REFLECTING ON THE REFLECTION

1 The challenge

When working in computer environments for geometric explorations you usually make the constructions by clicking and dragging. This was the case with your explorations on geometric transformation in GeoGebra. Here you will learn to "talk to the computer" in a geometrics language it understands – the language of Geomland. You will use the basic words (the so called *primitives*) in-built in its vocabulary (e.g. `point`, `segment`, `line`, `ray`, `circle`, `length`, `heading`, `radius`, `center`, etc) and will explain to it what *reflection* means in geometrical context.

2 Warming up

All examples in this document are written in Elica Geomland – this is Geomland reimplemented as a library of Elica. To run these examples you need to download and install Elica on your computer. Note that Elica is a free application and it is tuned for Windows XP operating system It can be downloaded from `www.elica.net`. To install it execute the downloaded file (it should be named `Elica56Setup.zip` for Elica 5.6) and follow the installation instructions.

The examples from this document together with the Geomland library are not part of Elica installation package, but are provided as a ZIP file accompanying this document. Unzip the file in a directory of your choice. To run an example double click on its name. Alternatively you can start examples from Elica's main menu: File|Open (or Ctrl-O) to open the file, and Run|Start (or F9) to run the example.

In case it is not possible to install or run Elica you can still view all examples (*.ELI files) with a standard text editor.

In order to construct a geometric object in Geomland you specify the name of the object (proceeded by double quotes `"`), its type (e.g. `point`, `line`, etc.) and its value. Here are some examples for you to start with:

2.1 Constructing points

To construct a point you use the command `ob` (from object), the name of the point (e.g. O) and specify its coordinates (e.g. `100 50`):

```
ob "O point 100 50
```

Task 1 Construct at least 3 points of your own choice.

Hint: You could start from scratch, or you could edit a previous command, e.g:

```
ob "O1 point 100 -50
```

2.2 Constructing lines

Similarly to constructing a point you use the command `ob`, the name of the line and specify the values of its components. If you would like to construct a line passing through two points which already exist, you could use their values in the construction, e.g.:

```
ob "L line :O :O1
```

Note that the values are denoted by "**:**"**(** a column) in front of the point's name.

Task 2 Construct at least 3 lines of your own choice.

Hint: You could start from scratch, or you could edit a previous command, e.g.

```
ob "L1 line :O1 :O2
```

where the points O2 and O3should have been already constructed.

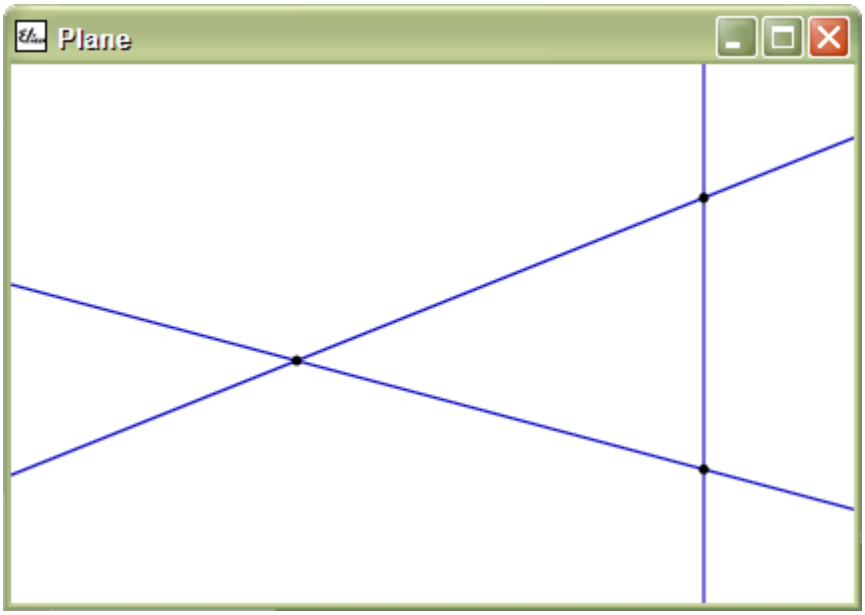

Figure 1 Three points and three lines

Questions for you to reflect on:

Q1 Do you guess how to construct a line from scratch (without first constructing the points it passes trough)?

Hint: You could use the primitive `point` in the construction of a line, e.g.

```
ob "L1 line point 30 30 point 100 50
```

Enter the command and explain what happens. Will there be any difference if you first construct the points $O_2(30, 30)$ and $O_3(100, 50)$ and then use the command

```
ob "L1 line :O2 :O3
```

Q2 Could the names of the points and lines be more exotic, e.g. Banana, Giraffe, *etc.* Try and explain why you don't see them in the standard textbooks.

Q3 How many points are needed for constructing a line?

Q4 Are 3 points too many? Try to connect 3 points which should be on the same line and explain the message you get if you force `line` to use three points.

Q5 Is one point not sufficient? Try the reaction of the computer and discuss it with a peer.

Yes, three are too many and one point is not sufficient as the message shows. The real question is*: Does the second input for LINE have to be a point?* Not really.

There is another way to construct a line – specifying a point through which it passes and its heading, e.g.

```
ob "L1 line :O 30
```

Q6 How many lines can pass through a single point? Try to convince your peers.

2.2 Constructing parallel and perpendicular lines

Task 3 Construct the Cartesian coordinate system.

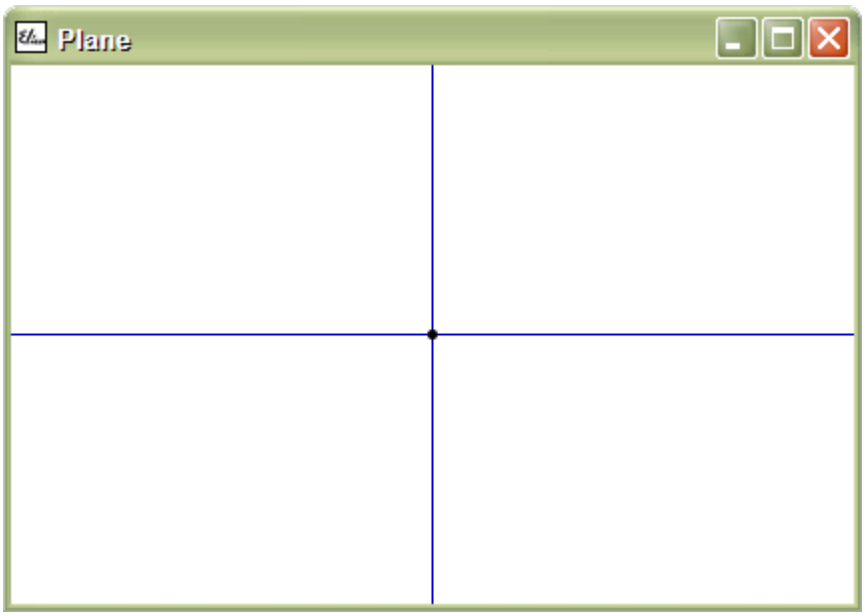

Figure 2 Cartesian coordinate system

Hint: You could first edit the command for constructing the existing point O by changing only the coordinates and then pass the axis Ox through it by the command:

```
ob "O point 0 0
ob "Ox line :O 0
```

or you could construct it directly (without using an already existing point):

```
ob "Ox line (point 0 0) 0
```

The parentheses are put only for readability; the computer knows how many inputs `point` and `line` need.

Note that the heading of a line is specified by the number of degrees of the angle between the axis Ox and the line under construction in counter clock direction. In order to construct the ordinate axis you could edit the command for the line Ox by changing the name to Oy and the degrees from 0 to 90.

Task 4 Construct two perpendicular lines passing through the point (30, 30). Can you do it in different ways?

Task 5 Construct two parallel lines passing through the point (30, 30). Can you do it in different ways?

Questions for you to reflect on:

Q7 Consider two parallel lines you have constructed and change the direction of the first one. What happens to the other one – does it remain parallel? Explain.

Q8 Consider two perpendicular lines you have constructed and change the direction of the first one. What happens to the other one – does it remain parallel? Discuss your finding with a peer.

3 Teaching the computer how to reflect

3.1 Defining procedures for reflecting a point with respect to a line

Task 6 Construct a point which is the reflection P_1 of a given point P with respect to a given line L.

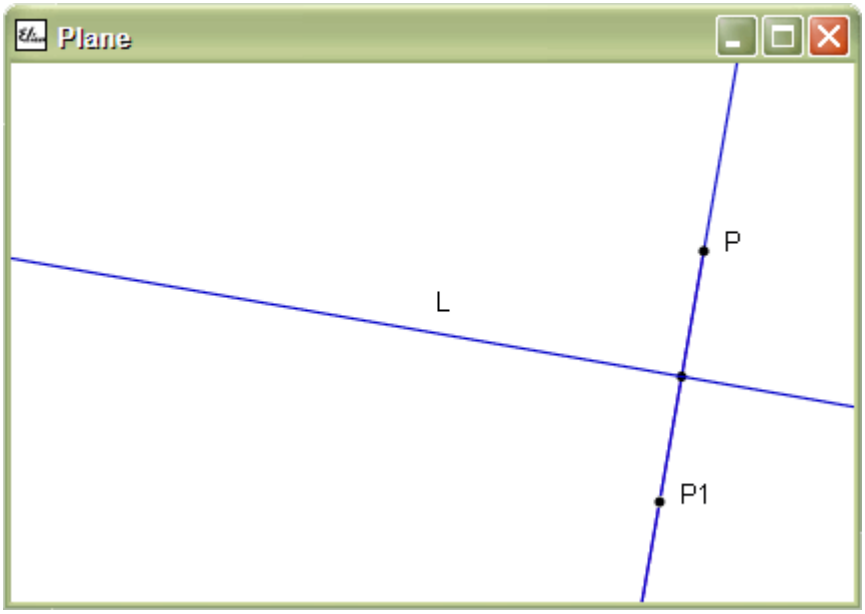

Figure 3 Reflection of a point with respect to a line

Hint. As you already know the reflection point could be constructed by the following steps:

- constructing a **line** L_1 passing through **point** P and **perpendicular** to L
- constructing the **intersecting** point (O) of the two lines
- constructing a **point** which is **on** the **ray** from P to O at a distance twice as long as the distance between P and O.

You know how to translate this algorithm in the language of Geomland, right? Here is one way to do this:

```
ob "L1 line :P 90+heading :L
ob "O isec :L :L1
ob "R ray :P :O
ob "P1 pointon :R 2*distance :P :O
```

What is very interesting in Geomland is that you could teach the computer to find the reflection point of any point with respect to any line by means of the following procedure:

```
to reflect_pl :P :L
   local "L1 "O "R
   ob "L1 line :P 90+heading :L
   ob "O isec :L :L1
   ob "R ray :P :O
   output pointon :R 2*distance :P :O
end
```

The suffix `_pl` stands for *point* and *line*. These are the types of the objects that can be used as inputs. We will define several other reflects, so this suffix is just a reminder to us.

Now you can construct an arbitrary point (e.g. Q) and an arbitrary line (e.g. M) and find the reflection point Q_1 of Q with respect to M by the command:

```
ob "Q1 reflect_pl :Q :M
```

Task 7 Construct the reflection point of the point Q (100, 50) with respect to the line M passing through the point (0, 0) and:

- passing through the point (-30, 30)
- with a heading 60.

Q9 Can you do this without first constructing M and Q?

Task 8 Edit the procedure `reflect_pl` by omitting the intermediate steps.

Hint: Replace the corresponding values of the objects in the output line different from P and L:

```
to reflect_pl :P :L
   output (pointon ray :P isec :L line :P 90+heading :L
            2*distance :P isec :L line :P 90+heading :L)
end
```

This procedure makes use of the functionality of the Geomland language and reflects more closely the corresponding mathematical definition.

Questions for you to reflect on

Here is another version of the reflect_pl procedure which seems synonymous to the first one and even more economical.

```
to reflect_pl :P :L
   local "L1 "O "R
   make "L1 line :P 90+heading :L
   make "O isec :L :L1
   output pointon :L1 2*distance :P :O
end
```

Q10 Does it always work properly? What would the reflection point P with respect to L be? Does it coincide with P? Check your answer with all the versions of the `reflect_pl` procedure and explain.

3.2 Defining procedures for reflecting a line with respect to a line

Task 9 Define a procedure for reflecting an arbitrary line L_1 with respect to a given line L.

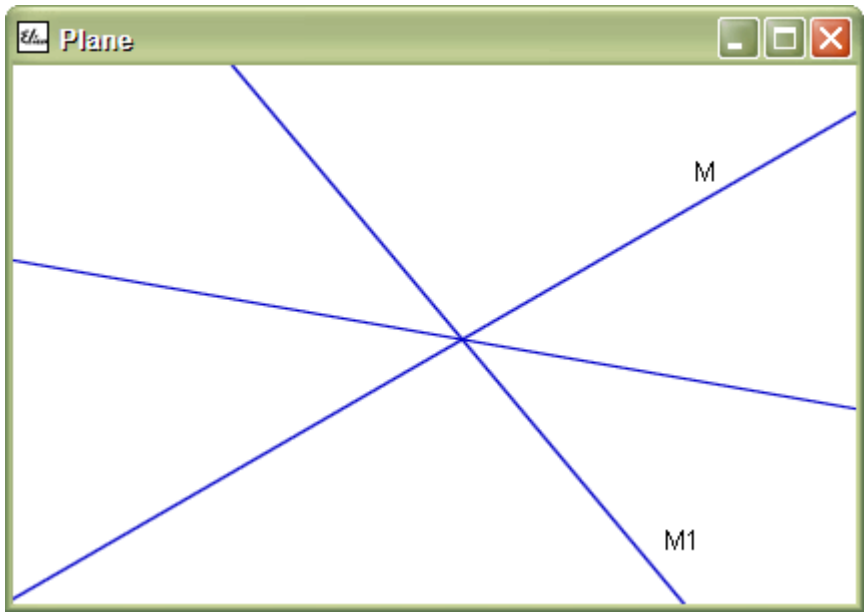

Figure 4 Reflection of a line with respect to a line

Hint: Consider the procedure `reflect_ll` below and try to reduce it to a single line (as above):

```
to reflect_ll :L1 :L
      local "A "B
      make "A initial :L1 0
      make "B pointon :L1 100
      output line reflect_pl :A :L reflect_pl :B :L
end
```

3.3 Defining procedures for reflecting an object with respect to a line

Task 10 Define a procedure for reflecting;

- a segment
- a circle
- a ray with respect to a given line L.

Task 11 Generalize your procedures to reflect an arbitrary object with respect to a line.

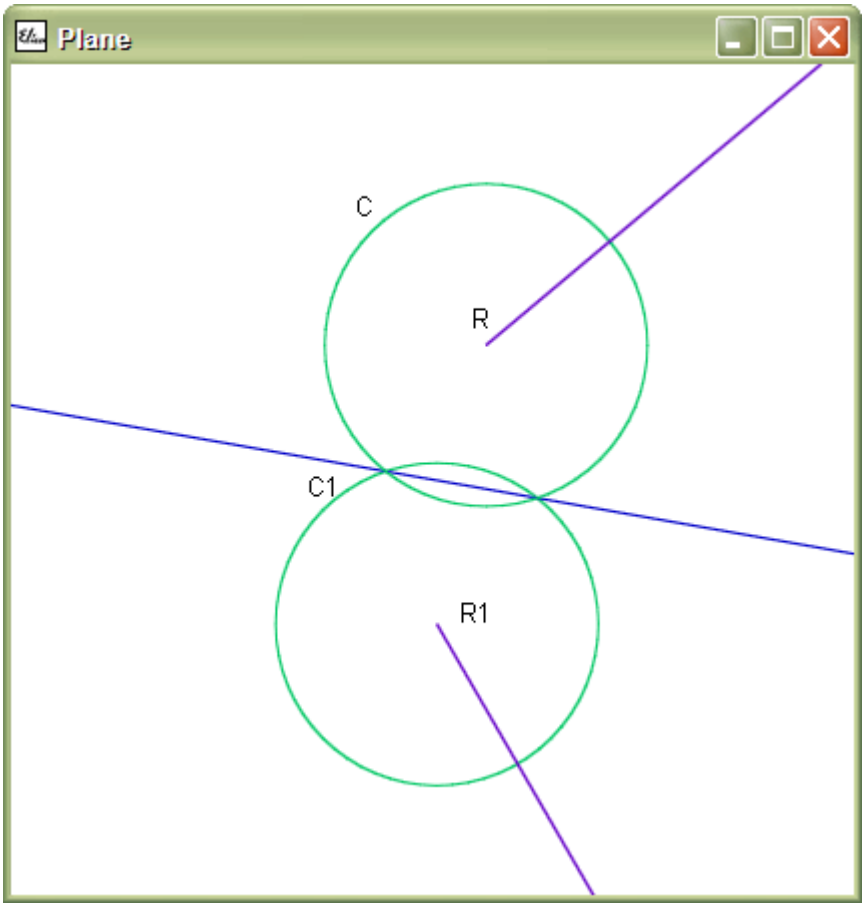

Figure 5 Reflection of a circle and a ray

Hint: The first approach is to consider the cases of the different objects you have already a procedure for:

```
to reflect_l :O :L
   if :O2.type="point   [output reflect_pl :O1 :O2]
   if :O2.type="segment [output reflect_sl :O1 :O2]
   if :O2.type="line    [output reflect_ll :O1 :O2]
   if :O2.type="circle  [output reflect_cl :O1 :O2]
   if :O2.type="ray     [output reflect_rl :O1 :O2]
end
```

or to use some operations on words and lists and make it as short as one line!

```
to reflect_l :O :L
   output run se (word "reflect_ first :O.type "l) [:O :L]
end
```

4 Project

Generalize your procedures to reflect an arbitrary object with respect to an arbitrary object (a point or a line).

Task 12 Develop procedure for a reflection about a point (point symmetry) starting for the cases of the first input being:

- a point
- a line, a ray or a segment
- a circle

Task 13 Generalize the procedures from Task 12 into a single procedure `reflect_p`.

Task 14 Combine both procedures `reflect_l` and `reflect_p` into a single procedure `reflect` which outputs the reflection of a given object O_1 (point, line, ray, segment or circle) with respect to object O_2 (point or line).

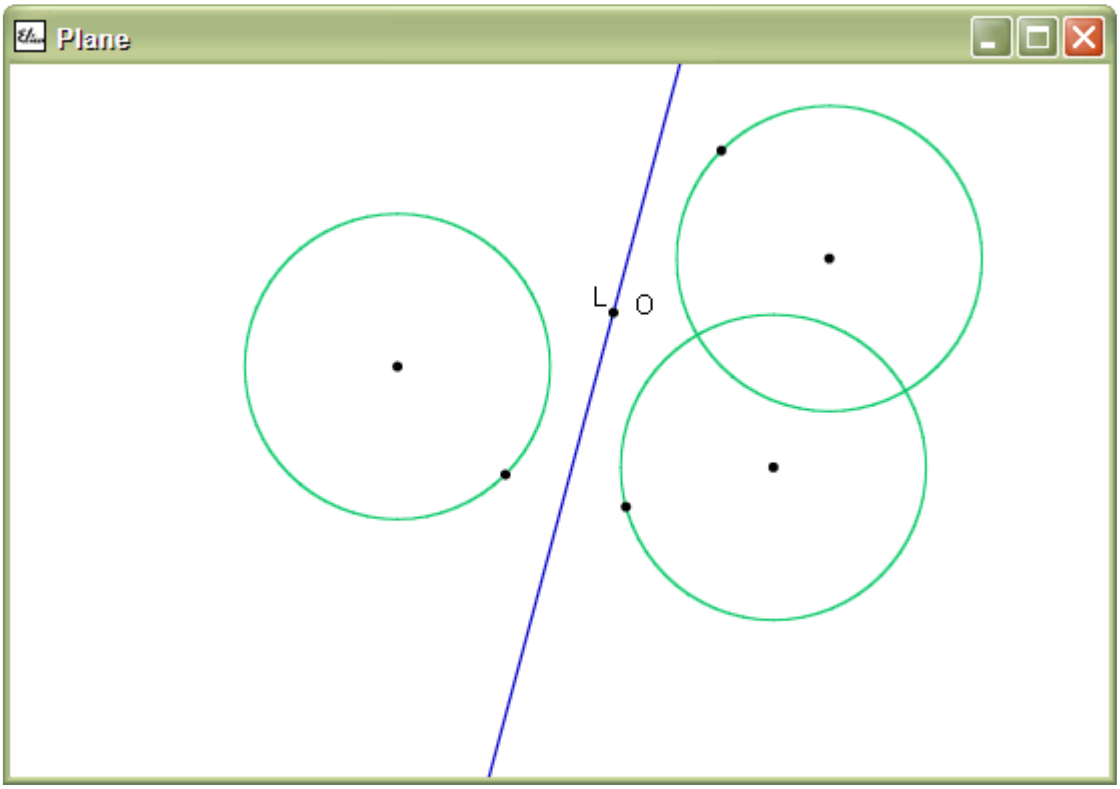

Figure 6 Reflection of a circle with respect to line L and point O

Hint: Here is a possible description

```
to reflect :O1 :O2
   if :O2.type="point
     [reflect_p :O1 :O2]
     [reflect_l :O1 :O2]
end
```

Task 15 Rewrite `reflect` so that the first input could be a set of any number of points, lines, rays, segments and circles.

Hint: Here is a possible description

```
to reflect :O1 :O2
      output if :O1.type="set
      [
        if (count :O1)=0
          [(set)]
          [(set reflect bf :O1 :O2 reflect first :O1 :O2)]
      ] [
        if :O2.type="point
          [reflect_p :O1 :O2]
          [reflect_l :O1 :O2]
      ]
end
```

All procedures for reflection could be grouped into a software package that can be used when needed. This makes programs much shorter, because it is not required to include the definition of all procedures in every program. In respect to reflection we already have more than a dozen of procedures, so it is time to put all of them in a library called `reflect.eli`. From now on, instead of loading Geomland and defining all `reflect` variants and subvariants, we only need to load `reflect` library.

Task 16 Create `reflect` library and rewrite the program from Task 15 to use this library.

Task 17 Create a picture and its reflection with respect of a sloped line.

Task 18 Create a picture and its reflection in a mirror.

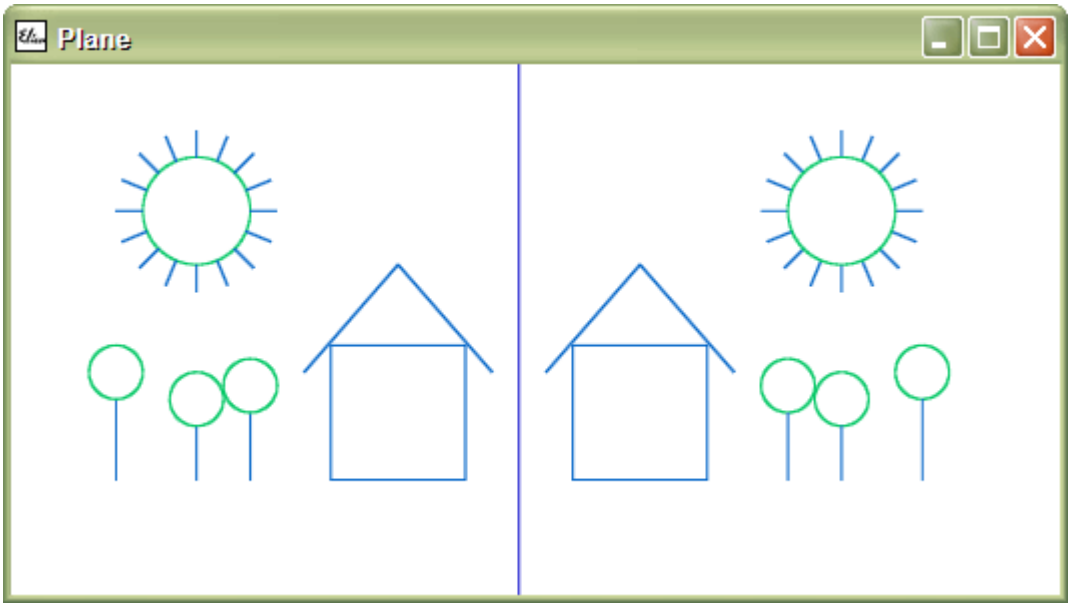

Figure 7 Reflection in a mirror

Task 19 Create a picture and its reflection in a water surface.

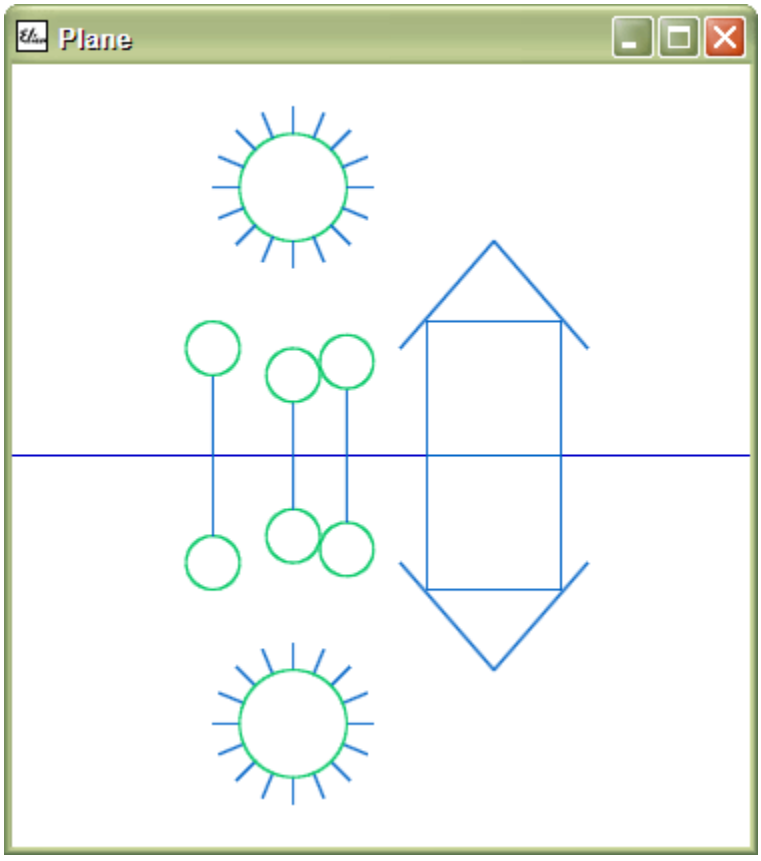

Figure 8 Reflection in water surface

Task 20 Create a picture and its reflection with respect to a point. Have you seen such type of reflection? Do you know any physical phenomenon which is based on it? And any biological?

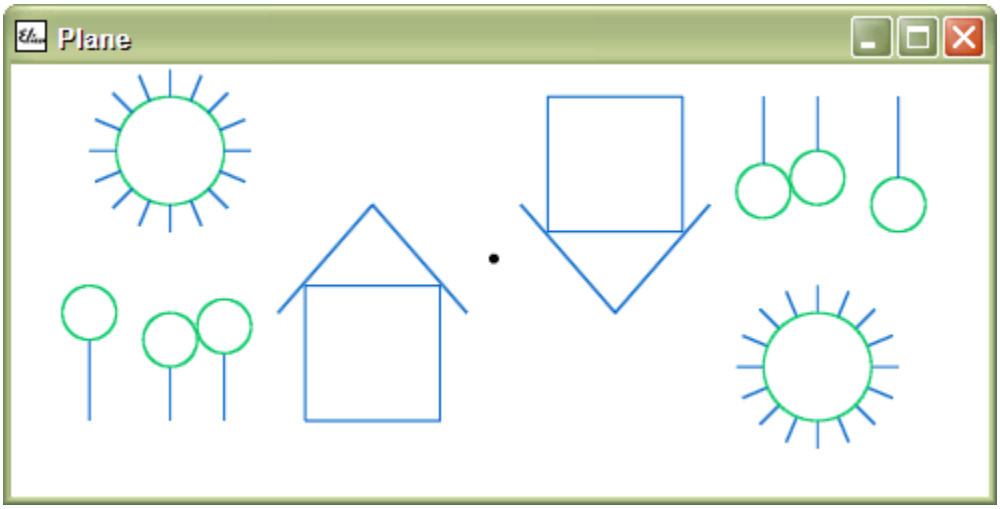

Figure 9 Reflection with image flip

Hint. Seach for more information about the lens and the human eye.

5 Reverted reflections

In mathematics it is common to solve reversed problems – they are the opposite problem of another (usually well studied) problem. In all cases above we solve the same problem: given object O_1 find its reflection O_2 with respect to a point or a line. What could be a revert problem to this?

Task 21 Study what happens with object O when it is reflected twice with respect to a line or a point (i.e., O_1 is a reflection of O and O_2 is a reflection of O_1).

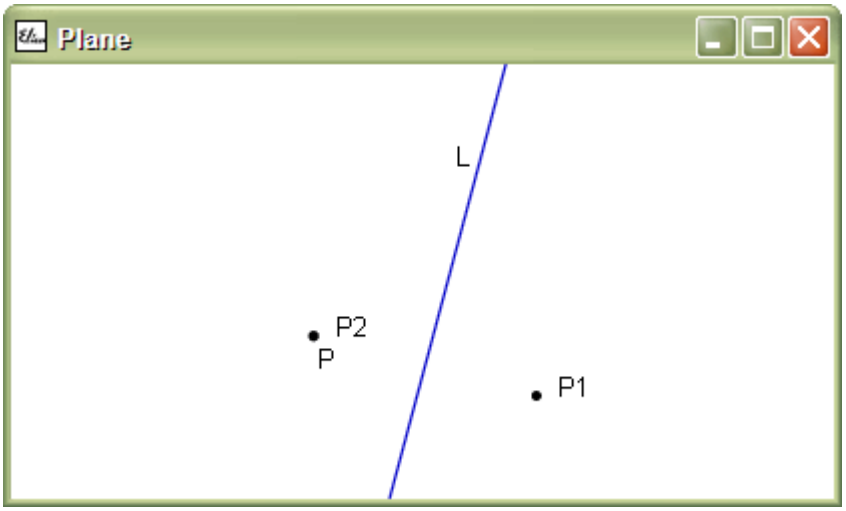

Figure 10 Double reflection with respect to a line

A double reflection with respect to a line or a point always gives the original object. However, there is another possibility to revert the problem:

Task 22 If point P and its reflection Q with respect to an unknown line L are given then find line L. What is the other name of this line in respect to segment PQ?

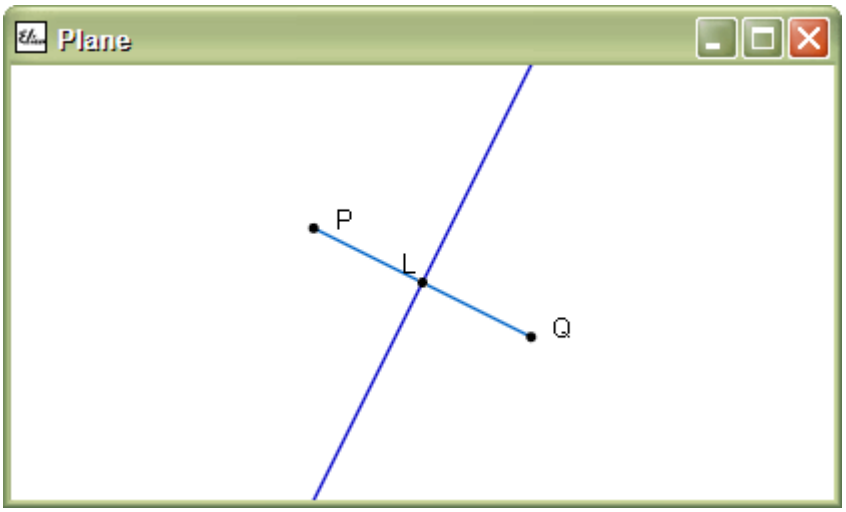

Figure 11 Perpendicular bisector as a line of reflection

Task 23 Find the line of reflection L if it is given a line M and its reflection N with respect to the unknown L. What is the other name of this line in respect to the angle between M and N?

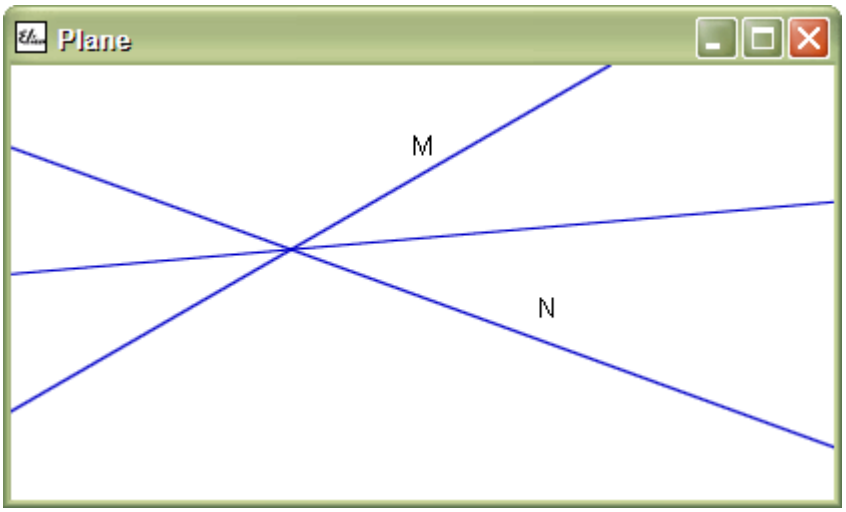

Figure 12 Angle bisector as a line of reflection

Hint: The problem has two solutions. Try to find both of them in case lines M and N intersect in a point. Study the solution when (1) M and N are parallel lines; (2) M and N coincide.

Task 24 Given are three distinct non-collinear points P_1, P_2 and P_3. Find reflection lines L_1, L_2 and L_3 so that L_1 reflects P_1 into P_2, L_2 reflects P_2 into P_3, and L_3 reflects P_3

back into P_1. Do these lines intersect in a single point? If yes, then what is this point? If not, then explain why this is not possible.

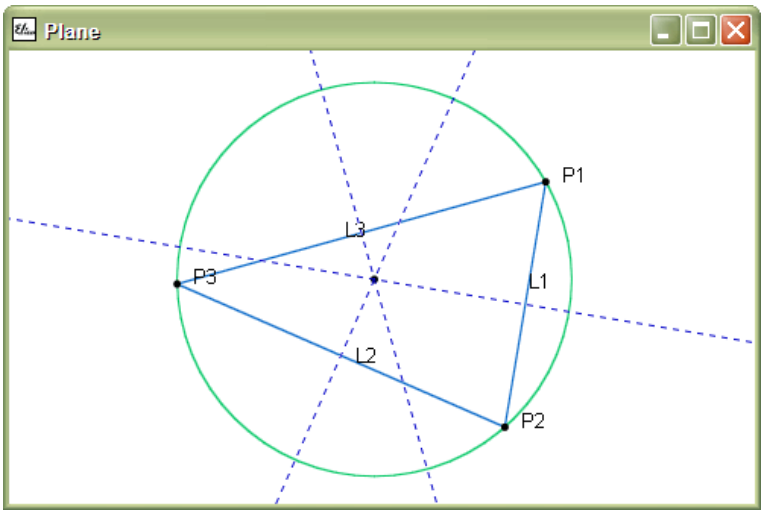

Figure 13 Circumscribed circle and dotted reflection lines of points P_1, P_2 and P_3

Task 25 Three distinct non-parallel lines L_1, L_2 and L_3 are given. Find reflection lines H_1, H_2 and H_3 so that H_1 reflects L_1 into L_2, H_2 reflects L_2 into L_3, and H_3 reflects L_3 back into L_1. Do these lines intersect in a single point? If yes, then what is this point? If not, then explain when this is not possible.

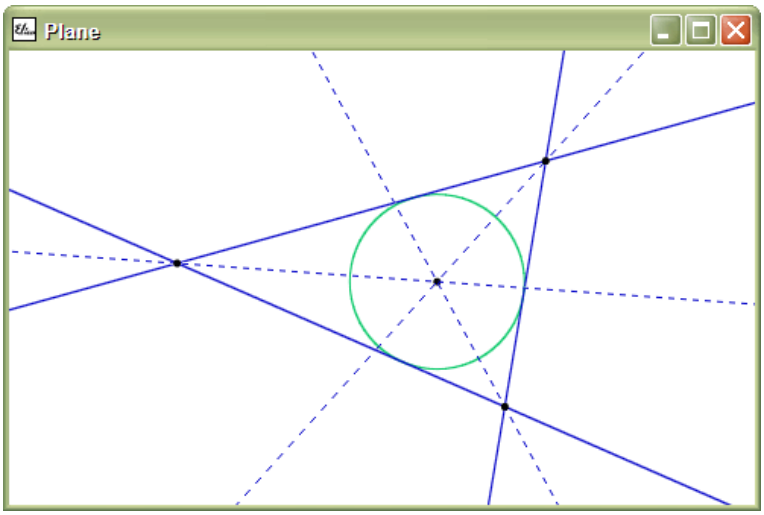

Figure 14 Inscribed circle and dotted reflection lines of points L_1, L_2 and L_3

Having in mind the solution of Task 23, what will happen if we pick some other reflection lines? Does an intersecting point exist? What is it?

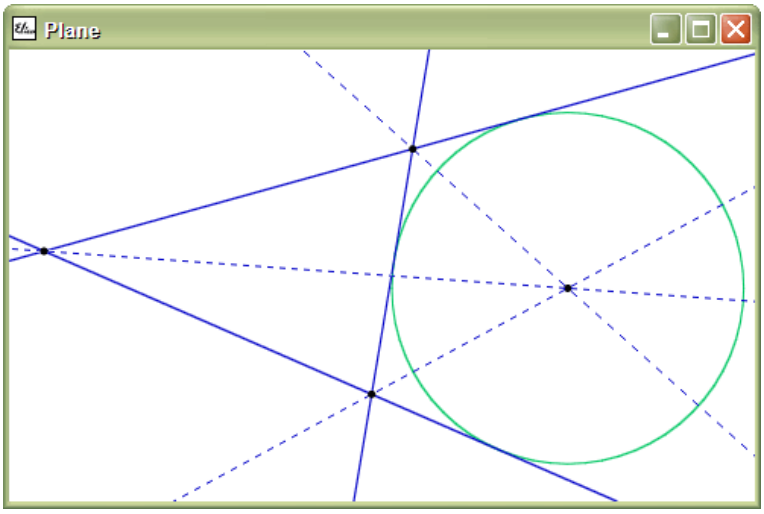

Figure 15 Escribed circle and dotted reflection lines of lines L_1, L_2 and L_3

If we consider all possible combinations of reflection lines, then one of them produces inscribed circle, another 3 produce escribed circles, and the rest correspond to situations with no single intersections. Figure 16 shows one of these situations. Find the others!

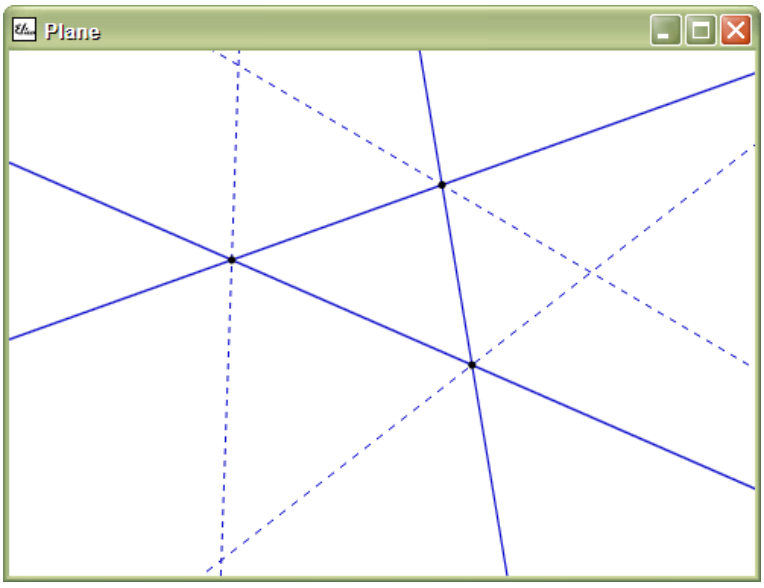

Figure 16 Reflection lines which do not intersect in a point

6 Exercise

Let us now make some exercises. Figure 17 represents the *jump-over-the-stick* game. The player stands in front of a stick (a). Then jumps over it landing in exactly the symmetrical point – on the other side of the stick (b). Then the stick is rotated on an arbitrary angle but around its center (c). After this the players jumps again (d), the stick is again turned (e), the player jumps (f), the stick is turned (g), the player jumps (h) and so on.

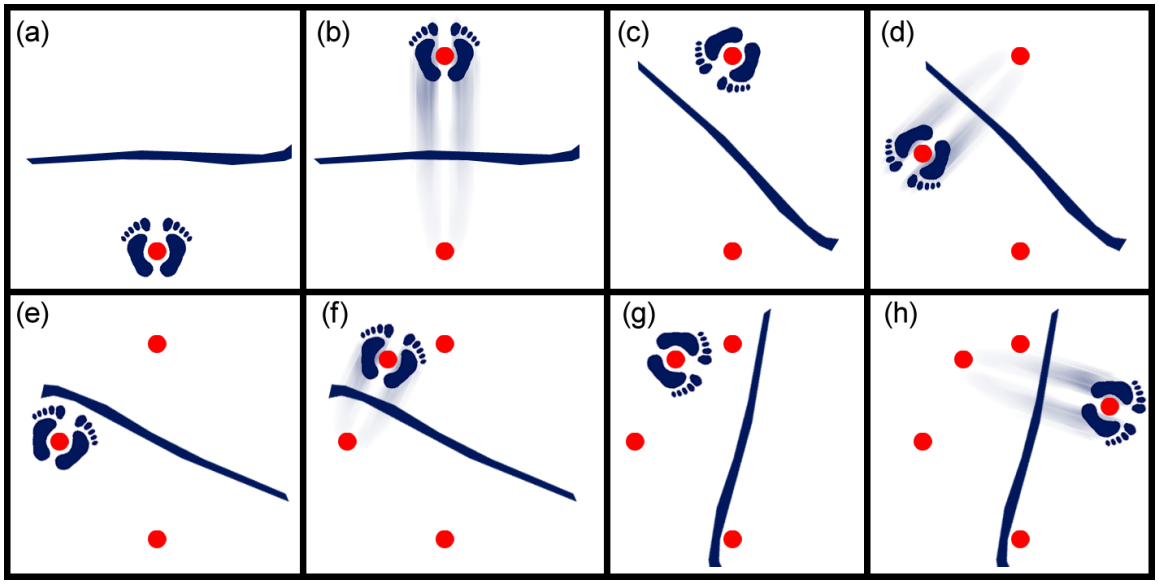

Figure 17 Jump-over-stick game

Task 26 Figure out any regularity in all player's positions. This is a nice physical and mathematical exercise which can be solved either physically or mathematically. However, let us try to solve it programmatically by using the `reflect` library.

```
run "reflect
ob "Angle 0
ob "Player point 100 0 [historical]
ob "Stick line point 0 0 :Angle
ob "Player reflect :Player :Stick
repeat 50 [ ob "Angle random 360 ]
ob "Positions :history.Player
```

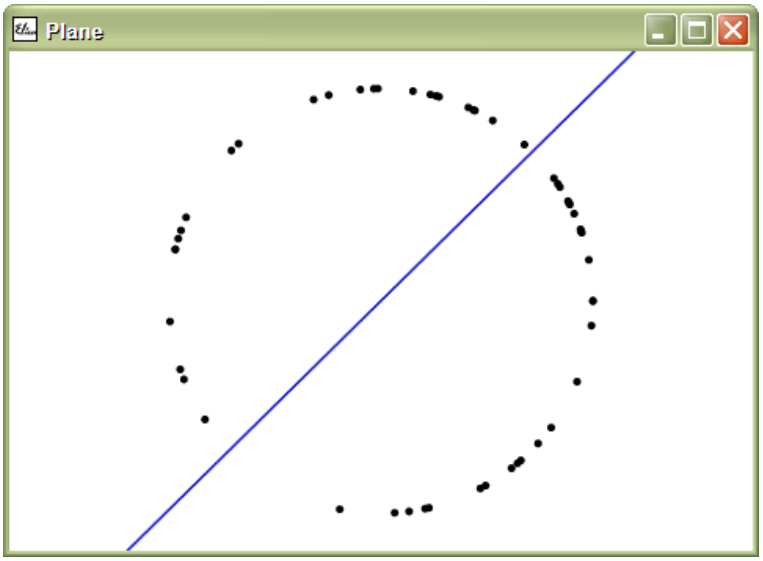

Figure 18 A history of 50 player's positions

Hint: Geomland remembers relationships between objects. For example, the program defines how `Stick` is calculated from `Angle`, and whenever Angle changes, Geomland automatically recomputes `Stick`. In the program above all geometrical objects are linked in a way, that any change of `Angle` forces reconstruction of a jump. That is why the cycle that "makes" `Player` jump 50 times over `Stick` modifies only `Angle`.

Additionally, `Player` is a historical object. In terms of Geomland this means that the system keeps a history of all previous positions of `Player`.

Task 27 Prove mathematically your observations from Task 26.

Task 28 Close the book, turn off the computer and go outside to play *jump-over-the-stick* with your friends.