings and briefly summarized. Each summary will describe what parameters the command needs if any, and its use and limitations. A complete summary is as follows:

ADD	*AND*	*AT*	*AUXILLIARY*	
BETWEEN	*BOTTOM*	*BOUNDON*	*BY*	*BYE*
CENTRE	*CHANGE*	*CLEAR*	*COLOUR*	*COMMMAND*
DEVICE	*ERASE*	*EXIT*	*FONT*	*FROM*
HELP	*IF*	*INTERRUPT*		
LABEL	*LIMIT*	*LIST*	*LOCAL*	*LOG*
MAGNIFY	*NOPLOT*	*NOTEXT*	*OR*	*ORIGIN*
P5	*PAUSE*	*PLOT*	*POINT*	*PRINT*
REMOVE	*RESTORE*	*ROTATE*		
SCALE	*SIZE*	*SLOWHP*	*SOFT* Fonts	
TEXT	*TO*	*UNITS*	*WRITER*	
XFRAME	*YFRAME*			

PICTURE CONSTRUCTION / TRANSFORMATION COMMANDS

The following set of commands may be used in combination with each other to construct and / or add on to pictures. Most commands concern themselves with **transforming** an existing picture into an altered form.

add ***add pictureA pictureB***

PictureA is added on to pictureB. If pictureB already exists, it will become a combined version of the two, else it will become a copy of pictureA. A variety of transforms may accompany this command.

Limitations:
- PictureB may not be the picture **line**.
- Only the first six letters of a picture name are looked at. The rest are ignored.
- Defaults are Origin 0 0, limit 1, magnification 1 1 rotation 0
- The order of execution of transformation is origin, rotate, magnify and plot regardless of the order of typing.

origin ***origin x-coordinate y-coordinate***

This command specifies where the origin of the picture being transformed is to be placed in the picture being defined.

magnify ***magnify x-coordinate y-coordinate***

The X and Y coordinate of the picture being transformed are multiplied by the given amounts. This command is often used for squashing, stretching reflecting and changing the picture size.

- If only one number is given as a coordinate, then both the X and Y coordinate will take that value.

rotate *rotate degrees*

This command will rotate the picture about the origin the specified number of degrees.

between ***between x1,y1 x2,y2 x3,y3 x4,y4***

This command will pick up a picture by the points (x1,y1) and (x2,y2) and add it to an indicated picture between the points (x3,y3) and (x4,y4). Automatic rotation and magnification is done to fit the picture in.

limit *limit recursion_limit*

This command specifies the depth of recursion of a recursive picture.

local *local recursion limit*

A recursive picture may be added on to another picture with its recursion limit temporarily altered.

Limitations:
- Mutually recursive pictures will have their recursion limits unpredictably altered. This if a specific local is specified, the actual recursion levels may differ from the specified amount

if *if recursion-limit*

A picture may be added on to another picture at a given depth of recursion.

Limitations:
- One must be familiar with how Groper numbers its recursion levels to know how to use this command properly. Groper starts off the picture at the (recursion limit - 1) and goes down to 0.

INPUT / OUTPUT COMMANDS

Groper must have the ability to draw pictures on a variety of graphical devices. Each device may have different capabilities and formats. The following commands allows a user to specify when, how and where Groper should draw its pictures.

plot *plot picturename*

The plot command will draw your picture on the designated device. Groper will automatically center the picture and draw it as large as possible on a graphics terminal.

device *device device_name*

This command tells Groper where to draw the picture. For a description of devices, see Appendix III. *Device* typed by itself will list the available devices.

Limitations:
- Certain devices require that the X and Y frame be set to avoid clipping.
- If no device is given, Groper will try to draw the picture on the current terminal.

clear *clear*

This is a logical switch which tells Groper to clear the screen before it draws the picture. This is used when the user is interacting with Groper and drawing pictures on a single terminal.

xframe *xframe lowerbound upperbound*

This command scales the raster X coordinates of the Groper picture to the bounds indicated. This is used to avoid clipping of a picture and to scale the figure to a desired size. A more in-depth detailing of the parameters to the device is given in Appendix II.

yframe *yframe lowerbound upperbound*

units *units actual units*

This command is used before xframe, yframe, to set the units specified by those commands. The following actual units may be specified: *cms*, *inches*, *raster*. *Raster* is the default and indicates the units of the device.

LIBRARY COMMANDS

log *log filename*

All commands written after this point in the Groper session are saved in the named file. The net result is that picture definitions are saved. In addition, these saved files may be edited by conventional techniques. The actual file will be suffixed with a .grp.

Limitations:
- A file of that name may not already exist.
- The file name must be five letters or less.

restore *restore filename*

Groper commands in the designated file are run. The net result is that the pictures in the saved session are restored.

Limitations:
- The file being restored may also have a restore command which may also have a restore command, etc. to a maximum of 6 levels.
- Picture elements having the same name as the restored elements will be added on to.

noplot	*noplot*

This command is a logical switch which tells Groper to draw or not to draw pictures. Recommended use is before a *restore*, which causes plot commands in the restored file to be ignored.

Limitations:
- If the restored file has a *noplot* command, then the switch is turned off.

READABILITY COMMANDS

Readability commands are a set of words which makes the Groper language English-like and thus easier to read. These commands are ignored by the system and may thus appear anywhere. The following is a list of these words:

and	*at*	*by*
for	*from*	*in*
is	*to*	*with*

EDITING COMMANDS

The following commands are usually used to interactively edit pictures. The user is reminded that it is also possible to edit a saved picture with conventional techniques.

list *list*

All pictures that have been created are listed, including the predefined line.

erase *erase picturename*

The named picture is totally removed. This avoids clutter and leaves the name free for re-use.

print *print picturename*

All transforms / picture elements are listed, showing the origin of the picture element in the main picture and all the attached labels.

label *label labelname*

This allows you to label a picture element / transform with your own label. However, Groper will automatically label all of your picture elements.

Limitations:
- Only the first five letters of your label name are saved. The rest are ignored.

colour *colour label red value, green value, blue value*

The label command attaches a user defined label to a transform. The colour command allows the user to associate a colour with that label.

e.g.:

add line to picture label yellow

If a picture contains a transform of line label yellow then the command

colour yellow 100 100 0

the colour command associates 100 units of red, 100 units of green and 0 units of blue with the label. On a pen plotter the pen number would be specified by the value associated with red. Note this method of specifying colour will be rationalised on a future version of groper.

remove *remove labelname picturename*

All transforms / picture elements with the designated labelname will be removed from the named picture. This instruction is most often used to remove trial and error transforms. If you do not know the label name, use the *print* command.

MISCELLANEOUS COMMANDS

bye *bye*

This command allows you to exit Groper. A file groper.grp, which is a copy of your session is automatically saved.

help *help command*

help followed by a command gives the description of the commands found in this appendix. *help help* gives a list of the commands by category.

APPENDIX III - DEVICES AND PARAMETERS

The following is a description of some of the devices that may be used to plot a Groper picture. This description will usually be implementation - dependant. For an up to date device description, see the Groper implementer.

Available devices are:

tk	-	Tektronix 4010
ae	-	AED 512
cv	-	Corvus Cocept
rt	-	Raster Tec. 120
p5	-	Plot 5 format.
rm	-	Remote 4010
vt	-	VT100
aa	-	Ann Arbor Amb.
cc	-	Calcomp 718.
c8	-	Calcomp 81
hp	-	HP 7470A
pp	-	Paper.plt

<no device given>

If no device command is given, Groper will try to plot the picture on the current terminal. This must be a graphical terminal, otherwise meaningless characters will be produced. Clear should be switched to true to clear the screen before each plot. X and Y frames need not be set. The default terminal is the Tektronix 4010. (A Visual 550 will emulate this.)

device aa

This designates the printer port of the Ann Arbor Ambassador terminal. Any tektronix - compatible device should be able to plot the pictures correctly. X and Y frames need not be set.

device p5

This plots the picture in a file called groper.p5 in the UNIX plot5 format. The user may designate the file name by the command: *p5 filename*. X and Y frame may be set, depending on where the p5 file will eventually be plotted.

Note that the devices **aa** and **vt** are login devices not plotting devices. If a plotter or graphics terminal is plugged into the auxilliary port of one of these devices they may be accessed using the *AUX* command for example:

```
device c8

plot device c8 - Calcomp 81     selected

aux

graphical output will be sent to the auxilliary port of the
```

aa - Ann Arbor Ambassador terminal.

If device vt had been specified as the login device then the *aux* command would use the auxilliary port of the vt100.

Other devices exist, but have not yet been included in this appendix.

APPENDIX IV - SOME GROPER HINTS

Radians

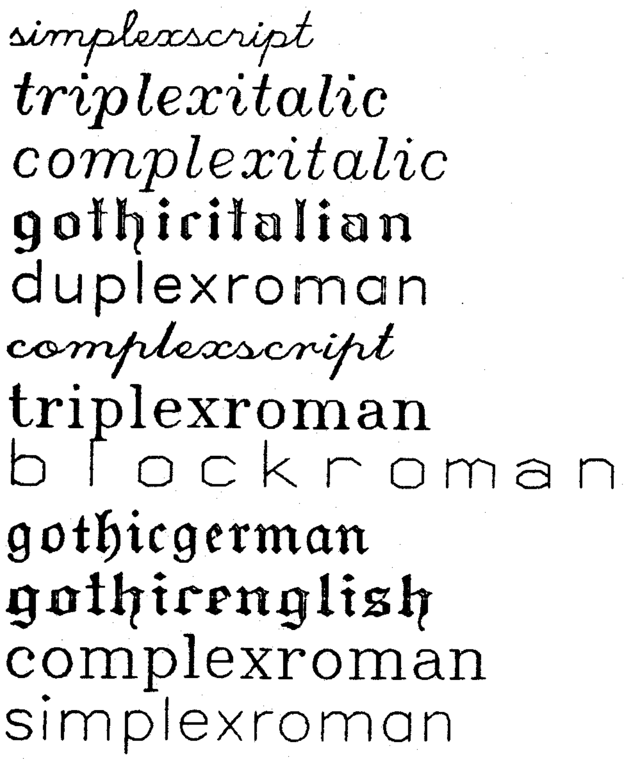
Frequently it is useful to multiply an angle by pi/180 to bring the angle to radians. This is done by simply following the angle by the letter R e.g: *add square to picture rotate by 1R* A good trick when defining a circle is to use the R convention to translate coordinates into polar notation:

add line to circle between 0,0 1,5R and 0,0 1,-5R add circle to circle rotate 10 limit 36

The above example produces a circle of radius 1 and centre at 0,0

Order of transformation commands.

Remember that whatever order the origin, rotate and magnify commands are entered for a single transformation the order will be stored as : Scaling (Magnify) Rotation Translation (Origin) If you wish to change the effective order an intermediate picture must be defined.



The Groper fonts

A final note

From the command list you can see that Groper contains commands not described here. These are mainly concerned with text and devices. The *help* command will give information on these commands.

Good luck!!

Brian Wyvill & Saul Greenberg (October 1983)