Implementation of Stereographic and Spherical Projection in Structural Geology with JavaScript and the web through **SvgNet**

Arijit Laik

M.Sc. Semester 1,
Department of Geology,
Durgapur Goverment College

Abstract

Projections are a "prescribed" part of structural analysis and visualizations, with the change in day and age numerous computer applications have been written to aid this. Most of them have taken care in explaining and elaborating the mathematical and programmatic aspects of their implementation. These applications and/or programs all have attained a robust and well accepted status in the field. The previously developed programs have mostly taken a traditional approach in code and design, i.e. the use of programming languages like C/C++, Fortran, Java, Python or packages like Matlab, Mathematica etc. The Open Source Web application discussed here takes up the task of removing system dependencies of the code by the use of JavaScript, the language of the web-browsers, for the entirety of the programming, and improves portability of the output by **directly** plotting into Scalable Vector Graphics(SVG).

1 Introduction

Representation of orientation of linear and planar elements in spherical and stereographic projection inevitably aid Structural Geologists with their research and other objectives. Thus a proper approachable method is to be followed when projections are dealt with, fundamentals of projections have deep seated roots inside mathematics and analytical geometry which are generally not discussed in structural geology textbooks, however literature like Goodman and Shi [1985] and Pollard and Fletcher [2005] respective describe and review the basic concepts, the analytical expressions necessary to plot the attitudes of planar and linear structural elements in the equal-angle and the equal-area lower hemispherical projections, and discuss the construction techniques for those nets, which serve as the backbone of the algorithms used in the discussed application. Majority of available orientation analysis packages(like Cardozo and Allmendinger [2013], Vollmer [2015]) are developed on system dependent native programming languages or scripting packages and produce raster output which may be exported into the vector format. **SvgNet.js** removes this hurdle by generating *SVG(Scalable Vector Graphics)* markup directly from input data as inline SVG in an HTML Document Object, via a JavaScript code, more commonly referred to as Document Object Model (DOM) which is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. In simple terms its a JavaScript application program interface(API) which generates a vector description(inline SVG elements) representing the graphic output, from the data i.e. linear or planar attitudes within a simple web-browser interface.

2 Expressions and Relations

Leaving aside the rigorous analytical and geometrical procedures discussed in Goodman and Shi [1985] and Pollard and Fletcher [2005] the expression that are necessary, and have been used, for the programmatic implementation of the graphical projections are discussed below. The reference Cartesian or polar coordinate system used in all of the following equations have their center coinciding with the center of the reference sphere and primitive circle.

2.1 Equal-Angle Lower Hemispherical Projection

2.1.1 Planar Attitudes

The stereographic projection of the inclined plane is a circle whose center is at C, and whose radius, r, are accordingly determined by, From [Goodman and Shi, 1985, p. 67-68] :

$$r = \frac{R}{\cos\alpha} \tag{1}$$

and the coordinates of the center of the circle representing the plane

$$(C_x, C_y) = (R\tan\alpha\sin\beta, R\tan\alpha\cos\beta) \tag{2}$$

where R is the radius of the primitive circle, α and β are the dip and dip direction, respective, of the planar element.
It is to be noted that the scope of projection is for one hemisphere only hence the projection of a planar elements on the Wulff net is an arc which starts at the point (R, θ) and ends at $(R, \theta + \pi)$, where θ is the strike of the plane i.e. according to the Right Hand Rule $(\beta - \pi/2)$.

2.1.2 Linear Attitudes

These are the equations used to plot the projection of a linear element on a stereonet of radius R, given the azimuth of plunge, α , and angle of plunge, ϕ , from [Pollard and Fletcher, 2005, p. 59],

$$(x, y) = \left(R\tan\left(45^\circ - \frac{1}{2}\phi\right)\sin\alpha, R\tan\left(45^\circ - \frac{1}{2}\phi\right)\cos\alpha\right) \tag{3}$$

this equation 3 is generally used for plotting linear and planar elements,as a family of linear elements, but **SvgNet.js** does't use this for equal-angle projection of planes, for the sake of simplicity, reduced number of steps in plotting a great circle directly as a circle due to the availability of a vector elliptical arc implementation in the SVG `path` element.

2.1.3 Small Circles

The locus of lines (through a common origin) making an equal angle with a given direction through the origin is a cone. This cone pierces a sphere about the origin along a circle; it is denoted a small circle because it can be generated also by the intersection of the reference sphere with a plane that does not contain the origin. By the fundamental property of stereographic projection[Goodman and Shi, 1985, p. 93-94], any small circle on the reference sphere in the projection plane[Goodman and Shi, 1985, p. 71]. A bit geo-logically a pair small circles in a stereonet reference grid can be imagined as the locus of projection of all line which have the same *rake* on a plane with a fixed *strike* and varying i.*dip* and ii. *dip direction*, which can be expressed as function of semi-epcial angle ω and the R, the radius of the primitive circle as a circle with radius,s and center (S_x, S_y), [Goodman and Shi, 1985, p. 75]

$$s = R\tan\omega \tag{4}$$

$$(S_x, S_y) = \left(0, \frac{\pm R}{\cos \omega}\right) \tag{5}$$

2.2 Equal-Area Lower Hemispherical Projection

This projection system is weakly developed presently in the application as the statistical implications are not yet worked out with in the API, yet this projection is done in a fairly cumbersome manner by the use of a single expression that relates the xy coordinates of the projection to the trend and plunge of a linear attitude. Hence, planar elements are projected as the a line through a fixed number of points on the beizer cubic curve i.e. the projection of planes in this system. Although SVG supports the drawing of these curves though the four control points a proper computational approach of plotting them as the ideal cubic curve is not well written about in any literature. Mostly due to the fact that this projection is used for statistical analysis where it is generally more convenient,owing to a generic good sample size, to plot the poles to a plane than the plane itself .
The coordinates of a point on the Schmidt net of radius R representing the orienta- tion of a linear element with plunge direction, α , and plunge angle, ϕ (foliations or other planar fabrics the azimuth and plunge of the normal) are , from [Pollard and Fletcher, 2005, p. 66],

$$(x, y) = \left(R\sqrt{2}\sin\left(45^\circ - \frac{1}{2}\phi\right)\sin\alpha, R\sqrt{2}\sin\left(45^\circ - \frac{1}{2}\phi\right)\cos\alpha\right) \tag{6}$$

3 JavaScript,SVG and the DOM

3.1 SVGs

Since **SVG** is the graphical component of the discussed application, here is a brief description of the tools the vector graphic language provides, that have been integrally used in the program.
According to Dailey [2010] **Scalable Vector Graphics (SVG)** is a Web graphics language. SVG defines markup and APIs for creating static or dynamic images, capable of interactivity and animation, including various graphical effects. It can be styled with CSS, and combined with HTML.
The default coordinate system in SVG is much the same as in HTML. It works as a two-dimensional $x - y$ plane. The origin (where $x = 0$ and $y = 0$) is the *upper left-hand corner*. As we move right from there, x increases. As we move downward, y increases.

3.1.1 SVG Elements

The `<line>` object draws a line between two specified points: $(x1, y1)$ and $(x2, y2)$. In order to see the line, it must have a stroke (i.e., a color). Hence, a sort of minimal line consists of code such as the following:

```
<line x1="5" y1="5" stroke="red" x2="90" y2="90" />
```

The `<circle>` does have a slightly simpler syntax the simplest circle requires only a center point (cx, cy) and a radius,r:

```
<circle cx="80" cy="50" r="40"/>
```

`<path>` is a very flexible drawing option. It renders the movement of a stylus through two dimensions, with both pen-up and pen-down options, including straight and curved segments joined together at vertices which are either smooth or sharp. One other aspect of the `<path>` deserves mention. That is the **elliptical arc**. It might seem that an arc would be a very simple topic, but that given any two points in the plane and two elliptical radii, there often are two ellipses that traverse those points with specified radii and those points specify two different arcs for each ellipse. The arc subcommand of the `<path>` has the following syntax: A `rx ry XAR large-arc-flag sweep-flag x y`. The

arc begins at the current point (determined by the last coordinate specified, e.g. by the M subcommand), and ends at (x, y). The ellipse will have radii of `rx and ry`, with the x-axis of the ellipse being rotated by `XAR` degrees. The particular ellipse (of the two possible) is specified by the `large-arc-flag (0 or 1)` and the particular segment of the ellipse is specified by the `sweep-flag. (0 or 1).`

It begins by specifying where the drawing will begin by inserting as the first element of d a notation such as`x y` for numbers `x and y`. It might be though of as `M x y` as meaning *move pen to the coordinate x y.* From there the options of moving (with pen still down on the canvas)

- linearly (L),
- quadratically (Q),
- cubically (C)
- or through an elliptic arc (A).

For example, `d="M 100 100 L 200 200"` would succeed in drawing a diagonal line from the point $(100, 100)$ to the point $(200, 200)$ as shown.

```
<path stroke="black" d="M 100 100 L 200 200"/>
```

3.2 SvgNet.js

The javascript file which is the core of all the functionalities of this application is named SvgNet.js, accessible via the open-source repository at `https://github.com/arijitlaik/SvgNet/`, not only provides API level functions which can be used to generate plots from input data like strike, dip, pitch plunge, rakes etc but also provides some basic SVG plotting API function, all of these could be used in a browser JavaScript console. Thus successfully providing for all the functionalities like

1. minimal, non dependent API methods to create svgs
2. functions to create svg elements from input data of attitudes
3. Objects oriented approach and access e.g. the plane, with their inherent properties like strike,dip and other properties like colour and thickness of its plot, which is another object with its intrinsic properties)

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc. JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard. This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63. Consequently these are passed on to the SVG attribute values, giving a more precise plot representation, although this precision does add a size constrains of polyline elements with lots of data points.

3.3 Usage

The SvgNet.js is intended to be a standalone wrapper for structural geology projection plots, and could find usage a an plotting API or an application after binding it with an standardised user interface and utility provisions. By virtue of its base in web based standards its inter-portability and device independent nature could aid both field based and later stages of structural analysis or as a library for the real-time cross-client display of orientation data.

3.3.1 Description of function objects and prototypes

Following is a brief discussion of the function objects, object prototypes and variables used in the API

`SchmidtNet_Flag`: the global *boolean* variable to determine projection type

`Pt(x_cord, y_cord)` : object prototype for a set of Cartesian coordinates with members x and y, as specified by the argument

`center`: the global variable of type *Pt* representing the center of the figure in the svg coordinate system

`radius_primitive`: global variable storing the radius of the primitive circle

`Svg_obj(in_id, id)`: is the object constructor for a new SVG object, i.e. creates a SVG within the *in_id* element with the argument specified *id*

`cart2svg(cart_cords)` : returns the SVG coordinate transformation of Cartesian coordinate arguments

`torad(degrees)` : returns the radian approximation of *degrees*.

`todeg(radians)` : returns the degree approximation of *radians*.

`linText(start,end)` : returns the d attribute for a straight line from the point $(start.x, start.y)$ to the point $(end.x, end.y)$, calls the `cart2svg` for coordinate transformation

`polytext(points)` : returns the d attribute for a polyline through the points in $(points[i].x, points[i].y)$ where $i = 0, 1, 2, 3 \ldots points.length$

`arcText(start_angle, radius, end_angle, ccw)` : returns the d attribute for a circular arc with `radius` and between the polar coordinates $(r, start\_angle)$ and (r, end_angle) the argument `ccw` is boolean indicating the sense of movement between the specified points

`Path_obj(d, stroke, stroke_wth, fill, deg, id)`: Object prototype of a SVG path element which creates a <path>, with the given arguments as its attributes, the d being generated by or a direct call to the d attribute generator functions such as `arcText(), polyText, lineText`, as child of a Svg_obj, intended to be used with plotting non-point projected elements like lines, poles to planes etc

`Circ_obj(cen, radius, stroke, stroke_wth, fill, deg, id)` Object prototype of a SVG circle element which creates a <circle>, with the given arguments as its attributes, as child of a Svg_obj

`Wtp2cart(plunge, trend)` : applies the equation 3 to convert the `trend and plunge` to cartesian coordinates returning it object of type `Pt`

`Stp2cart(plunge, trend)` : applies the equation 6 to convert the `trend and plunge` to cartesian coordinates returning it object of type `Pt`

`Ssdr2cart(strike, dip, rake, op_flag)` : this utilizes a trigonometric conversion from rake data to trend and plunge with the help the equations

$$trend = strike + \tan^{-1}(cos(dip))(\tan(rake)) \tag{7}$$

$$Plunge = \sin^{-1}(\sin(dip))(\sin(pitch)) \tag{8}$$

this function also take into consideration the strike end from which the rake, conventionally maximum 90°, is measured w.r.t. the RHR strike put in as the argument, and does necessary augmentations

`Plane(strike, dip, clr, lwidth, id)`: Apart from the necessary properties like strike, dip, colour, stroke width, and id, this prototype has a member function `draw()` which creates the property `plot` an object of type `Path_obj` depending on the dip and projection type. The plot is a arc of radius defined in equation 1 in a *Wulff net* and a poly-line estimation of points that are projections of lines which have rake of 0° to 90° measured from both the strike ends of the Plane, in a *Schmidt net* and a line when $dip = 90$. This `draw()` functions call `Ssdr2cart()` for the conversion of rake type data to Cartesian coordinate data used in the poly line estimation. The `modify` function of this object allows

modification of a previously created *Plane* object after its strike/dip are changes

Object prototypes for attitudes that plot as point objects are

1. `Line(trend, plunge, clr, id)` : the simple line object constructor
2. `LineonPlane(onPlane, pitch, op_flag, clr, id)`: the line on a plane, i.e parameter `onPlane` is an object of type `Plane` but has not plot object, as `Plane.draw()` is called only when the colour: `clr` is passed on to as argument of parameter, passes the values obtained from equn. 8 and equn. 7 and the `op_flag` to `Stp2cart and Wtp2cart`.
3. `PoletoPlane(ofPlane, clr, id)`: the pole to plane which similarly uses a no-plot `Plane` object as a parameter to calculate the trend and plunge of the pole.

These prototypes hold design similarity to `Plane` in context of the the member functions `draw()`: which creates `Circ_obj` with radius 2 or 1 pixels, with centres determined by equations 6 or 3; and `modify()` function .

The object prototypes `WulffNet()` and `SchmidtNet()` are the for the creation of nets of the respective types, with grid intervals of 5°s .
Wulff Net great circles and small circles are `Path_obj` function calls with the "d" parameter arguments being returned by `arcText` function with radii according to equations 1 and 4 respectively.
Schmidt Net being draw by loci line estimations of cubic curves, estimated from the rake on a plane to trend/plunge data then to Cartesian Coordinates by the use of the i `Ssdr2cart` which then uses ii. `Stp2cart`, finally all the data points 5°intervals of rakes for planner projections, 5°intervals of dip and a single dip direction change for conical surface(equivalent to small circles) are passed as an array of `Pt` type objects to the `polytext` function which returns the *d* attribute to the `Path_obj`.

4 Conclusion and Roadmap

4.1 Using the SvgNet.js in the SvgNet app

The context of this article has mainly dealt in the procedural part of the application i.e. the design of the API that aids in the building the app, thus User Interface part need little discussion. The API has been designed in such away that it paves the an easy path for HTML5 single page application development work-flow. The function provided can be blinded to HTML elements with much easy and without the use of any external libraries, although input parser and export to raster formats need not be done from bare-bones. The SvgNet.js script combined with other input parser would allow the implementation of data import from cvs and spreadsheets. A minimal Sample Application build on the SvgNet.js is called SvgNet and can be accessed online at `https://arijitlaik.github.io/SvgNet` it provides data input and the option of saving the thus generated inline SVG for use with any generic Vector Graphics editor like Corel Draw, Inkscape and Adobe Illustrator.

4.2 The Road Ahead

The API is a standalone lightweight library that can be used to render simple Structural geology projections on the Web. The web platform being the to-go place for all the technological diaspora, and the Fluid, Open and System Independent Platform leaves us with a lots of new ways of exploring data, and hence the urge to port exiting native applications to the web is but a necessity. This application being the first of its(Free and Open Source) kind in this field, is still at conceptual Alpha level release. Advanced analytical procedures such as Density Contouring, statistical implications, exporting of

the data via JSON (JavaScript Object Notation), a lightweight data-interchange format, for humans to read and write and for machines to parse and generate, needs to be groomed out from the present state. The Developer of the application would be humbly grateful for the reviews(at laikarijit@gmail.com) and advice on taking this code to framework level altitudes which would imply to be a completely robust and portable web infrastructural interface for Structural Analysis on an open and accessible platform.

Important links

Source-code: `https://github.com/arijitlaik/SvgNet`
Bugs/Issues: `https://github.com/arijitlaik/SvgNet/issues`
Sample WebApp: `http://arijitlaik.github.io/SvgNet/`
and its compiled Mobile App: `https://build.phonegap.com/apps/1624590/share`

References

N. Cardozo and R. W. Allmendinger. Spherical projections with {OSXStereonet}. *Computers & Geosciences*, 51:193 – 205, 2013. ISSN 0098-3004. doi: http://dx.doi.org/10.1016/j.cageo.2012.07.021. URL `http://www.sciencedirect.com/science/article/pii/S0098300412002646`.

D. Dailey. An svg primer for today's browsers, 2010. URL `http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.htm`.

R. Goodman and G. Shi. *Block theory and its application to rock engineering*. Prentice-Hall international series in civil engineering and engineering mechanics. Prentice-Hall, 1985. ISBN 9780130781895. URL `https://books.google.co.in/books?id=AclRAAAAMAAJ`.

D. Pollard and R. Fletcher. *Fundamentals of Structural Geology*. Cambridge University Press, 2005. ISBN 9780521839273. URL `https://books.google.co.in/books?id=9atcPwpwgJ0C`.

F. W. Vollmer. Orient 3: Spherical projection and orientation data analysis program, 2015. URL `http://www.frederickvollmer.com/orient/`.