OPENSCAD

A 2D and 3D CAD system, employing its own language. Only 2D capabilities are addressed here.

The system is available at:-

www.openscad.org

and documentation is located at:-

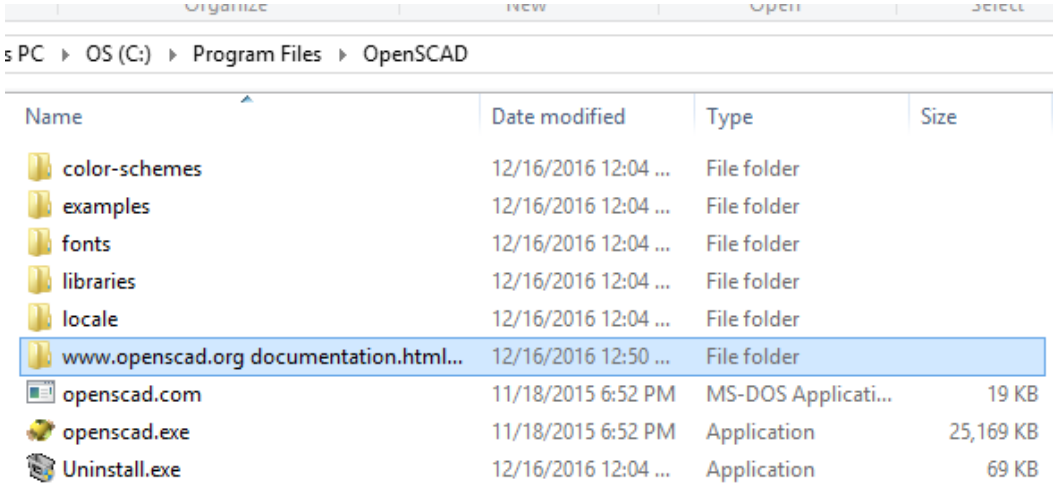
https://en.wikibooks.org/wiki/OpenSCAD_User_Manual

however, there is more at the online version of the printed documentation at:-

https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Print_version

and there are notes that each contains material not in the other.

Installation iss very easy, it works first time, no additional libraries needed, and a folder is created like:-



Clicking “openscad.com” or “openscad.exe” loads the system. It is that easy.

Creating a program for “openscad” is simple.

FILE
NEW

lets you create a file, and

DESIGN
PREVIEW

lets you run it. Programs are saved as “xxxxxx.scad”. DESIGN then RENDER will show in one color all the objects of a drawing made very easy to see, whereas DESIGN then PREVIEW shows objects as programmed, and true to the programmed scale and color.

However, beyond that, there is some stuff that must be emphasized…

ASSIGNMENT OF VARIABLES

A key point to understand right at the start is how it does variable assignment. Assignments are made at compile time, not run time. This is stated in the documentation.

```
// example 1
echo (a);
a=1;
a=3;
```

produces

```
Saved backup file:
C:/Users/simon/Documents/OpenSCAD/backups/swsTest1-backup-
FSpy8264.scad
Compiling design (CSG Tree generation)...
ECHO: 3
Rendering Polygon Mesh using CGAL...
```

Further, messing with a defined variable causes problems

```
// example 2
a=1;
a=a+1;
echo(a);
```

produces

```
Saved backup file:
C:/Users/simon/Documents/OpenSCAD/backups/swsTest1-backup-
FSpy8264.scad
Compiling design (CSG Tree generation)...
WARNING: Ignoring unknown variable 'a'.
ECHO: undef
```

So to increment a variable, you must use another variable

```
// example 3
a=1;
echo(a);
b=a+1;
echo("then ",b);
b=a+6;
echo("finally ",b);
```

but note how both printouts of “b” produce the same number, “7”. In other words assignments must be considered carefully, and the documentation does say this.

```
Saved backup file:
C:/Users/simon/Documents/OpenSCAD/backups/swsTest1-backup-
FSpy8264.scad
Compiling design (CSG Tree generation)...
ECHO: 1
ECHO: "then ", 7
ECHO: "finally ", 7
```

Even in a FOR loop, things are done be compile time assignment:-

```
// example 4
a=1;
echo(a," then...");
for (i=[a:1:5])
{
    j=i*3;
    echo(j);
}
```

which results in:-

```
Saved backup file:
C:/Users/simon/Documents/OpenSCAD/backups/swsTest1-backup-
kkmI2080.scad
Compiling design (CSG Tree generation)...
ECHO: 1, " then..."
ECHO: 3
ECHO: 6
ECHO: 9
ECHO: 12
ECHO: 15
```

But once “j” is assigned, don’t re-assign it:-

```
// example 5
a=1;
echo(a," then...");
for (i=[a:1:5])
{
    j=i*3;
    j=j+100;
    echo(j);
}
```

results in the second “j” assignment “j=j+100” being in error

```
Saved backup file:
C:/Users/simon/Documents/OpenSCAD/backups/swsTest1-backup-
kkmI2080.scad
Compiling design (CSG Tree generation)...
ECHO: 1, " then..."
WARNING: Ignoring unknown variable 'j'.
ECHO: undef
WARNING: Ignoring unknown variable 'j'.
ECHO: undef
WARNING: Ignoring unknown variable 'j'.
ECHO: undef
WARNING: Ignoring unknown variable 'j'.
ECHO: undef
WARNING: Ignoring unknown variable 'j'.
ECHO: undef
```

Again, this is documented, it is just not quite the same as many other languages. However, this is how TurboCAD handles the “parametric script”, so it is not totally unknown. So what this all means is that you must be very careful with variable assignment, and changes to them.

Variables are set at compile-time, not run-time

Because OpenSCAD calculates its variable values at compile-time, not run-time, the last variable assignment, within a scope will apply everywhere in that scope, or inner scopes thereof. It may be helpful to think of them as override-able constants rather than as variables.

```
// The value of 'a' reflects only the last set value
   a = 0;
   echo(a);  // 5
   a = 3;
   echo(a);  // 5
   a = 5;
```

While this appears to be counter-intuitive, it allows you to do some interesting things: For instance, if you set up your shared library files to have default values defined as variables at their root level, when you include that file in your own code, you can 're-define' or override those constants by simply assigning a new value to them.

Scope of variables

When operators such as translate() and color() need to encompass more than one action (actions end in ;), braces {} are needed to to group the actions, creating a new, inner scope. When there is only one semicolon, braces are usually optional.

Each pair of braces creates a new scope inside the scope where they were used. **Since 2015.03**, new variables can be created within this new scope. New values can be given to variables which were created in an outer scope . These variables and their values are also available to further inner scopes created within this scope, but are **not available** to any thing outside this scope. Variables still have only the last value assigned within a scope.

https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Print_version#Conditional_and_Iterator_Functions

Language

Why am I getting an error when writing a = a + 1?

First of all, the question why we have these "limitations" will become more clear once we start better exploiting the opportunities.

We need a "reduce" function to help collecting information depending on a list of input. Recursion is fine, but people tend to struggle with it and we could offer some help. We should probable disallow any attempt of reassignment, to make it more clear what's going on. The only real reason we partially allow it is to allow cmd-line variable overrides.

To help think about things:

Imagine every expression in OpenCAD being executed in parallel. Any dependency of existing expressions must be made explicit by hierarchical grouping. This will kill the idea of iterating in order to accumulate information.

In terms of functions: Imagine a function expression being something you'd type into a spreadsheet cell. Not totally mappable, but it might help framing it.

Now, we could add all kinds of sugar to help people apply their existing programming problem solving skills. Question is more if it really helps us, secondary who will spearhead the design of such language extensions, as we currently don't really have attachment for these ideas on the dev-team.

If you think about the OpenSCAD language as something similar to HTML, but for 3D modeling, you'd still have a need for various programs generating code in this language (similar to the plethora of HTML generators out there). There exist a number of tools for helping with OpenSCAD code generation from existing programming languages (python, ruby, C++, haskell, clojure off the top of my head) and there are tools offering Javascript interfaces for similar purposes (OpenJSCAD, CoffeeSCAD). Until we have a really good reason to do so in OpenSCAD proper, and a really good candidate for which language to support, I think it's better to keep these things separate.

SCOPE

The scope of a variable is within the current scope and further nested scopes.

```
a=1;
if (a==1) {
   a=2;
   echo (a);
}
echo (a);
```

will produce

```
Saved backup file: C:/Users/simon/Documents/OpenSCAD/backups/…
Compiling design (CSG Tree generation)...
ECHO: 2
ECHO: 1
Rendering Polygon Mesh using CGAL...
WARNING: No top level geometry to render
```

This means that within a FOR or IF block of code, that code may not modify a variable higher up the food chain. This is significant for example in the example horizontal sundial program where corrections are made for the four segments, 0000-0559, 0600-1159, 1200-1800, and 1800-2359. It means that some IF statements cannot simply change an “X” and “Y” ordinate, and then have one block of code draw the line and place the text, rather, it means that each IF block of code must do everything. This means duplicating lots of code.

READING USER INPUT

It would be nice if one could read answers from the console, replying to “echo”, for example allowing the program to ask the human for a latitude, and so on.

It would be nice, but we don’t always get what we want, and openScad does not support user data input at the time of execution. Data can be read from a file, however it is just as simple to put that data insode the program.

However, openJScad, the web variant, does allow on screen parameters.

CONDITIONAL TESTING

The use of “= =” in a conditional statement is consistent with many languages:-

```
a = 123;                    // “=” is an assignment of 123 to the variable “a”

if (a==123) {               // “==” is not an assignment, it is a test
}
```

There can be multiple conditions in an IF, such as “&&”:-

```
if ((i==12) && (lng<=ref) ) {
}
```

A BASIC LIMITED HORIZONTAL SUNDAL PROGRAM THAT IS INCOMPLETE

The obvious question then, is how does one program something like a horizontal dial in 2D using this system.

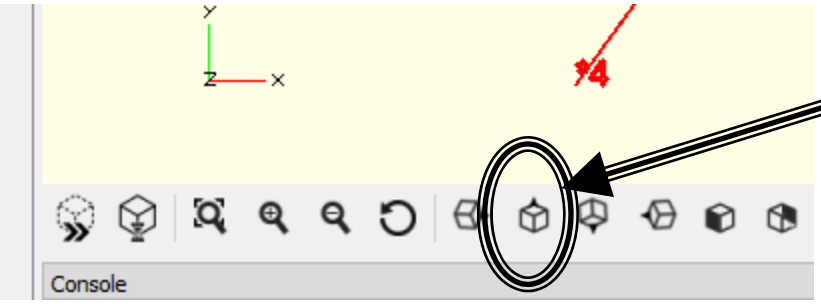
```
// 2d hDial
st=6.01;
en=19.01;
lat=33;
text("      33");
for (i=[st:1:en])
{
    hla=atan(sin(lat)*tan(15*i));
    //echo (lat,i,hla);
    l=100;
    x=sz*sin(hla);       // or x=cos…
    y=sz*cos(hla);       // or y=sin…
    echo (lat,i,hla,"....",x,y);
    polygon([[0,0],[x,y],[0.2,0.1]]);
}
// HORIZONTAL SUN DIAL
//am  pm    am    pm
//12.00     12.00 0.00  0.00
//11.00     1.00  8.30  8.30
//10.00     2.00  17.46 17.46
//9.00      3.00  28.57 28.57
//8.00      4.00  43.33 43.33
```

Produces

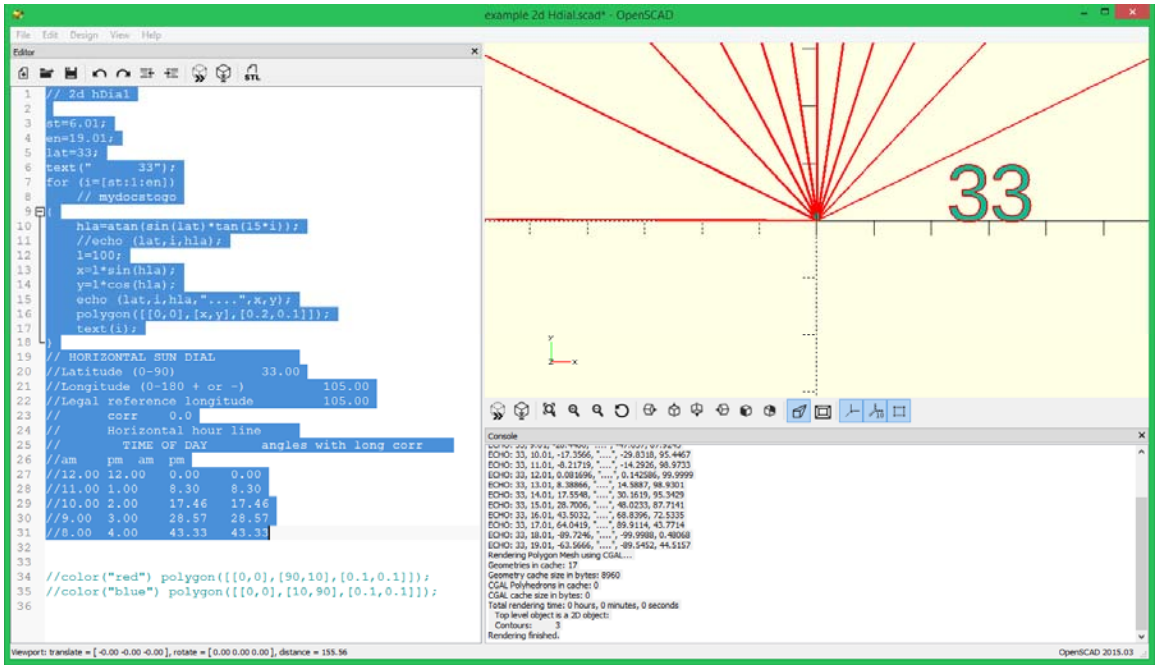
```
Saved backup file:
C:/Users/simon/Documents/OpenSCAD/backups/example 2d Hdial-
backup-WUjo4924.scad
Compiling design (CSG Tree generation)...
ECHO: 33, 6.01, -89.7246, "....", -99.9988, 0.48068
ECHO: 33, 7.01, -63.5666, "....", -89.5452, 44.5157
ECHO: 33, 8.01, -43.1574, "....", -68.4005, 72.9477
ECHO: 33, 9.01, -28.4486, "....", -47.637, 87.9245
ECHO: 33, 10.01, -17.3566, "....", -29.8318, 95.4467
ECHO: 33, 11.01, -8.21719, "....", -14.2926, 98.9733
ECHO: 33, 12.01, 0.081696, "....", 0.142586, 99.9999
ECHO: 33, 13.01, 8.38866, "....", 14.5887, 98.9301
ECHO: 33, 14.01, 17.5548, "....", 30.1619, 95.3429
ECHO: 33, 15.01, 28.7006, "....", 48.0233, 87.7141
ECHO: 33, 16.01, 43.5032, "....", 68.8396, 72.5335
ECHO: 33, 17.01, 64.0419, "....", 89.9114, 43.7714
ECHO: 33, 18.01, -89.7246, "....", -99.9988, 0.48068
ECHO: 33, 19.01, -63.5666, "....", -89.5452, 44.5157
Rendering Polygon Mesh using CGAL...
Geometries in cache: 17
Geometry cache size in bytes: 8960
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Total rendering time: 0 hours, 0 minutes, 0 seconds
Top level object is a 2D object:
Contours: 3
Rendering finished.
```

Do DESIGN and then PREVIEW, or in this case below, RENDER

Let us analyze this.



select the correct view



First, the trigonometric functions use degrees rather than radians, so that is simple. However, the 2D functions are limited to:-

```
square
circle
polygon
```

and there is no “line” function, so the “polygon” is used:-

```
polygon([[0,0],[x,y],[0.1,0.1]]);
```

but the polygon needs to end at a slightly place than it started, otherwise the line may cancel itself out, hence the “,[0.1,0.1]]);” ending. (see later for How To Draw A Line)

The reason for 6pm not showing up was because of the sign of the hour line angle variable (hla). This was fixed with some conditional statements.

TRIGONOMETRY AND SIGNS OF SIN, COS, ETC

OpenScad works well with trigonometric signs as can be seen below:-

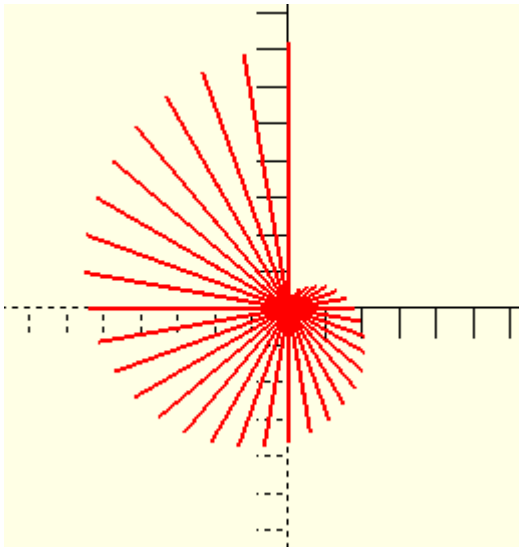
```
// sample 6

st=0;
en=360;
text("      sample 6");
for (i=[st:10:en])
{
    hla=i;
    l=i*2;
    x=l*sin(hla);
    y=l*cos(hla);
    echo (i,,"....",x,y);
    polygon([[0,0],[x,y],[0.1,0.1]]);
}
```

producing

Saved backup file: C:/Users/simon/Documents/OpenSCAD/backups/examples 6 thru nn-backup-GPwa3784.scad

```
Compiling design (CSG Tree generation)...
ECHO: 0, "....", 0, 0
ECHO: 10, "....", 3.47296, 19.6962
ECHO: 20, "....", 13.6808, 37.5877
ECHO: 30, "....", 30, 51.9615
ECHO: 40, "....", 51.423, 61.2836
ECHO: 50, "....", 76.6044, 64.2788
ECHO: 60, "....", 103.923, 60
ECHO: 70, "....", 131.557, 47.8828
ECHO: 80, "....", 157.569, 27.7837
ECHO: 90, "....", 180, 0
ECHO: 100, "....", 196.962, -34.7296
ECHO: 110, "....", 206.732, -75.2444
ECHO: 120, "....", 207.846, -120
ECHO: 130, "....", 199.172, -167.125
ECHO: 140, "....", 179.981, -214.492
ECHO: 150, "....", 150, -259.808
ECHO: 160, "....", 109.446, -300.702
ECHO: 170, "....", 59.0404, -334.835
ECHO: 180, "....", 0, -360
ECHO: 190, "....", -65.9863, -374.227
ECHO: 200, "....", -136.808, -375.877
ECHO: 210, "....", -210, -363.731
ECHO: 220, "....", -282.827, -337.06
ECHO: 230, "....", -352.38, -295.682
ECHO: 240, "....", -415.692, -240
ECHO: 250, "....", -469.846, -171.01
ECHO: 260, "....", -512.1, -90.2971
ECHO: 270, "....", -540, 0
ECHO: 280, "....", -551.492, 97.243
ECHO: 290, "....", -545.022, 198.372
ECHO: 300, "....", -519.615, 300
ECHO: 310, "....", -474.948, 398.528
ECHO: 320, "....", -411.384, 490.268
ECHO: 330, "....", -330, 571.577
ECHO: 340, "....", -232.574, 638.991
ECHO: 350, "....", -121.554, 689.365
ECHO: 360, "....", 0, 720
Rendering Polygon Mesh using CGAL...
Geometries in cache: 132
Geometry cache size in bytes: 64848
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Total rendering time: 0 hours, 0 minutes, 0 seconds
Top level object is a 2D object:
Contours: 15
Rendering finished.
```

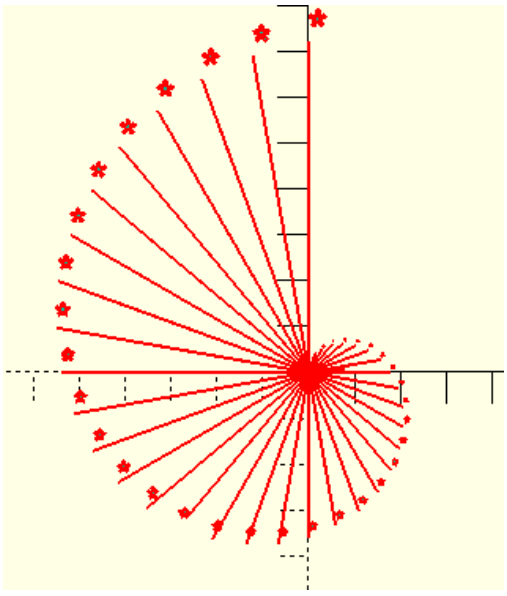
and the graphical display is correct:-



Now the next problem to solve is relocating starts of objects, placing text in the display area, and changing colors. It might help if I read the provided documentation.

Text is positioned using “translate”. The size of the text is varied with the second parameter of “text” in this example and uses “i/5”, so “text” and “translate” solve that problem.

```
// sample 7
st=0;
en=360;
text("      sample 7");
for (i=[st:10:en])
{
    hla=i;
    l=i*2;
    x=l*sin(hla);
    y=l*cos(hla);
    echo (i,,"....",x,y);
    polygon([[0,0],[x,y],[0.1,0.1]]);
    translate([x, y]) {
        text ("*",i/5);
    }
}
```



Now, how does one change color?

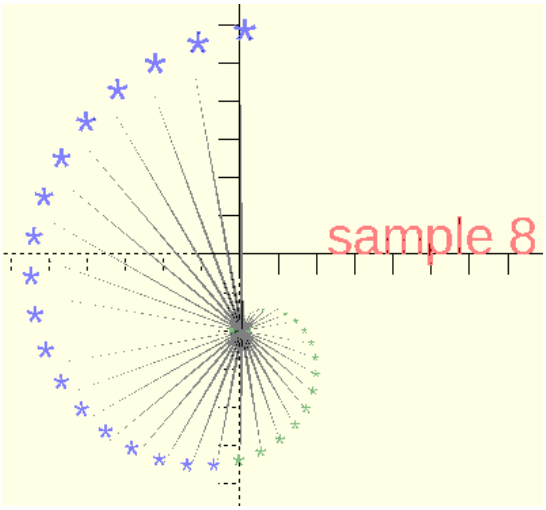
THE USE OF COLOR (DESIGN, PREVIEW SHOWS COLORS)

```
// sample 8
st=0;
en=360;

color ("red") {
    text("      sample 8",100);
}

translate([0, 0-200]) {
    color ("green") {
        circle (30);
    }
}

for (i=[st:10:en])
{
    hla=i;
    l=i*2;
    x=l*sin(hla);
    y=l*cos(hla);
    echo (i,,"....",x,y);

    color ("black") {
        polygon([[0,-200],[x,y-200],[10,-200]]);
    }

    if (i<181){
        color ("green") {
                translate([x-20, y-200-20]) {
                    text ("*",i/3);
                }
        }
    } else
    {
        color ("blue") {
                translate([x-20, y-200-20]) {
                    text ("*",i/3);
                }
        }
    }
}
```

In the above code, the “Y” ordinate is dropped down, and “color” is used. Also, the end point of the lines (polygon) is widened, the lines are of course actually narrow triangles in the above code.

Only “VIEW”, “PREVIEW” shows any color other than the default of red. And after a DESIGN then RENDER, the VIEW then PREVIEW doesn’t show the result unless you do a SAVE. Alternatively, do a DESIGN then PREVIEW.

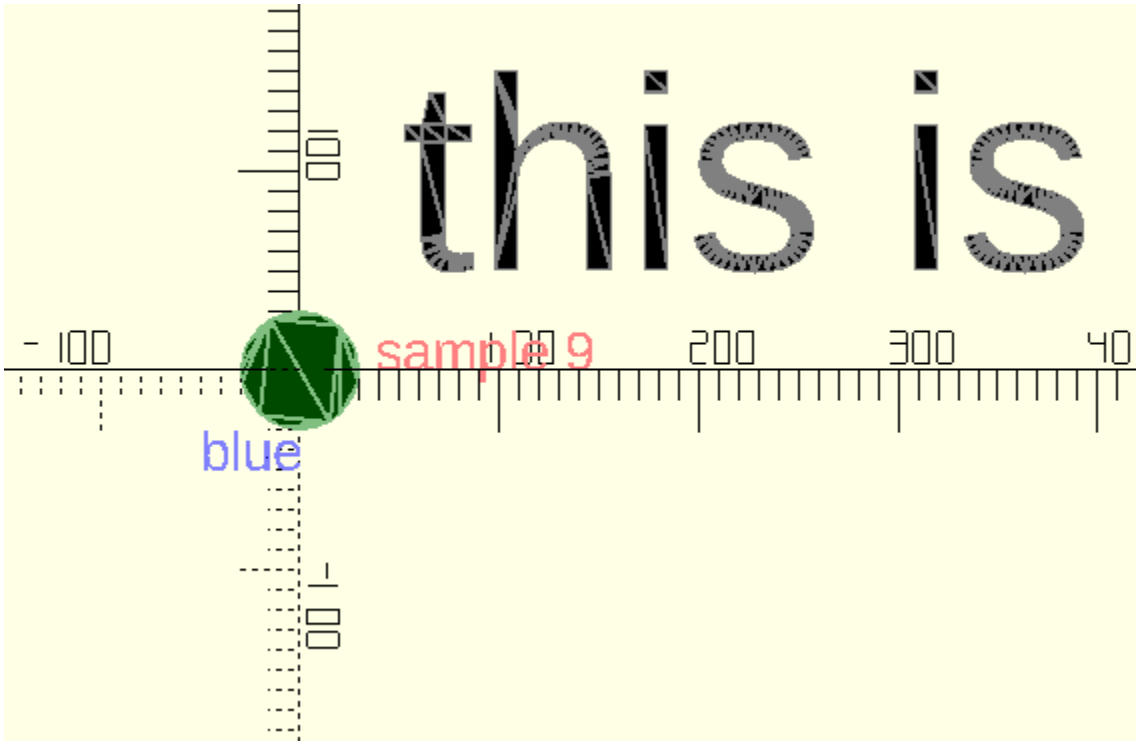
Note green “*” for 0 to 180, blue “*” 181 to 360, and black lines.



ALLOWING VARIABLE DATA TO APPEAR IN A TEXT COMMAND (str)

Below, color is used as well as the “str” feature to display variable text.

```
// sample 9
st=0;
en=360;

color ("red") {
    text("     sample 9",20);
}

translate([0, 0]) {
    color ("green") {
        circle (30);
    }
}

translate([-50, -50]) {
    color ("blue") {
        text("blue",20);
    }
}

jj=10;
translate([ 50,  50]) {
    color ("black") {
        text(str("this is ",jj),100);
    }
}
```



At this point, all the elements of coding for OpenScad are in place for drawing a 2D horizontal dial, so here it is:-

```
// 2d hDial v4
// = = = = = = = = = variables
lat=33.5;
lng=108.2;
ref=105;
// = = = = = = = = =
st=2;
en=23;

translate([20,-30]) {
    color("black") {
        text(str("lat ",lat),6);
    }
}
translate([20,-40]) {
    color("black") {
        text(str("lng ",lng),6);
    }
}
translate([20,-50]) {
    color("black") {
        text(str("ref ",ref),6);
    }
}
translate([-80, 0]) {square(size=[160,2]);}
for (i=[st:1:en]) {
    hla=atan(sin(lat)*tan(15*i-(lng-ref)));
    sz=80;
    x=sz*sin(hla);       // or x=cos…
    y=sz*cos(hla);       // or y=sin…
    //                   // if so, change below code
    echo (lat,i,hla,"....",x,y);

    // AM
    // 0600 to 1200
    if ((i<=12) && (x<0) && (y>0)) {
        //x=-x;    y=-y;
        color("black") {
            polygon([[0,0],[x,y],[.6,.6]]);
            translate([x, y]) {
                color("black"){
                    text(str("*",i),4);
                }
            }
        }
    }
    // 0600 or earlier
    if ((i<=6) && (x>0) && (y>0)) {
        x=-x;    y=-y;
        color("green") {
            polygon([[0,0],[x,y],[.6,.6]]);
            translate([x, y]) {
                color("green"){
                    text(str("*",i),4);
                }
            }
        }
    }

    // PM
    // 1800 or later
    if ((i>=18) && (x<0) && (y>0)) {
```

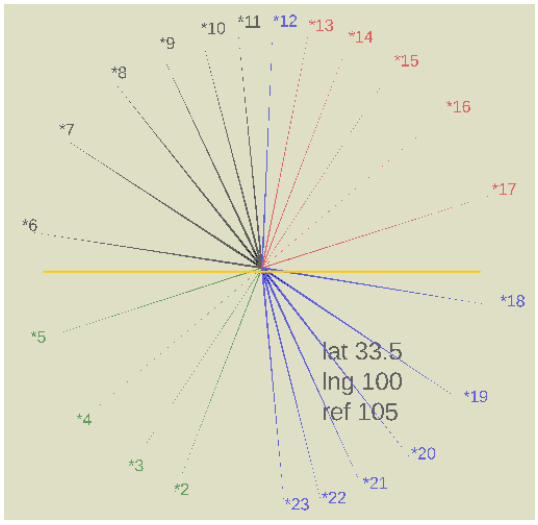
```
            x=-x;    y=-y;
            color("blue") {
                polygon([[0,0],[x,y],[.6,.6]]);
                translate([x, y]) {
                    color("blue"){
                        text(str("*",i),4);
                    }
                }
            }
        }
        // 1200 or later
        if ((i>12) && (i<=18) && (x>0) && (y>0)) {
            color("red") {
                polygon([[0,0],[x,y],[.6,.6]]);
                translate([x, y]) {
                    color("red"){
                        text(str("*",i),4);
                    }
                }
            }
        }

        // special case for i=12 and lng<=ref
            if ((i==12) && (lng<=ref) ) {
            color("blue") {
                polygon([[0,0],[x,y],[.6,.6]]);
                translate([x, y]) {
                    color("blue"){
                        text(str("*",i),4);
                    }
                }
            }
        }


        // special case for i=6 and lng=ref
            if ((i==6) && (lng==ref) ) {
            color("blue") {
                polygon([[0,0],[-x,y],[.6,.6]]);
                translate([-x, y]) {
                    color("blue"){
                        text(str("*",i),4);
                    }
                }
            }
        }

    }   // END OF FOR loop

    // *** END ***
```

Several points.

- The replication of special case code to draw hour lines and text is because openScad scope rules are that a change to a variable within an IF does not change the value outside the IF
- Remember ssignment is done at compile time where possible, unfortunately.

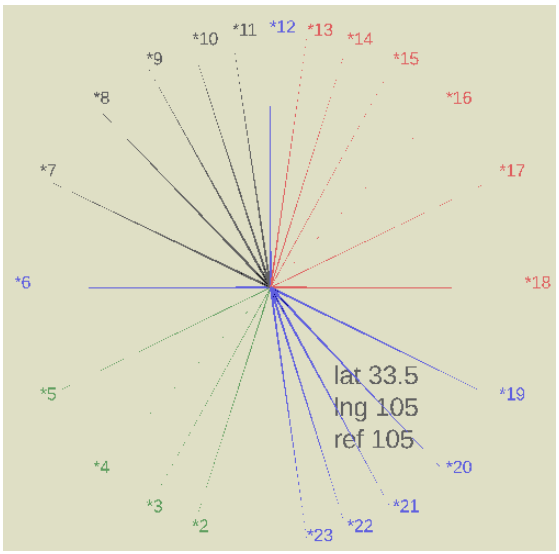
```
// sample 10
st=0;
echo (st);          ECHO: 360       (not 0)
st=360;
echo (st);          ECHO: 360
```

- Data cannot be read from the console, hence lat/long coded in the program

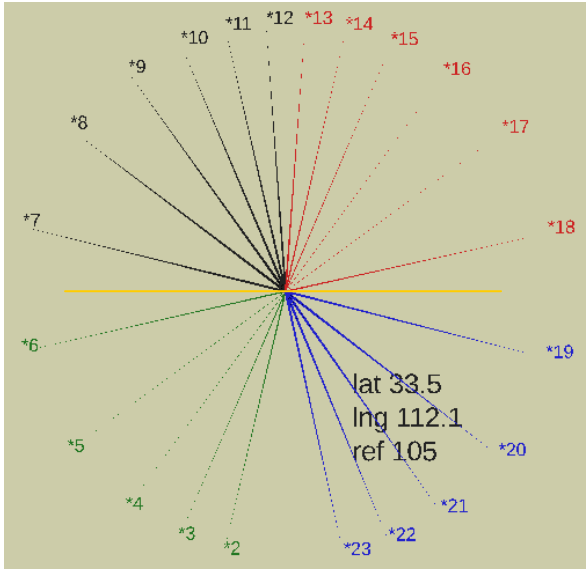


longitude east of reference

no longitude correction



longitude west of reference



HOW TO DRAW A LINE

While a polygon was used earlier for drawing a line, is it the best solution? In the scheme of things, the polygon uses [x,y] and so also does the text function, so they go well together.

Having said that, a line can be drawn with a square that is elongated, and rotated by an angle rather than [x,y].

```
// 2d example 11 ~ lines by rotates square not polygon
st=1;
en=360;
sz=20;

translate([20,-30]) {
    color("black") {
        text(str("start ",st),6);
    }
}
translate([20,-40]) {
    color("black") {
        text(str("end ",en),6);
    }
}

for (i=[st:10:en]) {
        color("black") {
            rotate(a=[0,0,i])
                square(size=[1,sz+(i/20)]);}
}
// *** END ***
```