Massachusetts Institute of Technology
Artificial Intelligence Laboratory

Working Paper No. 214 May, 1981

Logo Turtle Graphics for the Lisp Machine

Henry Lieberman

This paper is a manual for an implementation of Logo graphics primitives in Lisp on the MIT Lisp Machine. The graphics system provides:

Simple line drawing and erasing using "turtle geometry"
Flexible relative and absolute coordinate systems, scaling
Floating point coordinates
Drawing points, circles, boxes, text
Automatically filling closed curves with patterns
Saving and restoring pictures rapidly as arrays of points
Drawing on color displays, creating new colors
Three dimensional perspective drawing, two-color stereo display

Table of Contents

		Page
Section 1	Introduction	1
Section 2	Initializing the Display	3
Section 3	The Turtle	4
Section 4	Moving the Turtle	6
Section 5	Turning the Turtle	7
Section 6	The Pen	8
Section 7	The Eraser	8
Section 8	Drawing in XOR Mode	9
Section 9	Examining and Modifying the Turtle's State	10
Section 10	Multiple Turtles	11
Section 11	Global Navigation	12
Section 12	Trigonometry	13
Section 13	Text	14
Section 14	Points and Circles	14
Section 15	Scaling	15
Section 16	Saving Pictures	17
Section 17	Shading	20
Section 18	Shading Patterns	21
Section 19	Color	22
Section 20	Three Dimensional Turtle	25
Index to Primitives		29

Logo Turtle Graphics for the Lisp Machine

Henry Lieberman

Artificial Intelligence Laboratory
and Laboratory for Computer Science
Massachusetts Institute of Technology

Section 1. Introduction

This paper is a manual for an implementation of Logo graphics primitives in Lisp on the MIT Lisp Machine. The graphics system provides:

Simple line drawing and erasing using "turtle geometry"
Flexible relative and absolute coordinate systems, scaling
Floating point coordinates
Drawing points, circles, boxes, text
Automatically filling closed curves with patterns
Saving and restoring pictures rapidly as arrays of points
Drawing on color displays, creating new colors
Three dimensional perspective drawing, two-color stereo display

The program has been converted to Lisp Machine Lisp from MacLisp on the PDP-10, where it was part of Lisp Logo. This paper is a revision of the section on the TV Turtle in the Lisp Logo memo (AI memo 307, Logo memo 11). Familiarity with Lisp and the Logo graphics concepts is assumed.

The primitives available are designed to be as compatible as possible with LLOGO's turtle commands for the Knight TV's, the 340 and GT40 vector displays, and 11LOGO's. Some deviance from prior implementations of the LOGO turtle was necessary, however, to take advantage of the unique features and respect the limitations of the environment. Most of the incompatibilities are noted in the descriptions of the primitives below.

One global change is that in LLOGO, most global variables accessible to the user are preceded by a colon. Since Lisp Machine Lisp uses colon to denote package prefixes, global variables have been prefixed by a star instead in this implementation. So LLOGO's :XCOR variable is here *XCOR.

The facilities described in this paper are available by loading the file AI:LLOGO;LMT ("Lisp Machine Turtle") for the black-and-white version, or AI:LLOGO;LMCT ("Lisp Machine Color Turtle") for the color version.

Section 2. Initializing the Display

STARTDISPLAY {Abbreviation: SD}

Initializes the screen. The user is supplied with a single turtle, located at the center of the screen, with its pen down and an initial heading of zero. It creates a graphics window for displaying turtle pictures. This command is also useful as a means of reinitializing and restarting everything when things get hopelessly fouled up. STARTDISPLAY should restore the entire state of the turtle's world to what it was initially.

STARTDISPLAY allows the user to create a new window on the screen to display the turtle's pictures. If the newly created window overlaps the window in which the user was typing, he is given the opportunity to reshape that window to get it out of the way. This window is kept as the variable TVRTLE-WINDOW.

STARTDISPLAY-WITH <window>

Same as STARTDISPLAY, except the user can supply a window to be used for the turtle's output.

NODISPLAY {Abbreviation: ND}

Announces the user's intention to stop using turtle primitives. NODISPLAY removes the turtle graphics window, so that program output may occur in any part of the screen. Turtle commands executed after a NODISPLAY will cause graphic output to appear, but no assurance is given that graphic output and printed output will not interfere with each other. NODISPLAY also clears the screen. Use CLEARSCREEN if you want to return to split screen mode after executing a NODISPLAY.

WIPE

Erases the picture on the screen, except that it does not affect any turtles which are being displayed.

CLEARSCREEN {Abbreviation: CS}

Equivalent to HOME WIPE, but faster.

Section 3. The Turtle

The turtle marker is displayed as an isoceles triangle, with a line from the center to the vertex between the equal sides; this line points in the direction of the heading. The triangle turtle is XOR'ed in with the displayed picture to show or hide the turtle: points which are displayed when the turtle is not over them are turned off, and points where nothing is displayed are turned on when the turtle is over them. This allows the turtle to be more visible against a background consisting of a complex picture, or shaded area. LLOGO's turtle cursor provides an extra bit of information to the user about the turtle's state: The center of the triangle indicates what will happen if the turtle is moved. If the pen is down, a filled-in box is displayed at the center of the triangle. If the eraser is down, an outlined box appears. If XOR mode is in effect, an "X" is displayed at the center of the turtle. If XOR mode is not in effect and both the pen and the eraser are up, only the triangle will be displayed. This state indicates that the turtle will not draw or erase lines when moved.

HIDETURTLE {Abbreviation: HT}

Makes the turtle disappear. Only lines drawn by the turtle will be seen, and no marker will be drawn to indicate the turtle's position and heading.

SHOWTURTLE {Abbreviation: ST}

Brings the turtle back to life. A turtle marker will be drawn to indicate the state of the turtle.

*SEETURTLE

A global variable which is T if the turtle is being displayed, else NIL. Don't modify this variable yourself using SETQ. The value should only be changed by calls to HIDETURTLE and SHOWTURTLE. Unless explicitly stated otherwise, the global variables mentioned in this section, such as *XCOR, *HEADING, and *PENSTATE, should not be directly modified by assignment. Use instead the functions which are provided for that purpose.

MAKETURTLE <draw-turtle-procedure> <erase-turtle-procedure>

This allows the user to substitute procedures for drawing the turtle marker, to be used instead of the system's default triangle turtle. This feature could be used to substitute a more lifelike picture to represent the turtle, to print state information on the screen instead of drawing a picture, or to record the turtle's wanderings. MAKETURTLE takes as input the names of two procedures, the first to be called whenever the system wants the turtle to appear, the second to be called when the system wants the turtle marker to vanish. These procedures will be called in *SEETURTLE mode when the turtle's state changes, i.e. by FORWARD, RIGHT, PENDOWN, etc. The procedures should examine the turtle state variables, such as *XCOR, *YCOR, *HEADING, *PENSTATE, etc. to decide how and where the turtle marker is to be displayed. The procedures will be executed with *SEETURTLE bound to NIL, to prevent infinite recursion. All turtle state variables are rebound during the execution of user supplied turtle marker procedures, so that you can change them in the course of drawing a turtle. If there is more than one turtle, each turtle can be given a separate set of procedures for drawing and erasing itself.

*DRAWTURTLE

Global variable containing the name of the procedure being used to draw the current turtle. NIL means the standard system triangle turtle is in use. Set by MAKETURTLE.

*ERASETURTLE

Like *DRAWTURTLE, but contains procedure used to erase the turtle.

TRIANGLETURTLE

The procedure used to draw the standard system triangle turtle. If you want to do something just slightly different than the standard turtle, you might have a procedure which calls TRIANGLETURTLE. Since TRIANGLETURTLE draws the turtle in XOR mode, the same procedure is used both to draw and to erase the turtle.

Section 4. Moving the Turtle

FORWARD <steps> {Abbreviation: FD}

Moves the turtle <steps> in the direction it is currently pointed.

BACK <steps> {Abbreviation: BK}

Moves the turtle <steps> opposite to the direction in which it is pointed.

SETX <x>

Moves the turtle to (<x>, YCOR).

SETY <y>

Moves the turtle to (XCOR, <y>).

SETXY <x> <y>

Moves the turtle to (<x>, <y>).

DELX <dx>

Moves turtle to (XCOR+<dx>, YCOR).

DELY <dy>

Moves turtle to (XCOR, YCOR+<dy>).

DELXY <dx> <dy>

Moves turtle to (XCOR+<dx>, YCOR+<dy>).

HOME {Abbreviation: H}

Moves turtle home to its starting state, at (0, 0) with a heading of 0.

WRAP

Movement of the turtle past the boundaries of the screen by FORWARD, SETXY, etc. is an error, unless WRAP is done. This causes movement off one edge to result in the turtle's reappearance at the opposite edge, as if the screen was a torus.

NOWRAP

Turns off wraparound mode. NOWRAP makes sure that the turtle's coordinates are within the boundaries of the screen. Any subsequent attempt to move beyond the boundaries of the screen will cause an error.

*WRAP

A global variable containing T iff wraparound mode is in effect, NIL otherwise.

Section 5. Turning the Turtle

RIGHT <angle> {Abbreviation: RT}

Turns the turtle clockwise <angle> degrees.

LEFT <angle> {Abbreviation: LT}

Turns the turtle counter-clockwise <angle> degrees.

SETHEAD <angle>

The turtle is turned to a heading of <angle>.

Section 6. The Pen

PENDOWN {Abbreviation: PD}

The turtle's pen is lowered. This means that if the turtle is moved, a line will be drawn between the turtle's old and new positions. A filled in box is displayed at the center of the turtle if in SHOWTURTLE mode, to show the user that the pen is down.

PENUP {Abbreviation: PU}

The pen is raised. The turtle will not draw a line when moved. If SHOWTURTLE mode is on, the filled in box displayed at the center of the turtle to indicate PENDOWN will disappear.

*PENSTATE

A global variable which is T iff the pen is down, else NIL.

Section 7. The Eraser

As well as having a "pen" which can be raised or lowered to control drawing of lines when the turtle is moved, it also has an "eraser". When the eraser is down, if the turtle retraces a line which has been previously drawn with the pen down, the line disappears. This can also be thought of as "drawing in the same color ink as the background". Note that this means that if a line is drawn with the eraser down, any point lying on that line will be turned off, even though another line might have passed through the same point.

ERASERDOWN {Abbreviation: ERD}

The eraser is lowered. When the turtle moves, lines are erased which were drawn with the pen down. Note that the pen and the eraser can't be down at the same time. ERASERDOWN therefore will automatically do a PENUP, and PENDOWN will do an ERASERUP. An outlined box is displayed at the center of the turtle when in SHOWTURTLE mode as long as the eraser is down.

ERASERUP {Abbreviation: ERU}

The eraser is raised.

*ERASERSTATE

Global variable which is T iff the eraser is down, NIL otherwise.

Section 8. Drawing in XOR Mode

In addition to drawing with the pen down, which turns on points along the line being drawn, and drawing with the eraser down, which turns off points along the line being drawn, there exists another option, useful in certain circumstances. The turtle can be used to draw in XOR mode -- points along the line being drawn are turned on if they were previously off, and off if they were formerly on. This mode of operation is used to display the triangle turtle in SHOWTURTLE mode. It allows the same procedure to draw a line and erase it, leaving what was there before it undisturbed.

XORDOWN {Abbreviation: XD}

XORUP {Abbreviation: XU}

*XORSTATE

Analogous to the corresponding primitives for the pen and the eraser.

DRAWMODE <mode>

The screen memory has a feature which enables any attempt to write a word in the memory to result in a specified boolean function of the word being written and the word previously there. DRAWMODE changes that specification according to <mode>, which should be an integer representing the mode chosen from the values of one of the following symbols: IOR, XOR, ANDC, SAME, COMP, EQV, SETO, SETZ, SET. IOR is for PENDOWN mode, ANDC for ERASERDOWN. DRAWMODE returns the number describing the mode previously in effect.

*DRAWMODE

A global variable containing the current mode number as set by the last call to DRAWMODE.

Section 9. Examining and Modifying the Turtle's State

*XCOR

A global variable containing the turtle's current X location. X coordinates increase rightward, and the origin is in the center of the screen [but can be changed via SETHOME]. This variable is always a floating point number. If wraparound mode is in effect, this variable indicates distance from the origin as if on an infinite plane. If the right edge of the screen is 500, and SETX 600 is done, *XCOR will be 600.0, but the turtle will appear 400 units to the left of the origin.

*YCOR

Like *XCOR, but holds the value of the Y coordinate. Y coordinates increase upward.

*HEADING

Holds the value of the turtle's heading, in floating point. A heading of zero corresponds to pointing straight upward, and heading increases clockwise. This variable always gives the absolute heading, not reduced modulo 360. After SETHEAD 400, *HEADING is 400.0, not 40.0, although the turtle is pointing in the same direction as SETHEAD 40.

XCOR

Outputs the X coordinate of the turtle as an integer. If wrapaound mode is in effect, this function will output the position of the turtle as it appears on the screen. After SETX 600, XCOR would return -400.

YCOR

Like XCOR, but outputs the Y coordinate of the turtle.

HEADING

Outputs the heading of the turtle as an integer, modulo 360. After SETHEAD 400, HEADING would return 40.

HERE

Outputs (LIST XCOR YCOR HEADING). Useful for remembering the turtle's state via (SETQ 'TURTLESTATE (HERE)). A turtle state saved in this manner can be restored using SETTURTLE.

MOUSE-HERE

Returns a list of the X and Y positions of the mouse.

SETTURTLE <state> {Abbreviation: SETT}

Sets the state of the turtle to <state>. <state> is a sentence of X coordinate, Y coordinate, and heading. The heading may be omitted, in which case it is not affected. SETTURTLE is the inverse of HERE.

Section 10. Multiple Turtles

Initially, the user is supplied by LOGO with one unique turtle, which remembers its position and heading, and is capable of drawing or erasing lines when moved. The ability to create any number of these creatures and to switch the attention of the system between them makes possible such things as assigning a turtle locally to each one of several programs.

HATCH <turtle-name>

Creates a new turtle, christened <turtle-name>. The turtle created by HATCH starts out in a state identical to that of the original turtle present after a STARTDISPLAY; It is located at its home, at the center of the display area, its heading points straight up, and its pen is down. The newly created turtle becomes the current turtle, and will respond to all turtle commands. The state of any previously created turtle, including the one originally supplied by STARTDISPLAY, remains unaffected by HATCH, or any turtle command referring to the new turtle.

USETURTLE <turtle-name> {Abbreviation: UT}

Selects the named turtle to be the current turtle; this means that all subsequent turtle commands [FORWARD, RIGHT, ...], and turtle state variables [*HEADING, *XCOR, *YCOR, ...] now will refer to the selected turtle until changed again by another call to USETURTLE or a call to HATCH. The state of the previously selected turtle is preserved so that if it is ever selected again, its state will be restored. The turtle which is provided initially by STARTDISPLAY is named LOGOTURTLE.

*TURTLE

Global variable which contains the name of the currently selected turtle.

*TURTLES

Global variable which contains a list of the names of all the turtles in existence.

Section 11. Global Navigation

BEARING, RANGE, and TOWARDS return integers if all inputs are integers, otherwise they return floating point numbers. The numbers returned are always positive, and BEARING and TOWARDS return headings modulo 360.

RANGE <x> <y>

RANGE <sentence-of-x-and-y>

Outputs the distance from the turtle to a point specified either by two inputs which are x and y coordinates respectively, or by a sentence of x and y coordinates.

BEARING <x> <y>

BEARING <sentence-of-x-and-y>

Outputs the absolute direction from the turtle to a point specified in a format acceptable to RANGE. (SETHEAD (BEARING <x> <y>)) points the turtle in the direction of (<x>,<y>).

TOWARDS <x> <y>

TOWARDS <sentence-of-x-and-y>

Outputs the relative direction from the turtle to the point specified. (RIGHT (TOWARDS <x> <y>)) points the turtle in the direction of (<x>, <y>).

Section 12. Trigonometry

COSINE <angle>

Cosine of <angle> degrees.

SINE <angle>

Sine of <angle> degrees.

ARCTAN <x> <y> {Abbreviation: ATANGENT}

Angle whose tangent is <x>/<y>, in degrees.

[SIN, COS, and ATAN are the corresponding functions which input or output in radians]

Section 13. Text

MARK <text>

Prints text in at the turtle's current location. PRINC is used to print the text (rather than PRINT).

Section 14. Points and Circles

[These are displayed whether or not the pen or the eraser is down]

POINT

Displays a point at the turtle's current location.

POINT <T or NIL>

Turns the point at HERE on if its input is not NIL, off if it is NIL.

POINT <x> <y> <T or NIL>

Turns the point at (<x>, <y>) on or off as specified by its input. The third input is optional, and defaults to T [e.g., turn the point on] if omitted.

Note: These conventions for POINT differ slightly from those used in the LLOGO 340 turtle, to accommodate the capability of turning a point off as well as on.

POINTSTATE

Returns T or NIL, depending on whether the point at the turtle's current location is on or off. The turtle marker is hidden temporarily during the execution of POINTSTATE, so that display of the turtle will not interfere with the point being tested. POINTSTATE will return whether the point being tested is on, regardless of how it was caused to appear -- by a line drawn by the turtle, text printed, shading, etc.

POINTSTATE <x> <y>

Tests the point at the specified coordinates.

ARC <radius> <degrees>

Draws an arc of a circle of the given radius, and extending for the given number of degrees around the circle centered on the turtle's current location. The arc drawn begins at the point on the circle where the turtle's heading is pointing, and is drawn in a clockwise direction [in the direction of increasing heading].

CIRCLE <radius>

Equivalent to (ARC <radius> 360).

Section 15. Scaling

Two functions are provided for changing the size of the graphic display area at the top of the screen and the area for typein and typeout at the bottom of the screen, and the dimensions of the display area in turtle coordinates. TVSIZE controls the actual size of the display area, and operates in terms of raster display points. TURTLESIZE is used to establish the mapping from the specified TVSIZE into turtle coordinates -- the numbers given to and returned by the turtle primitives. It does not have any effect on the visual size of the area used for graphic display output.

TVSIZE

Returns a list containing the horizontal and vertical sizes of the display area in raster points.

TVSIZE <new-size>

Sets both the horizontal and vertical sizes of the display area to <new-size>. Modifying the TVSIZE causes a CLEARSCREEN to be performed. The size of the area at the bottom of the screen for typein and typeout is adjusted to take up as much space as possible on the screen not being used for graphic output. Changing the TVSIZE will not have any effect on pictures previously saved by MAKEWINDOW [see Section 16].

TVSIZE <new-x-size> <new-y-size> Sets the horizontal and vertical sizes independently. If either of the two inputs is NIL, the corresponding size remains unchanged.

TURTLESIZE

Returns a list containing the horizontal and vertical sizes of the display area in turtle coordinates. These are in floating point. The initial default is 1000 x 1000, and the origin is always at the center of the screen -- so turtle coordinates initially range from -500 to +500. If wraparound mode is in effect, turtle coordinates are allowed above and below the range set by TURTLESIZE, and will be mapped to appropriate points on the screen.

TURTLESIZE <new-size>

Sets the dimensions of the screen in turtle coordinates to <new-size> turtle steps. If the display area is not square [that is, if the horizontal and vertical TV size parameters are not equal], then <new-size> is taken to be the number of turtle steps for the minimum dimension of the screen, and the other dimension is adjusted accordingly. In particular, you can't specify TURTLESIZE independently in each direction, so that a turtle step always corresponds to the same number of TV points. Changing TURTLESIZE has no effect on the picture currently being displayed, or on any pictures saved by MAKEWINDOW.

SETHOME {Abbreviation: TURTLEHOME}

SETHOME <new-x-home> <new-y-home> {Abbreviation: TH}

Changes the origin of turtle coordinates to the specified location, defaulting to the turtle's present position. That position on the screen will then correspond to an XCOR and YCOR of zero for all subsequent turtle commands. The home location is local to each turtle, so that each of several turtles may be assigned different homes on the screen.

TV-X <turtle-x>

TV-Y <turtle-y>

TURTLE-X <tv-x>

TURTLE-Y <tv-y>

Conversion between TV and turtle coordinate systems.

Section 16. Saving Pictures

In creating pictures which consist of repeating patterns of smaller pictures, and creating animated cartoons, it is often useful to be able to save displayed pictures drawn by a series of turtle commands, and operate upon them as a unit, displaying and erasing them, moving them to other parts of the screen, etc. We provide such a facility, allowing the user to save rectangular portions of the screen as arrays of points. These arrays can be displayed and erased at any location on the screen, although they cannot be automatically rotated.

These saved pictures are called "windows", not to be confused with the Lisp Machine system windows. (The terminology conflict is unfortunate, but the turtle graphics software predates the Lisp Machine.)

This facility is somewhat different from the SNAP command in the LLOGO 340 turtle and 11LOGO. The SNAP operation saves the picture as display lists, essentially a vector representation, while the window saves an array of points. For large, sparse pictures, the vector representation consumes less space, while the point array representation favors small, complex pictures. Saving point arrays makes it possible to redisplay pictures much more rapidly than redrawing them with the commands used to originally generate the picture, since recomputation of points lying along vectors is unnecessary. It is therefore ideal for programs which want to make only few, spatially localized changes to a picture, but need the maximum possible speed for dynamic updating of the screen. It also has the advantage that the amount of space and time used for creating and redisplaying pictures is insensitive to the complexity of a picture within an area. These characteristics make an array representation more suitable than a vector representation for, say, a space war program, where the space ship must be redisplayed rapidly, and consists of perhaps a large number of vectors confined to a small area of the screen. It also provides a "clipping" facility.

Saving point arrays has a property not shared by picture saving commands on vector displays -- "What you see is what you get". Everything within the designated area is included, regardless of how it was caused to appear -- vectors, text, points, other WINDOWs, etc. This means that you can always tell what will be included in a saved picture simply by looking at the screen.

MAKEWINDOW <name> <size> {Abbreviation: MW}

Creates a "window", i.e., an array of points, and names it <name>. The <name> should be a word, and should be chosen so as not to conflict with existing functions or arrays. The window is centered on the turtle's current location, and extends for <size> turtle steps horizontally and vertically from the center. The location of the center of the window and its size are remembered.

MAKEWINDOW <name> <x-size> <y-size>

Creates a window centered on the turtle's current location, but sets the horizontal and vertical sizes of the window independently, so the area saved can be rectangular instead of square, as in the one input mode.

MAKEWINDOW <name> <x> <y> <x-size> <y-size>

Creates a window centered on the specified location, of the specified size. If the <y-size> is omitted it is assumed identical to the <x-size>.

ERASEWINDOW <name> {Abbreviation: EW}

Destroys the window specified by <name>. If the window is no longer needed, this permits the space that it occupied to be reclaimed.

ERASEWINDOWS {Abbreviation: EWS}

Erases all currently defined windows.

*WINDOWS

Global variable which contains a list of all currently defined windows.

WINDOWFRAME . . . . {Abbreviation: WF}

Takes inputs like MAKEWINDOW, except for the window name. That is, it takes from one to four inputs specifying a size and optionally a center location. WINDOWFRAME displays a box on the screen which indicates the extent of the picture which would be saved by a MAKEWINDOW of the corresponding size and location. This is useful in deciding how large a window is necessary before using MAKEWINDOW. The box is XORed into the screen, so that giving the WINDOWFRAME command again will cause the box to disappear. If no inputs are given to WINDOWFRAME the size and location default to the last ones specified.

SHOWWINDOW <name> {Abbreviation: SW}

Causes the specified window to be displayed at the location at which it was originally created. Currently, wraparound is not allowed; display of the picture is not allowed to cross the edge of the display area. Changing TVSIZE and TURTLESIZE have no effect on the size of saved pictures.

SHOWWINDOW <name> <new-center-x> <new-center-y>

Causes the window to be displayed at the new location specified.

HIDEWINDOW . . . . {Abbreviation: HW}

Accepts arguments like SHOWWINDOW, but displays the window turning off any point which was on in the window when it was created. The effect of this is as if the picture were redrawn in eraser mode. If a call to SHOWWINDOW displayed the window on a blank area, a similar call to HIDEWINDOW will erase it. If SHOWWINDOW superimposed the window on something already displayed, the old picture is not guaranteed to remain intact after the window is hidden.

XORWINDOW . . . . {Abbreviation: XW}

Like SHOWWINDOW and HIDEWINDOW, but XOR's the picture into the screen.

WINDOWHOME <name> {Abbreviation: WH}

WINDOWHOME <name> <new-x-home> <new-y-home>

Changes the home location associated with a window to the specified location, defaulting to HERE. This is the location where the center of the window will be displayed if only the name of the window is given as input to SHOWWINDOW, HIDEWINDOW, etc.

SAVEWINDOWS <filespec> {Abbreviation: SWS}

Creates a file on the disk which saves all currently defined windows in binary. They can be reloaded at a later time with GETWINDOWS. The file specification follows the same format as other LLOGO file commands such as READFILE, and LISP's UREAD. The filenames are not evaluated.

GETWINDOWS <filespec> {Abbreviation: GW}

Reloads windows from a disk file created by SAVEWINDOWS.

EXPAND <small window> <expansion factor> <big window>

Expands the size of a window, creating a new window which is larger by some integer number of times. The last argument is the name of the new window. It expands the window by replicating each point horziontally and vertically.

Section 17. Shading

A unique advantage of the TV displays over vector oriented displays is that in addition to the display of line drawings, they make feasible the creation of pictures using shaded areas. Patterns of points of varying densities can be used to fill regions, creating the effect of a "gray scale". Our shading facility is aimed toward creating a convenient and efficient means of specifying areas to be shaded, and patterns to be used in shading. The basic idea is that regions to be shaded are indicated by drawing a closed curve around them in PENDOWN mode, and placing the turtle inside the region before issuing the SHADE command, with an argument determining the pattern to be used. Several simple patterns are supplied by the system, but the user has the opportunity of defining new ones.

SHADE <pattern name>

Shades the area enclosing the turtle's current location. The input is a pattern to be used in shading the area, and defaults to the SOLID pattern if omitted. The turtle must be sitting in an empty area [not on a line or in a filled in region], or an error results. The effect of this primitive is to fill in the region surrounding the turtle's location with the shading pattern given [by inclusive OR'ing it in with the existing picture]. The region to be shaded must be bounded by a closed curve; SHADE works by filling in the pattern starting from the turtle's location, and stopping when a boundary is reached. If the region is not closed, the entire screen will be shaded!

Section 18. Shading Patterns

Shading patterns are represented as functions which tell the SHADE primitive how to shade an area. The system provides a group of predefined shading patterns, described below. These will probably be sufficient for most simple uses of shading, i.e. distinguishing a few neighboring regions with different shading patterns, etc. Those needing more sophisticated capabilities can define their own patterns. The predefined shading patterns are slightly different in this implementation than in LLOGO.

SOLID

A shading pattern which fills in every point. This pattern is the default used if no argument is given to SHADE.

CHECKERS

A pattern which makes a checkerboard by filling in alternating squares.

LINES

A pattern consisting of vertical lines.

CIRCLES

Repeating small open circles, packed close together.

DOTS

Small filled in circles.

PIGNOSES

(I'm not responsible for this one, blame Bernie Greenberg!)

The name of a window may also be given to SHADE for use as a shading pattern. This provides the capability of using arbitrary pictures as shading patterns. The effect will be to fill the closed curve to be shaded with the picture specified by the window. If area beyond the extent of the original picture is to be shaded, the picture will be repeated horizontally and vertically as many times as is necessary to fill the area.

The representation of a shading pattern is as a function of three arguments, so that it is possible for a user to supply his own function to SHADE. The function has the responsibility of shading in a horizontal line, given starting X and Y coordinates, and ending X coordinate, in TV points. (This differs slightly from LLOGO.)

Section 19. Color

This section describes a version of the turtle for machines equipped with a color graphics display.

Some of the primitives listed above for the black-and-white version are not available with the color version. These include XOR mode, XGP output, and MARK (which may be implemented later). Shading patterns (except SOLID) don't work in color yet.

Colors are created by specifying the amount of red, green, and blue light which make the color. The colors WHITE, BLACK, RED, GREEN, BLUE, YELLOW, MAGENTA, CYAN are supplied initially, and new ones can be created

MAKECOLOR <color name> <red> <green> <blue> {Abbreviation: MC}

Where <color name> is an atomic name for the color, and <red>, <green>, and <blue> are floating point numbers between 0.0 and 1.0 which say how much of each of the primaries is included in the color being defined. YELLOW was defined by (MAKECOLOR 'YELLOW 1.0 1.0 0.0).

*COLORS

A list of all currently defined colors.

REDPART <color name>

GREENPART <color name>

BLUEPART <color name>

These retrieve the corresponding intensities of the primaries in that color. They are between zero and one, as for MAKECOLOR.

The system always keeps track of two default colors, the PENCOLOR and the ERASERCOLOR. The PENCOLOR is the color the turtle draws in, so that all lines drawn by the turtle when its pen is down appear on the screen in that color. The initial pen color is WHITE.

PENCOLOR <color name> {Abbreviation: PC}

Changes the pen color, where <color name> has been previously defined. There can, of course, be many different turtles, each with its own color pen. The ERASERCOLOR is the color used by the turtle to draw when its eraser is down. It is used as the background color, so lines drawn with the pen down are made to disappear by drawing them in the same color as the background. CLEARSCREEN erases everything on the screen, filling the screen with the ERASERCOLOR.

ERASERCOLOR <color name> {Abbreviation: ERC}

This changes the eraser color, and has the effect of immediately changing the background color on the screen as well, whereas changing PENCOLOR only affects future drawing by the turtle. [maybe this is wrong?] The initial eraser color is black.

*PENCOLOR

*ERASERCOLOR

Variables which hold the current pen color, and eraser color.

POINTCOLOR

Returns the name of the color of the point specified.

POINTSTATE

If the point is in the eraser color, POINTSTATE will return NIL. This is for compatibility with the black and white version, so that conditionals like IF POINTSTATE ... will be true if the point is in some color other than the background color. Similarly, PENCOLOR NIL will choose the current eraser color.

There is a limit of 16. different colors visible on the screen at once, including the *PENCOLOR and *ERASERCOLOR [More precisely, you can only do PENCOLOR with 15. different colors between CLEARSCREEN's]. The currently visible colors are kept in an array called the PALETTE, and colors are added to free slots in the PALETTE whenever you do a PENCOLOR mentioning a color not included in the PALETTE. Clearing the screen removes all colors from the palette except for the current pen and eraser colors. The index of the *PENCOLOR in the PALETTE is *PENNUMBER and the index of the *ERASERCOLOR is *ERASERNUMBER [*ERASERNUMBER never changes]. The following primitives allow explicit manipulation of the current set of colors.

REPLACECOLOR <old color> <new color> {Abbreviation: RC}

changes all visible drawings currently on the screen in <old color> to be instantly changed to <new color>. It is much faster than changing the PENCOLOR and redrawing the objects.

MAKEPALETTE <number> <color>

Changes the color in position <number> in the PALETTE to be <color>. It is like REPLACECOLOR of whatever color was formerly in that position.

PENCOLOR will also accept a number as input, in which case it will select whatever color is in that slot in the PALETTE, i.e. it is like (PENCOLOR PALETTE <number>).

*CAREFULTURTLE

This is a flag controlling whether displaying the triangle turtle cursor saves the picture underneath it. It is normally on, so that displaying the turtle cursor doesn't mess up the picture. However, this feature is expensive, so setting this switch to NIL will disable it when speed is more important.

Section 20. Three Dimensional Turtle

There's also a three dimensional version of Logo graphics. On the color display, it draws perspective views of three dimensional drawings in two colors, red and blue. By wearing special glasses, you can view the picture in depth, like the old 3D movies. Jim Stansfield programmed an early version of a 3D turtle in 11LOGO.

The 3D turtle lives on AI:LLOGO;3D.

The turtle cursor is displayed as a little "airplane", instead of a triangle for the flat turtle, and the commands fly the airplane thru the three dimensional space. There are three kinds of rotations, one for each axis. The heading and rotations are kept as matrices, matrix multiplication used to compose rotations. The heading is the rotation transformation necessary to align the coordinate system with the direction in which the turtle is facing.

WALKFORWARD <steps> {Abbreviation: WFD}

Like FORWARD, moves in the direction the turtle is currently pointed, drawing if the pen is down. The "nose" of the airplane representing the turtle points in the direction of the heading.

WALKBACK <steps> {Abbreviation: WBK}

Like BACK.

`FALLFORWARD <turns> {Abbreviation: FFD}`

The turtle falls forward some number of degrees. Like "pitch" on an airplane.

`FALLBACK <turns> {Abbreviation: FBK}`

Rotate in the opposite direction to FALLFORWARD.

`LEANRIGHT <turns> {Abbreviation: LRT}`

`LEANLEFT <turns> {Abbreviation: LLT}`

Leans to the right or left, like "yaw". This rotation doesn't change the way the turtle will draw if a WALKFORWARD immediately follows, since it rotates the turtle about the axis of its heading.

`TURNRIGHT <turns> {Abbreviation: TRT}`

`TURNLEFT <turns> {Abbreviation: TLT}`

Turns to the right or left, like "roll". This is like 2D RIGHT and LEFT.

`3DSETHEADING <new-heading> {Abbreviation: 3DSH}`

`3DHEADING`

The heading is a list of nine numbers, representing a three-by-three matrix. These functions access and retrieve it.

```
3DSETXYZ <x> <y> <z> {Abbreviation: SETXYZ}
3DXCOR
3DYCOR
3DZCOR
3DSETX <x>
3DSETY <y>
```

3DSETZ <z>
3DSETTURTLE <turtle-state> {Abbreviation: 3DSETT}
3DDELX <x>
3DDELY <y>
3DDELZ <z>

Like SETXY, etc., but takes a z, too.

3DHOME
3DPENUP {Abbreviation: 3DPU}
3DPENDOWN {Abbreviation: 3DPD}
3DERASERUP {Abbreviation: 3DERU}
3DERASERDOWN {Abbreviation: 3DERD}
3DXORUP {Abbreviation: 3DXU}
3DXORDOWN {Abbreviation: 3DXD}
3DWRAP
3DNOWRAP
3DCLIP
3DNOCLIP
3DSHOWTURTLE {Abbreviation: 3DST}
3DHIDETURTLE {Abbreviation: 3DHT}
3DDRAWSTATE
3DSTARTDISPLAY {Abbreviation: 3DSD}
3DWIPECLEAN {Abbreviation: 3DWIPE}
3DCLEARSCREEN {Abbreviation: 3DCS}
3DPOINT
3DARC <radius> <degrees>
3DCIRCLE <radius>

These are all like their 2D counterparts.

DISPARITY <new-disparity>

*DISPARITY

The disparity is the distance by which the two images are shifted with respect to each other. Higher values give more depth but make it harder to fuse the two images.

EYEDISTANCE <new-eyedistance>

*EYEDISTANCE

The distance "between your eyes".

*APERTURE

How much of the 3d space the observer can view.

*LEFTCOLOR

*RIGHTCOLOR

The colors used to display the views of each turtle. Currently RED and BLUE. You should change it if you have red and green glasses instead.

Index to Primitives

*APERTURE 28
*CAREFULTURTLE 25
*COLORS 23
*DISPARITY 27
*DRAWMODE 10
*DRAWTURTLE 5
*ERASERCOLOR 24
*ERASERSTATE 9
*ERASETURTLE 5
*EYEDISTANCE 28
*HEADING 4, 10, 12
*LEFTCOLOR 28
*PENCOLOR 24
*PENSTATE 4, 8
*RIGHTCOLOR 28
*SEETURTLE 4
*TURTLE 12
*TURTLES 12
*WINDOWS 19
*WRAP 7
*XCOR 4, 10, 12
*XORSTATE 9
*YCOR 10, 12

3DARC 27
3DCIRCLE 27
3DCLEARSCREEN 27
3DCLIP 27
3DCS 27
3DDELX 27
3DDELY 27
3DDELZ 27
3DDRAWSTATE 27
3DERASERDOWN 27
3DERASERUP 27
3DERD 27
3DERU 27
3DHEADING 26
3DHIDETURTLE 27
3DHOME 27
3DHT 27
3DNOCLIP 27
3DNOWRAP 27
3DPD 27
3DPENDOWN 27
3DPENUP 27
3DPOINT 27
3DPU 27
3DSD 27
3DSETHEADING 26
3DSETT 27
3DSETTURTLE 27
3DSETX 26
3DSETXYZ 26
3DSETY 26
3DSETZ 27
3DSH 26
3DSHOWTURTLE 27
3DST 27
3DSTARTDISPLAY 27
3DWIPE 27
3DWIPECLEAN 27
3DWRAP 27
3DXCOR 26
3DXD 27
3DXORDOWN 27
3DXORUP 27
3DXU 27
3DYCOR 26
3DZCOR 26

ARC 15
ARCTAN 13
ATANGENT 13

BACK 6
BEARING 13
BK 6
BLUEPART 23

CHECKERS 21
CIRCLE 15
CIRCLES 22
CLEARSCREEN 4
COSINE 13
CS 4

DELX 6
DELXY 6
DELY 6
DISPARITY 27
DOTS 22
DRAWMODE 9

ERASERCOLOR 23
ERASERDOWN 8
ERASERUP 9
ERASEWINDOW 18
ERASEWINDOWS 18
ERC 23
ERD 8
ERU 9
EW 18
EWS 18
EXPAND 20
EYEDISTANCE 28

FALLBACK 26
FALLFORWARD 26
FBK 26
FD 6
FFD 26
FORWARD 6, 7, 12

GETWINDOWS 20
GREENPART 23
GW 20

H 7
HATCH 12
HEADING 11
HERE 11, 14, 20
HIDETURTLE 4
HIDEWINDOW 19
HOME 4, 7
HT 4
HW 19

LEANLEFT 26
LEANRIGHT 26
LEFT 7
LINES 21
LLT 26
LOGOTURTLE 12
LRT 26
LT 7

MAKECOLOR 23
MAKEPALETTE 24
MAKETURTLE 5
MAKEWINDOW 15, 16, 18, 19
MARK 14

MC 23
MOUSE-HERE 11
MW 18

ND 3
NODISPLAY 3
NOWRAP 7

PC 23
PD 8
PENCOLOR 23
PENDOWN 8
PENUP 8
PIGNOSES 22
POINT 14
POINTCOLOR 24
POINTSTATE 14, 15, 24
PRINC 14
PRINT 14
PU 8

RANGE 13
RC 24
READFILE 20
REDPART 23
REPLACECOLOR 24
RIGHT 7, 12, 13
RT 7

SAVEWINDOWS 20
SD 3
SETHEAD 7, 13
SETHOME 10, 16
SETQ 4
SETT 11
SETTURTLE 11
SETX 6
SETXY 6, 7
SETXYZ 26
SETY 6
SHADE 21
SHOWTURTLE 4
SHOWWINDOW 19
SINE 13
SNAP 17
SOLID 21
ST 4
STARTDISPLAY 3, 12
STARTDISPLAY-WITH 3
SW 19
SWS 20

TH 16
TLT 26
TOWARDS 13
TRIANGLETURTLE 5
TRT 26
TURNLEFT 26
TURNRIGHT 26
TURTLE-X 17
TURTLE-Y 17
TURTLEHOME 16
TURTLESIZE 16
TV-X 16
TV-Y 17
TVSIZE 15

UREAD 20
USETURTLE 12
UT 12

WALKBACK 25
WALKFORWARD 25
WBK 25
WF 19
WFD 25
WH 20
WINDOWFRAME 19
WINDOWHOME 20
WIPE 3
WRAP 7

XCOR 6, 10, 16
XD 9
XORDOWN 9
XORUP 9
XORWINDOW 19
XU 9
XW 19

YCOR 6, 11, 16