

Visual Thinking and Spatial Reasoning

Sep.16.02

Computer Form Making

Formwriter

FormWriter is a simple and powerful programming language for generating three-dimensional (3-D) geometry, intended for architectural designers with little programming experience to be able to generate three dimensional forms algorithmically without writing complex code.

FormWriter Environment

Figure 1 shows the FormWriter working environment: on the left is a display window with browsing controls, where the visual output will be shown in 3-D; on the right is an editor window for writing code. You type code in the editor window; to see the result you press the “xeq” (execute) button. The resulting geometry can be saved as a file for further processing. FormWriter also allows you to load saved files for further modification. When you open a saved file, the code of that file will be shown in the editor window.

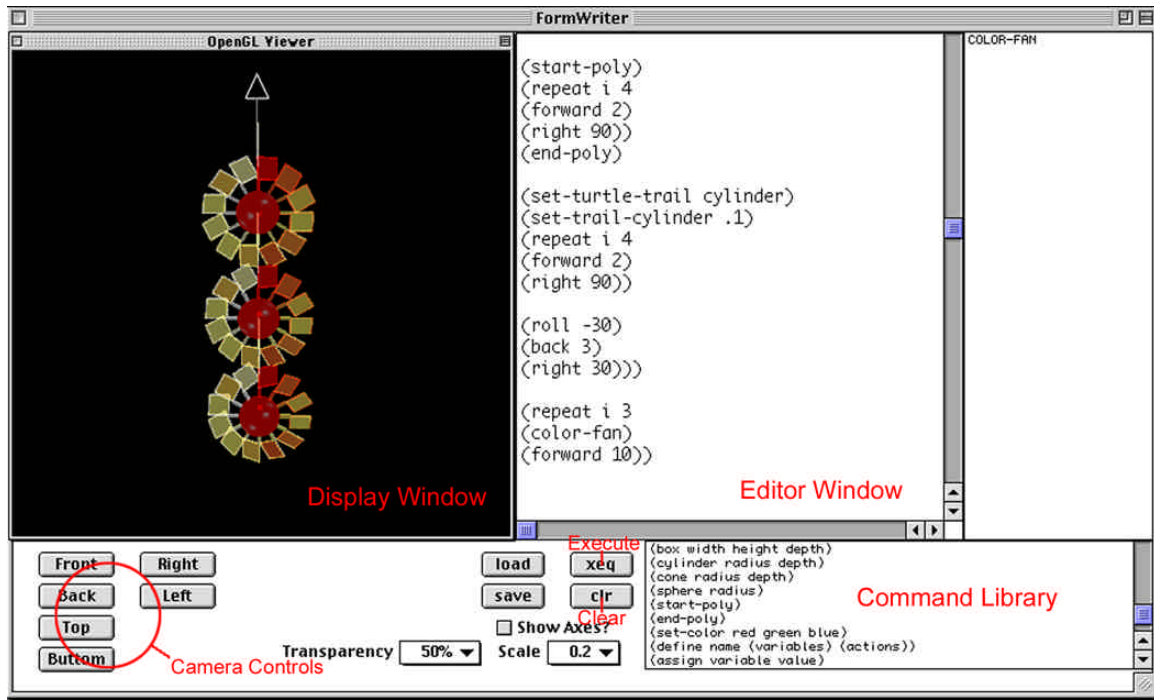


Figure 1

The Flying Turtle

After you open FormWriter, the display window shows a fixed global coordinate system that indicates + X, + Y, and + Z axes, Figure 2. You can also choose to hide the axes. The initial view on the display window shows the X-Y plane with + Z axis facing out of the screen (toward the user). Right now you should be able to see the flying turtle, actually a red triangle frame, standing at the origin with the head pointing to the + Y direction and waiting for your commands to make something happen. The flying turtle can move forward and back, and turn (right and left), pitch (up and down), and roll (side to side).

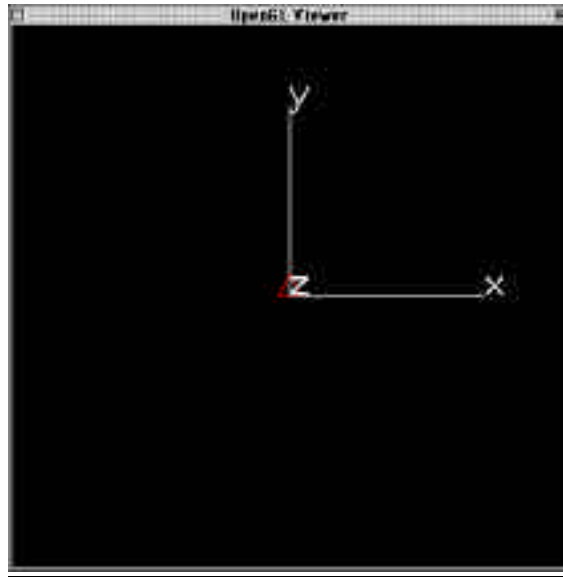


Figure 2

Basic Turtle Operations

(forward *distance*)

move turtle by *distance* in +y-axis direction

(back *distance*)

move turtle back by *distance* in -y-axis direction

(right *degrees*)

rotate turtle in a clockwise direction by *degrees* along the +z-axis

(left *degrees*)

rotate turtle in a counterclockwise direction by *degrees* along the +z-axis

(pitch *degrees*)

pitch turtle in a counterclockwise direction by *degrees* along the +x-axis

(roll *degrees*)

rotate turtle in a counterclockwise direction by *degrees* along the +y-axis

(set-global-pos *x y z*)

move turtle with **(penup)** to position (*x y z*) in relation to the global coordinate

(set-local-pos *x y z*)

move turtle with (**penup**) to position (x y z) in relation to the current local coordinate

(global-pos)

show the current values of (x y z) in relation to the global coordinate, the values will show in **Listener**

(get-global-pos *position*)

position: p0, p1, p2, ... , p20

record the current global values of (x y z) and assign the values to *position*

(move-to *position*)

move turtle with (**penup**) to *position*) in relation to the global coordinate

(home)

move turtle back to the origin (0.0 0.0 0.0) in relation to the global coordinate

FormWriter Syntax System

Before we move on to other FormWriter commands, here is how to write the code.

FormWriter currently uses a Lisp syntax system that has a set of simple rules. The first rule is that each single command is represented by an expression that is always enclosed in a pair of parentheses.

For example, the expression (**forward** 2), which commands FormWriter to draw a line, starts with an open-parenthesis and ends with a close-parenthesis. The **forward** is actually an operator, and the number 2 is a parameter that gives a value to the operator.

Here is another example of FormWriter syntax. When we do a simple calculation: 2 plus 3, we would naturally write the expression as 2 + 3. In FormWriter we should write it as (+ 2 3), which starts with an open-parenthesis, followed by the operator +, and then two parameters 2 and 3, and finally with a close-parenthesis. Perhaps you noticed that the operator **forward** takes only one parameter, but the operator + takes two. Some operators do need not just one parameter but many to complete the operation.

An expression actually can contain other expressions. For example, the expression (**forward** (+ 1 1)), which contains another expression (+ 1 1), does exactly the same thing as the expression (**forward** 2) does.

Turtle Trail

(penup)

move turtle without drawing turtle trail

(pendown)

move turtle with drawing turtle trail

(showturtle)

show turtle

(hideturtle)

hide turtle

(set-turtle-trail *trail_type*)

trail_type: *line, box, cylinder*

set current type of turtle trail to *trail_type*

(set-trail-box *width height*)

assign dimension to trail-box, *width*: x-axis, *height*: z-axis

(set-trail-cylinder *radius*)

assign *radius* to trail-cylinder

3D Geometry Primitives**(line *x1 y1 z1 x2 y2 z2*)**

draw a line from (*x1 y1 z1*) to (*x2 y2 z2*) in relation to the current local coordinate

(triangle *x1 y1 z1 x2 y2 z2*)

draw a triangle from the current local origin to (*x1 y1 z1 x2 y2 z2*)

(box *w h d*)

w: x-axis, *h*: z-axis, *d*: y-axis

(cylinder *r h*)

h: y-axis

(cone *r h*)

h: y-axis

(sphere *r*)

center is at the local origin

(start-poly)

start to draw a convex polygon by driving turtle to enclose a loop, non-convex polygon is not allowed

(end-poly)

finish drawing a convex polygon

Color**(set-color *red green blue*)**

red: 0 ~ 100, *green*: 0 ~ 100, *blue*: 0 ~ 100

Before we start to play with colors, we should be aware of one thing: whenever a particular geometric object is drawn, it is drawn using the currently specified color. In general, we can first set the color and then draw the objects. Until you change the color, all objects are drawn in that color. The initial system default

color is red (100, 0, 0). Therefore FormWriter will draw everything in red unless we change it to a different color.

To set a color, use the command `set-color`. It takes three parameters, all of which are numbers between 0.0 and 100.0. The parameters are, in order, the red, green, and blue "components" of the color. You can think of these three values as specifying a "mix" of colors: 0.0 means don't use any of that component, on the other hand, 100.0 means use all you can of that component. Here are eight basic colors:

<code>(set-color 100 0 0);</code>	red
<code>(set-color 0 100 0);</code>	green
<code>(set-color 0 0 100);</code>	blue
<code>(set-color 100 100 0);</code>	yellow
<code>(set-color 100 0 100);</code>	magenta
<code>(set-color 0 100 100);</code>	cyan
<code>(set-color 100 100 100);</code>	white
<code>(set-color 0 0 0);</code>	black

Visual Thinking and Spatial Reasoning

Sep.16.02

In-Class Exercises

1. Basic Turtle Operation

The easiest way to generate graphics in FormWriter environment is to directly drive the 3-D (flying) turtle's movement. When the turtle moves from one point to the next, it draws a trail between those two points. Try to drive the turtle to make a drawing as shown in Figure 1.1.

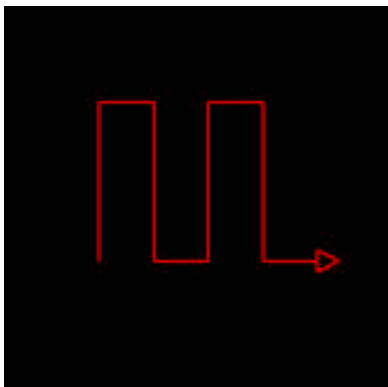


Figure 1.1

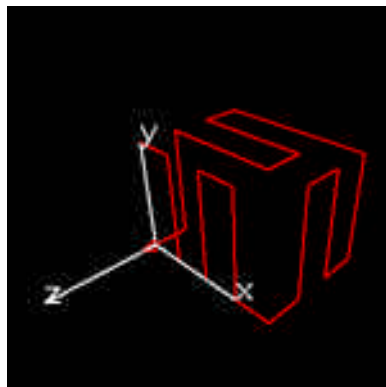


Figure 1.2

The turtle can move around the 3-D world. Try to navigate the turtle around 3-D world to get the result as shown in Figure 1.2.

However, the turtle trail is not always shown. We can use the pair of commands: **penup** and **pendown** to command FormWriter not to draw the turtle trail or to draw it. Play with **penup** and **pendown** to get results like those in Figure 1.3 and 1.4.

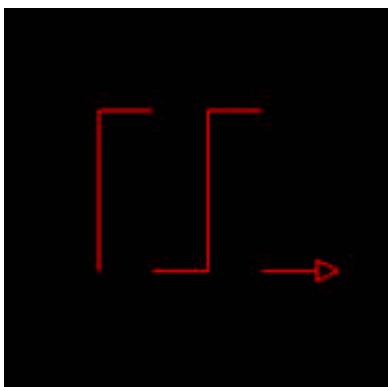


Figure 1.3

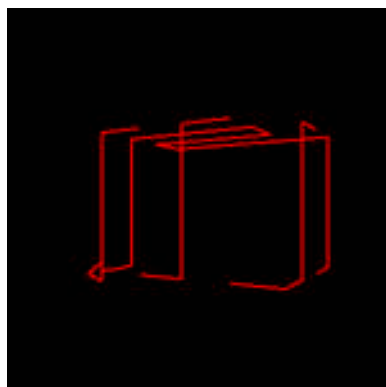


Figure 1.4

FormWriter offers three different types of turtle trail: line trail (as we used before), box trail, and cylinder trail. We can set the dimension of the trail box or cylinder. Change the turtle trail to get the result shown in Figure 1.5.

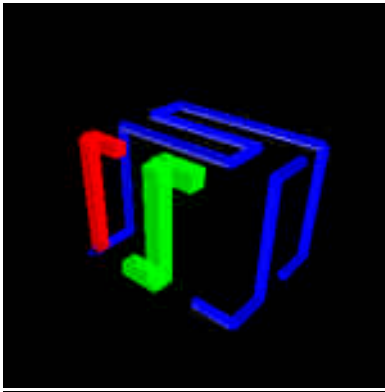


Figure 1.5

2. 3-D Geometric Primitives

In addition to drawing 3-D geometry by the turtle trail, FormWriter provides 3-D geometric primitives to generate 3-D objects: Box, Cylinder, Cone, and Sphere. Play with 3-D geometric primitives to build a robot as shown in Figure 2.

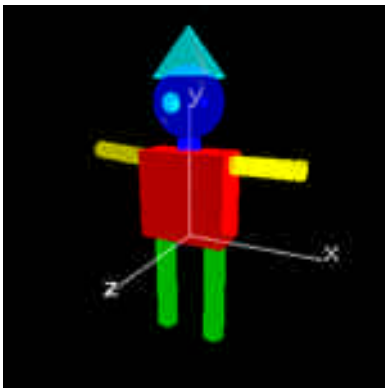


Figure 2

3. Procedure

You can define procedures that will be used repeatedly. Use the `define` command.

```
(define name (parameters)
  (action 1)
  (action 2)
  .
  .
  (action n))
```

First define a procedure to draw a square frame, Figure 3.1, then use this procedure in another procedure to create a cube frame, Figure 3.2.

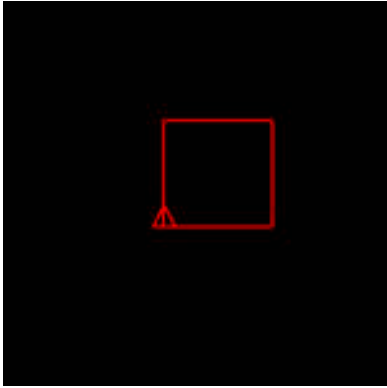


Figure 3.1

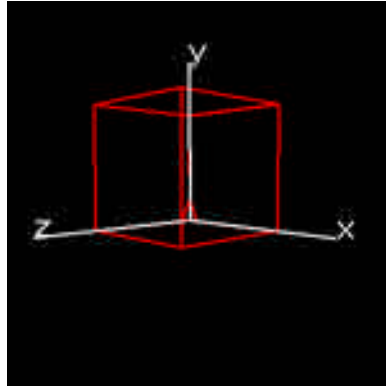


Figure 3.2

A procedure can take a number of parameters. Try to re-define the two previous procedures and make two squares and two cubes in different sizes, Figure 3.3 and 3.4.

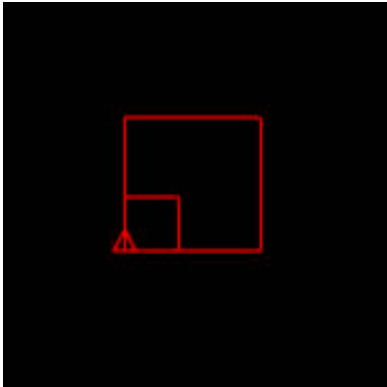


Figure 3.3

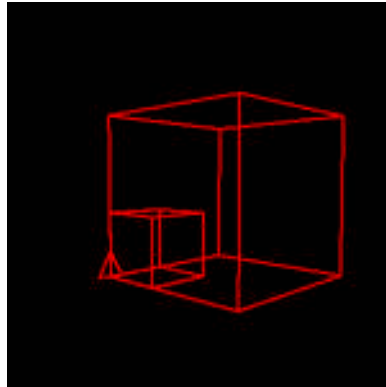
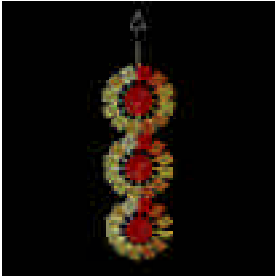


Figure 3.4

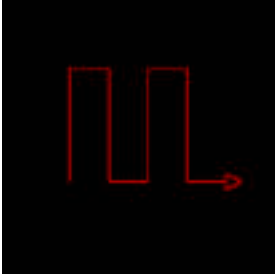
Visual Thinking and Spatial Reasoning

Sep.16.02

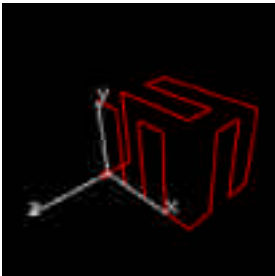
Original Codes for the Demo and In-Class Exercises



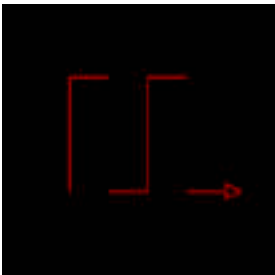
```
(define color-fan ()
  (set-color 100 0 0)
  (sphere 2)
  (repeat i 12
    (set-color 100 (* i 8) (* i 3))
    (set-turtle-trail box)
    (set-trail-box .3 .3)
    (forward 3)
    (roll 30)
    (start-poly)
    (repeat i 4
      (forward 2)
      (right 90))
    (end-poly)
    (set-turtle-trail cylinder)
    (set-trail-cylinder .1)
    (repeat i 4
      (forward 2)
      (right 90))
      (roll -30)
      (back 3)
      (right 30)))
  (repeat i 3
    (color-fan)
    (forward 10))
```



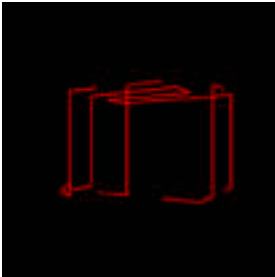
```
(forward 6)      (forward 6)
(right 90)       (left 90)
(forward 2)     (forward 2)
(right 90)
(forward 6)
(left 90)
(forward 2)
(left 90)
(forward 6)
(right 90)
(forward 2)
(right 90)
```



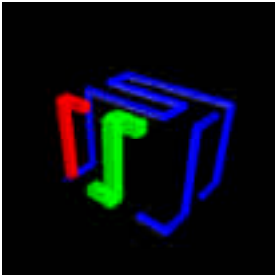
```
(forward 6)      (pitch -90)      (forward 2)
(right 90)       (forward 2)      (left 90)
(forward 2)     (left 90)         (forward 6)
(right 90)      (forward 6)       (right 90)
(forward 6)    (right 90)         (forward 2)
(left 90)      (forward 2)       (right 90)
(forward 2)   (right 90)         (forward 6)
(left 90)     (forward 6)       (pitch -90)
(forward 6)  (left 90)         (forward 6)
(right 90)   (forward 2)       (left 90)
(forward 2)  (left 90)         (forward 2)
(right 90)   (forward 6)
(forward 6)  (pitch -90)
(left 90)    (forward 8)
(forward 2)  (left 90)
```



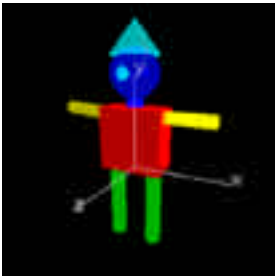
```
(forward 6)      (forward 2)
(right 90)       (right 90)
(forward 2)     (penup)
(right 90)      (forward 6)
(penup)        (left 90)
(forward 6)    (pendown)
(left 90)      (forward 2)
(pendown)
(forward 2)
(left 90)
(forward 6)
(right 90)
```



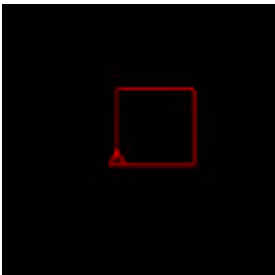
```
(forward 6)      (forward 2)      (forward 2)      (forward 2)
(right 90)       (right 90)       (right 90)       (left 90)
(forward 2)     (penup)          (penup)          (forward 6)
(right 90)     (forward 6)     (forward 6)     (right 90)
(penup)        (left 90)      (left 90)       (forward 2)
(forward 6)     (pendown)       (pendown)       (right 90)
(left 90)       (forward 2)   (forward 2)     (forward 6)
(pendown)      (pitch -90)    (left 90)       (pitch -90)
(forward 2)    (forward 2)   (forward 6)     (forward 6)
(left 90)      (left 90)   (pitch -90)     (left 90)
(forward 6)    (forward 6)   (forward 8)     (forward 2)
(right 90)     (right 90)   (left 90)
```



```
(set-color 100 0 0) (penup) (left 90)
(set-turtle-trail box) (forward 6) (forward 6)
(set-trail-box .6 .6) (left 90) (pitch -90)
(forward 6) (pendown) (forward 8)
(right 90) (set-color 0 0 100) (left 90)
(forward 2) (set-turtle-trail cylinder) (forward 2)
(right 90) (set-trail-cylinder .25) (left 90)
(penup) (forward 2) (forward 6)
(forward 6) (pitch -90) (right 90)
(left 90) (forward 2) (forward 2)
(pendown) (left 90) (right 90)
(set-color 0 100 0) (forward 6) (forward 6)
(set-turtle-trail box) (right 90) (pitch -90)
(set-trail-box .8 .8) (forward 2) (forward 6)
(forward 2) (right 90) (left 90)
(left 90) (penup) (forward 2)
(forward 6) (forward 6)
(right 90) (left 90)
(forward 2) (pendown)
(right 90) (forward 2)
```

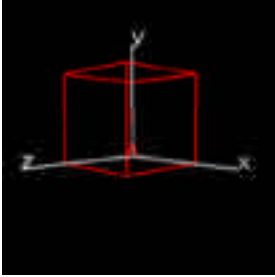


```
(set-color 100 0 0) (home) (home)
(penup) (forward 4) (forward 3.5)
(box 4 2 4) (cylinder 0.5 0.5) (right 90)
(set-color 0 100 0) (forward 2) (forward 2)
(right 90) (sphere 1.5) (cylinder 0.4 3)
(forward 1.3) (forward 1) (home)
(right 90) (set-color 0 100 100) (forward 3.5)
(cylinder 0.5 5) (cone 1.5 2) (left 90)
(home) (back 1) (forward 2)
(left 90) (pitch 90) (right 5)
(forward 1.3) (forward 1.5) (cylinder 0.4 3)
(left 90) (cone 0.4 0.4)
(cylinder 0.5 5) (set-color 100 100 0)
(set-color 0 0 100)
```

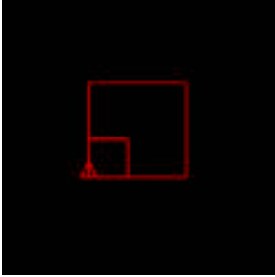


```
(define square ()
(repeat i 4
(forward 4)
(right 90)))

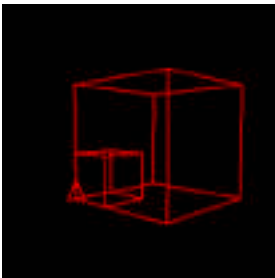
(square)
```



```
(define cube ()  
  (repeat i 4  
    (square)  
    (forward 4)  
    (pitch 90)))  
  
(cube)
```



```
(define square-2 (size)  
  (repeat i 4  
    (forward size)  
    (right 90)))  
  
(square-2 2)  
(square-2 5)
```



```
(define cube-2 (size)  
  (repeat i 4  
    (square-2 size)  
    (forward size)  
    (pitch 90)))  
  
(cube-2 2)  
(cube-2 5)
```