

# Turtle Metamorphoses

## ( From “FD 1” To 3D Animated Characters )

Pavel Boytchev

*Elica Team*  
*Bulgaria, Sofia*  
*pavel@elica.net*

### Abstract

Although the concept of the turtle evolved, modern turtles are still essentially the same. This paper presents ideas about user-defined turtles. It focuses on the creation of the ordinary triangular turtle, then shows how it can naturally grow into a turtle with a 3D shape, which later transforms into a true space turtle or spherical turtle. The paper also describes how turtles can live in different environments and operate in extremely abstract affine transformation spaces in order to define and animate 3D characters.

### Keywords

Logo, Elica, turtle graphics, space turtle, spherical turtle, 3D characters, animation

### 1. tur-tle ['tɜ:tl]

*a reptile of the order Testudinata (it's having been regarded in ancient times as an infernal creature)*

There were many discussions as to what Logo feature is the most recognisable. Without any doubt the turtle graphics is the thing that most people associate Logo with. Although not every Logo contains turtle graphics, it really appears to be one of the basic features of the majority of Logo implementations. The concept of the turtle evolved along the evolution of the Logo language itself [Abelson H and diSessa A, (1981)]. Since its birth almost 40 years ago, the language has more than a hundred implementations. One could think that such a history record should definitely mean that today's Logos are quite different, and the turtles – more advanced, more beautiful, more flexible.

Unfortunately, modern turtles have features almost equivalent to those of past turtles. Does it mean that the evolution hit a dead end [Tempel M (1995)]? Are turtles so perfect that there is no room for further improvement [Tempel M (1996)]? Or maybe there is something makes this impossible. The author explored various ideas of designing, implementing and using turtles with Logo. Results suggest that **the thing that obstructs turtle evolution is ... its integration with the Logo language.**

Elica is a Logo implementation that does not provide a system-embedded turtle, but a turtle entirely written in Logo [Elica Team (2003)]. So, let us forget about it and build our own turtle from a scratch.

## 2. con-ven-tion-al [kən'venʃənəl]

- (a) based on convention, custom or traditional usage
- (b) lacking spontaneity, originality, or individuality

To rebuild the conventional turtle, we need two of the libraries that come with Elica – LOGO, which defines a complete set of Logo commands, and GRAPHIX that is used to draw things on the screen. As a starting point, we will implement the two basic turtle movements – going forward (FD) and turning to the left (LT). There are several ways to do it [Silverman B and Tempel M (1991)], but we would use vectors because it will make further metamorphoses easier.

Vectors are used as containers of two numbers and thus their definition is like a command with an empty body. HOME creates and initialises turtle's vectors by using VECTOR as a class definition.

```
to vector :x :y
end

to home
  make "f vector 0 1
  make "r vector 1 0
  make "pos vector 0 0
end

to fd :n
  make "pos :pos+:n*:f
end

to lt :a
  local "ff "rr
  make "ff (cos :a)*:f - (sin :a)*:r
  make "rr (sin :a)*:f + (cos :a)*:r
  make "f :ff
  make "r :rr
end
```

When Elica executes VECTOR it defines X and Y as local variables, and instead of destroying the local stack it is returned to the caller<sup>1</sup>. Let's now focus on FD and LT. Their definitions could be pretty close to the mathematical representation<sup>2</sup>. To simplify the source, we use operator + to add vectors, and \* to multiply them by numbers.

As long as vectors has been just defined by us, Elica has no knowledge as to how to use them in expressions. In other Logo implementations this problem could be solved by either forcing the user to create functions for vector algebra, or if the Logo developers are generous enough, to have vector algebra hardcoded in the translator. Fortunately, this is not the case with Elica.

---

<sup>1</sup> This is effectively the same as creating variables f . x, f . y, r . x, r . y, pos . x and pos . y .

<sup>2</sup> A turtle is defined by 3 vectors:  $\vec{P}$  is the **turtle position**,  $\vec{F}$  is the **forward vector**, and  $\vec{R}$  is the **right vector**.  $\vec{F}$  and  $\vec{R}$  are orthogonal unit vectors, so  $\vec{F} \cdot \vec{R} = 0$  and  $|\vec{F}| = |\vec{R}| = 1$ . Going forward  $n$  units is:

$$(1) \vec{P}' = \vec{P} + n\vec{F},$$

where  $\vec{P}'$  is the new position of the turtle. Turning left  $\alpha$  radians changes vectors  $\vec{F}$  and  $\vec{R}$  in this way:

$$(2) \begin{cases} \vec{F}' = \vec{F} \cos \alpha - \vec{R} \sin \alpha \\ \vec{R}' = \vec{F} \sin \alpha + \vec{R} \cos \alpha \end{cases}$$

where  $\vec{F}'$  and  $\vec{R}'$  are the new forward and right vectors (note that the orientation of the co-ordinate system could swap the meanings of "right" and "left").

The LOGO library defines standard scalar `+`, `-` and `*`, but the user can upscale them to work with both numbers and vectors as it is shown in the definition of `+`. If the second input is a number, then the result is the same as the standard operator `+` from the LOGO library. Otherwise the assumption is that both inputs are vectors and the result is a new vector which components are sums of the corresponding components of the input vectors.

```

to :x + :y
  if number? :y
    [output :x 'logo.+ ' :y]
    [output vector :x.x+:y.x :x.y+:y.y]
end

```

To complete the conventional turtle we only need a few lines to initialise the graphical window, to draw the image of the turtle and to register its traces when the pen is down.

### 3. car-a-pace ['kærə,peɪs]

- (a) a bony or chitinous case or shield covering the back or part of the back of an animal
- (b) the entire shell of a turtle comprising the carapace and the plastron

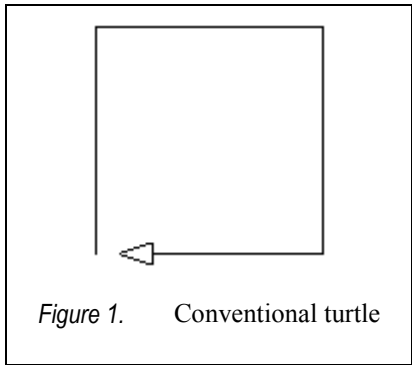


Figure 1. Conventional turtle

Having a conventional turtle ready, it's time to change its appearance. This has nothing to do with assigning a set of bitmaps for different headings, but using a true image [Platinum Pictures Multimedia (2003)], a Logo-generated image.

Because we define our turtle from a scratch, it is possible to change its image using all the power of Elica and to make turtle graphics abstractions more concrete [Pilkington R and Groat A].

The standard Elica installation included the `TURTLE` library, which contains two predefined turtle shapes. They are both entirely written in Logo, so they can be modified if needed.

The primary shape is the standard triangle on Figure 1. The other shape, shown on Figure 2, is a 3D image of a turtle. By changing the viewpoint it is possible to see the turtle from the other side or from below. Turtle anatomy is pretty simple. Its shell is made of a greenish texture projected onto two semiellipsoids. All the rest visible body parts are made of smaller brown ellipsoids.

A 3D shape for a turtle moving on a plane raises the issue of the benefits. If users always look at the turtle from above, a 3D shape does not differ much from a properly shaded bitmap. To resolve this Elica turtle has several modes.

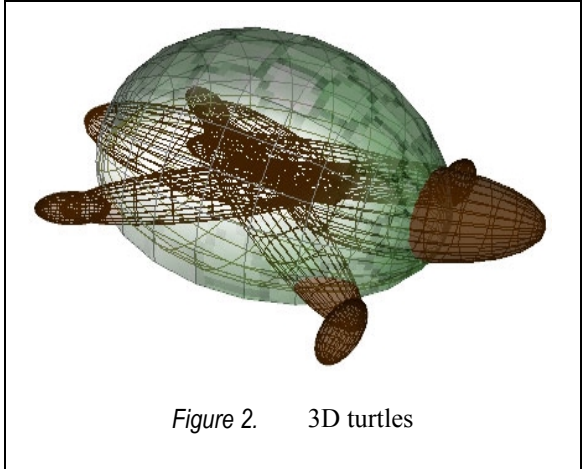


Figure 2. 3D turtles

In `FAST` mode a turtle goes as fast as pos-

sible, while in SLOW mode every action is broken down into smaller fragments. When `FD 100` is executed, the turtle does not jump 100 units ahead, but smoothly moves forward.

In another mode, called CENTERED, the system automatically moves the viewpoint to keep the turtle in the centre of the screen. Visually this looks like as if the turtle is pinned and its drawing is moving around it.

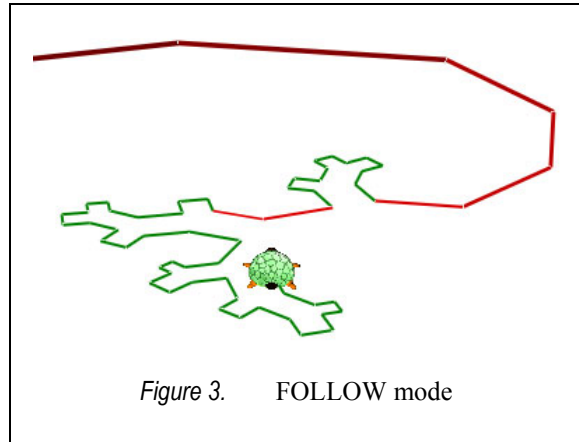


Figure 3. FOLLOW mode

A similar mode is FOLLOW where the viewpoint is elevated behind the turtle – Figure 3. This not only gives a perspective view of the drawing, but also helps the user to self-associate with the turtle (here turning left is always left, independent on the position and the heading).

#### 4. space ['speis ]

*a three-dimensional entity that extends without bounds in all directions*

The next natural metamorphosis it to make our 3D looking turtle walk in space. Using vectors to do this makes the task easy<sup>3</sup>. In addition to a new vector, we only need to change the definition of VECTOR by adding a third component and to initialise the new vector in HOME.

```
to vector :x :y :z
end

to home
  make "f vector 0 1 0
  make "r vector 1 0 0
  make "u vector 0 0 1
  make "pos vector 0 0 0
end
```

As for turtle movements, FD and LT need no changes, because they are already written using vector algebra. To complete the basic procedures, we need to define the two new rotations (using LT as a template), and to update vector operators to work in 3D.

There are many amazing figures that can be drawn in the space [Prusinkiewicz P (1995)]. One typical example is the fern fractal shown in Figure 4.

<sup>3</sup> We need to add an **up unit vector**  $\vec{U}$  that points towards the up direction and is orthogonal to both  $\vec{F}$  and  $\vec{R}$  [Abelson H. and diSessa A (1984), Loethe H (1992), Nehaniv C, Tremblay C (1997)].

Going forward is handled by the same equation (1). Yaw rotation already implemented in equation (2) turns directional vectors around the up axis, pitch rotation turns vectors around the right axis using equation (3), and roll rotation turns vectors around forward axis using equation (4).

$$(3) \begin{cases} \vec{U}' = \vec{U} \cos \alpha - \vec{F} \sin \alpha \\ \vec{F}' = \vec{U} \sin \alpha + \vec{F} \cos \alpha \end{cases}$$

$$(4) \begin{cases} \vec{R}' = \vec{R} \cos \alpha - \vec{U} \sin \alpha \\ \vec{U}' = \vec{R} \sin \alpha + \vec{U} \cos \alpha \end{cases}$$

where  $\vec{F}'$ ,  $\vec{R}'$  and  $\vec{U}'$  are the new direction vectors. Equations (2), (3) and (4) look much better in a matrix form, but even with the current representation they show the symmetry between all rotations.

## 5. spherical ['sferikl]

*having the form of a sphere or one of its segments*

Once we have a turtle that can go and turn in all directions it is easy to generate some exotic mutations. Just as an example, we will see how to make a turtle that lives on a sphere [White J (2002)].

For an external viewer, the turtle will act as a 3D one, but from turtle's point of view it will be like a flat one.

The spherical turtle can do only what a 2D turtle can do – going forward or backward, and turning left or right. As before we are not interested on other activities like changing skins or manipulating pen colours.

Constraining turtle's movements on the surface of a sphere can be done mathematically, but the description would become long and complex. Instead we will use one of the basic tricks a turtle can do since its birth – making circles.

If a turtle steps forward and turns left, then again steps forward and turns, after a given number of repetition it will make a circle and will return to its initial position. The circle will be drawn at the left side of the turtle. If we apply this to a 3D scene, replacing every turn to the left with a turn downwards, the circle will be drawn underneath the turtle. But

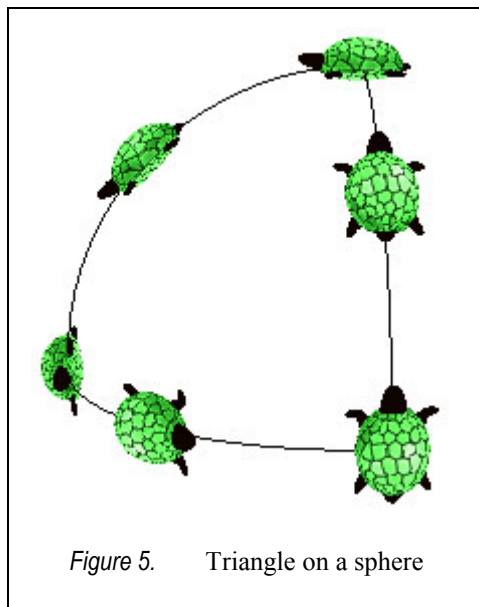


Figure 5. Triangle on a sphere

how this circle can be turned into a sphere? Imagine the turtle draws a vertical circle and when it returns to the initial position it turns to the left and draws another circle, then again turns to the left and draws a circle. Repeating this will result in a family of circles that frame a sphere.

Without a strict mathematical proof this can only give us a clue as to how to build a spherical turtle: whenever the turtle goes forward pitch it down.

There is one favourite example in introductory course of spherical geometry – the triangle in Figure 5. The turtle at the lower right corner goes forward and turns left and repeats these actions twice more. At the end it reaches its initial position.

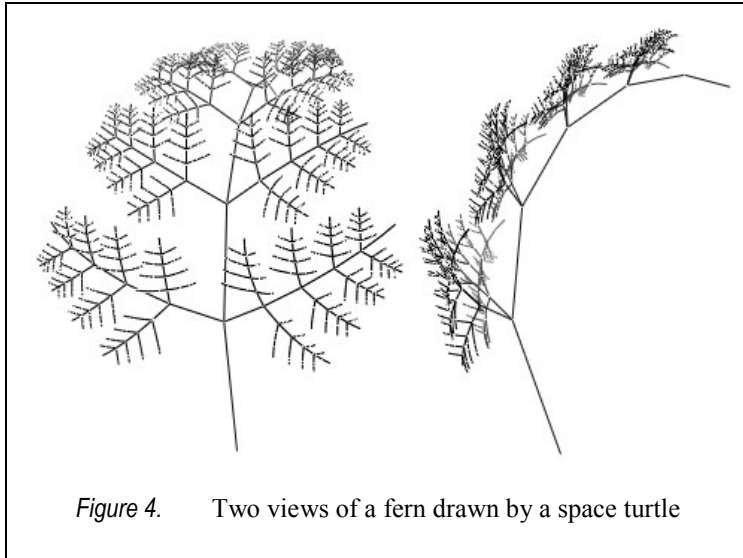


Figure 4. Two views of a fern drawn by a space turtle

The implementation of a spherical turtle is almost trivial. Every FD command should be decomposed into a series of micro forward steps with fixed length  $\mu$ . For example, the command FD :X will be executed as  $\lceil x/\mu \rceil$  separate steps of length  $\mu$ . If  $x/\mu$  is not integer, then there is a need to compensate with one additional shorter step of length  $x - \mu\lceil x/\mu \rceil$ .

Each step is accompanied by a down pitch, which is as many degrees as the length of the corresponding step. There are several ways of distributing these rotations. Experiments show that a relatively precise one is when half of the rotation is done before the step, and the rest is done afterwards. The Logo code assumes the ratio of the length of a step and turn angle is 1:1. This ratio defines the radius of the surface and can be set to a different value.

```

to myfd :x
  while :x>=:M
  [
    down :M/2
    fd :M
    down :M/2
    make "x :x-:M
  ]
  down :x/2
  fd :x
  down :x/2
end

```

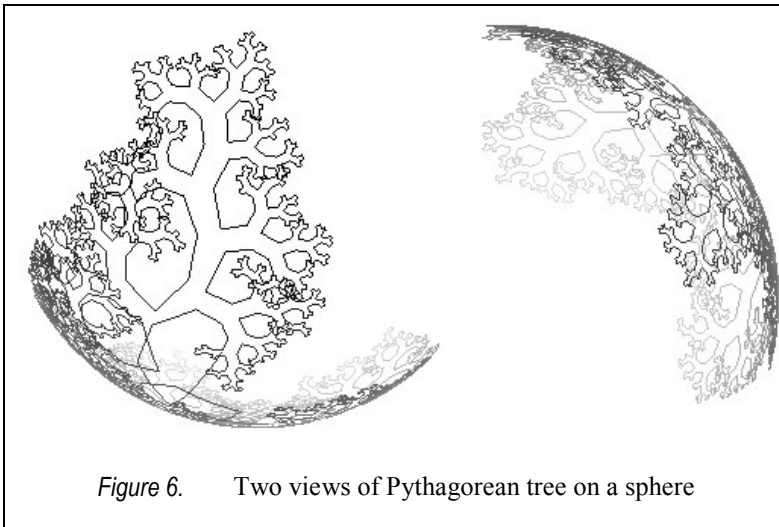


Figure 6. Two views of Pythagorean tree on a sphere

Spherical turtles can be used in almost any area where a flat one can be used [Wilensky U (2002)]. Two snapshots of Pythagorean fractal tree are shown on Figure 6. Shapes that occupy a small area on the surface look normally, but large shapes are deformed by the curve of the surface because the geometrical distortion becomes more noticeable.

## 6. a-ni-ma-ted [ 'ænimetid ]

- (a) *having the appearance of something alive*
- (b) *made in the form of an animated cartoon or of animation*

For this final turtle metamorphosis we will use Logo to define and animate 3D characters in two different ways. The 3D turtle seems quite suitable for this purpose, but as it happens, it can be used only for some basic and simple structures.

The early attempts of the author to build a 3D character resulted in a wire-frame model shown on Figure 7. Although it is not hard to build and animate such a model, it becomes very difficult if one wants to put some flesh on the bones. For example, in a wire-frame mode, if wires are drawn by a turtle there are no problems to draw them relative to each other, hence this is one of the key features of turtle graphics. However, if drawn parts are

more complex, it is not clear how to handle it. The problem becomes even harder if some parts are not drawn by the turtle, but with other tools.

The solution is do decompose the 3D character into parts where each part does not change its geometry. Different parts are connected at joints and can turn relatively to each other [Gross M (2001)].

Each joint is accompanied by a local co-ordinate system as shown on Figure 8. All parts that are drawn in respect to this system do not change their position in it. Thus, when the co-ordinate system of the knee is turned downwards, all bound co-ordinate systems are turned down<sup>4</sup>. To render the image, each part of the body is drawn in respect to a transformation matrix defined by its local matrix.

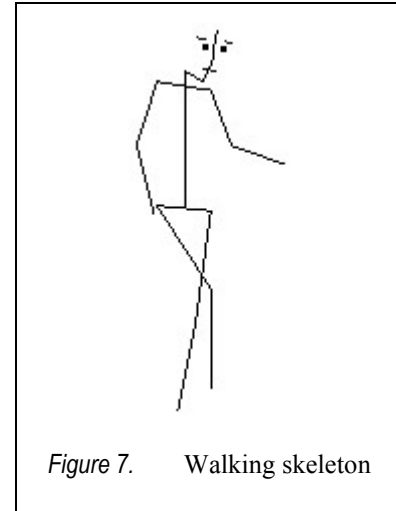


Figure 7. Walking skeleton

Skipping most of the math background, it is enough to note that if we have a turtle reaching the foot<sup>5</sup>, its direction vectors will match the foot local matrix as represented relatively to the knee co-ordinate system. However both systems depend on the hip co-ordinate system which is at the hip, which in turn may depend on the body co-ordinate system that defines the position of the whole body.

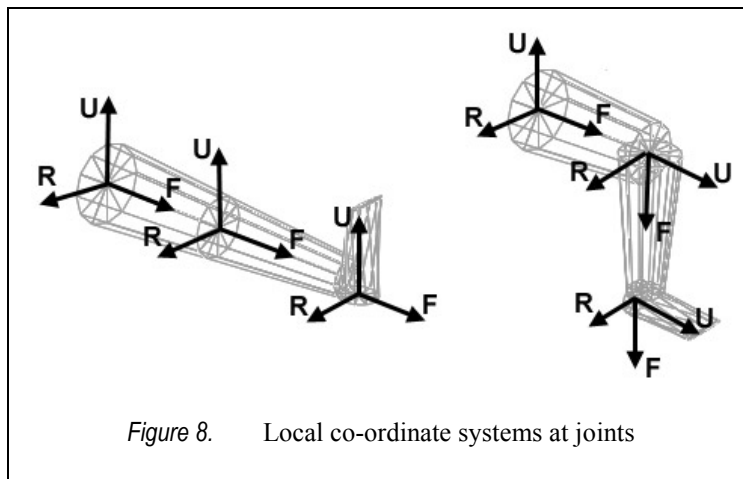


Figure 8. Local co-ordinate systems at joints

There are a couple of tricks that make all this work. First of all, the actual rendering of a body part requires that it be rendered as if it has been transformed in respect to all binding matrices. Thus, to render the foot, we need to multiply at least four transformation matrices – the first one will transform from body to hip system, the second one from hip to knee system, and the third from

the knee to the foot. Finally, the last system will orient the foot relatively to its default position. Elica turtles automatically handle this multiplication if the body is represented as hierarchy of nested body parts

<sup>4</sup> Actually, bound co-ordinate systems are not only turned, but also translated as long as rotation is not relative to their own centres.

<sup>5</sup> Here we assume that the turtle started from the hip joint, then went to the knee joint and finally reached the foot joint.

The other trick is about the calculation of a particular transformation matrix of a local coordinate system. Because of the automatic multiplication, we only need to remember that each matrix is relative only to its parent matrix (this is the matrix of the parenting body part). The parent matrix is assumed to be identity in respect to the child matrix. So, the child matrix will be the same as the transformation matrix that projects the parent system onto the child system. Also and more importantly, its axes will be the same as a turtle's directions if the turtle moves from the parent to the child.

When building a given posture, a turtle walks along body parts leaving copies of itself in every joint. The directional vectors of these clonings are later used to represent transformation matrices that help building the body.

There are some other issues that need to be resolved. How a Logo program will control all these things? It does not look like a simple task even if we ignore essential concerns about speed and real-time animation.

```
to LEG
  to THIGH
    make local "thigh sphonoid 3.5 2.5 12
    make local "calf CALF
    construct [thigh fd :thigh.length calf]
  end

  to CALF
    make local "calf sphonoid 2.5 1.5 12
    make local "foot FOOT
    construct [calf fd :calf.length foot]
  end

  to FOOT
    make local "foot conoid 1.5 1.5 0.2 2 7
    construct [fd 1 up 90 cw 90 fd -1 foot fd 7]
  end

  make local "thigh THIGH
  main_construct [thigh]
end
```

Fortunately, customisable turtles solve this problem. Unfortunately, the detailed explanation how this is done goes out of the scope of the paper.

The source of `LEG` shown above is the actual and the complete definition of the leg from Figure 8. Most of the work is hidden from the end user in a library called `ANDROID`. Also, there is a library called `HUMANBODY` that uses `ANDROID` and defines an object that represents the human body.

Each body part is defined as a simple command. It contains definitions of the graphical objects that represent the body part and a command `CONSTRUCT` that places these objects in space. For example, the construction of `THIGH` will create a sphonoid, go forward to the end of the sphonoid and create there a `CALF`. One really important thing here, is that every

definition of a body part assumes that it works with its own turtle which is always placed at the correct position.

This turtle is in fact the clone left by the turtle that walked along the body parts. The animation of the character is done by changing the orientation of one or more turtle clones. When a single clone is changed, the body part attached to it will change its orientation. This change will also affect the rendering of all bound body parts although their clones will be kept the same.

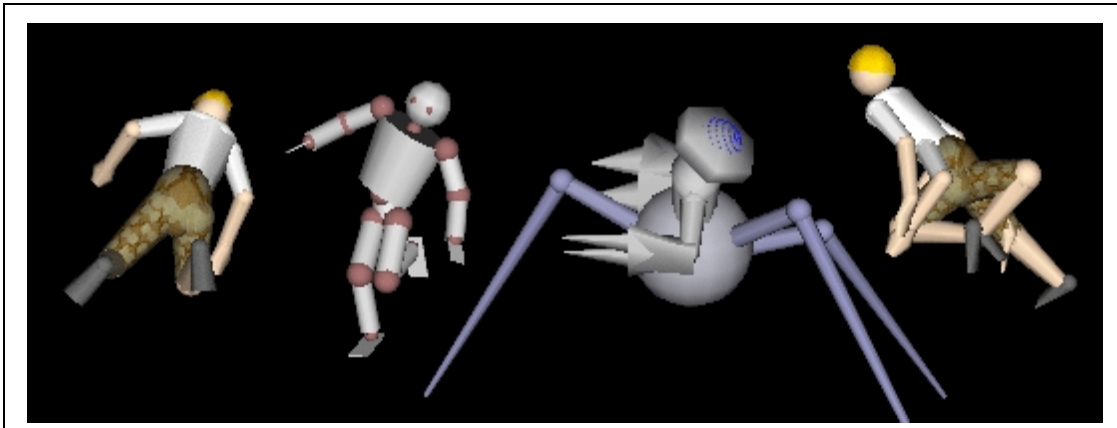


Figure 9. 3D characters – a human, an android, a robot-warrior, and a mutant

Because 3D characters are object-oriented by design and by implementation, it is straightforward to make not only human-like bodies, but also other creatures met in video games and movies – androids, robots, mutants (see Figure 9).

Elica libraries handling 3D creatures are designed so that the user can do even more sophisticated tasks, like, combining different parts and rearranging body structure (see the mutant – the last image in Figure 9). It is also possible to store the complete tensor-space of a body. This tensor-space is a hierarchical structure built up of all local affine transformation matrices and the 3D turtle clones play the role of “points” in the space. If such tensor-space is stored in a variable, it can be used to recover the posture of the body. And additionally, there is an interpolating procedure that morphs one tensor-space into another, thus generating all intermediate complex of movements that convert one posture into another. This is how creatures can be animate if users do not want to control manually every joint.

## 7. con-clu-sion [kən'klu:zhən]

(a) *reasoned judgement or an expression of one*

(b) *the last part of anything*

Turtle graphics has been parented by Logo language for quite a long time. If we let turtles free, they could find a bigger place, a world where the sky is higher and the boundaries are farther. Logo could only benefit if turtles are allowed to do all the things that people think of as impossible to do with Logo.

The main concern behind the writing of this paper is not what a turtle can do, but what the end user can do with a turtle. This paper presented some ideas about turtle metamorphoses when turtles are fully designable and implementable by the user. Turtles are not limited by nature. Logo language is not limited too. It's the design of a particular Logo implementation that will either put the turtle in chains or will catalyse its abilities.

## 8. References

- Abelson H. and diSessa A (1984), *Turtle Geometry*, MIT Press, 144
- Elica Team (2003), *Elica*, <<http://www.elica.net>> (Mar 2003)
- Gross M (2001), *FormWriter - A Little Programming Language for Generating Three-Dimensional Form Algorithmically*, CAAD Futures 2001, p. 577-588, <[http://depts.washington.edu/dmgftp/publications/pdfs/Paper\\_97.pdf](http://depts.washington.edu/dmgftp/publications/pdfs/Paper_97.pdf)> (Mar 2003)
- Loethe H (1992), *Conceptually Defined Turtles*, Learning Mathematics and Logo, MIT Press, 55-95
- Nehaniv C, *Turtle Commands*, <<http://homepages.feis.herts.ac.uk/~nehaniv/CM/turtle.html>> (Mar 2003)
- Platinum Pictures Multimedia (2003), *Animals / Insects*, <<http://www.3dcafe.com/asp/animals.asp>>, Platinum Pictures Multimedia, Inc. (Mar 2003)
- Prusinkiewicz P, Hammel M, Mech R (1995) *The Artificial Life of Plants. Artificial life for graphics, animation, and virtual reality*, volume 7 of SIGGRAPH Course Notes.
- Pilkington R and Groat A, *Styles of Learning and Organisational Implications*, <<http://cbl.leeds.ac.uk/~rachel/papers/styles.html>> (Mar 2003)
- Silverman B and Tempel M (1991), *Fuzzy Logo*, <[http://el.media.mit.edu/logo-foundation/pubs/papers/fuzzy\\_logo.html](http://el.media.mit.edu/logo-foundation/pubs/papers/fuzzy_logo.html)>, LCSI and Logo Foundation (Mar 2003)
- Tempel M (1995), *The Turtle Is Dead - Rethinking Logo in the Age of Kid Pix*, <<http://el.media.mit.edu/logo-foundation/pubs/logoupdate/v4n1.html#dead>>, LogoUpdate, volume 4, number 1, Logo Foundation (Mar 2003)
- Tempel M (1996), *The State of the Turtle*, <<http://el.media.mit.edu/logo-foundation/pubs/logoupdate/v6n1/v6n1.html#turtle>>, LogoUpdate, volume 6, number 1, Logo Foundation (Mar 2003)
- Tremblay C (1997), *Plant Modelling Using Parallel Graph Grammar Languages (L-systems)*, <<http://www.cs.mcgill.ca/~carlos/>> (Mar 2003)
- White J (2002), *Mathwright Microworld – Spherical Logo*, <[http://www.mathwright.com/book\\_pgs/book625.html](http://www.mathwright.com/book_pgs/book625.html)> (Mar 2003)
- Wilensky U (2002), *NetLogo 3D Shapes model*. <<http://ccl.northwestern.edu/netlogo/models/3Dshapes>>, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. (Mar 2003)