SVGLAB v1.0

Handbook

Author：Zhang Shichao

zhshchao@163.com

Chapter1 preface

1.1、Why writing this module

Generate SVG pictures in a simple and meaningful way.

1.2、Name of the module

1, SVG and a combined meaning of MATLAB. Draw SVG image in MATLAB style;
2, and take the meaning of SVG laboratory;
3, never heard of SVG, never once used MATLAB, does not affect the use of this module.

1.3、Guiding ideology

Make least definition, do as more things.

1.4、Definitions and functions' names

Lines, circles, etc, called SVG elements.

Nearly all the functions for drawing pictures or set the picture properties are named after SVG's elements, or the corresponding MATLAB function name.

1.5、Install

Just copy the module to a directory where perl can find it.
In fact, the following command gives all such directories in your machine:

```
perl -e 'print"@INC"';
```

If there is anything mistake, please email to me.

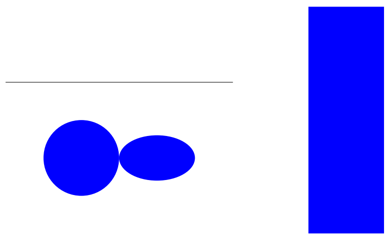
Chapter2 Getting started

Before creating a figure, you need to know the module defines Y axis bottom-up; and by default, one unit is the length of a pixel.

2.1、Create one figure

A figure-figend pair generates a figure：

```
#!/usr/bin/perl

use SVGLAB;

figure;
        line(100,300,400,300);
        rect(500,400,100,300);
        circle(200,200,50);
        ellipse(300,200,50,30);
figend;
#Generates SVGLAB1.svg
```

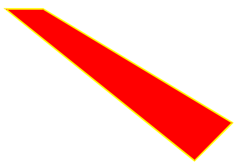


The first example

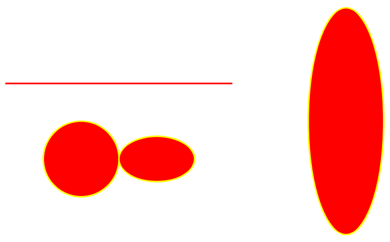
2.2、Create more in one time

More figure-figend pairs generates more figures

```
#!/usr/bin/perl
use SVGLAB;

figure;
        $x=[200,450,500,250];
        $y=[300,100,150,300];
        polygon($x,$y,'red',2,'yellow');
figend;
#Generates SVGLAB1.svg
figure;
        line(100,300,400,300,2,'red');
        rect(500,400,100,300,10,10,'red',2,'yellow');
        circle(200,200,50,'red',2,'yellow');
        ellipse(300,200,50,30,'red',2,'yellow');
figend;
#Generates SVGLAB2.svg
```



polygon



Many elements in one figure

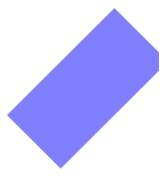
2.3、Figure parameters

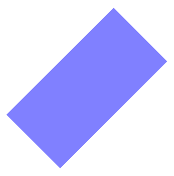
2.3.1、Local parameters－figure() Function

1，Using 'figure()' set the size and name of a figure

```
#!/usr/bin/perl
use SVGLAB;

figure(500,500);#Set the size to be 500*500 pixels
        rotate(45,300,200);#Rotate 45 degrees around(300,200).
        shq("fill-opacity='0.5'");
        rect(300,400,100,200);
figend;
#SVGLAB1.svg,500*500
```

```
#!/usr/bin/perl
use SVGLAB;

figure('mySVG.svg');#or figure('/home/usr/SVGLAB_eg/mySVG.svg')
        rotate(45,300,200);
        shq("fill-opacity='0.5'");
        rect(300,400,100,200);
figend;
#mySVG.svg,800*500
```


SVGLAB1.svg,500*500


mySVG.svg,800*500

'figure('name',width,height)' can set name and size at the same time.

2，How to use 'figure()':

```
figure;
figure(width,height);
figure('name');
figure('name',width,height)
```

2.3.2、Global setting－SVGLAB()

1，SVGLAB() set size and prefix of figure name:

SVGLAB() can set the global size and prefix, e. g :

```
#!/usr/bin/perl
use SVGLAB;
SVGLAB('mySVG');
        figure;
        figend;

        figure;
        figend;
SVGLAB('smallSVG',200,100);
        figure;
        figend;

        figure;
        figend;
```

Four empty figure will be generated:
mySVG1.svg、mySVG2.svg，800*500 pixels;
smallSVG1.svg、smallSVG2.svg，200*100 pixels .

2，How to use 'SVGLAB()':

SVGLAB() must be out of figure-figend pairs:

```
SVGLAB;              #Back to default: 800*500 pixels ,with prefix 'SVGLAB';
SVGLAB(width,heitht);     #e. g :SVGLAB(2000,1500);width:2000，height:1500;
SVGLAB('prefix');   #e. g: SVGLAB('mySVG'); with prefix: 'mySVG';
SVGLAB('prefix',width,heitht);    #e. g: SVGLAB('mySVG',2000,1500);
```

Chapter3 SVG elements & their properties

3.1 Elements

Properly speaking, 3.11 to 3.14 aren't SVG elements, but they can be used in the same style, so I list them in this section .

3.1.1、Line

line($x1,$y1,$x2,$y2,line_width,line_color);#($x1,$y1),($y1,$y2) are the beginning and ending points of the line.
Simplest call:
line($x1,$y1,$x2,$y2);
(black, line width 1 pixel)

3.1.2、Rect

rect($x,$y,width,height,x_round,y_round,fill_color,line_width,line_color);#($x,$y) is the upper left corner.
Possible calling:
rect($x,$y,width,height);

3.1.3、Circle

circle($cx,$cy,$r,fill_color,line_width,line_color);#center: ($cx,$cy), radius: $r
Possible calling:
circle($cx,$cy,$r);

3.1.4、Ellipse

ellipse($cx,$cy,$rx,$ry,fill_color,line_width,line_color);#center: ($cx,$cy), radius: $rx, $ry
Possible calling:
ellipse($cx,$cy,$rx,$ry);

3.1.5、Polyline

polyline(x_coordinates,y_coordinates,line_width,line_color,style);
"x_coordinates" is reference of an array, which stores x coordinates of all points; and "y_coordinates" the same.
e. g:

```
$x=[200,450,500,250];
$y=[300,100,150,300];
polyline($x,$y,'red',2,'yellow');
```

3.1.6、Polygon

polygon(x_coordinates,y_coordinates,line_width,line_color,style);
Nearly the same as the using of polyline().

3.1.7、Path(temporarily lacking)

Path is lacking in this version.

3.1.8、Text

text(text,$x,$y,size,font,color,stroke_color);
Possible calling:
text(text,$x,$y)

3.1.9、Link

Adding links to the figure, calling:

```
alink('location','style');
   some elements;
aend;
```

Letting elements between alink-aend pair links to 'location', open these in 'style '. SVG provides four kind of 'style's: 'new'、'replace'、'_black'、'_top', different from which browser is been used, so best omit it.
Possible calling:

```
alink('location');
aend;
```

Note: alink generates SVG <a> element, just because 'a' is too short, so I name it 'alink'.

3.1.10、Group

Create group element, using:

```
gp;
   some SVG element;
gpend;
```

3.1.11、Vertical text

vtext(text,$x,$y,size,font,color,stroke_color);
I find it redundancy sometimes, maybe I will delete it in following versions.

3.1.12、Plot function image

plot(x,y,line_width,line_color,style);
If there is variables:
$x=linspace(0,6.28,100);

```
$y=vsin($x);#These two functions is inside SVGLAB.
e. g. 1:
plot($x,$y);
e. g. 2:
plot($x,$y,'','','*');
('style' can be any character or string)
```

3.1.13 stem() math figure

```
stem(x_coordinates,y_coordinates,line_width,line_color);
```

Using is nearly the same as plot()

3.1.14 label

```
xlabel(text,size,font,$x,$y,color,stroke_color);
ylabel(text,size,font,$x,$y,color,stroke_color);
title(text,size,font,$x,$y,color,stroke_color);
```

Possible calling:

```
xlabel(text);
ylabel(text);
title(text);
```

3.2 Properties

3.2.1 element properties

1，rotate(degree,x,y);

Function: let the nearest SVG element below it rotate 'degree' round (x,y). e. g . see 2.3.1.

2，shq('properties');

Function: let the 'properties' used in the nearest SVG element below it.

3.2.2 figure properties

3.2.2.1 local figure properties

1, figure() set figure's name and size

Four kind of using:

```
figure;
figure(width,height);
figure('name');
figure('name',width,height);
```

2, axis() set figure coordinate system

In this module, Y axis is bottom up, which is different from SVG's top down. If you must use SVG's original coordinate, see AXIS in next section.

Four kind of using:

```
axis;                   #auto adapt, let the coordinates adapt to
                        #all the elements above it.
axis(1);                #scale maintain proportion.
axis(x1,x2,y1,y2);      #set the axis range, which will cover the edge.
axis('on');             #show axis arrow
```

If there are more axis() commands(except the axis('on') command) inside one figure-figend pair, the last one works.

3, grid property in mathematic figure

Five kind of using:

```
grid;           #or grid('') or grid('xy');
grid('x');      #grids perpendicular to the x-axis, aligned to the scale.
grid('y');      #grids perpendicular to the y-axis, aligned to the scale.
grid(m,n);      #grids perpendicular to the x&y-axis respectively, not necessarily align to the scale.
grid(m);        #grids perpendicular to the x&y-axis respectively, not necessarily align to the scale, the same as grid(m,m);
```

e. g. please see 4.3.1.

3.2.2.2 global figure properties

Global figure properties is set outside the figure-figend pairs, and will effect all the figures below, until new properties is set.

1，SVGLAB set figure's name prefix and size

Four kind of using:

SVGLAB() must be out of figure-figend pairs:

```
SVGLAB;                 #Back to default: 800*500 pixels ,with prefix 'SVGLAB';
SVGLAB(width,heitht);      #e. g :SVGLAB(2000,1500);width:2000，height:1500;
SVGLAB('prefix');    #e. g: SVGLAB('mySVG'); with prefix: 'mySVG';
SVGLAB('prefix',width,heitht);     #e. g: SVGLAB('mySVG',2000,1500);
```

2，AXIS set global coordinate system

Five kind of using:

```
AXIS;           #default, or back to default
AXIS(0);        #use SVG's original coordinate system, and no scale.
AXIS(num);   #e. g.: AXIS(0.8);set scale parameter 0.8.
AXIS(bool,bool,bool,bool)  #means : AXIS(translate_axis_or_not, maintain_proportion_or_not, scale_or_not, cover_or_not)

AXIS(num,bool,bool,bool)#means: AXIS(scale_parameter, maintain_proportion_or_not, scale_or_not[always ture], cover_or_not), e. g.:
AXIS(0.5,1,1,1);       #the same as AXIS(0.5,1,0,1)
```

Chapter4 Mathematic figure

4.1、vector generating

It's important to illustrate how to define SVGLAB vector before using mathematic function. It happed that the MATLAB vector is the same as perl's reference ijn formal, so I use it as the definition of a vector. e. g.:

```
$x=[1,2,3,4,5];
```

specify a vector. (only line vector is defined)

Currently, two function: vector() , and linspace() which is defined from MATLAB:

```
$x=vector(begin, step, end);          #Equivalent as  x=begin:step:end in MATLAB.
$x=linspace(begin, end, length);     #The same as using in MATLAB.
```

In addition, '$x=[a..b];' can generate Integer vector whose step is 1.

4.2、Math function

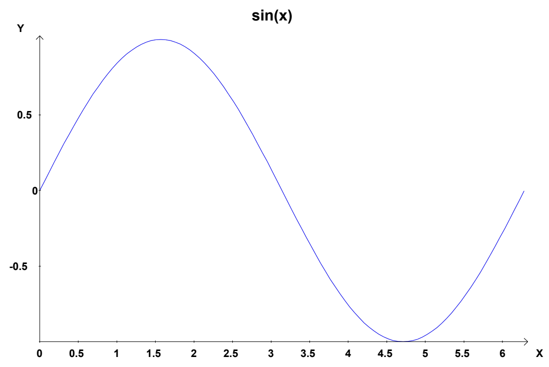
Now three function available:
$y=vsin($x)
$y=vcos($x)
$y=vpoisson($lamda,$k)
$y is a vector with the same dimension as $x.

4.3、Figure and its' properties

plot() and stem() is imitated from MATLAB.

4.3.1、plot example

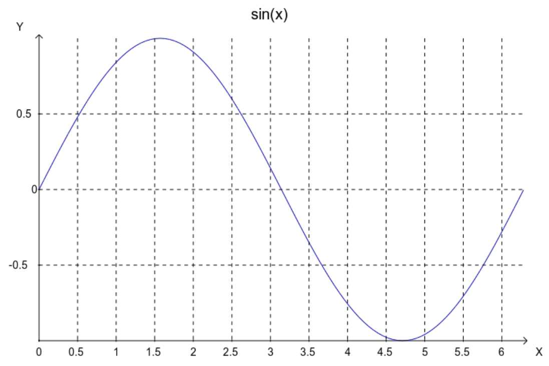
```
#!/usr/bin/perl
use SVGLAB;

$x=linspace(0,6.28,100);# 100 points from 0 to 6.28.
$y=vsin($x);
figure;
        plot($x,$y);
        axis;
        axis('on');
        title('sin(x)');
figend;
```



plot Sine function

You can also add 'grid' property:

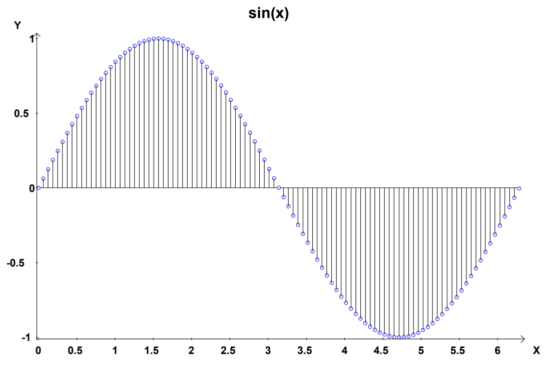
```
#!/usr/bin/perl
use SVGLAB;

$x=linspace(0,6.28,100);#100 points from 0 to 6.28.
$y=vsin($x);
figure;
        plot($x,$y);
        axis;
        axis('on');
        title('sin(x)');

        grid;
figend;
```



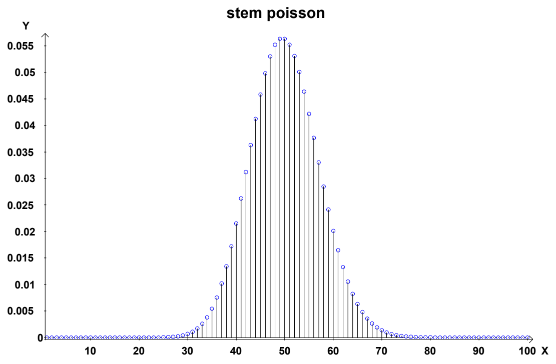
Sine function with grids

4.3.2、 stem() example

```
#!/usr/bin/perl
use SVGLAB;
$x=linspace(0,6.28,100);# 100 points from 0 to 6.28.
$y=vsin($x);
figure;
        stem($x,$y);
        axis;
        axis('on');
        title('sin(x)');
figend;

$x=[1..100];
$y=vpoisson(50,$x);
figure;
        stem($x,$y);
        axis;
        axis('on');
        title('stem poisson');

figend;
```



stem Sine function



stem poisson distribution

Chapter5 Special effects — shq() function

5.1、some shq() effects

5.1.1、shq() set properties of elements

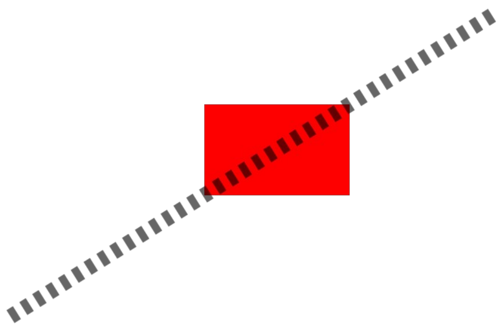
shq() set properties of the nearest element below it. Properties string is the same as SVG's original. e. g. :

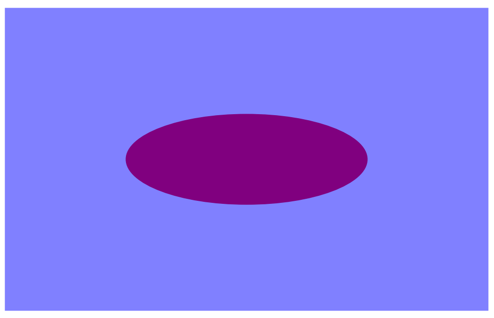
1，line dash & opacity

```
#!/usr/bin/perl
use SVGLAB;
figure;
        rect(1.4,1.7,0.3,0.3,'','','red');      # reference rect

        shq('stroke-dasharray="10"');
        shq('opacity="0.6"');
        line(1,1,2,2,20);      #stroke with 20 pixel

        axis;   #auto axis. (May not maintain the aspect ratio)
figend;
```

2，fill opacity

```
#!/usr/bin/perl
use SVGLAB;

figure;
        ellipse(2,2,0.5,0.3,'red');   #reference ellipse
        shq("fill-opacity='0.5'");
        rect(1,3,2,2);
        axis;   #auto axis.
figend;
```


dashed and transparent line


transparent rect

5.1.2、shq() effects

In fact, shq doesn't simplify the generation of special effects, here I only want to illustrate that these effects can be achieved by shq. Hope a more convenient way can be find latter.

5.1.2.1 Gradient

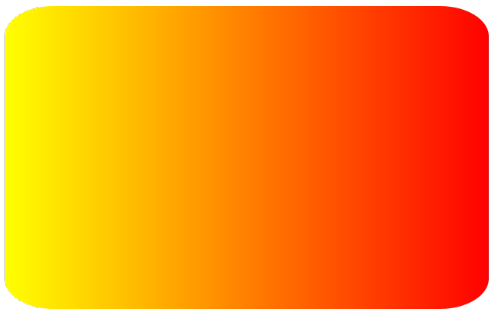
Put the 'id' defined by shq() into 'url()', use it as color value of some element. This is also how SVG does.

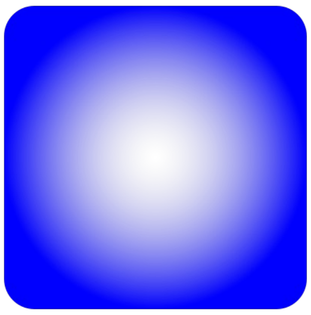
1， Linear gradient

```
#!/usr/bin/perl
use SVGLAB;
figure;
        shq('<defs>
                <linearGradient id="orange_red" x1="0%" y1="0%"
                         x2="100%" y2="0%">
                <stop offset="0%" style="stop-color:rgb(255,255,0);
                stop-opacity:1"/>
                <stop offset="100%" style="stop-color:rgb(255,0,0);
                stop-opacity:1"/>
                </linearGradient>
                </defs>
        ');#The definition of Gradual change above is from:
          #http://www.w3school.com.cn/svg/svg_grad_linear.asp

        rect(1,3,2,2,0.2,0.2,'url(#orange_red)');
        axis;#auto axis
figend;
```

2， Radial gradient

```
#!/usr/bin/perl
        use SVGLAB;
        figure;
                shq('<defs>
                        <radialGradient id="grey_blue" cx="50%"
                                 cy="50%" r="50%"
                        fx="50%" fy="50%">
                        <stop offset="0%" style="stop-color:rgb(200,200,200);
                        stop-opacity:0"/>
                        <stop offset="100%" style="stop-color:rgb(0,0,255);
                        stop-opacity:1"/>
                        </radialGradient>
                        </defs>
                ');#The definition of Gradual change above is from:
                  #http://www.w3school.com.cn/svg/svg_grad_radial.asp

                rect(1,3,2,2,0.2,0.2,'url(#grey_blue)');
                axis(1);#auto axis and maintain the aspect ratio.
        figend;
```


Linear gradient


Radial gradient

5.1.2.2 Blur

Put the 'id' defined by shq() into 'url(#)', then put the 'url(#id)' into another shq(). Because blur is not a kind of color, so is different as gradiant.

```
#!/usr/bin/perl
use SVGLAB;
figure;
        shq('<defs>
                <filter id="Gaussian_Blur">
                <feGaussianBlur in="SourceGraphic" stdDeviation="10"/>
                </filter>
                </defs>
        ');#definition of blur above is from:
          #http://www.w3school.com.cn/svg/svg_filters_gaussian.asp

        shq('filter="url(#Gaussian_Blur)"');
        rect(1,3,2,2,0.2,0.2);
        axis;
figend;
```

Gaussian blur

Following are blurs defined by SVG:
   * feBlend
   * feColorMatrix
   * feComponentTransfer
   * feComposite
   * feConvolveMatrix
   * feDiffuseLighting
   * feDisplacementMap
   * feFlood
   * feGaussianBlur
   * feImage
   * feMerge
   * feMorphology
   * feOffset
   * feSpecularLighting
   * feTile
   * feTurbulence
   * feDistantLight
   * fePointLight
   * feSpotLight

5.1.2.3 Animating

```
#!/usr/bin/perl
use SVGLAB;
AXIS(0);#back to SVG's original coordinate system.
figure(240,240);#SVG clock
```

```
		shq('transform="translate(120,120)  rotate(180)"');
		gp;

			circle(0,0,102,'none',2,'green');#clock's edge

			shq('   <animateTransform attributeName="transform"
				 type="rotate"
				repeatCount="indefinite" dur="12h" by="360" />
			');
			gp;#hour hand
				shq('opacity="0.5"');
				line(0,0,0,80,5,'black');
				circle(0,0,7,'black');
			gpend;
			shq('   <animateTransform attributeName="transform"
				 type="rotate"
				repeatCount="indefinite" dur="60min" by="360" />
			');
			gp;#minute hand
				shq('opacity="0.9"');
				line(0,0,0,95,4,'red');
				circle(0,0,6,'red');
			gpend;
			shq('   <animateTransform attributeName="transform"
				 type="rotate"
				repeatCount="indefinite" dur="60s" by="360" />
			');
			gp;#second hand
				line(0,0,0,100,2,'blue');
				circle(0,0,4,'blue');
			gpend;
		gpend;
		axis;

	figend;#The clock above followed from: http://kb.operachina.com/node/161
```

You can check SVG animating using opera9.

5.1.3、shq generate elements

You can use SVG's original description of elements by shq(), you can even use shq() to generate the whole picture:

```
	#!/usr/bin/perl
	use SVGLAB;
	figure;#shq() generate the whole picture
		shq('<line x1="100" y1="200" x2="400" y2="200" stroke-width="1" 
stroke="black" ></line>');
		shq('<rect x="500" y="100" width="100" height="300" rx="0" ry="0" 
fill="blue"  ></rect>');
		shq('<circle cx="200" cy="300" r="50"  fill="blue"  ></circle>');
		shq('<ellipse cx="300" cy="300" rx="50" ry="30"  fill="blue" 
></ellipse>');
	figend;
```

5.2、How shq works

Shq() put the string into proper position: if shq()'s parameter is properties of element, then put it into[pos 1]; if the parameter is animating description, then put it into [pos 2]; if the parameter is element effect such as blur, then put it into [pos 3].

The 'pos's above can be illustrated in the following SVG file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/REC-
SVG-20010904/DTD/svg10.dtd">
<svg width="800" height="500" xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">
<line x1="80" y1="450" x2="720" y2="250" stroke-width="1" stroke="black" [pos
1]>[pos 2]</line>
[pos 3]
<line x1="80" y1="450" x2="720" y2="50" stroke-width="1" stroke="black" ></line>
        <!--
        Generated using the Perl SVGLAB Module V1.0
        by Zhang Shichao
        email:zhshchao@163.com
 -->
</svg>
```

5.3、 Big SHQ

What is Big SHQ？ Though shq() is convenient, but is quite long sometimes, and is difficult to remember. So SHQ() appears. It simplified some long properties' name, such as 'fo' represents 'fill-opacity'. Take a look at the following command:

SHQ('xx:10')

Equivalent as shq('stroke-dasharray="10"');

And SHQ('sd:10;o:0.6')

Equivalent as two shq()s:

shq('stroke-dasharray="10"') and shq('opacity="0.6"');

Only three is defined currently:

sd	==stroke-dasharray
fo	==fill-opacity
o	==opacity

Separated the property and its value by ':' and several blanks.

If there is more than one properties in SHQ(), separate them by ';' and several blanks.

More simplify name will be added in the following version.

Chapter6 Improving in the future

The module should improve in the following aspects:
1、Support more SVG elements and effects.
2、Simulate more MATLAB figure function.
3、Simulate more MATLAB properties, such as vertical vector, matix.
4、More high level functions.

7.1 Support more SVG elements and effects

Support of path;
Sector function;
Triangle function;
Add more SHQ() simplify names.

7.2 Simulate more MATLAB figure function

Subplot;
Figure handle support;
3D figure(will be cool if achieved).

7.3 Simulate more MATLAB properties

Vexp;
vector calculation.

7.4 More high level functions

Pie figure;
Some frequently used figures for working.

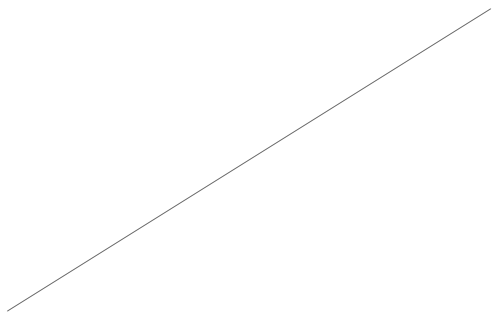
Appendix1 Some examples

e. g. 1、axis() auto adapt the axis

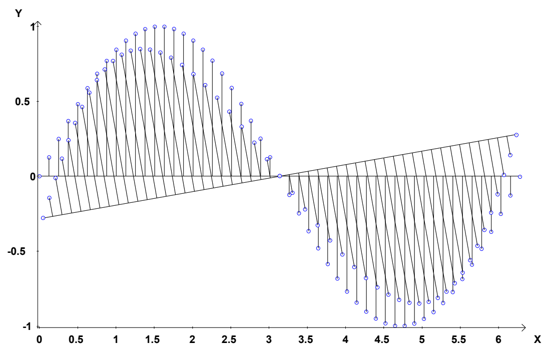
```
#!/usr/bin/perl
use SVGLAB;
figure;
        line(-1,-1,0,0);    #line from (-1,-1) to (0,0)
        axis;               #auto adapt
figend;
```

e. g. 2、rotate() on elements

```
#!/usr/bin/perl
use SVGLAB;
figure;
        $x=linspace(0,6.28,50);
        $y=vsin($x);
        stem($x,$y);
        rotate(-10,3.14,0);   #rotate 10 degrees around (3.14,0).
        stem($x,$y);
        axis;                 #auto adapt
        axis('on');           #axis arrow on
figend;
```



axis() auto adapt axis.



Rotate e.g.

e. g. 3、Using of group

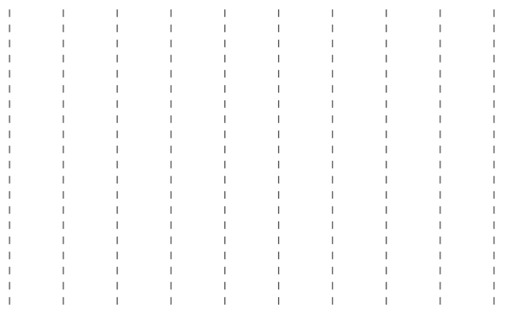
One can apply same properties to elements in one group.

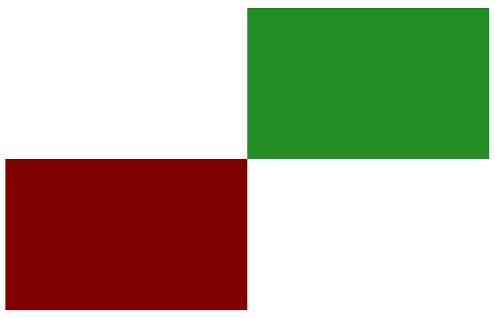
```
#!/usr/bin/perl
use SVGLAB;

    figure;
        SHQ('xx:10');       #or shq('stroke-dasharray="10"');
        gp;             #10 lines below in one group
        for$i(1..10){
            line($i,1,$i,10);
        }
        gpend;          #group end
        axis;
    figend;
```

e. g. 4、Color value

Color value can be values SVG support, such as 'red'、'rgb(128,0,0)'、'#238e23', and so on:

```
#!/usr/bin/perl
use SVGLAB;
    figure;
        rect(1,3,2,2,'','','rgb(128,0,0)');
        rect(3,5,2,2,'','','#238e23');
        axis;
    figend;
```


Same properties inside a group


Example of color values

Appendix2 Test of several important functions

Filename: test.pl
contents:

```
#!/usr/bin/perl -w
use strict;
use utf8;
use SVGLAB;

my ($x,$y);



##==================axis===================##

sub axis_test{
	figure;#SVGLAB1.svg
		$x=linspace(0,6.28,100);
		$y=vsin($x);
		plot($x,$y);	#can't see.
	figend;

	figure;#SVGLAB2.svg
		$x=linspace(0,6.28,100);
		$y=vsin($x);
		plot($x,$y);
		axis;#auto axis
	figend;#You can see a sine function inside the figure.

	figure;#SVGLAB3.svg
		$x=linspace(0,6.28,100);
		$y=vsin($x);
		plot($x,$y);
		axis;
		axis('on');#axis arrow.
	figend;

	figure;#SVGLAB4.svg
		$x=linspace(0,6.28,100);
		$y=vsin($x);
		plot($x,$y);
		axis(0,3.14,0,1);
		axis('on');
	figend;

	figure;#SVGLAB4.svg
		$x=linspace(0,6.28,100);
		$y=vsin($x);
		plot($x,$y);
		axis(0,3.14,0,1);
		axis('on');
		axis(1);			#keep aspect ratio.
	figend;
}
#axis_test();
##==================axis end==================##
```

```
##==================SVGLAB==================##
sub svglab1{
	SVGLAB;#default.
		figure;
		figend;

		figure;
		figend;

		figure;
		figend;
#Three files named: SVGLAB1.svg、SVGLAB2.svg、SVGLAB3.svg, 800*500 pixels.
}
#svglab1();
sub svglab2{
	SVGLAB(2000,1500);
		figure;
		figend;

		figure;
		figend;

	SVGLAB(200,100);
		figure;
		figend;

		figure;
		figend;
#Two files: SVGLAB1.svg、SVGLAB2.svg, 200*100;
#SVGLAB resets the file number, so the former two was covered;
}
#svglab2();
sub svglab3{
	SVGLAB('scaffold');
		figure;
		figend;

		figure;
		figend;
	SVGLAB('contig');
		figure;
		figend;

		figure;
		figend;
#scaffold1.svg、scaffold2.svg
#	contig1.svg、contig2.svg, 800*500.
}
#svglab3();
sub svglab4{
	SVGLAB('scaffold',2000,1500);
		figure;
		figend;

		figure;
		figend;
	SVGLAB('contig',200,100);
		figure;
		figend;
		figure;
		figend;
#Four files: scaffold1.svg、scaffold2.svg, 2000*1500;
#	contig1.svg、contig2.svg, 200*100.
}
#svglab4();
sub svglab5{
	SVGLAB('scaffold',2000,1500);
		figure;
		figend;

		figure;
		figend;
	SVGLAB;			#back to default
		figure;
		figend;

		figure;
		figend;
#Four files: scaffold1.svg、scaffold2.svg, 2000*1500;
#	SVGLAB1.svg、SVGLAB2.svg, 800*500.
}
#svglab5();
##===================SVGLAB end===================##

##===================AXIS===================##
sub AXIS_test1{
	AXIS;#default.
		figure;
			line(100,100,700,400);
			#axis;#('on');
		figend;

	AXIS(0);#using SVG's original coordinate system.
		figure;
			line(100,100,700,400);
		figend;
	AXIS(0.5);#zone zooming.
		figure;
			line(100,100,700,400);
		figend;

#Four files: SVGLAB1.svg, a line upward;
#	SVGLAB2.svg, a line downward;
#	SVGLAB3.svg, a line upward, half size of the line in SVGLAB1.svg.
}
#AXIS_test1();
sub AXIS_test2{
	AXIS;
		figure;#SVGLAB1.svg
			rect(300,300,5000,5000);# rectangle  out of the figure.
		figend;

	AXIS;
		figure;#SVGLAB2.svg
			rect(300,300,5000,5000);
			axis;
		figend;
```

```
        AXIS;
                figure;#SVGLAB3.svg
                        rect(300,300,5000,5000);
                        axis;
                        axis(1);
                figend;#a square.


        AXIS(0.8,1,1,1);
                figure;#SVGLAB4.svg
                        rect(300,300,5000,5000);
                        axis;
                figend;#a square.

        AXIS(0.8,0,1,1);#do not keep the aspect ratio.
                figure;#SVGLAB5.svg
                        rect(300,300,5000,5000);
                        axis;
                        axis(1);        #keep the aspect ratio,cover the global 'do not'.
                figend;#a square.

        AXIS(0.99);#zooming, 0.99 of figure;
                figure;#SVGLAB6.svg
                        rect(300,300,5000,5000);
                        axis;
                figend;
}
#AXIS_test2();
sub AXIS_test3{

        AXIS;
                figure;#SVGLAB2.svg
                        $x=linspace(0,6.28,100);
                        $y=vsin($x);
                        plot($x,$y);
                        axis;
                        axis(0,3.14,-1,1);#setting axis range.
                        axis('on');
                figend;

        AXIS(1,1,1,0);#do not cover the edge.
                figure;#SVGLAB3.svg
                        $x=linspace(0,6.28,100);
                        $y=vsin($x);
                        plot($x,$y);
                        axis;
                        axis(0,3.14,-1,1);
                        axis('on');
                figend;
}
#AXIS_test3();
##===================AXIS end===================##
```