Logo to SVG

Vladimir Batagelj

Department of mathematics, FMF, University of Ljubljana
Jadranska 19, 1000 Ljubljana, Slovenia
e-mail: vladimir.batagelj@uni-lj.si

Abstract

A picture produced by logo on the screen can be mirrored into its SVG (Scalable Vector Graphics) description on the file. These files provide a high quality records (snap-shots) of logo screen to be included in other documents – web pages.
Logo2SVG is a collection of logo commands, for MSW Logo, that by redefining turtle commands support the mirroring of turtle movement into the SVG.

Keywords: Logo, SVG, screen snap-shot, redefining logo primitives, MSW Logo

1. Introduction

The usual approach to get snap-shots of logo screen for inclusion in different documents (manuals, tutorials, papers, books) is to save (a part of) the screen as a bitmap. Adjusting such bitmap picture to the dimensions required by a document we can considerably lose on its quality.

In the paper ***Logo to Postscript*** [1] another approach was proposed. Using **Logo2PS** (a collection of redefined basic logo commands) a picture produced by logo on the screen can be mirrored into its PostScript description on the file. These files provide a high quality snap-shots of logo screen and can be included in other documents, clipped and scaled to the available space.

In this paper we present a parallel collection **Logo2SVG** that supports the mirroring of turtle movement into the SVG.

2. SVG – Scalable Vector Graphics

2.1. What is SVG ?

SVG - Scalable Vector Graphics [6] is a new web graphics standard. The SVG development group published in October 1998 the requirements on SVG and in February 1999 the first draft. Several improved versions followed. The last version was published in November 2000 as a candidate release.

Technically, SVG is a 2D-graphics markup language based on XML [5]. It is compatible with other web standards: HTML, XML Namespace, Xlink, Xpointer, CSS 2, DOM 1, Java, ECMA/Javascript, Unicode, SMIL 1.0, ...[4, 8, 7, 9]. It allows us to include in HTML documents pictures described by their structure – composition of curves and shapes. Since the SVG viewer is not integrated yet into web browsers we need, to view SVG pictures, to install it as a plug-in. An excellent SVG plug-in for Windows was produced by Adobe [10].

The SVG pictures are not static (as standard bitmaps GIF, JPEG, PNG). The SVG viewer provides options to zoom in (to see details) and out (to see global view), to move the picture, to search for text, ... Besides this, using built-in animation capabilities or Javascript program support, the pictures can be made alive and interactive, SVG files are relatively small and independent of output devices and computer platforms.

To get some impression about SVG see some examples from:
`http://sio.edus.si/list/1/svg/svg06.htm`

2.2. Applications of SVG

SVG pictures can be produced by drawing tools. On Windows we can use last versions of Adobe Illustrator, Corel Draw, WebDraw (by Jasc) and Mayura [11, 12, 13, 14]. But special programs for visualization of obtained data/results will produce most SVG pictures. See, for example: Social patterns and structures in Vienna [15] and European countries [16].

Other applications include: data visualization, presentations (like Power Point), maps (GIS), layouts, educational pictures, ...

Using **`Logo2SVG`** we can produce SVG pictures also with MSW Logo.

2.3. A simple example in SVG

Here is a simple example of picture description in SVG

```
<svg>
 <circle cx="120" cy="65" r="30"
         style="fill:yellow;stroke:black;stroke-width:3;"/>
 <text style="fill:red;" x="100" y="55">EuroLogo 2001</text>
</svg>
```

It creates yellow circle with black border containing red inscription `EuroLogo 2001`.

2.4. Embedding SVG pictures in web pages

To insert a SVG picture into a HTML document we use the `EMBED` tag. For the picture from our simple example we have:

```
<EMBED SRC="./svgfiles/simple.svg" NAME="simplex"
  HEIGHT="100" WIDTH="300"
  TYPE="image/svg-xml"
  PLUGINSPAGE="http://www.adobe.com/svg/viewer/install/">
```

The attribute `SRC` determines the location (URL) of the SVG file; `NAME` becomes important in advanced applications using Javascript or Java. The attributes `HEIGHT` and `WIDTH` are obligatory and determine the size of rectangle in which the picture is rendered. The value of `TYPE` is the MIME-type of the file – for a SVG file it can be `image/svg` or `image/svg-xml`. The attribute `PLUGINSPAGE` directs the user that has not a SVG viewer installed on his computer, to the web site from which he can obtain a viewer.

3. Redefining logo primitive commands

Berkeley logo [2] and its adaptation for Windows, MSWlogo [3] allow the user to redefine also the primitive commands. In general this is a dangerous, but sometimes very useful, practice. To enable this possibility we have first to switch-on the special (system) variable `REDEFP`

```
MAKE "REDEFP "true
```

To redefine commands we shall use two logo commands

`COPYDEF` *newname oldname*

that makes *newname* a procedure identical to *oldname*; and

`DEFINE` *procname* `[` *params body* `]`

that defines a procedure with name *procname*, parameters list *params* and *body* – sequence of lists containing instruction lines.

Since we want to add the generation of SVG code to the turtle moving and related commands, the general pattern of redefinitions will be

```
COPYDEF ".cmd "cmd
DEFINE "cmd [ [ params ] [ ext1 ] [ .cmd params ] [ ext2 ] ]
```

We first save the original definition of the primitive command *cmd* as `.cmd`. Afterwards we redefine *cmd* by surrounding the application

`.cmd` *params*

of original command with additions *ext1* and *ext2*.

For example the commands `FORWARD` and `SETPENCOLOR` are redefined as follows

```
COPYDEF ".forward     "FORWARD
DEFINE "FORWARD [[d][.forward :d .move]]
COPYDEF ".setpencolor    "SETPENCOLOR
DEFINE "SETPENCOLOR [[c]
  [IF AND :.draw :.shape [ messagebox [Logo2SVG]
    [SetPC Ignored. Not allowed in Shape] STOP ]]
  [.setpencolor :c]
  [.EndPath  MAKE ".pc .conv :c .BegPath ]
]
```

4. Logo to SVG

To use **`Logo2SVG`** we simply load the `Logo2SVG.LGO` file. It defines the following commands:

`SVGInit`
: Redefines turtle commands to mirror the turtle movement into the SVG. `SVGInit` is executed on **`Logo2SVG`** load.

`SVGExit`
: Erases **`Logo2SVG`** commands and variables.

`BegPic :p` `:bbox :pen :d`
: Initializes the file p`.SVG` for a new SVG picture. $bbox = [xll, yll; xur, yur]$ determines the screen bounding box; d determines the precision – decimal places of real numbers written to p`.SVG`; and c selects the type of line caps (`butt`, `round`, `square`).

  The parameters $bbox$, pen and d are optional. Their default values are
  $bbox = [-300\ -300\ 300\ 300]$, $pen =$ `round`, $d = 2$

`EndPic`
: Ends the current picture on the SVG file.

`BegShape :w :pc :fc`
: Starts a new shape with border width w, border color pc and fill color fc. Between `BegShape` and `EndShape` these attributes should not be changed; also `PU` command is not allowed.

`EndShape :t`
: Ends the drawing of a shape. The list t contains logo commands that determine a point in the interior of the shape, required by the logo `FILL` command – the `(FILL "true)` is used. After the fill the position before the t moves is restored.

and some auxiliary commands.

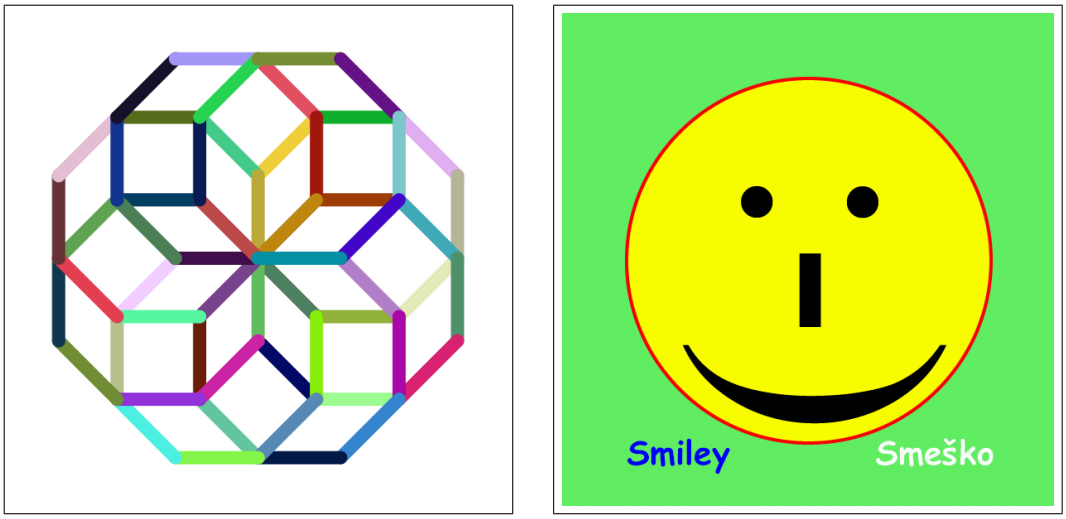

Figure 1: Pictures from examples.

4.1. Examples

To start tracing the turtle movement to the SVG file(s) we execute the `BegPic` *filename* command. Then we run the logo commands that produce the picture. We finish tracing by command `EndPic`

For example:

```
BegPic "test
 CS SetPenSize [8 8]
 REPEAT 8 [ REPEAT 8 [
   SetPenColor (LIST random 256 random 256 random 256)
   FD 50 RT 45 ] RT 45 ]
EndPic
```

produces the picture presented on the left side of Figure *1*.

The shape commands `BegShape` and `EndShape` were introduced because the logic of filling in logo is different from that in SVG.

`Logo2SVG` partially supports also the use of fonts. The use of both is illustrated with the picture on the right side of Figure *1*, obtained by the following command:

```
TO Smiley
  IF NOT definedp "svginit [PRINT "|Load  Logo2SVG.LGO  first| STOP]
  BegPic "smiley
    Cs Ht SETSCREENCOLOR [100 240 100] PU SETPOS [0 0] PD
    BegShape 4 [255 0 0] [255 255 0]
      CIRCLE 220
    EndShape [ SETPOS [0 0] ]
    PU SETPC [0 0 0] SETPOS [238 120] SETHEADING 180
    SETTEXTFONT [[Times New Roman] -350 0 0 400 0 0 0 0 3 2 1 34]
    LABEL "|:-)|
    SETPC [0 0 255] SETPOS [-220 -210] SETHEADING 90
    SETTEXTFONT [[Comic Sans MS] -40 0 0 800 0 0 0 0 3 2 1 18]
    LABEL "Smiley
    SETPC [255 255 255] SETPOS [80 -210]
    (LABEL "Smesko "Sme&#353\;ko)
  EndPic
END
```

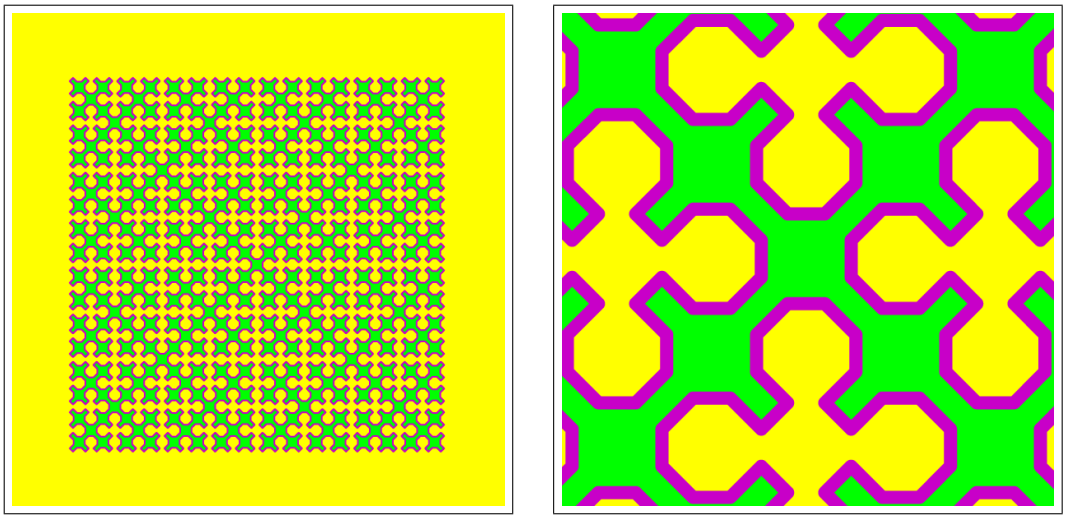

Figure 2: Sierpinski curve of order 9 and a Zoom-in.

Note that the eyes, nose and mouth of Smiley are produced by **`LABEL "|:-)|`**; and that the font size is given by a negative number.

The following example shows how to draw magenta Sierpinski curve of order 9 filled with green – see the left side of Figure *2*. On the right side a zoom-in to the obtained SVG picture is presented.

```
TO Sierp :n :a :h :k
  IF :n = 0 [ FD :k STOP ]
  RT :a Sierp :n - 1 (-:a) :h :k LT :a FD :h
  LT :a Sierp :n - 1 (-:a) :h :k RT :a
END
TO Sierpinski :n :d :w
  IF NOT definedp "svginit [PRINT "|Load  Logo2SVG.LGO  first| STOP]
  BegPic (WORD "Sierpinski :n)
    PU SETPOS [ -230 -230 ] PD SetScreenColor [255 255 000]
    BegShape :w [200 000 200] [000 255 000]
      REPEAT 4 [ Sierp :n 45 :d/sqrt 2 5*:d/6
        RT 45 FD :d/sqrt 2 RT 45 ]
    EndShape [ SetPos [-228 -230] ]
  EndPic
END
Sierpinski 9 8 2
```

5. Some Logo2SVG implementation details

5.1. Control

The **Logo2SVG** logic is controlled by the following control variables.

`.draw` – true when SVG mirroring is active. Needed for internal interruptions of mirroring – for example in `EndShape`.

`.path` – true when new path/shape is built.

`.act` – true for nonempty path.

`.shape` – true when new shape is built.

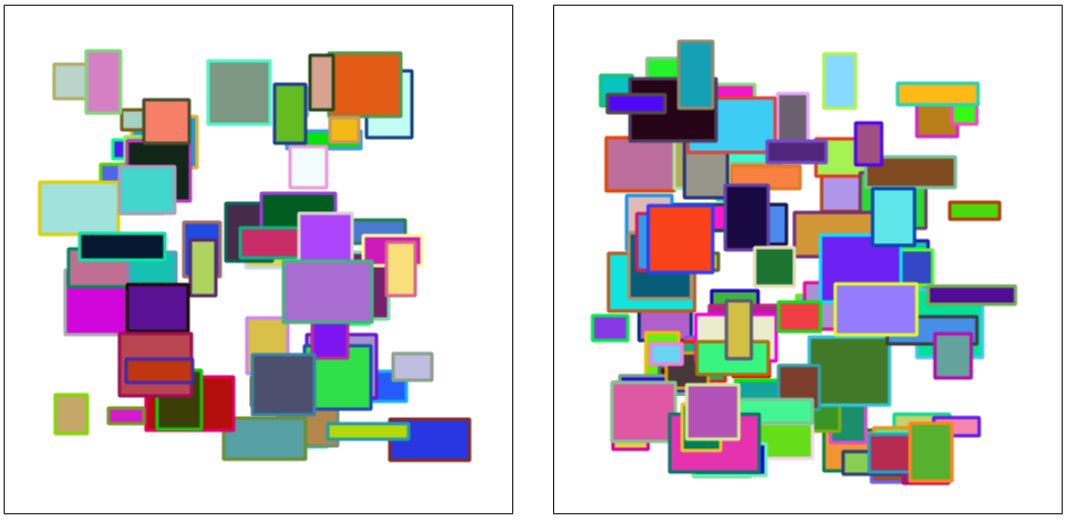

Figure 3: Random patterns.

5.2. Coordinate system

The Logo and SVG coordinate systems are different. In Logo the origin of the coordinates is placed in the center of the screen. The directions of axes follow the mathematical conventions. In SVG the origin is in the upper left corner and the y-axis is directed downwards.

In **Logo2SVG** all computations with coordinates are done by regular Logo movements. To get the corresponding SVG coordinates the Logo coordinates are used (`XCOR`, `YCOR`) and transformed into SVG coordinate system as in the command `.move`:

```
TO .move
  IF :.draw [
    IFELSE PENDOWNP [MAKE ".act "true .SVGon TYPE "\ L ]~
    [ .EndPath  TYPE "|<path d="M| MAKE ".path "true ]
    (TYPE FORM XCOR+:.x0 :.m :.d "\  FORM :.y0-YCOR :.m :.d)
    .SVGoff
  ]
END
```

(`x0`, `y0`) are the coordinates of the Logo origin in the SVG coordinates. The command `.move` builds a path by segments using SVG path commands `M` (move to) and `L` (line to).

5.3. Paths and shapes

Turtle graphics is essentially a sequence of paths and shapes. Every change of path parameters (color, width, pen position - `PU`, `PD`) starts a new path. In descriptions of shapes these changes are not allowed. Besides this, shapes are also filled with color.

New shapes are overlaying the existing, as can be seen from examples in Figure *3* produced by the command `Wall`:

```
TO RndColor
  OP (LIST RANDOM 256 RANDOM 256 RANDOM 256)
END
TO Wall :n
  CS HT PU
```

```
  BegPic "wall
    REPEAT :n [
      BegShape 3 RndColor RndColor
        MAKE "a 21 + RANDOM 60 MAKE "x (RANDOM 320) - 200
        MAKE "b 14 + RANDOM 47 MAKE "y (RANDOM 340) - 200
        SETXY :x :y PD
        REPEAT 2 [FD :b RT 90 FD :a RT 90]
      EndShape [FD :b/2 RT 90 FD :a/2]
      PU
    ]
  EndPic
END
Wall 100
```

5.4. Label

The LABEL command also needs a special treatment. In Logo the current position determines the upper left corner of the text box; while in SVG it determines the origin of the baseline. In this version of **Logo2SVG** only fonts with negative font size are supported.

SVG supports also Unicode characters. To allow the Unicode output to SVG the LABEL command was extended with additional optional parameter. The command (LABEL *screen unicode*) writes the text *screen* on Logo screen and *unicode* into SVG file. An example of the use of extended LABEL is given at the end of the command Smiley – š is an entity representing the character š with Unicode code 353.

```
COPYDEF ".label   "LABEL
DEFINE "LABEL [[t [u :t]]
  [MAKE ".ls POS .label :t]
  [IF :.draw [ .EndPath MAKE ".draw "false
    LOCALMAKE "ps PENDOWNP PU RT 90 FD 0.9*:.fs LT 90
    (TYPE "|<g transform="translate(| FORM XCOR+:.x0 :.m :.d ",
      FORM :.y0-YCOR :.m :.d "|)">| )
    (TYPE "|<g transform="rotate(| heading-90 "|)| ) PRINT "|">|
    (TYPE "|<text x="0" y="0"|
      "| style="font-family:| :.font "|;font-size:| :.fs
      "|;font-weight:| :.fw "|;fill:| :.pc )
    IF :.italic [ TYPE "|;font-style:italic|]
    PRINT "|;">|
    PRINT :u
    PRINT "|</text></g></g>|
    SETPOS :.ls IF :ps [ PD ] MAKE ".draw "true
    .BegPath
  ]]
]
```

6. Conclusion

Further implementation details on **Logo2SVG** can be seen from its code. The last version of **Logo2SVG** and some examples of SVG files produced with it are available at:
http://vlado.fmf.uni-lj.si/educa/logo/logo2svg/

References

[1] Batagelj V.: *Logo to Postscript*. Proceedings of Eurologo97. Budapest, 1997, p. 333-341.
http://vlado.fmf.uni-lj.si/educa/logo/logo2ps/

[2] Harvey B.: *Berkeley Logo*: http://http.cs.berkeley.edu/~bh/

[3] Mills G.: *Logo (Berkeley) for Windows*, ver. 6.4h. Program doc file in Word, July 2000.
http://www.softronix.com/logo.html

[4] *The World Wide Web Consortium – W3C*: http://www.w3.org

[5] *XML*: http://www.w3.org/XML/

[6] *W3C/SVG*: http://www.w3.org/TR/SVG/index.html

[7] *ECMA/JavaScript*: http://www.ecma.ch/ecma1/stand/ecma-290.htm

[8] *Java*: http://java.sun.com/

[9] *Unicode*: http://www.unicode.org/

[10] *Adobe SVG Viewer*: http://www.adobe.com/svg/viewer/install/

[11] *Adobe Illustrator*:
http://www.adobe.com/products/illustrator/main.html

[12] *Corel Draw*: http://www.corel.com/svg

[13] *Jasc Software, WebDraw*: http://www.jasc.com/webdraw

[14] *Mayura*: http://www.mayura.com/

[15] *Vienna - Social patterns and structures*:
http://www.karto.ethz.ch/ an/cartography/vienna/

[16] *Europe*: http://www.carto.net/papers/svg/eu/index.html