Use of MaLT

The final version of MaLT consists of five components namely:

- *Turtle Scene (TS).* The main part of this component provides representations of geometric objects allowing navigation within the 3D virtual space (Figure 1).

- *Logo Editor (LE).* This component is the Logo-like programming interface. The user is able to write, run and edit Logo programs. For a detailed description of all the available commands, see the Specifications of the First Version of MaLT, www.remath.cti.gr in Forum, WP2) (Figure 1).

- *Variation Tools*. This component is the dynamic manipulation feature of the computer environment providing means to represent and handle variation of variable values. In the final version of MaLT it consists of the *Uni-dimensional Variation Tool (1dVT), the Two-dimensional Variation Tool and the Vector Variation Tool.*

  - *Uni-dimensional Variation Tool (1dVT).* This component provides means to handle and represent variation of variable values in MaLT. The main part of this component consists of 'number line'-like sliders, each corresponding to one of the variables used in a Logo procedure. 1dVT is thus energized only when the user defines a Logo procedure, run it for a specific value for each variable and then click on any part of the figure (Figure 2). In the first version the menu is only in greek but all the available commands in english.

  - *Two-dimensional Variation Tool.* The 2dVT is a 2d bi-axial coordinate system with two axies (x and y) in the form of an orthogonal pad. It can be used for defining the co-variation of two variables selected by the user and thus it appears only if we edit and define a procedure with at least two variables.

  - *Vector Variation Tool (VVT).* The VVT allows the co-variation of three variables by using two 2d representations of a vector defined by these variables according to an (r, φ, θ) polar semantic in 3d space (for a detailed presentation of the VVT see in pages 6-8 of the manual).

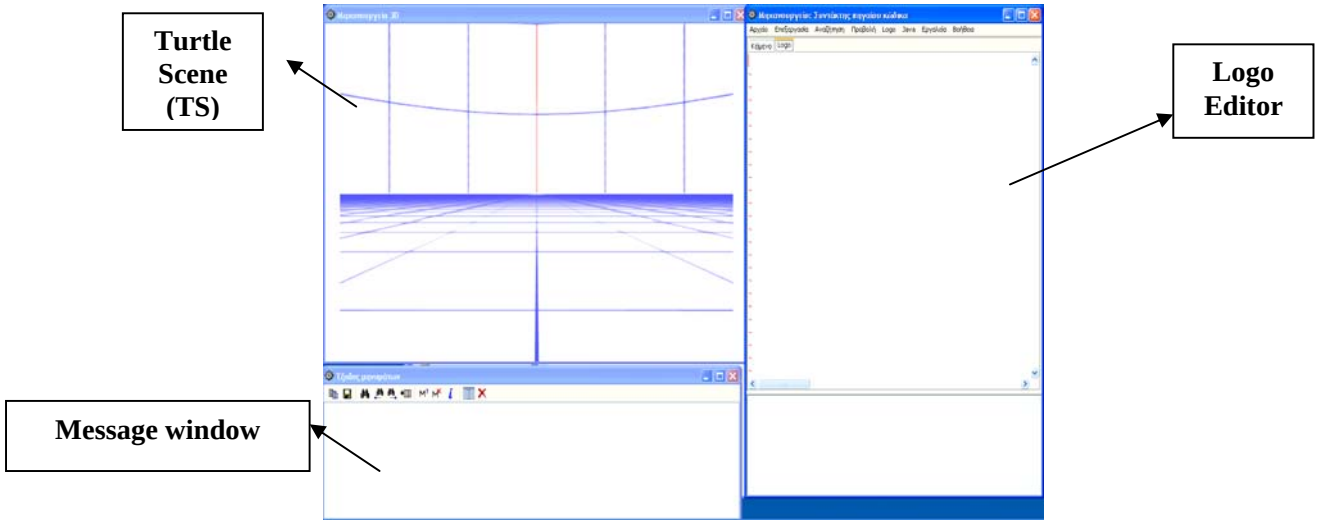

Figure 1: The main part of the final version of MaLT without any procedure (and thus any of the variation tools).

First use instructions

1. ***The turtle.*** When the user opens Malt the turtle doesn't appear. He/she must write at least one command at the Logo Editor to see the result on the TS (top left screen).

2. ***Execution of commands.*** Execution of commands is achieved by pressing **Insert** when the cursor is in the same line with the command.

3. ***Working with 1dVT.*** The user has to write and run a Logo procedure with at least one variable.

Step 1: Write a Logo procedure with at least one variable.

For example:

```
to rect :a :b :c
repeat 2 [fd(:a) rt(:c) fd(:b) rt(180-:c)]
end
```

Step 2: Run this procedure for one value for each variable.

Write the name of the procedure and one arithmetic value for each variable.
For the above procedure, e.g. `rect 20 30 40`
Then press F5.
The figure (rectangle) corresponding to these specific values will be designed in TS.

Step 3: Click on any part of the figure. (Be careful not to resize the TS before clicking on the figure).

After a while the 1dVT will appear on the left bottom of the screen.

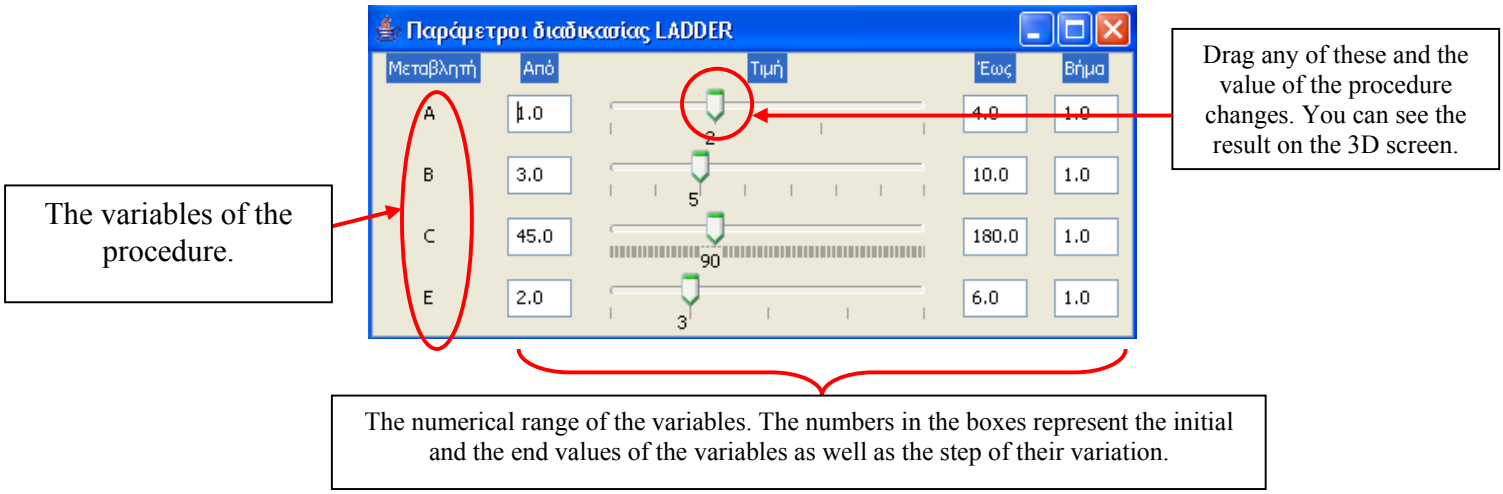

Figure 2. The 1dVT.

4. Cleaning the TS so as to execute new commands/procedures.

See at the following figure.

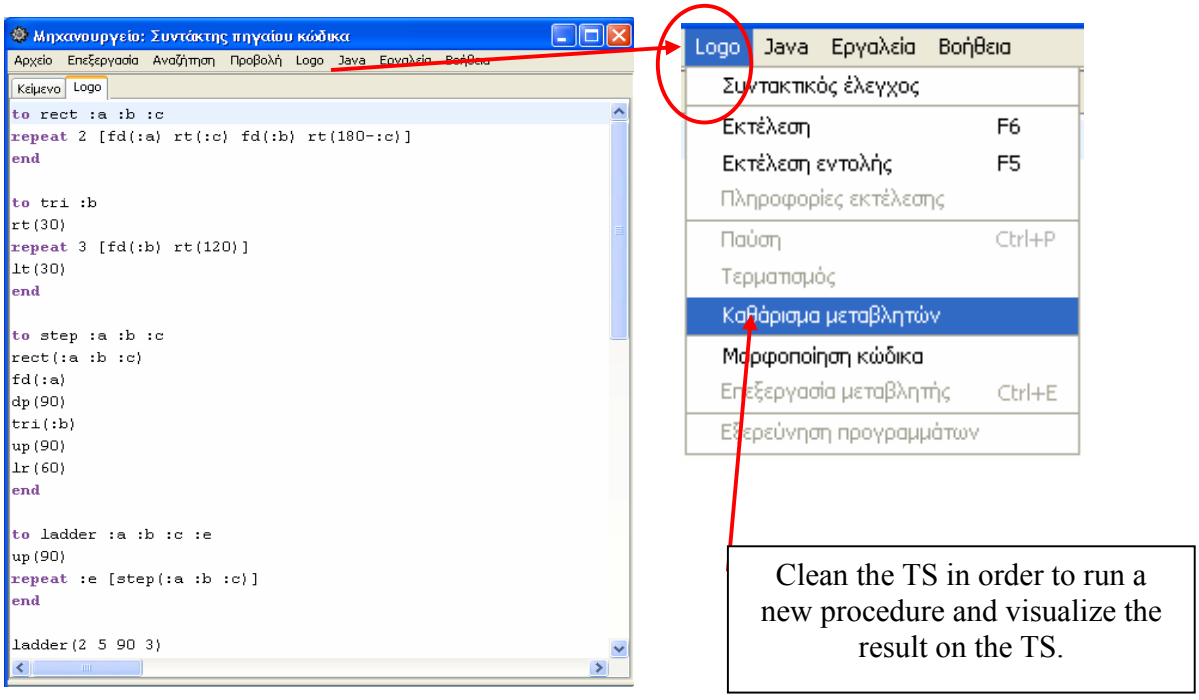

Figure 3. Cleaning the TS.

5. Opening and Saving a File. See at the following Figure.

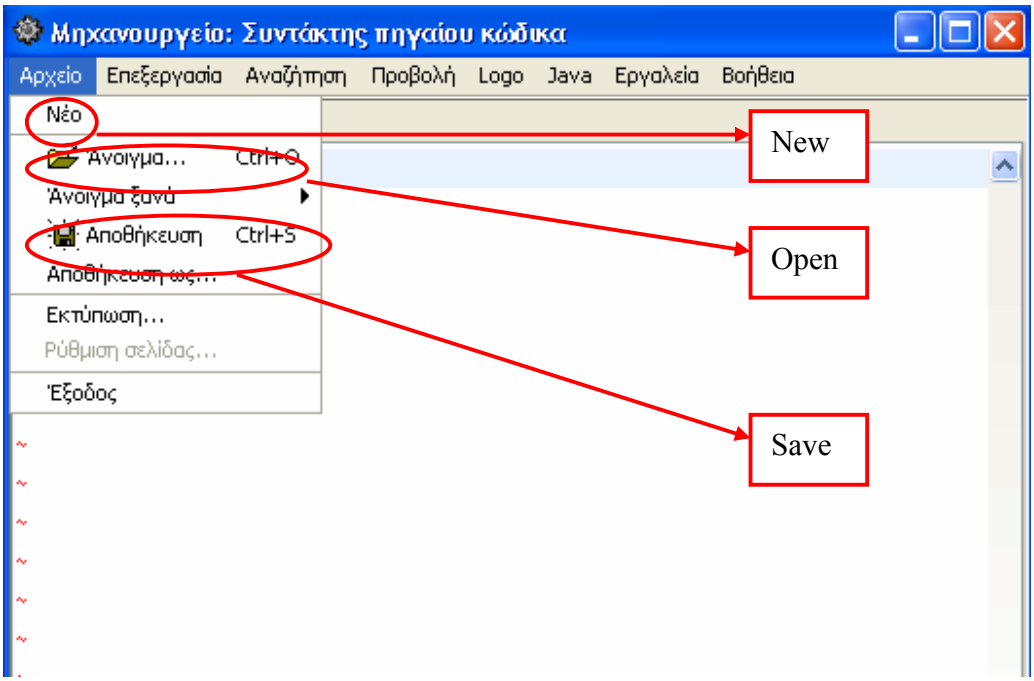

Figure 4. Opening and saving a file.

6. Working with 2dVT

The 2dVT is a 2d bi-axial coordinate system with two axies (x and y) in the form of an orthogonal pad. It can be used for defining the co-variation of two variables selected by the user and thus it appears only if we edit and define a procedure with at least two variables. It is activated through the 1dVT after selecting two variables and clicking on one of the two axes of an icon representing an orthogonal bi-axial system, which is placed next to each slider.

In the following example (see next Figure) we have selected in 1dVT variable ‘a’ to represent the y-axis in the 2dVT (clicking on the vertical axis in the ‘little’ orthogonal bi-axial system next to the slider for variable ‘a’ in 1dVT) and variable ‘b’ to represent the x-axis in 2dVT (clicking on the horizontal axis in the ‘little’ orthogonal bi-axial system next to the slider for variable ‘b’ in 1dVT). Clicking on these ‘little’ axies for the selection of the respective variables there has been a change in their colour from red to green.

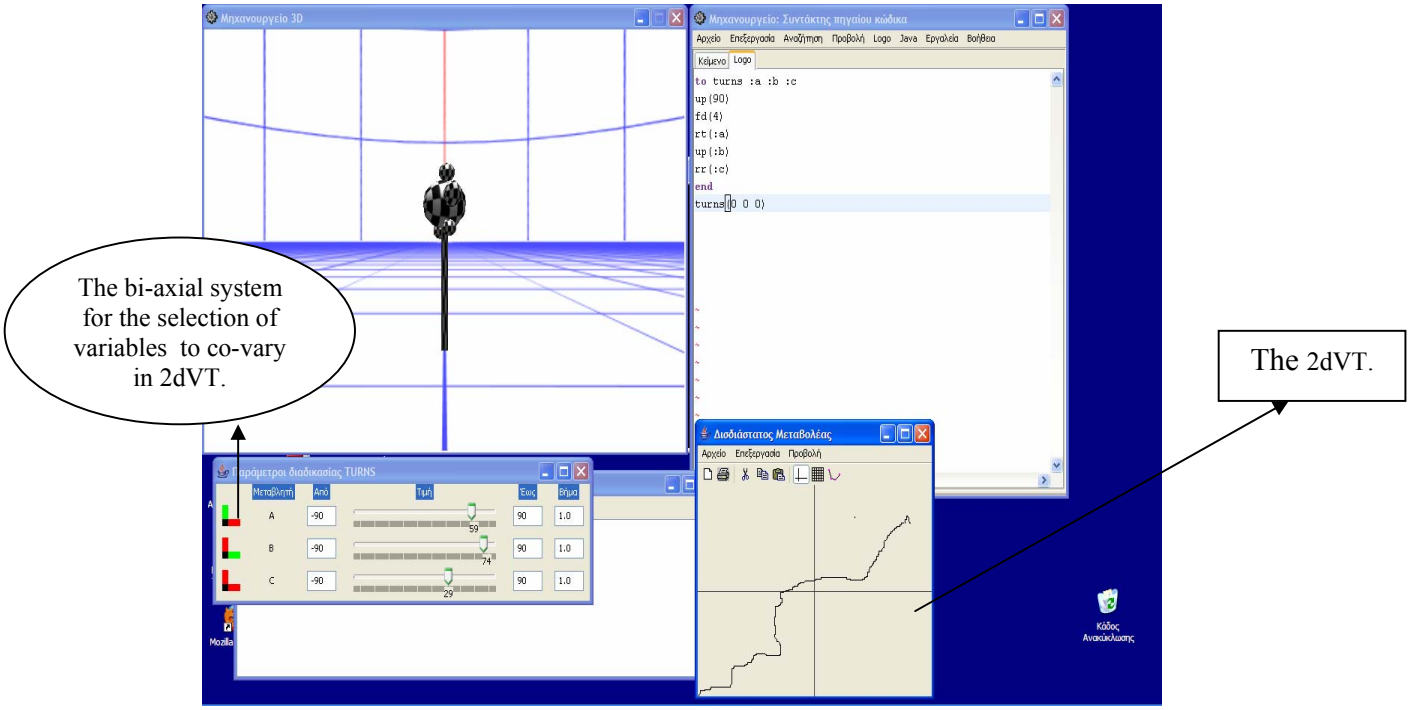

Figure 5: Selection of variables a and b in the 1dVT to co-vary in the 2dVT.

The effect of using the 2dVT is that of co-variation of two variables. The numeric domain of the two variables is that defined in the 1dVT. Any change in the numeric domain and the step in the 1dVT is transferred automatically to the system of coordination on the 2dVT.

Each position on the pad represents a value for each of the two selected variables, one representing the x-axis and one the y-axis, respectively. A trace is drawn when the mouse is dragged. The numeric changes in 2dVT can be seen simultaneously to the respective sliders on the 1dVT.

If one coordinates the dragging to approach – or to define or to guess after experimenting - a functional relationship underlying the co-variation of two variables then the result would be the graph of this function.

7. Working with VVT

The VVT requires a procedure with at least three variables. Next to each slider in 1dVT there is another icon to activate the VVT. Clicking on this icon activates a pop-up menu with three choices: vector r, angle θ and angle φ (Figure 3). The semantic behind this type of representation is (r, φ, θ) which means that r stands for length, θ for the angle between the vector's projection on the zx plane and the z-axis and φ for the angle between the vector and the zx plane (See figures 7, 8, 9).

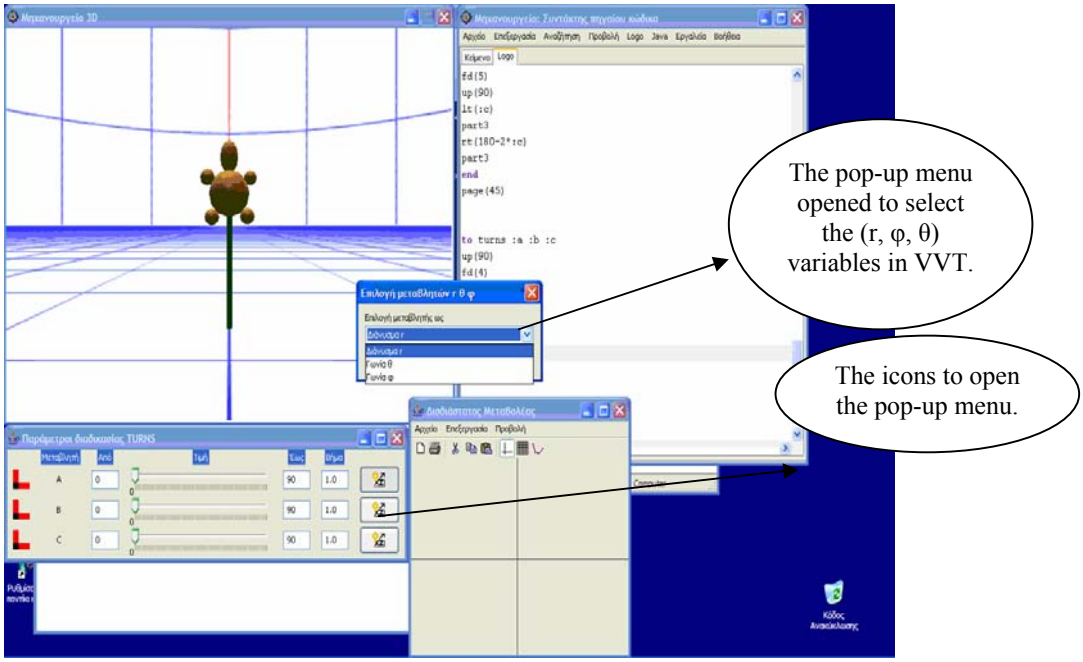

Figure 6: The (r, θ, φ) semantics for energizing the VVT using any three variables.

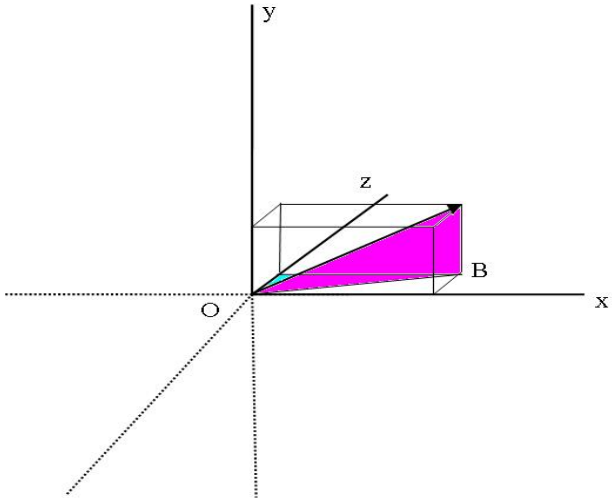

Figure 7: A schematic representation of the (r, θ, φ) semantic.

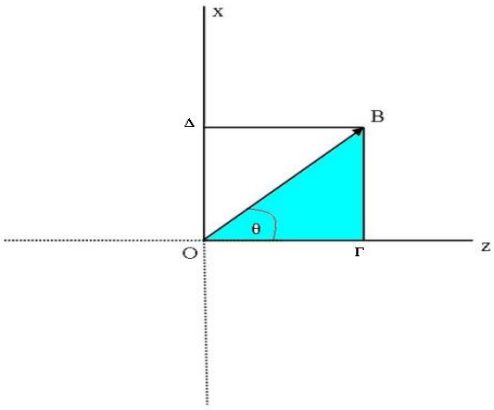

Figure 8: A schematic representation of the angle θ.

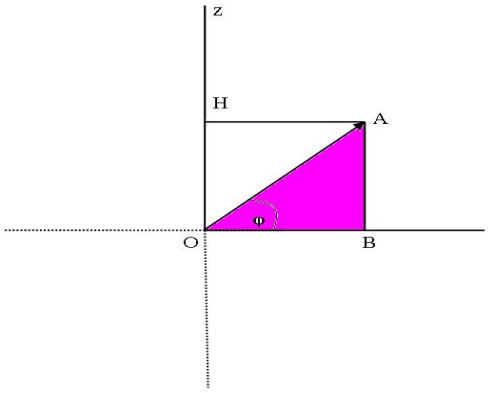

Figure 9: A schematic representation of the angle φ.

The user can select which variables will stand for r, θ and φ respectively. The VVT vector tool consists of two vector-like representations, which appear as two square windows having a common side. One representation stands for the projection of the vector on the horizontal plane zx. The user can dynamically manipulate the length of the vector's projection on the zx-plane and rotate it with respect to its inclination with the x-axis, angle θ.

The second representation stands for the plane formed by the vector and the projection of the vector on the horizontal plane zx and the user can again manipulate the vector's length and rotate it with respect to its inclination with the zx-plane, angle φ. These two representations sit side by side on the screen. This means that the VVT does not allow the manipulation of the resultant vector value, but only of the constituent projections on the two planes. The values seen on the VVT will thus be different to the ones observed on the 1dVT.

The VVT has a third component (Figure 7), where the resultant vector appears in a cube-like box. This component is not manipulable; its function consists of merely representing what happens to the resultant vector as either of the two projections change (Figures 8 and 9).

The polar values are represented graphically by the vector itself and numerically by digits in corresponding text boxes. The cartesian ones are represented by the visible projections of the vector on the two axes and digitally again by text boxes (Figure 10).

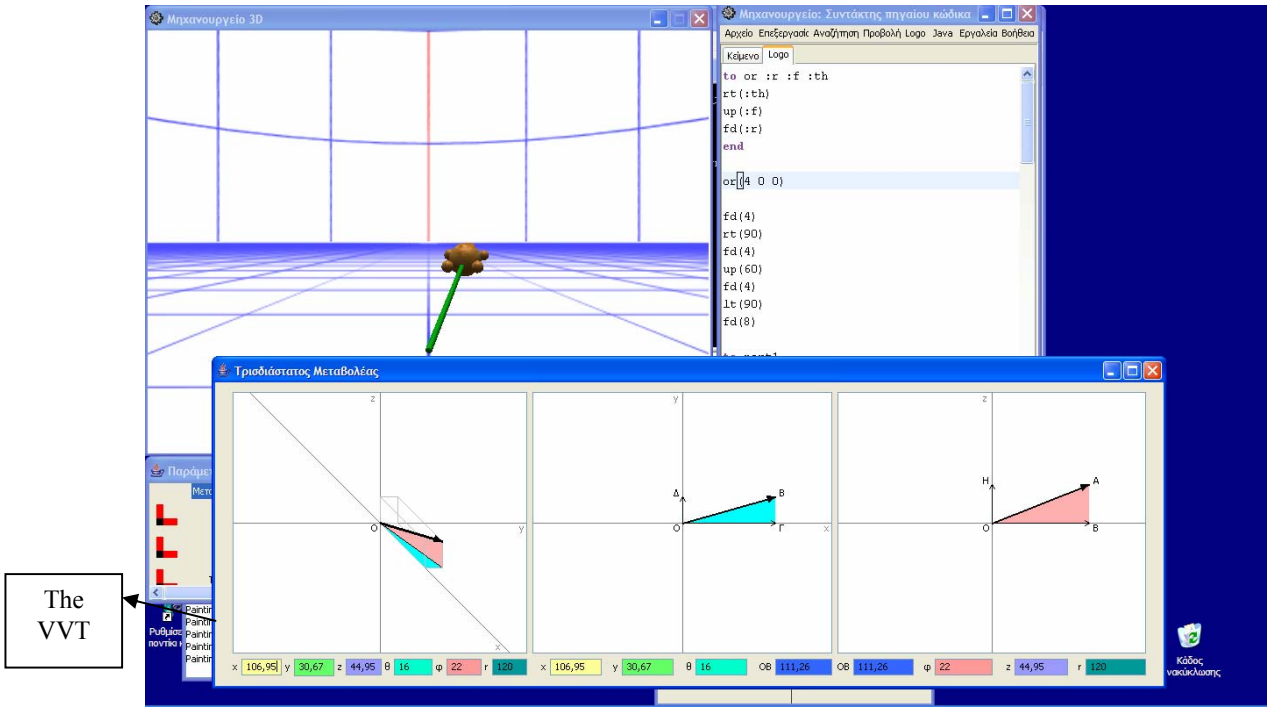

Figure 10: The VVT. The second and third representations on the right are manipulable while the first -which represents the vector in 3d space- changes only as a result of manipulating either of the latter two. The two square window projections contain both polar and cartesian values represented in the small colored boxes below each representation.

8. Insert 3d objects in the Turtle Scene

Type

The 3d object should be type .x (if we have objects of another type we have to change them using an appropriate 3d graphics program that supports this type of objects).

Loading

In order to load a 3d object at the Scene we have to use the following procedure:

```
to loadObject :filename
 localmake "result
 getObject(engine.CreateMesh("|media\models\|+:filename))
```

```
 getMesh(:result).setVisible(true)
 output :result
end
```

This procedure can be used for any object named “filename” so it is better to use it once in our program and then load many objects using another procedure.

Letters in red, specify the path that the object is located. We usually use this as the default path to locate our objects, but we can also use another path, writing the exact address each time. If an object is located in another folder except MaLT’s folders, we use the absolute path.

For example, if the object ball.x is in a folder named MaltModels in our desktop, we replace the path with: C:\Documents and Settings\user\desktop\MaltModels\.

In order to load the ball at the scene we use the following procedure:

```
to ball
  make "ball loadObject("|ball.x|)
end
```

Placing

To place the object (in our example the ball) in a specific place in 3d space of Malt we use the following procedure:

```
to place :ball :x :y :z
 getMesh(:ball).setLocation(engine.createVector3(:x :y :z))
end
```

Using this procedure we can place our object in any place. For example if we want our ball to be at x=1, y=3, z=5 (coordinates) in our program, we can use the command: place(:ball 1 3 5)

9. Different camera viewpoints

MaLT has a default viewpoint of seeing the turtle or the objects inserted. In order to change the viewpoint we have to move the camera in 3d space. In order to do this, the first step is to define the camera’s location using the following procedure:

```
to placeCamera :X :Y :Z
 camera.setLocation(engine.CreateVector3(:X :Y :Z))
end
```

Different camera viewpoints can be controlled through the x, y, z coordination system. So we must define how the coordinates will change and in which way. For example we can

specify keyboard buttons that define changes in camera's coordinates. The following example is indicative the way camera moves.

Procedure to initialize variables

```
to init
make "camstep .2
make "xcam 1
make "ycam 1
make "zcam 1
end
```

explanation:

- make "camstep .2 → defines the step that camera moves
- make "xcam 1 → defines the initial camera x coordinate
- make "ycam 1 → defines the initial camera y coordinate
- make "zcam 1 → defines the initial camera z coordinate

Procedure to define buttons' actions and camera's changes according to coordinates

```
to run
if engine.KeyDown(DIK_LEFTARROW) [make "xcam :xcam-:camstep]
if engine.KeyDown(DIK_RIGHTARROW) [make "xcam :xcam+:camstep]
if engine.KeyDown(DIK_UPARROW) [make "ycam :ycam+:camstep]
if engine.KeyDown(DIK_DOWNARROW) [make "ycam :ycam-:camstep]
if engine.KeyDown(DIK_A) [make "zcam :zcam+:camstep]
if engine.KeyDown(DIK_Z) [make "zcam :zcam-:camstep]
placecamera(:xcam :ycam :zcam)
if engine.keyDown(DIK_ESCAPE) [setCallBacksEnabled(false)]
camera.lookAt(engine.createVector3(0 0 0) FAST_SPEED)
end
```

explanation:

- First six commands define the buttons that change camera's coordinates.
- placecamera(:xcam :ycam :zcam) → places camera in coordinates
- if engine.keyDown(DIK_ESCAPE) [setCallBacksEnabled(false)] → defines that escape button stops navigation with the buttons
- camera.lookAt(engine.createVector3(0 0 0) FAST_SPEED) → initializes camera's viewpoint

Editing the procedures described above and running the 'run' procedure you can move camera with the arrows, A and Z buttons. With Escape button deactivates the camera.

Examples of Logo Procedures

1. Construction of a twisted ladder

The construction of the twisted ladder (Figure 11) can be built by using the following procedures:

a. Construction of a rectangle.

```
to rect :a :b :c
repeat 2 [fd(:a) rt(:c) fd(:b) rt(180-:c)]
end
```

b. Construction of an equilateral triangle.

```
to tri :b
rt(30)
repeat 3 [fd(:b) rt(120)]
lt(30)
end
```

c. Construction of the step of a twisted ladder, where steps are equilateral triangles.

```
to step :a :b :c
rect(:a :b :c)
fd(:a)
dp(90)
tri(:b)
up(90)
lr(60)
end
```

d. Construction of the ladder.

```
to ladder :a :b :c :e
up(90)
repeat :e [step(:a :b :c)]
end
```

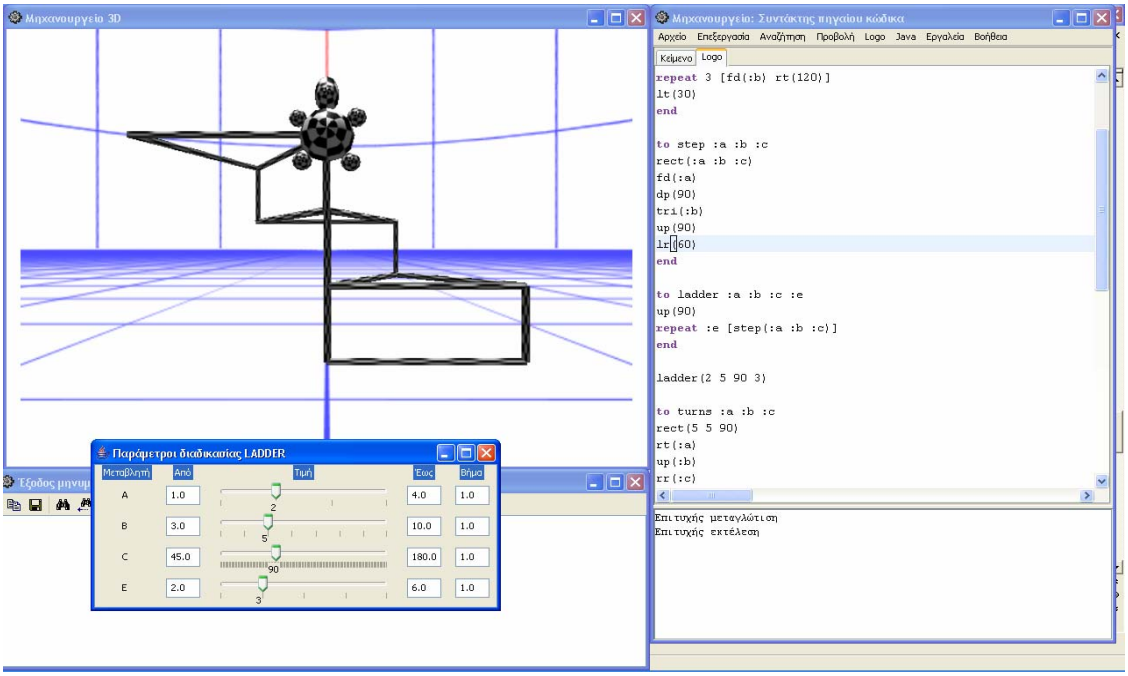

Figure 11. The construction of the twisted ladder.

2. The construction of swiveled rectangles.

By moving the sliders (Figure 12) the user can dynamically manipulate the size as well as the turning movements of the construction.

```
to swivelrects :a :b :c :d :x :y :z
up(90)
rt(:x)
dp(:y)
rr(:z)
multirect(:a :b :c :d)
end

swivelrects(2 5 30 10 90 90 90)
```

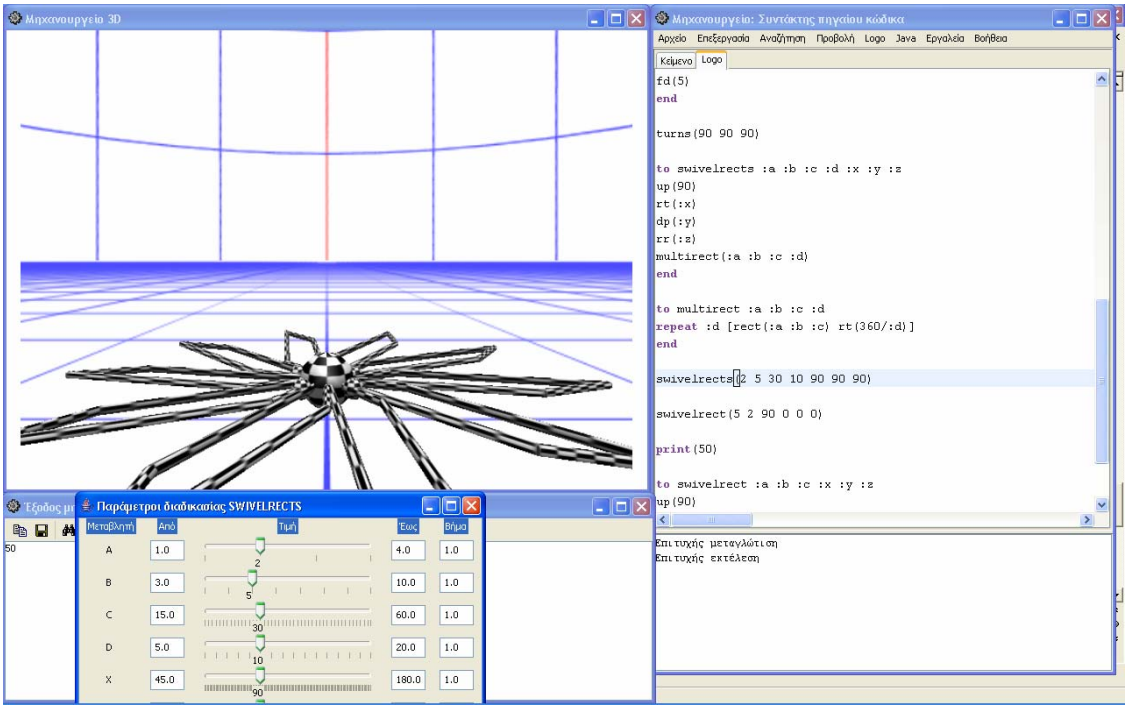

Figure 12. The construction of the swiveled rectangles.