Dr. Geo tutorial RMLL 2005

Hilaire Fernandes

CRDP Aquitaine - OFSET

July 2005

What is Dr.Geo ?

- Dr.Geo is a free dynamic geometry software
- It can be used in primary and secondary education
- It has original features as :

What is Dr.Geo ?

- Dr.Geo is a free dynamic geometry software
- It can be used in primary and secondary education
- It has original features as :
  - The macro-constructions

What is Dr.Geo ?

- Dr.Geo is a free dynamic geometry software
- It can be used in primary and secondary education
- It has original features as :
  - The macro-constructions
  - The Scheme scripts

What is Dr.Geo ?

- Dr.Geo is a free dynamic geometry software
- It can be used in primary and secondary education
- It has original features as :
  - The macro-constructions
  - The Scheme scripts
  - Sketch programmed with Scheme

Where to find Dr.Geo ?

- Its website : `http ://www.ofset.org/drgeo`
- In Freeduc-cd, the OFSET live cd-rom : `http ://www.ofset.org/freeduc-cd`

What is a macro-construction ?

Definition

In short, it is a tool to learn Dr. Geo new complex constructions.

What is a macro-construction ?

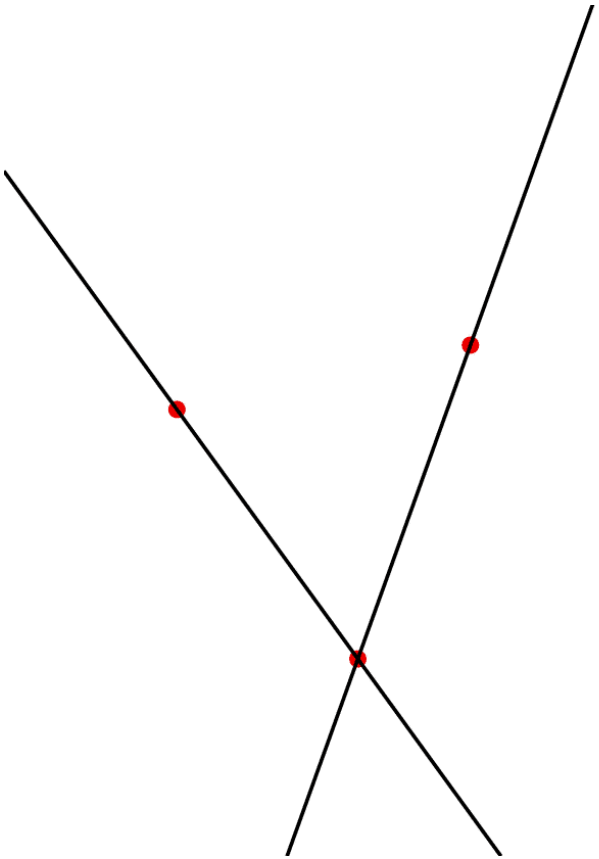
Definition

In short, it is a tool to learn Dr. Geo new complex constructions.

Definition

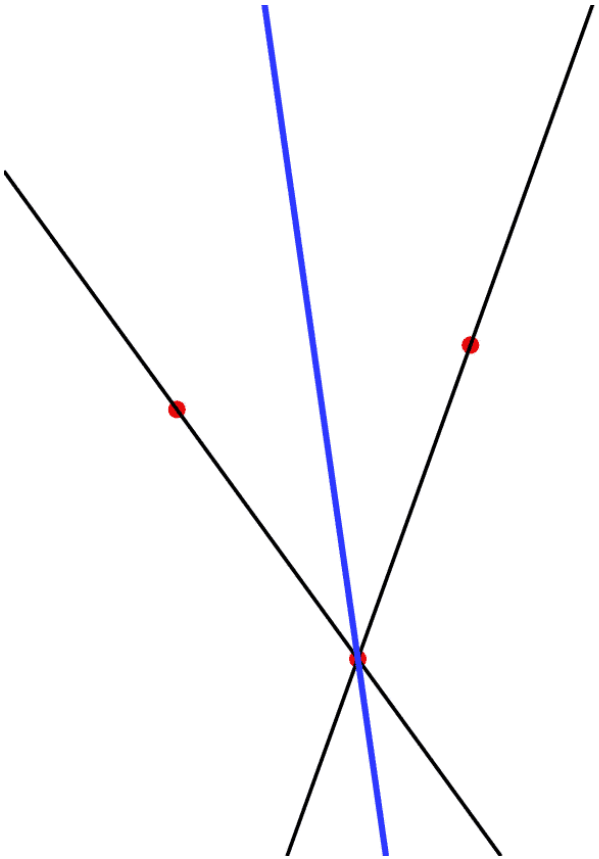
1. *It is a set of constructions recorded in one construction instruction. The user defines all the construction set during the construction phase of the macro.*
2. *When a macro is used, all the construction set is realized in one shot. It is the execution phase of a macro.*
3. *The construction set can be recorded in a file, so it can be used later.*

A bisector, starting situation



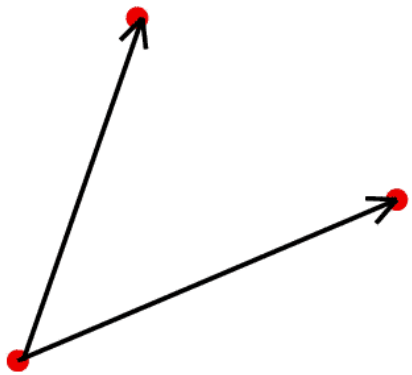
Starting situation : three points and two lines.
It is about learning Dr. Geo how to construct a bisector given three points.

A bisector, expected result



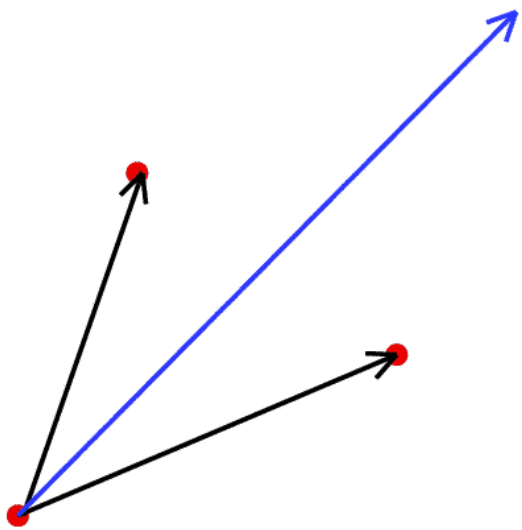
The macro-construction execution over the three points.

Addition of two vectors, starting situation



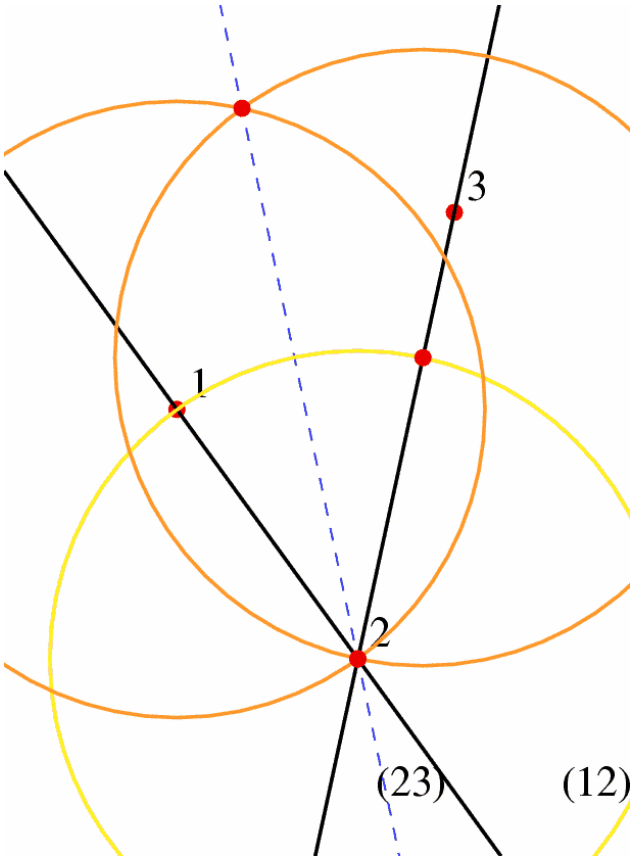
Starting situation : three points and two vectors.
It is about learning Dr. Geo how to construct a vector addition given one point and two vectors.

Addition of two vectors, expected result



The macro-construction execution over a point and two vectors.

Bisector macro-construction

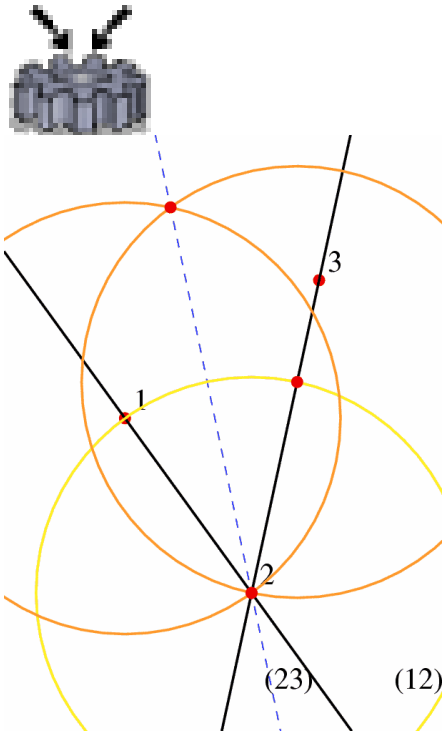


The user resolves the problem once.
For example, using circles to construct two points of the bisector.

Bisector macro-construction, construction

Choose the construction tool of the macro-construction

Bisector macro-construction, construction



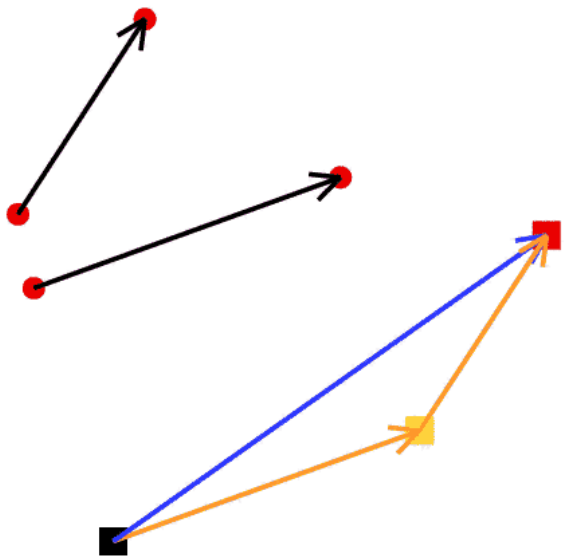
Choose the construction tool of the macro-construction

1. Select the three points 1, 2 et 3 as input objects of the macro-construction
2. Select the line as the output object
3. Set a name and a description to the macro
4. Validate the macro-construction

Bisector macro-construction, use

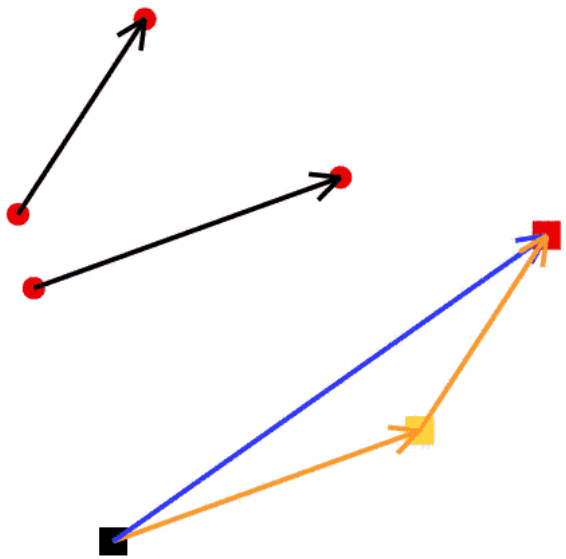
1. Choose the execution tool of the macro-construction
2. Select our macro-construction
3. Click over three points – the input objects
4. Result : the macro-construction immediately creates the bisector

Macro-construction to add two vectors



The user resolves the problem, for example with the translation tool.

Macro-construction to add two vectors



The user resolves the problem, for example with the translation tool.

1. the input objects :
   - the point (black and square)
   - the two vectors (black)
2. the output object, the vector (blue)

What to remember ?

- Creating a macro-construction it is learning Dr. Geo how to conduct the given constructions.

What to remember ?

- Creating a macro-construction it is learning Dr. Geo how to conduct the given constructions.
- A macro-construction expects input objects.

What to remember ?

- Creating a macro-construction it is learning Dr. Geo how to conduct the given constructions.
- A macro-construction expects input objects.
- It returns output objects, result of constructions deduced from the input objects.

What to remember ?

- Creating a macro-construction it is learning Dr. Geo how to conduct the given constructions.
- A macro-construction expects input objects.
- It returns output objects, result of constructions deduced from the input objects.
- All the intermediate objects of the construction are hidden.

What to remember ?

- Creating a macro-construction it is learning Dr. Geo how to conduct the given constructions.
- A macro-construction expects input objects.
- It returns output objects, result of constructions deduced from the input objects.
- All the intermediate objects of the construction are hidden.
- The macro-constructions are saved on file, individually or in set.

What is a script ?

Definition

1. *A script is a function written in the Scheme language.*

What is a script ?

Definition

1. *A script is a function written in the Scheme language.*
2. *As a function, it receives input arguments.*

What is a script ?

Definition

1. *A script is a function written in the Scheme language.*
2. *As a function, it receives input arguments.*
3. *Still as a function, it returns value.*

What is a script ?

Definition

1. *A script is a function written in the Scheme language.*
2. *As a function, it receives input arguments.*
3. *Still as a function, it returns value.*
4. *This value is alway printed in the sketch. It can be used for the coming after constructions.*

What is a script ?

Definition

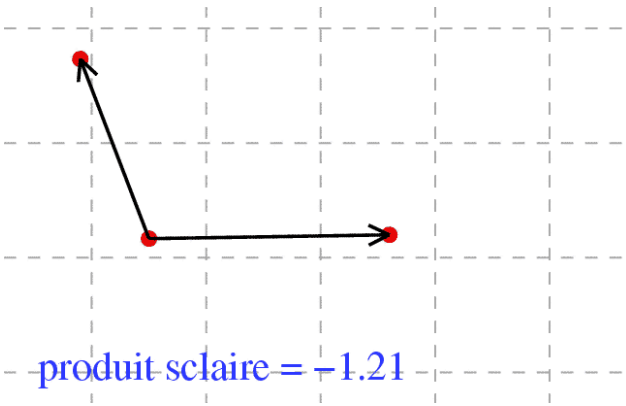
1. *A script is a function written in the Scheme language.*
2. *As a function, it receives input arguments.*
3. *Still as a function, it returns value.*
4. *This value is alway printed in the sketch. It can be used for the coming after constructions.*
5. *The input arguments are references to other objects in the current sketch.*

What is a script ?

Definition

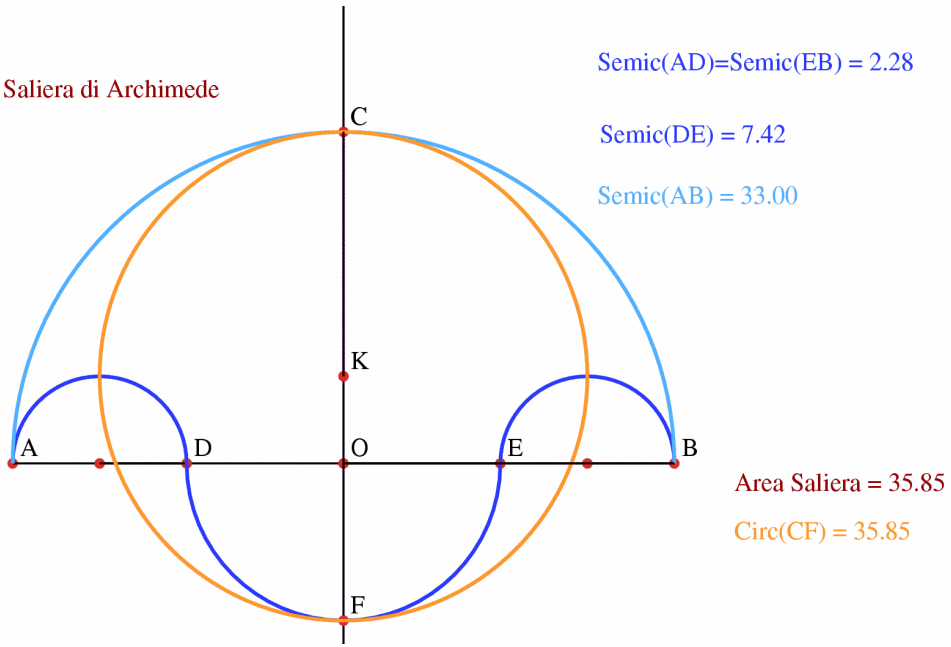
1. *A script is a function written in the Scheme language.*
2. *As a function, it receives input arguments.*
3. *Still as a function, it returns value.*
4. *This value is alway printed in the sketch. It can be used for the coming after constructions.*
5. *The input arguments are references to other objects in the current sketch.*
6. *Dr. Geo proposes an API – application programming interface – to request information from the objects.*

Dot product calculus



Starting situation : two vectors.
With the help of a script, we want to compute the dot product of these two vectors.

Surface calculus and comparison

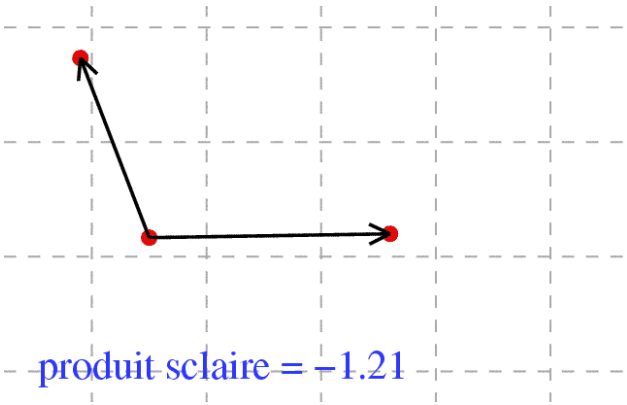


Starting situation : La saliera of Archimedes.
We want to calculate the delimited surfaces.
We can use scripts in cascade to achieve these calculus.

Dot product calculus, creating the script

Choose the tool to create script.

Dot product calculus, creating the script



Choose the tool to create script.

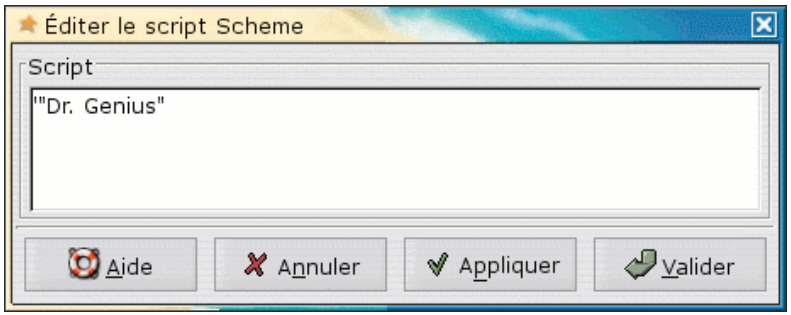
1. In the sketch, select the two vectors as input arguments of the script.
2. Click somewhere in the sketch background to set the script there.
3. The message *Dr. Genius* is printed at this place, it is where is anchored the script.

Dot product calculus, edit the script

Choose the property tool and click over the script.

Dot product calculus, edit the script

Choose the property tool and click over the script.



Replace the content of the script with :

```
(let  ((u (getCoordinates a1))
       (v (getCoordinates a2)))
   (+ (* (car u) (car v))
      (* (cadr u) (cadr v))))
```

then apply the modifications.

Dot product calculus, explanations 1/2

```
(let  ((u (getCoordinates a1))
       (v (getCoordinates a2)))
```

- a1 et a2 are the two arguments of the script. These are references to our two vectors.
- `getCoordinates` is a function returning a list of coordinates.

Dot product calculus, explanations 2/2

```
(+ (* (car u) (car v))
   (* (cadr u) (cadr v))))
```

`car` et `cadr` get the 1st and 2nd element of the coordinates list : abscissa and ordinate.

Associate script and macro-construction

This extend Dr.Geo with *macro-scripts* to :

- calculate the dot product of two vectors
- calculate an algebraic measure
- construct a polygon
- and may others things...

What to remember ?

- A script is a function which returns a value printed in the sketch.

What to remember ?

- A script is a function which returns a value printed in the sketch.
- I receives input arguments, references to other objects in the sketch.

What to remember ?

- A script is a function which returns a value printed in the sketch.
- I receives input arguments, references to other objects in the sketch.
- Dr. Geo proposes an API to request, from a script, information from the objects.

What to remember ?

- A script is a function which returns a value printed in the sketch.
- I receives input arguments, references to other objects in the sketch.
- Dr. Geo proposes an API to request, from a script, information from the objects.
- The scripts help to extend macro-constructions : it is the *macro-scripts*.

What is a programmed Scheme sketch ?

Definition

1. *It is a sketch described in a semi-natural language.*

What is a programmed Scheme sketch ?

Definition

1. *It is a sketch described in a semi-natural language.*
2. *The computer language is Scheme, a variant of Lisp.*

What is a programmed Scheme sketch ?

Definition

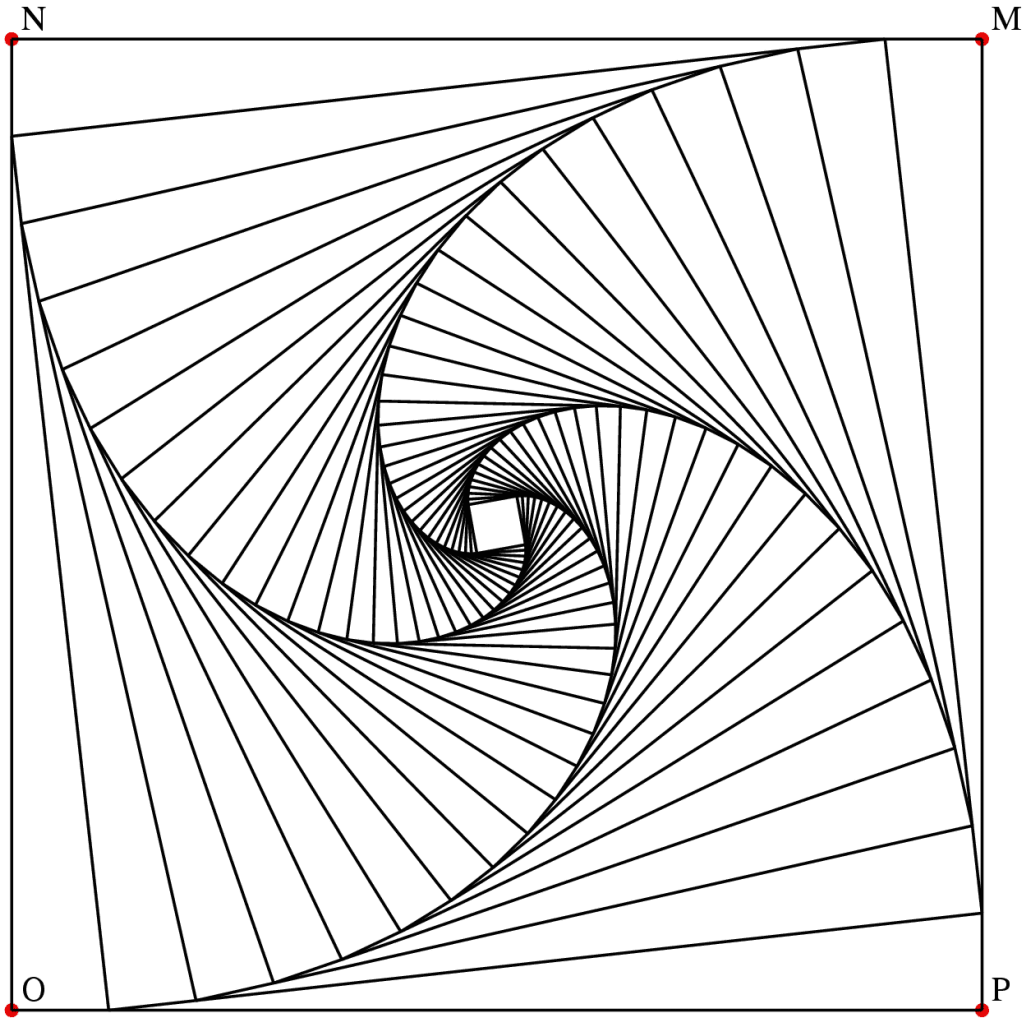
1. *It is a sketch described in a semi-natural language.*
2. *The computer language is Scheme, a variant of Lisp.*
3. *The language allows iterative or recursive construction.*

What is a programmed Scheme sketch ?

Definition

1. *It is a sketch described in a semi-natural language.*
2. *The computer language is Scheme, a variant of Lisp.*
3. *The language allows iterative or recursive construction.*
4. *Dr. Geo proposes a well documented API – application programming interface.*

Spiral – Sketch



Spiral – Description

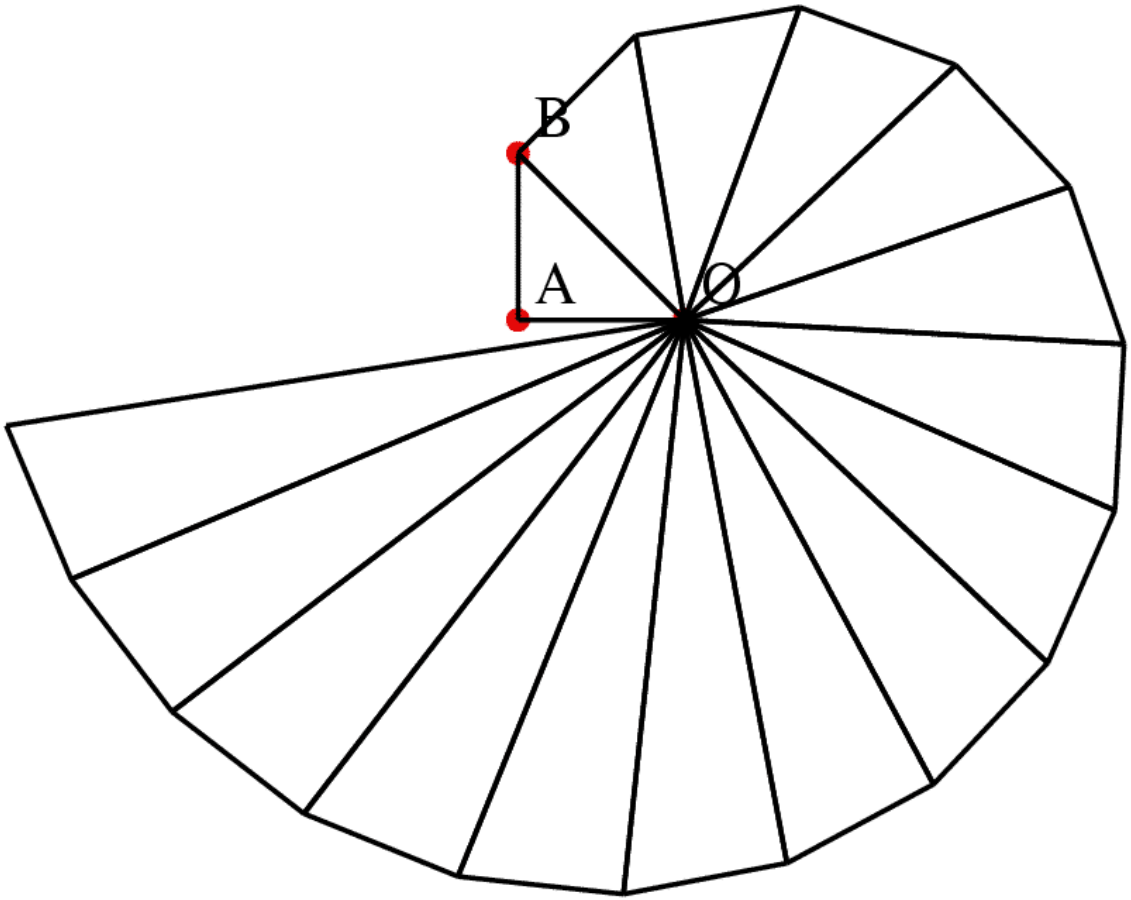
```
(new-figure "Spiral")

(define (square p1 p2 p3 p4 n)
   (let* ((s1  (Segment "" extremities     p1 p2))
          (s2  (Segment "" extremities     p2 p3))
          (s3  (Segment "" extremities     p3 p4))
          (s4  (Segment "" extremities     p4 p1))
          (A   (Point   "" on-curve    s1   1/10))
          (B   (Point   "" on-curve    s2   1/10))
          (C   (Point   "" on-curve    s3   1/10))
          (D   (Point   "" on-curve    s4   1/10)))
     (send A masked)
     (send B masked)
     (send C masked)
     (send D masked)
 (if (> n 0)
     (square  A B C D (- n 1)))))

(lets Point "M" free  5  5)
(lets Point "N" free -5  5)
(lets Point "O" free -5 -5)
(lets Point "P" free  5 -5)

(square M N O P 30)
```

Construction of square root – Sketch



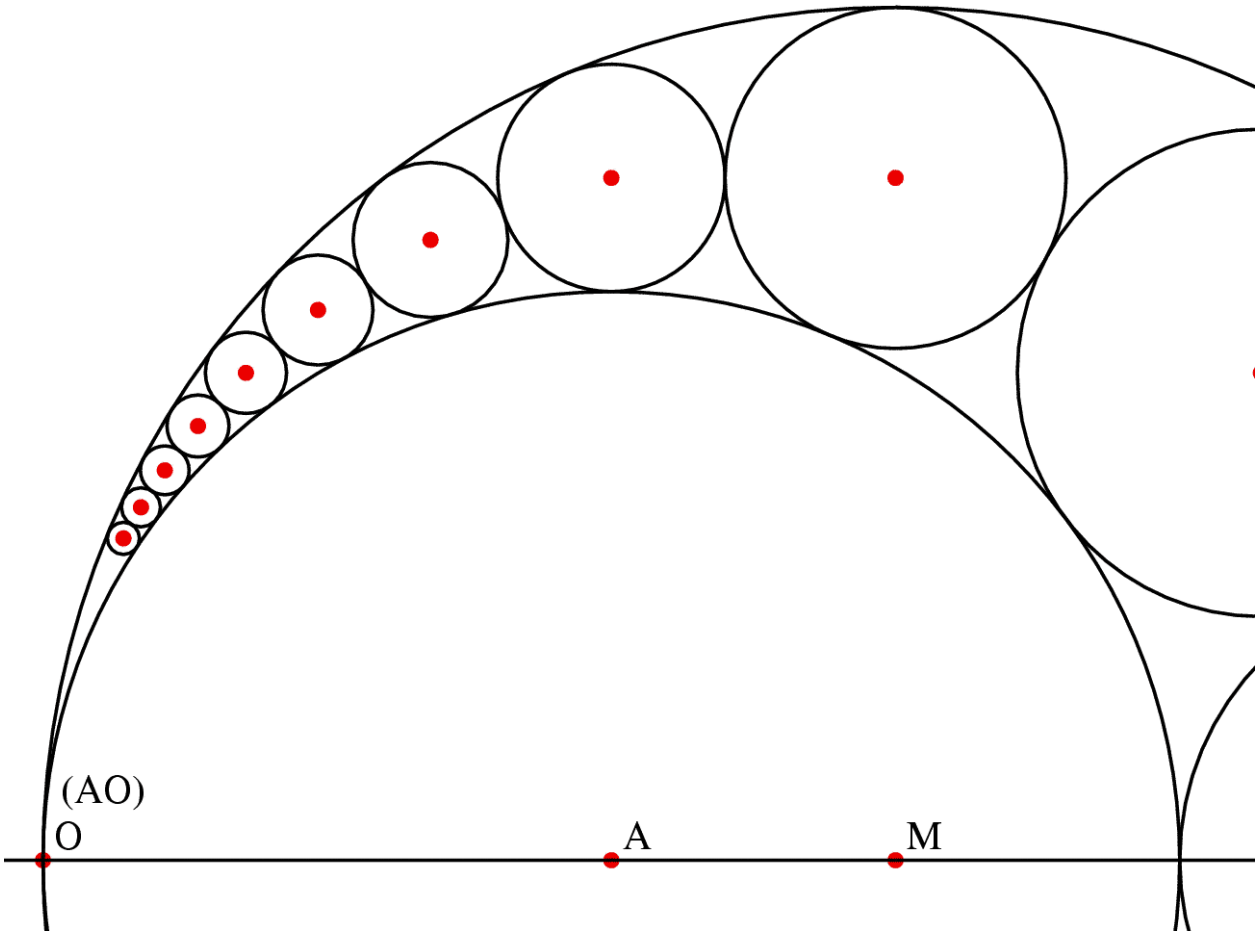
Construction of square root – Description

```
(nouvelle-figure "Triangle")

(define (triangle p1 p2 p3 n)
  (let* ((s1 (Segment "" extremities    p1 p2))
         (s2 (Segment "" extremities    p2 p3))
         (s3 (Segment "" extremities    p3 p1))
         (pe (Line    "" orthogonal  p3 s3))
         (ci (Circle  "" center-segment p3 s2))
         (p4 (Point   "" intersection2  pe ci)))
    (send pe masked)
    (send ci masked)
    (send p4 masked)
    (if (> n 0)
       (triangle p1 p3 p4 (- n 1)))))

(lets Point "O" free  0  0)
(lets Point "A" free -1  0)
(lets Point "B" free -1  1)
(triangle O A B 15)
```

Pappo – Sketch

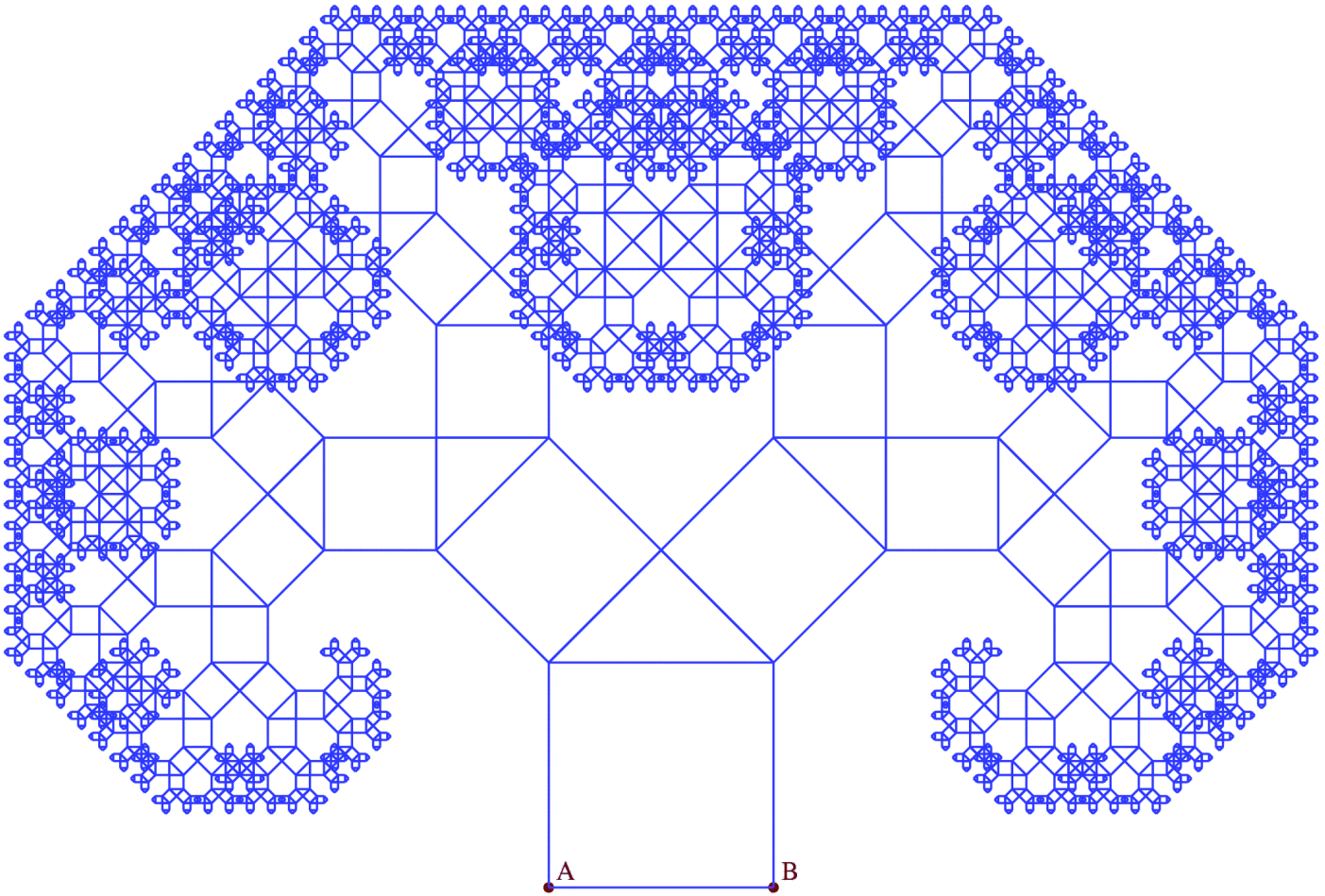


Pappo – Description

```
(new-figure "Pappo")
(define (circle n)
(let*((r (Numeric "" free 0 0 (/ 15 ( + 6 (* n n )))))
      (c (Point   "" free (* 5 (/ 15 ( + 6 (* n n ))))
                          (* 2 (* n (/ 15 ( + 6 (* n n )))))))
      (p (Circle "" centre-rayon c r)))
  (send r masked)
  (if (> n 0)
    (circle (- n 1)))))

(circle 10)
(lets Point "A" free 5 0)
(lets Point "O" free 0   0)
(lets Point "B" free 15 0)
(lets Point "M" middle-2pts B O)
(lets Circle "" 2points M O)
(lets Circle "" 2points A O)
(lets Line   "" 2points A O)
```

Fractal – Sketch

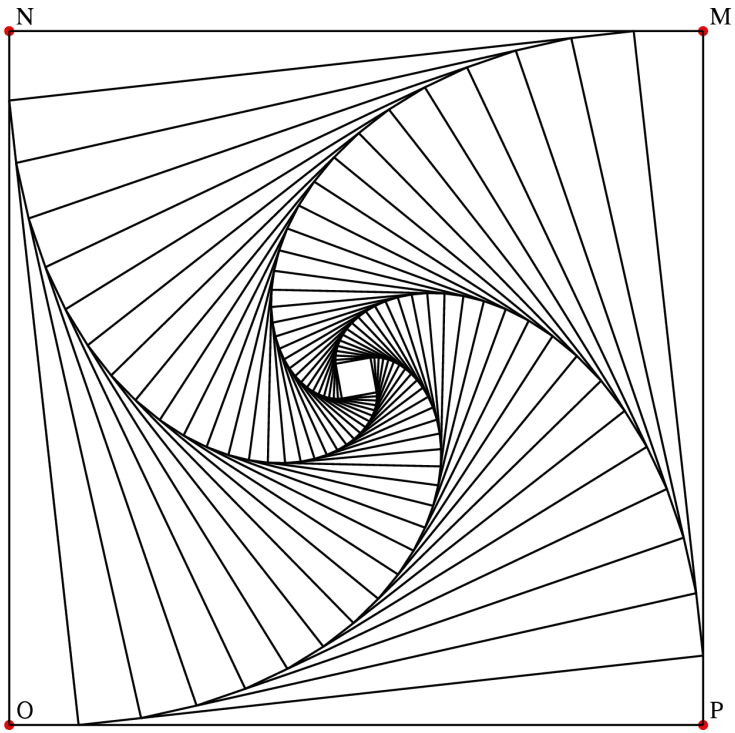


Fractal – Description

```
(new-figure "Une belle Fractale")
(lets Numeric "a1" free 1 2 (acos 0))
(lets Numeric "a2" free 1 3 (- 0 (acos 0)))
(send a1 masked)
(send a2 masked)
(define (pythagore p1 p2 n)
   (send p1 masked)
   (send p2 masked)
   (let* (
      (p4 (Point "" rotation p2 p1 a1))
      (p3 (Point "" rotation p1 p2 a2))
      (s1 (Segment ""extremities p1 p2))
      (s2 (Segment "" extremities p2 p3))
      (s3 (Segment "" extremities p3 p4))
      (s4 (Segment "" extremities p4 p1))
      (O  (Point "" middle-2pts p3 p4))
      (M  (Point "" rotation p3 O a1))
      (s5 (Segment "" extremities p4 M))
      (s6 (Segment "" extremities M p3)))
      (send p1 masked)
      (send p2 masked)
      (send p3 masked)
      (send p4 masked)
      (send O masked)
      (send M masked)
      (if (> n 0)
         (begin
            (pythagore p4 M (- n 1))
            (pythagore M p3 (- n 1))))))

(lets Point "A" free -5 0)
(lets Point "B" free -2 0)
(pythagore A B 10)
```

Spiral – Explanations



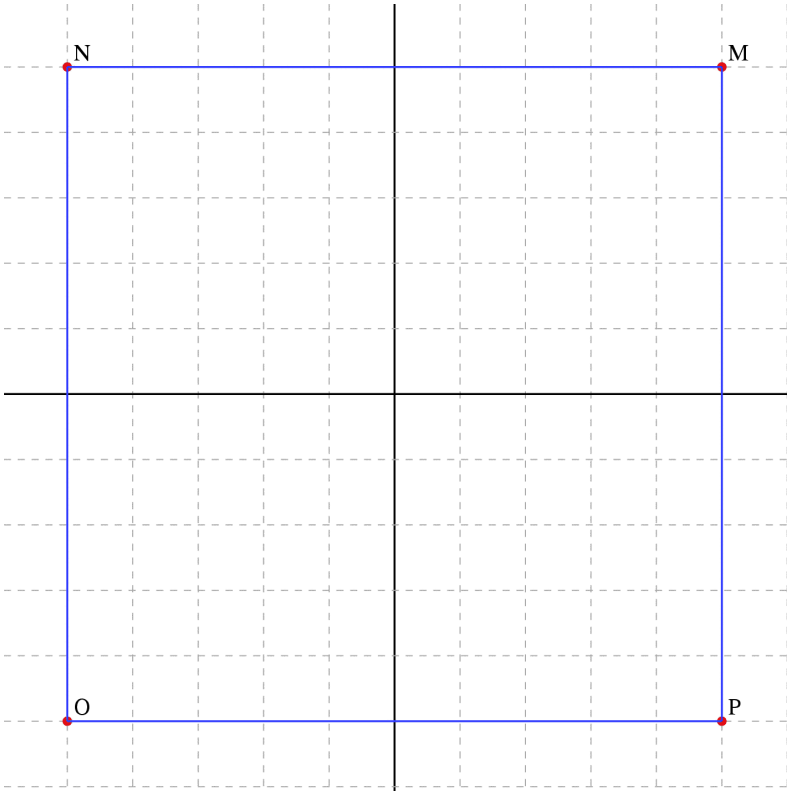

```
(new-figure "Spiral")

(define (square p1 p2 p3 p4 n)
   (let* ((s1  (Segment "" extremities     p1 p2))
          (s2  (Segment "" extremities     p2 p3))
          (s3  (Segment "" extremities     p3 p4))
          (s4  (Segment "" extremities     p4 p1))
          (A   (Point   "" on-curve    s1   1/10))
          (B   (Point   "" on-curve    s2   1/10))
          (C   (Point   "" on-curve    s3   1/10))
          (D   (Point   "" on-curve    s4   1/10)))
     (send A masked)
     (send B masked)
     (send C masked)
     (send D masked)
 (if (> n 0)
     (square  A B C D (- n 1)))))

(lets Point "M" free  5  5)
(lets Point "N" free -5  5)
(lets Point "O" free -5 -5)
(lets Point "P" free  5 -5)

(square M N O P 30)
```

Spiral – Recursion of depth 0



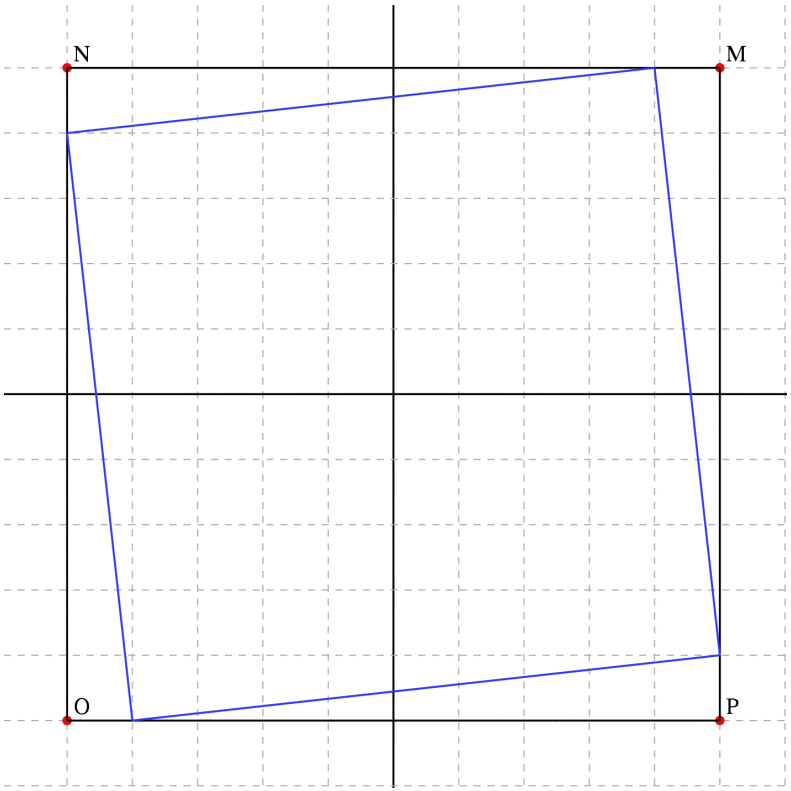

```
(new-figure "Spiral")

(define (square p1 p2 p3 p4 n)
   (let* ((s1  (Segment "" extremities     p1 p2))
          (s2  (Segment "" extremities     p2 p3))
          (s3  (Segment "" extremities     p3 p4))
          (s4  (Segment "" extremities     p4 p1))
          (A   (Point   "" on-curve    s1   1/10))
          (B   (Point   "" on-curve    s2   1/10))
          (C   (Point   "" on-curve    s3   1/10))
          (D   (Point   "" on-curve    s4   1/10)))
     (send A masked)
     (send B masked)
     (send C masked)
     (send D masked)
 (if (> n 0)
     (square  A B C D (- n 1)))))

(lets Point "M" free  5  5)
(lets Point "N" free -5  5)
(lets Point "O" free -5 -5)
(lets Point "P" free  5 -5)

(square M N O P 0)
```

Spiral – Recursion of depth 1



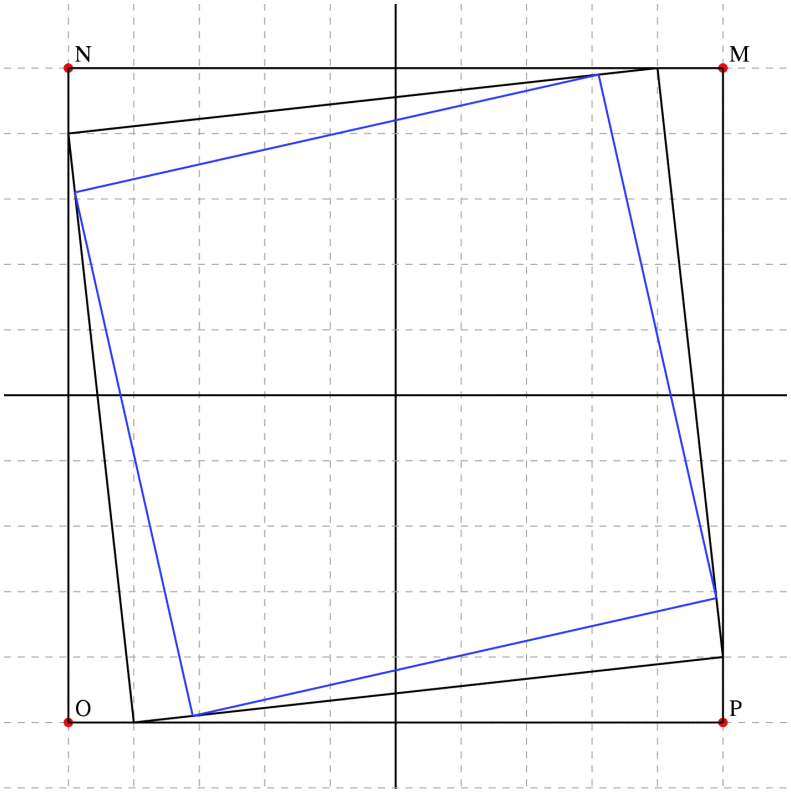

```
(new-figure "Spiral")

(define (square p1 p2 p3 p4 n)
   (let* ((s1  (Segment "" extremities     p1 p2))
          (s2  (Segment "" extremities     p2 p3))
          (s3  (Segment "" extremities     p3 p4))
          (s4  (Segment "" extremities     p4 p1))
          (A   (Point   "" on-curve    s1   1/10))
          (B   (Point   "" on-curve    s2   1/10))
          (C   (Point   "" on-curve    s3   1/10))
          (D   (Point   "" on-curve    s4   1/10)))
     (send A masked)
     (send B masked)
     (send C masked)
     (send D masked)
 (if (> n 0)
     (square  A B C D (- n 1)))))

(lets Point "M" free  5  5)
(lets Point "N" free -5  5)
(lets Point "O" free -5 -5)
(lets Point "P" free  5 -5)

(square M N O P 1)
```

Spiral – Recursion of depth 2




```
(new-figure "Spiral")

(define (square p1 p2 p3 p4 n)
   (let* ((s1  (Segment "" extremities     p1 p2))
          (s2  (Segment "" extremities     p2 p3))
          (s3  (Segment "" extremities     p3 p4))
          (s4  (Segment "" extremities     p4 p1))
          (A   (Point   "" on-curve    s1   1/10))
          (B   (Point   "" on-curve    s2   1/10))
          (C   (Point   "" on-curve    s3   1/10))
          (D   (Point   "" on-curve    s4   1/10)))
     (send A masked)
     (send B masked)
     (send C masked)
     (send D masked)
 (if (> n 0)
     (square  A B C D (- n 1)))))

(lets Point "M" free  5  5)
(lets Point "N" free -5  5)
(lets Point "O" free -5 -5)
(lets Point "P" free  5 -5)

(square M N O P 2)
```

Spiral – Main commands 1/

Code sample

```
(lets Point "M" free 5 5)
```

Declare a free point, named M with coordinates (5 ;5).
`lets` is a key word to create an object ; the created object is referenced in the variable `M`.

Spiral – Main commands 2/

Code sample

```
(let* ((s1  (Segment "" extremities p1 p2))
       (A   (Point   "" on-curve  s1   1/10))
```

`let*` contains a local variables declaration bloc, the variables scope is the function `square`.

In this example, the `lets` keyword is not used, instead functions calls are used to create the geometric objects .

The results of these function calls are affected in the variables `s1`, `A`, etc.

Spiral – Main commands 3/

Code sample

```
(Point   ""   on-curve   s1   1/10)
```

This call creates a point on the curve `s1`, with the curve abscissa 1/10.
Whatever the curve type (line, ray, circle, etc), a point belonging to a curve has an abscissa in the range [0 ;1].

Programmed sketch – Some advices

- Use a text editor adapted to the Scheme language – here we have used the SciTe editor. Your selected editor must handle at the minimum :

Programmed sketch – Some advices

- Use a text editor adapted to the Scheme language – here we have used the SciTe editor. Your selected editor must handle at the minimum :
  - syntax highlighting

Programmed sketch – Some advices

- Use a text editor adapted to the Scheme language – here we have used the SciTe editor. Your selected editor must handle at the minimum :
  - syntax highlighting
  - bracket matching – there is a lot in Scheme and it is mandatory to use a suited text editor.

Programmed sketch – Some advices

- Use a text editor adapted to the Scheme language – here we have used the SciTe editor. Your selected editor must handle at the minimum :
  - syntax highlighting
  - bracket matching – there is a lot in Scheme and it is mandatory to use a suited text editor.
- Study carefully the Dr. Geo documentation and the examples coming with the documentation.

Programmed sketch – Some advices

- Use a text editor adapted to the Scheme language – here we have used the SciTe editor. Your selected editor must handle at the minimum :
  - syntax highlighting
  - bracket matching – there is a lot in Scheme and it is mandatory to use a suited text editor.
- Study carefully the Dr. Geo documentation and the examples coming with the documentation.
- Train yourself with the Scheme language – Dr. Scheme is a free Scheme environment to learn it.

Thanks for your intention

http ://www.ofset.org/drgeo