

OpenSCAD Help

Table of contents

Introduction	12
What's new	13
Getting Started	13
Language Reference	14
Introduction	15
Comments	16
Values and Data Types	16
Numbers	16
Boolean Values	17
Strings	18
Ranges	18
The Undefined Value	19
Variables	19
Undefined variable	20
Scope of variables	20
Variables are set at compile-time, not at run-time	21
Special Variables	22
Vectors	22
Vector Operators (concat & len)	23
Matrix	24
Getting input	24
3D Objects	25
Primitive Solids	25
cube	25
sphere	26
cylinder	27
polyhedron	30
Debugging polyhedra	33
Mis-ordered faces	34
Point repetitions in a polyhedron point list	35
3D to 2D Projection	37
2D Objects	40
square	40
circle	41
ellipse	42
regular polygon	42
polygon	43
Text	46
Using Fonts & Styles	47
Alignment	47
Vertical Alignment	47
Horizontal Alignment	47
3D Text	48
3D to 2D Projection	48
2D to 3D Projection	48
Linear Extrude	48
Usage	48

Twist	48
Center	48
Mesh Refinement	48
Scale	48
Rotate Extrude	48
Parameters	48
Usage	48
Examples	48
Mesh Refinement	49
Extruding a Polygon	49
Description of extrude Parameters	49
Extrude parameters for all extrusion modes	49
Extrude parameters for linear extrusion only	49
Transforms	49
Basic concept	49
Advanced concept	49
scale	49
resize	49
rotate	49
Rotation rule Help	49
translate	49
mirror	50
Function signature:	50
Examples	50
multimatrix	50
More?	50
color	50
function signature	50
Example	50
Example 2	50
offset	50
minkowski	50
hull	50
Combining transformations	51
Boolean Combinations	51
Boolean overview	51
2D examples	51
3D examples	51
union	51
difference	51
difference with multiple children	51
intersection	51
render	51
Other Functions and Operators	51
Condition and Iterator functions	51
For loop	51
Intersection For loop	52
if statement	52
else if	52
Conditional ?	52

Recursive function calls	52
Assign Statement	52
Let Statement	52
Mathematical Operators	52
Scalar Arithmetical Operators	52
Relational Operators	52
Logical Operators	52
Conditional Operator	52
Vector-Number Operator	53
Vector Operators	53
Vector Dot-Product Operator	53
Matrix Multiplication	53
Mathematical Functions	53
Trigonometric Functions	53
cos	53
sin	53
tan	53
acos	53
asin	53
atan	53
atan2	53
Other Mathematical Functions	54
abs	54
ceil	54
concat	54
cross	54
exp	54
floor	54
in	54
len	54
let	54
log	54
lookup	54
max	54
min	55
norm	55
pow	55
rands	55
round	55
sign	55
sqrt	55
Infinities and NaNs	55
String Functions	55
str	55
chr	55
ord	55
Also see search()	55
List Comprehensions	56
Basic Syntax	56
Multiple generator expressions	56

for	56
each	56
if	56
if/else	56
let	56
Nested loops	56
Advanced Examples	56
Generating vertices for a polygon	56
Flattening a nested vector	56
Sorting a vector	56
Selecting elements of a vector	57
Concatenating two vectors	57
Other Language Features	57
Special Variables	57
\$fa, \$fs and \$fn	57
\$t	57
\$vp, \$vpt and \$vpd	57
\$preview	57
Echo Statements	57
Usage examples	57
Rounding examples	57
Small and large Numbers	57
HTML	57
Echo Function	58
Render	58
Surface	58
Text file format	58
Images	58
Examples	58
Search	58
Search Usage	58
Search Arguments	58
Search Usage Examples	58
Index values return as list	59
Search on different column; return index values	60
Search on list of values	61
Search on list of strings	62
Getting the right result	63
OpenSCAD Version	63
parent_module(n) and \$parent_modules	63
assert	63
Example	63
Checking parameters	63
Adding message	63
Using assertions in functions	63
User Defined Functions and Modules	63
Introduction	63
Scope	63
Functions	63
Recursive Functions	63

Function Literals	64
Modules	64
Object modules	64
Operator Modules	64
Children	64
Further Module Examples	64
Recursive Modules	64
Overwriting built-in modules	64
Overwriting built functions	64
Debugging Aids	64
Advanced Concept	64
Background Modifier	64
Debug Modifier	65
Root Modifier	65
Disable Modifier	65
Echo Statements	65
External Libraries and code files	65
Use and Include	65
Directory separators	65
Variables	65
Scope of Variables	65
Overwriting variables	65
Example "Ring-Library"	65
Nested Include and Use	65
Import	65
Parameters	66
Convexity	66
Notes	66
Import DXF	66
Import STL	66
surface	66
Parameters	66
Text file Format	66
Images	66
Examples	66
MCAD Library	66
regular_shapes.scad	68
2D regular shapes	68
regular_polygon(sides, radius)	68
n-gons 2D shapes	68
triangle(radius)	68
pentagon(radius)	68
hexagon(radius)	68
heptagon(radius)	69
octagon(radius)	69
nonagon(radius)	69
decagon(radius)	69
hendecagon(radius)	69
dodecagon(radius)	69
ring(inside_diameter, thickness)	69

ellipse(width, height)	69
egg_outline(width, thickness)	69
3D regular shapes	69
cone	69
oval_prism	69
oval_tube	70
cylinder_tube	70
triangle_prism	70
triangle_tube	70
pentagon_prism	70
pentagon_tube	70
hexagon_prism	70
hexagon_tube	70
heptagon_prism	70
heptagon_tube	70
octagon_prism	70
octagon_tube	70
nonagon_prism	70
decagon_prism	71
hendecagon_prism	71
dodecagon_prism	71
torus	71
torus2	71
oval_torus	71
triangle_pyramid	71
square_pyramid	71
egg	71
involute_gears.scad	71
bevel_gear_pair()	71
bevel_gear()	71
gear()	72
Tests	72
test_gears()	72
test_meshing_double_helix()	72
test_bevel_gear()	72
test_bevel_gear_pair()	72
test_backlash()	72
dotSCAD Library	72
2D modules	72
arc	72
pie	72
rounded_square	72
line2d	73
polyline2d	73
hull_polyline2d	73
hexagons	73
polytransversals	73
multi_line_text	73
voronoi2d	73
3D modules	73

rounded_cube	73
rounded_cylinder	73
crystal_ball	73
line3d	73
polyline3d	73
hull_polyline3d	74
function_grapher	74
sweep	74
loft	74
starburst	74
voronoi3d	74
Transformations	74
along_width	74
hollow_out	74
bend	74
shear	74
2D functions	74
in_shape	75
bijection_offset	75
trim_shape	75
triangulate	75
contours	75
2D/3D functions	75
cross_sections	75
paths2sections	75
path_scaling_sections	75
bezier_surface	75
bezier_smooth	75
midpt_smooth	75
in_polyline	75
Paths	76
arc_path	76
bspline_curve	76
bezier_curve	76
helix	76
golden_spiral	76
archimedean_spiral	76
sphere_spiral	76
torus_knot	76
Extrusion	76
box_extrude	76
ellipse_extrude	76
stereographic_extrude	77
rounded_extrude	77
bend_extrude	77
2D Shape	77
shape_taiwan	77
shape_arc	77
shape_pie	77
shape_circle	77

shape_ellipse	77
shape_square	77
shape_trapezium	77
shape_cyclicpolygon	77
shape_pentagram	77
shape_starburst	78
shape_superformula	78
shape_glue2circles	78
shape_path_extend	78
2D Shape extrusions	78
path_extrude	78
ring_extrude	78
helix_extrude	78
golden_spiral_extrude	78
archimedean_spiral_extrude	78
sphere_spiral_extrude	78
Utils	78
util/sub_str	79
util/split_str	79
util/parse_number	79
util/reverse	79
util/slice	79
util/sort	79
util/rand	79
util/fibseq	79
util/bsearch	79
util/has	79
util/dedup	79
util/flat	79
Matrix	79
matrix/m_cumulate	80
matrix/m_translation	80
matrix/m_rotation	80
matrix/m_scaling	80
matrix/m_mirror	80
matrix/m_shearing	80
Point Transformation	80
ptf/ptf_rotate	80
ptf/ptf_x_twist	80
ptf/ptf_y_twist	80
ptf/ptf_circle	80
ptf/ptf_bend	80
ptf/ptf_ring	81
ptf/ptf_sphere	81
ptf/ptf_torus	81
Turtle	81
turtle/turtle2d	81
turtle/turtle3d	81
turtle/t2d	81
turtle/t3d	81

Pixel	81
pixel/px_line	81
pixel/px_polyline	81
pixel/px_circle	81
pixel/px_cylinder	81
pixel/px_sphere	82
pixel/px_polygon	82
pixel/px_from	82
pixel/px_ascii	82
pixel/px_gray	82
Part	82
part/connector_jpg	82
part/cone	82
part/join_T	82
Surface	82
surface/sf_square	82
surface/sf_bend	82
surface/sf_ring	83
surface/sf_sphere	83
surface/sf_torus	83
surface/sf_solidity	83
Noise	83
noise/nz_perlin1	83
noise/nz_perlin1s	83
noise/nz_perlin2	83
noise/nz_perlin2s	83
noise/nz_perlin3	83
noise/nz_perlin3s	83
noise/nz_worley2	83
noise/nz_worley2s	83
noise/nz_worley3	84
noise/nz_worley3s	84
noise/nz_cell	84
BOSL Library	84
Commonly Used	84
transforms.scad	84
Translations	84
move()	84
xmove()	86
ymove()	87
zmove()	87
left()	87
right()	87
fwd() / forward()	87
back()	87
down()	87
up()	87
shapes.scad	87
masks.scad	87
threading.scad	88

paths.scad	88
beziers.scad	88
Standard Parts	88
involute_gears.scad	88
joiners.scad	88
sliders.scad	88
metric_screws.scad	88
linear_bearings.scad	88
nema_steppers.scad	88
phillips_drive.scad	88
torx_drive.scad	88
wiring_scad	89
Miscellaneous	89
constants.scad	89
math.scad	89
convex_hull.scad	89
quaternions.scad	89
triangulation.scad	89
debug.scad	89
Nut Job Library	89

Introduction

OpenSCADHelp

OpenSCAD Help Project is a compilation of help information about OpenSCAD and some libraries.

Any programming language has two parts to learn. The language syntax and himself instructions and the common libraries of constructed instructions.

OpenSCAD Help is a compilation all this help in a unique site, with a unique form of search and found help about almost the most important things of OpenSCAD.

The libraries included in this project are the next:

MCAD

Components commonly used in designing and mocking up mechanical designs

<https://github.com/openscad/MCAD>

dotSCAD

Reduce the burden of 3D modeling in mathematics. <https://github.com/JustinSDK/dotSCAD>

BOSL

The Belfry OpenScad Library - A library of tools, shapes, and helpers to make OpenSCAD easier to use. <https://github.com/revarbat/BOSL>

Syntax

```
var = value;
module name(_) { _ }
name();
function name(_) = _
name();
include <...scad>
use <...scad>
```

2D

```
circle(radius)
square(size,center)
square([width,height],center)
polygon([points])
polygon([points],[paths])
```

3D

```
sphere(radius)
cube(size)
cube([width,height,depth])
cylinder(h,r,center)
cylinder(h,r1,r2,center)
polyhedron(points, triangles, convexity)
```

Transformations

```
translate([x,y,z])
rotate([x,y,z])
scale([x,y,z])
mirror([x,y,z])
multmatrix(m)
color("colorname")
color([r, g, b, a])
hull()
minkowski()
```

Boolean operations

```
union()
difference()
intersection()
```

Modifier Characters

```
*  disable
!  show only
#  highlight
%  transparent
```

Mathematical

```
abs
sign
acos
asin
atan
atan2
sin
cos
floor
round
ceil
ln
len
log
lookup
min
max
pow
sqrt
exp
rand
```

Other

```
echo(_)
str(_)
for (i = [start:end]) { _ }
for (i = [start:step:end]) { _ }
for (i = [...]) { _ }
intersection_for(i = [start:end]) { _ }
intersection_for(i = [start:step:end]) { _ }
intersection_for(i = [...]) { _ }
if (..) { _ }
assign (..) { _ }
search(..)
import("...stl")
linear_extrude(height,center,convexity,twist,slices)
rotate_extrude(convexity)
surface(file = "...dat",center,convexity)
projection(cut)
render(convexity)
```

Special variables

```
$fa minimum angle
$fs minimum size
$fn number of fragments
$t animation step
```

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

What's new

2020 April 22th.

- First version of PDF help file.
- First version of HTML help site.
- First version of EPUB help file.

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

Getting Started

OpenSCAD is a software for creating solid 3D CAD objects.

It is **free software** and available for **GNU/Linux**, Microsoft Windows and Mac OS X.

OpenSCAD focuses on the **CAD** aspects as oposition to artistic aspects. So it might be the application you are looking for when you are planning to create 3D models of machine parts.

OpenSCAD is not an interactive modeler. It is something like a 2D/3D-compiler that reads in a program file that describes the object and renders the model from this file. This gives you (the designer) full control over the modeling process. This enables you to easily change any step in the modeling process and make designs that are defined by configurable parameters.

OpenSCAD has two main operating modes:

- Preview is relatively fast using **3D graphics** and the computer's **GPU**, but is an approximation of the model and can produce **artifacts**; Preview uses **OpenCSG** and **OpenGL**.
- Render generates exact geometry and a fully **tessellated mesh**. It is not an approximation and as such it is often a lengthy process, taking minutes or hours for larger designs. Render uses **CGAL** as its geometry engine.

OpenSCAD provides two types of 3D modelling:

- **Constructive Solid Geometry (CSG)**
- extrusion of 2D primitives into 3D space.

OpenSCAD can be downloaded from <https://www.openscad.org/>. **OpenSCAD** is a software for creating solid 3D CAD objects.

It is **free software** and available for **GNU/Linux**, Microsoft Windows and Mac OS X.

Unlike most free software for creating 3D models (such as the well-known application **Blender**), OpenSCAD does not focus on the artistic aspects of 3D modelling, but instead focuses on the **CAD** aspects. So it might be the application you are looking for when you are planning to create 3D models of machine parts, but probably is not what you are looking for when you are more interested in creating computer-animated movies or organic life-like models.

OpenSCAD, unlike many CAD products, is not an interactive modeler. Instead it is something like a 2D/3D-compiler that reads in a program file that describes the object and renders the model from this file. This gives you (the designer) full control over the modelling process. This enables you to easily change any step in the modelling process and make designs that are defined by configurable parameters.

OpenSCAD has two main operating modes, *Preview* and *Render*. Preview is relatively fast using **3D graphics** and the **computer's GPU**, but is an approximation of the model and can produce **artifacts**; Preview uses **OpenCSG** and **OpenGL**. Render generates exact geometry and a fully **tessellated mesh**. It is not an approximation and as such it is often a lengthy process, taking minutes or hours for larger designs. Render uses **CGAL** as its geometry engine.

OpenSCAD provides two types of 3D modelling:

- **Constructive Solid Geometry (CSG)**
- extrusion of 2D primitives into 3D space.

Autocad DXF files are used as the data exchange format for 2D outlines. In addition to 2D paths for extrusion it is also possible to read design parameters from DXF files. Besides DXF files, OpenSCAD can read and create 3D models in the **STL** and **OFF** file formats.

OpenSCAD can be downloaded from <https://www.openscad.org/>. More information is available on the **mailing list**. OpenSCAD can also be tried online at <http://openscad.net/>, which is a partial port of OpenSCAD for the web. More information is available on the **mailing list**. OpenSCAD can also be tried online at <http://openscad.net/>, which is a partial port of OpenSCAD for the web.

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

Language Reference

Language Reference

Table of Contents

1. [The OpenSCAD Language - General](#)
2. [3D Objects, Projection](#)
3. [2D Objects, Primitives, Text, Extrusion to 3D](#)
4. [Transformations](#)
5. [Boolean operations](#)
6. [Conditional and iterator functions](#)
7. [Mathematical operators](#)

8. [Mathematical functions](#)
9. [String functions](#)
10. **Type test functions**
11. [List comprehensions](#)
12. [Other language features](#)
13. [User defined functions and modules](#)
14. [Debugging aids - modifier characters](#)
15. **Importing geometry, Exporting geometry**

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Introduction

Introduction

OpenSCAD is a 2D/3D and solid modeling program which is based on a Functional programming language used to create models that are previewed on the screen, and rendered into 3D mesh which allows the model to be exported in a variety of 2D/3D file formats.

A script in the OpenSCAD language is used to create 2D or 3D models. This script is a free format list of action statements.

```
object();
variable = value;
operator() action();
operator() {action(); action();}
operator() operator() {action(); action();}
operator() {operator() action();
              operator() {action(); action();}}
```

Objects

Objects are the building blocks for models, created by 2D and 3D primitives. Objects end in a semicolon ';'.

Actions

Action statements include creating objects using primitives and assigning values to variables. Action statements also end in a semicolon ';'.

Operators

Operators, or transformations, modify the location, color and other properties of objects. Operators use braces '{}' when their scope covers more than one action. More than one operator may be used for the same action or group of actions. Multiple operators are processed Right to Left, that is, the operator closest to the action is processed first. Operators do not end in semicolons ';', but the individual actions they contain do.

Examples

```
cube(5);
x = 4 + y;
rotate(40) square(5,10);
translate([10,5]) { circle(5); square(4); }
rotate(60) color("red") { circle(5); square(4); }
```

```
color("blue") { translate([5,3,0]) sphere(5); rotate([45,0,45]) { cylinder(10); cube([5,6,7]); } }
```

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

Comments

Comments

Comments are a way of leaving notes within the script, or code, (either to yourself or to future programmers) describing how the code works, or what it does. Comments are not evaluated by the compiler, and should not be used to describe self-evident code.

OpenSCAD uses C++-style comments:

```
// This is a comment

myvar = 10; // The rest of the line is a comment

/*
Multi-line comments
can span multiple lines.
*/
```

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

Values and Data Types

Values and Data Types

A value in OpenSCAD is either a Number (like 42), a Boolean (like true), a String (like "foo"), a Range (like [0: 1: 10]), a Vector (like [1,2,3]), or the Undefined value (undef).

Values can be stored in variables, passed as function arguments, and returned as function results.

[OpenSCAD is a dynamically typed language with a fixed set of data types. There are no type names, and no user defined types. Functions are not values. In fact, variables and functions occupy disjoint namespaces.]

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

Numbers

Numbers

Numbers are the most important type of value in OpenSCAD, and they are written in the familiar decimal notation used in other languages. Eg, -1, 42, 0.5, 2.99792458e+8.

[OpenSCAD does not support octal or hexadecimal notation for numbers.]

In addition to decimal numerals, the following names for special numbers are defined:

- **PI**

OpenSCAD has only a single kind of number, which is a 64 bit IEEE floating point number.

[OpenSCAD does not distinguish integers and floating point numbers as two different types, nor does it support complex numbers.]

Because OpenSCAD uses the IEEE floating point standard, there are a few deviations from the behavior of numbers in mathematics:

- We use binary floating point. A fractional number is not represented exactly unless the denominator is a power of 2. For example, 0.2 (2/10) does not have an exact internal representation, but 0.25 (1/4) and 0.125 (1/8) are represented exactly.
- The largest representable number is about 1e308. If a numeric result is too large, then the result can be infinity (printed as `inf` by `echo`).
- The smallest representable number is about -1e308. If a numeric result is too small, then the result can be -infinity (printed as `-inf` by `echo`).
- If a numeric result is invalid, then the result can be Not A Number (printed as **nan** by `echo`).
- If a non-zero numeric result is too close to zero to be representable, then the result is -0 if the result is negative, otherwise it is 0. Zero (0) and negative zero (-0) are treated as two distinct numbers by some of the math operations, and are printed differently by `'echo'`, although they compare as equal.

The constants **'inf'** and **'nan'** are not supported as numeric constants by OpenSCAD, even though you can compute numbers that are printed this way by `'echo'`. You can define variables with these values by using:

```
inf = 1e200 * 1e200;
nan = 0 / 0;
echo(inf, nan);
```

The value `'nan'` is the only OpenSCAD value that is not equal to any other value, including itself. Although you can test if a variable `'x'` has the undefined value using `'x == undef'`, you can't use `'x == 0/0'` to test if `x` is Not A Number. Instead, you must use `'x != x'` to test if `x` is `nan`.

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

Boolean Values

Boolean Values

Booleans are truth values. There are two Boolean values, namely `true` and `false`. A Boolean is passed as the argument to conditional statement `'if()'`, conditional operator `'?:'`, and logical operators `'!'` (not), `'&&'` (and), and `'||'` (or). In all of these contexts, you can actually pass any quantity. Most values are converted to `'true'` in a Boolean context, the values that count as `'false'` are:

- `false`
- `0` and `-0`
- `" "`

- `[]`
- `undef`

Note that `"false"` (the string), `[0]` (a numeric vector), `[[]]` (a vector containing an empty vector), `[false]` (a vector containing the Boolean value false) and `0/0` (Not A Number) all count as true.

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

Strings

Strings

A string is a sequence of zero or more unicode characters. String values are used to specify file names when importing a file, and to display text for debugging purposes when using `echo()`.

Strings can also be used with the [text\(\) primitive](#).

A string literal is written as a sequence of characters enclosed in quotation marks `"`, like this: `"` (an empty string), or `"this is a string"`.

To include a `"` character in a string literal, use `\`. To include a `\` character in a string literal, use `\\`. The following escape sequences beginning with `\` can be used within string literals:

- `\"` → `"`
- `\\` → `\`
- `\t` → tab
- `\n` → newline
- `\r` → carriage return
- `\u03a9` → `Ω` - see `text()` for further information on unicode characters

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Ranges

Ranges

Ranges are used by [for\(\) loops](#) and [children\(\)](#). They have 2 varieties:

```
[<start>:<end>]
[<start>:<increment>:<end>]
```

Although enclosed in square brackets `[]`, they are not vectors. They use colons `:` for separators rather than commas.

```
r1 = [0:10];
r2 = [0.5:2.5:20];
echo(r1); // ECHO: [0: 1: 10]
echo(r2); // ECHO: [0.5: 2.5: 20]
```

You should avoid step values that cannot be represented exactly as binary floating point numbers. Integers are okay, as are fractional values whose denominator is a power of two. For example, 0.25 (1/4) and 0.125 (1/8) are safe, but 0.2 (2/10) should be avoided. The problem with these step values is that your range may have too many or too few elements, due to inexact arithmetic.

A missing *<increment>* defaults to 1.

A range in the form [*<start>*:*<end>*] with *<start>* greater than *<end>* generates a warning and is equivalent to [*<end>*: 1: *<start>*].

A range in the form [*<start>*:1:*<end>*] with *<start>* greater than *<end>* does not generate a warning and is equivalent to [].

The *<increment>* in a range may be negative.

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

The Undefined Value

The Undefined Value

The undefined value is a special value written as **undef**. It's the initial value of a variable that hasn't been assigned a value, and it is often returned as a result by functions or operations that are passed illegal arguments. Finally, **undef can be used as a null value**, equivalent to `null` or `NULL` in other programming languages.

All arithmetic expressions containing `undef` values evaluate as `undef`.

In logical expressions, `undef` is equivalent to `false`.

Relational operator expressions with `undef` evaluate as `false` except for `undef==undef` which is `true`.

Note that numeric operations may also return 'nan' (not-a-number) to indicate an illegal argument. For example, `0/false` is `undef`, but `0/0` is 'nan'. Relational operators like `<` and `>` return `false` if passed illegal arguments. Although `undef` is a language value, 'nan' is not.

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Variables

Variables

OpenSCAD variables are created by a statement with a name or **identifier**, assignment via an expression and a semicolon. The role of arrays, found in many imperative languages, is handled in OpenSCAD via vectors.

```
var = 25;
xx = 1.25 * cos(50);
y = 2 * xx + var;
logic = true;
MyString = "This is a string";
a_vector = [1,2,3];
rr = a_vector[2]; // member of vector
rangel = [-1.5:0.5:3]; // for() loop range
xx = [0:5]; // alternate for() loop range
```

OpenSCAD is a **Functional** programming language, as such **variables** are bound to expressions and keep a single value during their entire lifetime due to the requirements of **referential transparency**. In **imperative languages**, such as C, the same behavior is seen as constants,

which are typically contrasted with normal variables.

In other words OpenSCAD variables are more like constants, but with an important difference. If variables are assigned a value multiple times, only the last assigned value is used in all places in the code.

See further discussion at [Variables are set at compile-time, not run-time](#). This behavior is due to the need to supply variable input on the [command line](#), via the use of `-D variable=value` option. OpenSCAD currently places that assignment at the end of the source code, and thus must allow a variable's value to be changed for this purpose.

Values cannot be modified during run time; all variables are effectively constants that do not change. Each variable retains its last assigned value at compile time, in line with [Functional](#) programming languages. Unlike [Imperative](#) languages, such as C, OpenSCAD is not an iterative language, and as such **the concept of $x = x + 1$ is not valid**. Understanding this concept leads to understanding the beauty of OpenSCAD.

Variables can be assigned in any scope. Note that assignments are only valid within the scope in which they are defined - you are still not allowed to leak values to an outer scope. See [Scope of variables](#) for more details.

```
a=0;
if (a==0) {
    a=1; // but the value a=1 is confined to within the braces {}
}
```

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

Undefined variable

Undefined variable

A non assigned variable has the special value **undef**. It could be tested in conditional expression, and returned by a function.

```
Example
echo("Variable a is ", a); // Variable a is undef
if (a == undef) {
    echo("Variable a is tested undefined"); // Variable a is tested
    undefined
}
```

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Scope of variables

Scope of variables

When operators such as `translate()` and `color()` need to encompass more than one action (actions end in `;`), braces `{}` are needed 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. 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.

```

// scope 1
a = 6; // create a
echo(a, b); // 6, undef
translate([5, 0, 0]){ // scope 1.1
  a = 10;
  b = 16; // create b
  echo(a, b); // 100, 16 a=10; was overridden by
later a = 100;
  color("blue") { // scope 1.1.1
    echo(a, b); // 100, 20
    cube();
    b=20;
  } // back to 1.1
  echo(a, b); // 100, 16
  a = 100; // override a in 1.1
} // back to 1
echo(a, b); // 6, undef
color("red"){ // scope 1.2
  cube();
  echo(a,b); // 6, undef
} // back to 1
echo(a, b); // 6, undef

```

//In this example, scopes 1 and 1.1 are outer scopes to 1.1.1 but 1.2 is not.

Anonymous scopes are not considered scopes:

```
{ angle = 45; } rotate(angle) square(10);
```

For() loops are not an exception to the rule about variables having only one value within a scope. A copy of loop contents is created for each pass. Each pass is given its own scope, allowing any variables to have unique values for that pass. No, you still can't do `a=a+1`;

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

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

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

Because OpenSCAD calculates its variable values at compile-time, not run-time, **the last variable assignment, within a scope 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.

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

Special Variables

Special Variables

Special variables provide an alternate means of passing arguments to modules and functions. **All variables starting with a '\$' are special variables**, similar to special variables in lisp. As such they are more dynamic than regular variables. (for more details see [Other Language Features](#))

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Vectors

Vectors

A vector is a sequence of zero or more OpenSCAD values. Vectors are a collection (or list or table) of numeric or boolean values, variables, vectors, strings or any combination thereof. They can also be expressions which evaluate to one of these.

Vectors handle the role of arrays found in many imperative languages. The information here also applies to lists and tables which use vectors for their data.

A vector has square brackets, [] enclosing zero or more items (elements or members), separated by commas. A vector can contain vectors, which contain vectors, etc.

examples

```
[1, 2, 3]
[a, 5, b]
[]
[5.643]
["a", "b", "string"]
[[1, r], [x, y, z, 4, 5]]
[3, 5, [6,7], [[8, 9], [10, [11, 12], 13], c, "string"]
[4/3, 6*1.5, cos(60)]
```

use in OpenSCAD:

```
cube( [width, depth, height]);    // optional spaces shown for clarity
translate( [x, y, z] )
polygon( [ [x0, y0], [x1, y1], [x2, y2] ] );
```

creation

Vectors are created by writing the list of elements, separated by commas, and enclosed in

square brackets. Variables are replaced by their values.

```
cube([10, 15, 20]);
a1 = [1, 2, 3];
a2 = [4, 5];
a3 = [6, 7, 8, 9];
b = [a1, a2, a3]; // [ [1,2,3], [4,5], [6,7,8,9] ] note increased nesting depth
```

elements within vectors

Elements within vectors are numbered from 0 to n-1 where n is the length returned by `len()`. Address elements within vectors with the following notation:

```
e[5]           // element no 5 (sixth) at 1st nesting level
e[5][2]        // element 2 of element 5 2nd nesting level
e[5][2][0]     // element 0 of 2 of 5 3rd nesting level
e[5][2][0][1] // element 1 of 0 of 2 of 5 4th nesting level
```

example elements with lengths from `len()`

```
e = [ [1], [], [3, 4, 5], "string", "x", [[10, 11], [12, 13, 14], [[15, 16], [17]]] ]; // length 6
```

address	length	element
e[0]	1	[1]
e[1]	0	[]
e[5]	3	[[10,11], [12,13,14], [[15,16],[17]]]
e[5][1]	3	[12, 13, 14]
e[5][2]	2	[[15, 16], [17]]
e[5][2][0]	2	[15, 16]
e[5][2][0][1]	undef	16
e[3]	6	"string"
e[3][2]	1	"r"
s = [2,0,5];	a = 2;	
s[a]	undef	5
e[s[a]]	3	[[10, 11], [12, 13, 14], [[15, 16], [17]]]

alternate dot notation

The first three elements of a vector can be accessed with an alternate dot notation:

```
e.x //equivalent to e[0]
e.y //equivalent to e[1]
e.z //equivalent to e[2]
```

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Vector Operators (concat & len)

Vector Operators (concat & len)

concat

`concat()` combines the elements of 2 or more vectors into a single vector. No change in nesting level is made.

```
vector1 = [1, 2, 3]; vector2 = [4]; vector3 = [5,6];
new_vector = concat(vector1, vector2, vector3); // [1, 2, 3, 4, 5, 6]

string_vector = concat("abc", "def"); // ["abc", "def"]
one_string = str(string_vector[0], string_vector[1]); // "abcdef"
```

len

`len()` returns the length of vectors or strings. Indices of elements are from [0] to [length-1].

vector

- Returns the number of elements at this level.
- Single values, which are **not** vectors, return **undef**.

string

- Returns the number of characters in string.

```
a = [1, 2, 3]; echo(len(a)); // 3
```

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Matrix

Matrix

A matrix is a vector of vectors.

Example which defines a 2D rotation matrix

```
mr = [
    [cos(angle), -sin(angle)],
    [sin(angle), cos(angle)]
];
```

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

Getting input

Getting input

Now we have variables, it would be nice to be able to get input into them instead of setting the values from code. There are a few functions to read data from DXF files, or you can set a variable with the -D switch on the command line.

Getting a point from a drawing

Getting a point is useful for reading an origin point in a 2D view in a technical drawing. The function `dxf_cross` reads the intersection of two lines on a layer you specify and returns the intersection point. This means that the point must be given with two lines in the DXF file, and not a point entity.

```
OriginPoint = dxf_cross(file = "drawing.dxf",
```



```
layer = "SCAD.Origin",
origin = [0, 0], scale = 1);
```

Getting a dimension value

You can read dimensions from a technical drawing. This can be useful to read a rotation angle, an extrusion height, or spacing between parts. In the drawing, create a dimension that does not show the dimension value, but an identifier. To read the value, you specify this identifier from your program:

```
TotalWidth = dxf_dim(file = "drawing.dxf",
                        name = "TotalWidth",
                        layer = "SCAD.Origin",
                        origin = [0, 0],
                        scale = 1);
```

For a nice example of both functions, see Example009 and the image on the [homepage of OpenSCAD](#).

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

3D Objects

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

Primitive Solids

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

cube

cube

Creates a cube in the first octant. When center is true, the cube is centered on the origin. Argument names are optional if given in the order shown here.

```
cube(size = [x,y,z], center = true/false); cube(size = x , center
= true/false);
```

parameters:

size

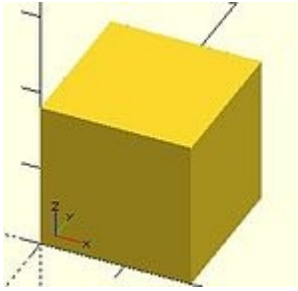
- single value, cube with all sides this length
- 3 value array [x,y,z], cube with dimensions x, y and z.

center

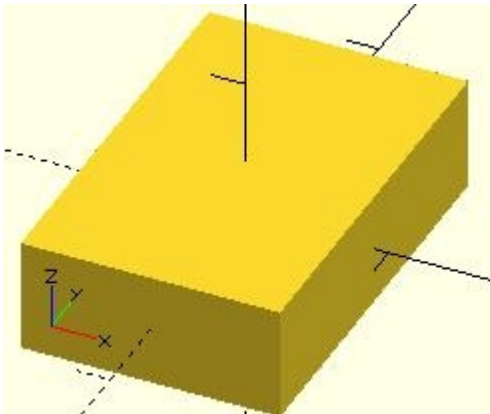
- false** (default), 1st (positive) octant, one corner at (0,0,0)
- true**, cube is centered at (0,0,0)

```
default values: cube(); yields: cube(size = [1, 1, 1], center =
false);
```

examples:



```
equivalent scripts for this example cube(size = 18); cube(18);
cube([18,18,18]); . cube(18,false); cube([18,18,18],false);
cube([18,18,18],center=false); cube(size = [18,18,18], center =
false); cube(center = false,size = [18,18,18] );
```



```
equivalent scripts for this example cube([18,28,8],true);
box=[18,28,8];cube(box,true);
```

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

sphere

sphere

Creates a sphere at the origin of the coordinate system. The *r* argument name is optional. To use *d* instead of *r*, *d* must be named.

Parameters

r

Radius. This is the radius of the sphere. The resolution of the sphere is based on the size of the sphere and the *\$fa*, *\$fs* and *\$fn* variables. For more information on these special variables look at: [OpenSCAD_User_Manual/Other_Language_Features](#)

d

Diameter. This is the diameter of the sphere.

\$fa

Fragment angle in degrees

\$fs

Fragment size in mm

\$fn

Resolution

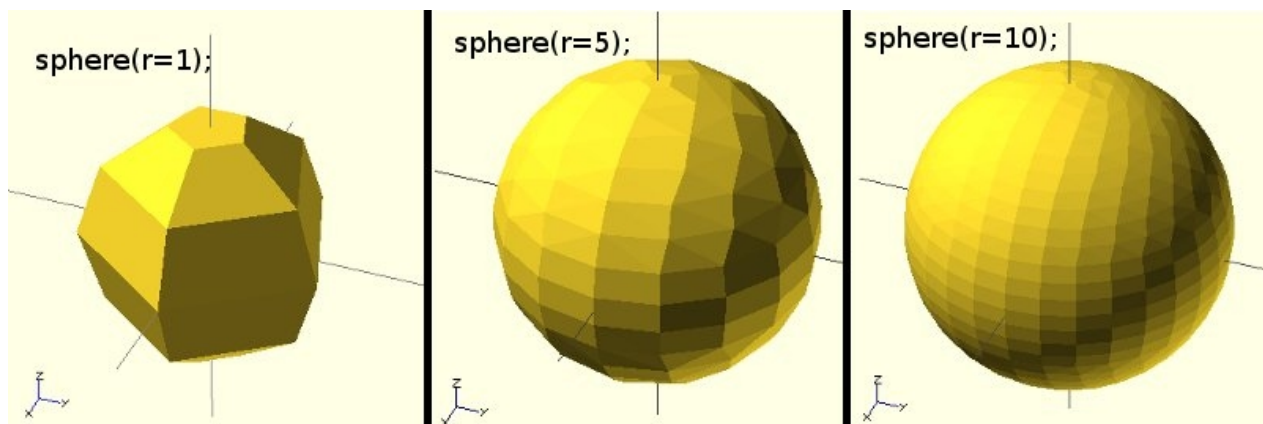
```
default values: sphere(); yields: sphere($fn = 0, $fa = 12, $fs = 2, r = 1);
```

Usage Examples

```
sphere(r = 1); sphere(r = 5); sphere(r = 10); sphere(d = 2);  
sphere(d = 10); sphere(d = 20);
```

```
// this creates a high resolution sphere with a 2mm radius  
sphere(2, $fn=100);
```

```
// also creates a 2mm high resolution sphere but this one // does  
not have as many small triangles on the poles of the sphere  
sphere(2, $fa=5, $fs=0.1);
```



Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

cylinder

cylinder

Creates a cylinder or cone centered about the z axis. When center is true, it is also centered vertically along the z axis.

Parameter names are optional if given in the order shown here. If a parameter is named, all following parameters must also be named.

NOTE: If r, d, d1 or d2 are used they must be named.

```
cylinder(h = height, r1 = BottomRadius, r2 = TopRadius, center =
```

```
true/false);
```

Parameters

h : height of the cylinder or cone

r : radius of cylinder. $r1 = r2 = r$.

r1 : radius, bottom of cone.

r2 : radius, top of cone.

d : diameter of cylinder. $r1 = r2 = d / 2$. **[Note: Requires version 2014.03]**

d1 : diameter, bottom of cone. $r1 = d1 / 2$. **[Note: Requires version 2014.03]**

d2 : diameter, top of cone. $r2 = d2 / 2$. **[Note: Requires version 2014.03]**

center

false (default), z ranges from 0 to h

true, z ranges from -h/2 to +h/2

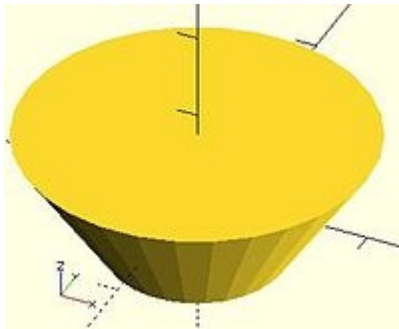
\$fa : minimum angle (in degrees) of each fragment.

\$fs : minimum circumferential length of each fragment.

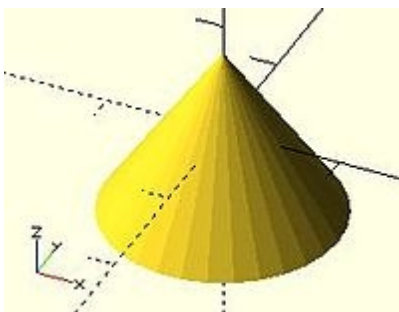
\$fn : **fixed** number of fragments in 360 degrees. Values of 3 or more override \$fa and \$fs

\$fa, \$fs and \$fn must be named. [click here for more details](#),.

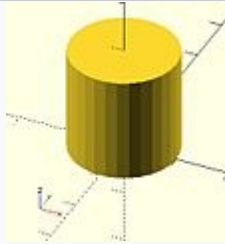
```
defaults: cylinder(); yields: cylinder($fn = 0, $fa = 12, $fs = 2,
h = 1, r1 = 1, r2 = 1, center = false);
```



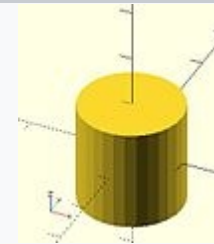
```
equivalent scripts cylinder(h=15, r1=9.5, r2=19.5, center=false);
cylinder( 15, 9.5, 19.5, false); cylinder( 15, 9.5, 19.5);
cylinder( 15, 9.5, d2=39 ); cylinder( 15, d1=19, d2=39 );
cylinder( 15, d1=19, r2=19.5);
```



```
equivalent scripts cylinder(h=15, r1=10, r2=0, center=true);
cylinder( 15, 10, 0, true); cylinder(h=15, d1=20, d2=0,
center=true);
```



center = false



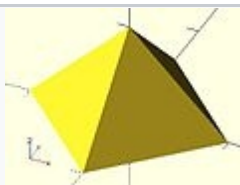
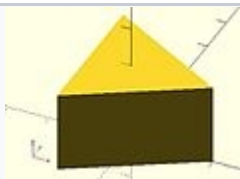
center = true

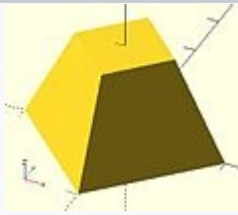
```
equivalent scripts cylinder(h=20, r=10, center=true);
cylinder( 20, 10, 10,true); cylinder( 20, d=20, center=true);
cylinder( 20,r1=10, d2=20, center=true); cylinder( 20,r1=10,
d2=2*10, center=true);
```

use of \$fn

Larger values of \$fn create smoother, more circular, surfaces at the cost of longer rendering time. Some use medium values during development for the faster rendering, then change to a larger value for the final F6 rendering.

However, use of small values can produce some interesting non circular objects. A few examples are show here:

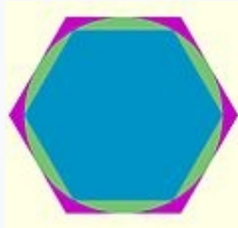




```
scripts for these examples cylinder(20,20,20,$fn=3);
cylinder(20,20,00,$fn=4); cylinder(20,20,10,$fn=4);
```

undersized holes

Using `cylinder()` with `difference()` to place holes in objects creates undersized holes. This is because circular paths are approximated with polygons inscribed within in a circle. The points of the polygon are on the circle, but straight lines between are inside. To have all of the hole larger than the true circle, the polygon must lie wholly outside of the circle (circumscribed). [Modules for circumscribed holes](#)



Notes on accuracy Circle objects are approximated. The algorithm for doing this matters when you want 3d printed holes to be the right size. Current behavior is [illustrated in a diagram](#) . Discussion regarding optionally changing this behavior happening in a [Pull Request](#)

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

polyhedron

polyhedron

A polyhedron is the most general 3D primitive solid. It can be used to create any regular or irregular shape including those with concave as well as convex features. Curved surfaces are approximated by a series of flat surfaces.

```
polyhedron( points = [ [X0, Y0, Z0], [X1, Y1, Z1], ... ], triangles
= [ [P0, P1, P2], ... ], convexity = N); // before 2014.03
polyhedron( points = [ [X0, Y0, Z0], [X1, Y1, Z1], ... ], faces =
[ [P0, P1, P2, P3, ...], ... ], convexity = N); // 2014.03 & later
```

Parameters

points

Vector of 3d points or vertices. Each point is in turn a vector, `[x,y,z]`, of its coordinates.

Points may be defined in any order. N points are referenced, in the order defined, as 0 to N-1.

triangles [**Deprecated:** *triangles will be removed in future releases. Use **faces** parameter instead*]

Vector of faces that collectively enclose the solid. Each face is a vector containing the indices (0 based) of 3 points from the points vector.

faces [**Note:** *Requires version 2014.03*]

Vector of faces that collectively enclose the solid. Each face is a vector containing the indices (0 based) of 3 or more points from the points vector.

Faces may be defined in any order. Define enough faces to fully enclose the solid, with no overlap.

If points that describe a single face are not on the same plane, the face is automatically split into triangles as needed.

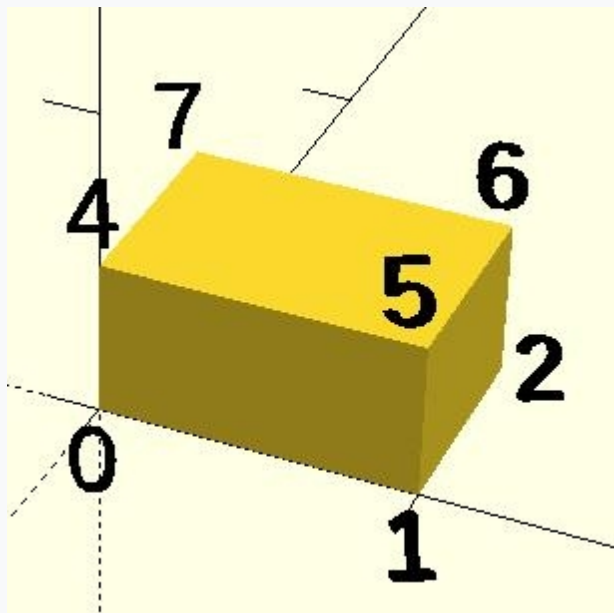
convexity

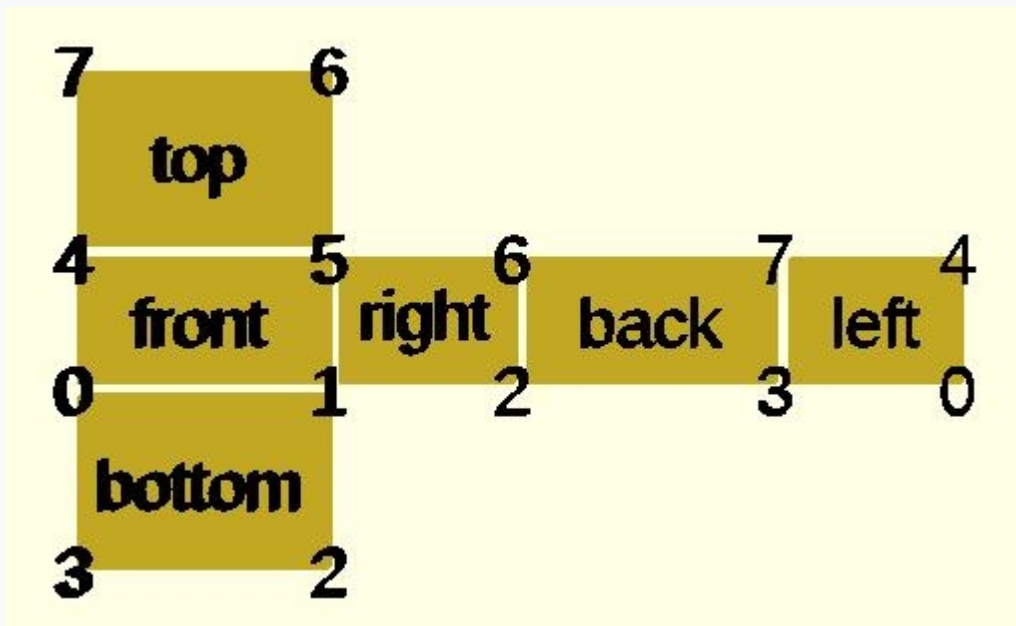
Integer. The convexity parameter specifies the maximum number of faces a ray intersecting the object might penetrate. This parameter is needed only for correct display of the object in OpenCSG preview mode. It has no effect on the polyhedron rendering. For display problems, setting it to 10 should work fine for most cases.

```
default values: polyhedron(); yields: polyhedron(points = undef,
faces = undef, convexity = 1);
```

All faces must have points ordered in the same direction . OpenSCAD prefers **clockwise** when looking at each face from outside **inward**. The back is viewed from the back, the bottom from the bottom, etc.

Example 1 Using polyhedron to generate cube([10, 7, 5]);



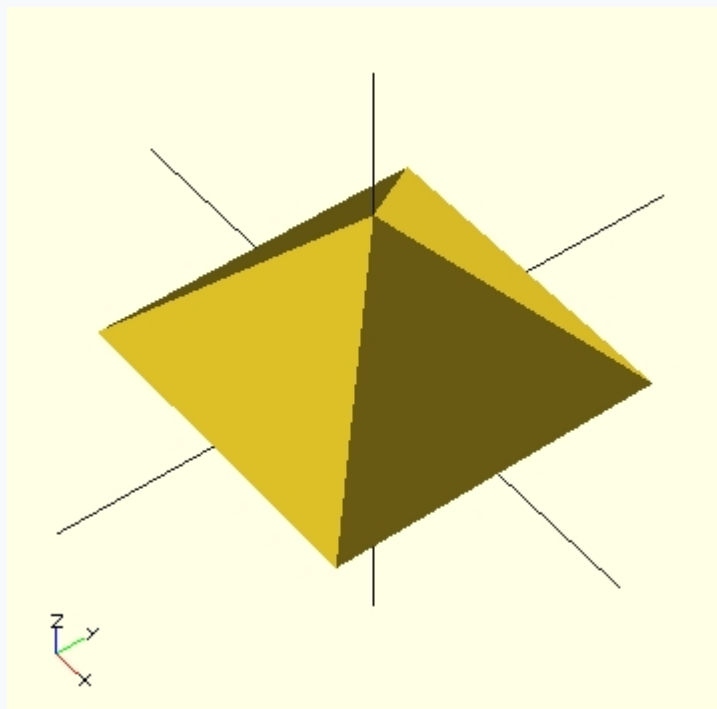


unfolded cube faces

```
CubePoints = [ [ 0, 0, 0 ], //0 [ 10, 0, 0 ], //1 [ 10, 7,
0 ], //2 [ 0, 7, 0 ], //3 [ 0, 0, 5 ], //4 [ 10, 0, 5 ], //5 [ 10,
7, 5 ], //6 [ 0, 7, 5 ]]; //7 CubeFaces = [ [0,1,2,3], // bottom
[4,5,1,0], // front [7,6,5,4], // top [5,6,2,1], // right
[6,7,3,2], // back [7,4,0,3]]; // left polyhedron( CubePoints,
CubeFaces );
```

equivalent descriptions of the bottom face [0,1,2,3], [0,1,2,3,0],
[1,2,3,0], [2,3,0,1], [3,0,1,2], [0,1,2],[2,3,0], // 2 triangles
with no overlap [1,2,3],[3,0,1], [1,2,3],[0,1,3],

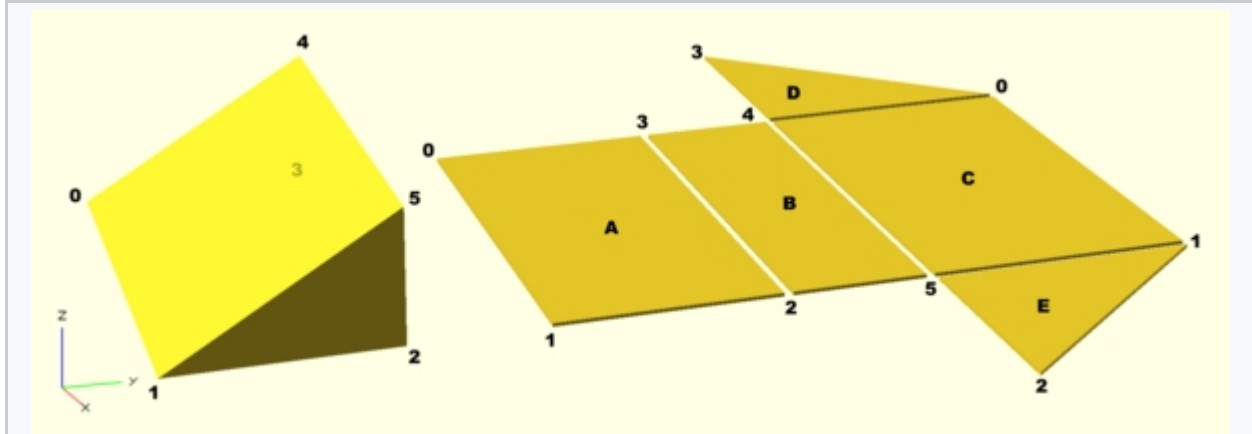
Example 2 A square base pyramid:



A simple polyhedron, square base pyramid


```
polyhedron( points=[ [10,10,0],[10,-10,0],[-10,-10,0],[-10,10,0], // the four points at base [0,0,10] ], // the apex point
faces=[ [0,1,4],[1,2,4],[2,3,4],[3,0,4], // each triangle side
[1,0,3],[2,1,3] ] // two triangles for square base );
```

Example 3 A triangular prism:



Apolyhedron triangular prism

```
module prism(l, w, h){ polyhedron( points=[[0,0,0], [1,0,0],
[1,w,0], [0,w,0], [0,w,h], [1,w,h]], faces=[[0,1,2,3],[5,4,3,2],
[0,4,5,1],[0,3,4],[5,2,1]] ); // preview unfolded (do not include
in your function z = 0.08; separation = 2; border = .2;
translate([0,w+separation,0]) cube([1,w,z]);
translate([0,w+separation+w+border,0]) cube([1,h,z]);
translate([0,w+separation+w+border+h+border,0])
cube([1,sqrt(w*w+h*h),z]);
translate([1+border,w+separation+w+border+h+border,0])
polyhedron( points=[[0,0,0],[h,0,0],[0,sqrt(w*w+h*h),0], [0,0,z],
[h,0,z],[0,sqrt(w*w+h*h),z]], faces=[[0,1,2], [3,5,4], [0,3,4,1],
[1,4,5,2], [2,5,3,0]] ); translate([0-
border,w+separation+w+border+h+border,0])
polyhedron( points=[[0,0,0],[0-h,0,0],[0,sqrt(w*w+h*h),0],
[0,0,z],[0-h,0,z],[0,sqrt(w*w+h*h),z]], faces=[[1,0,2],[5,3,4],
[0,1,4,3],[1,2,5,4],[2,0,3,5]] ); } prism(10, 5, 3);
```

point numbers for cube

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

Debugging polyhedra

Debugging polyhedra

Mistakes in defining polyhedra include not having all faces with the same order, overlap of faces and missing faces or portions of faces. As a general rule, the polyhedron faces should also satisfy manifold conditions:

- exactly two faces should meet at any polyhedron edge.
- if two faces have a vertex in common, they should be in the same cycle face-edge around the vertex.

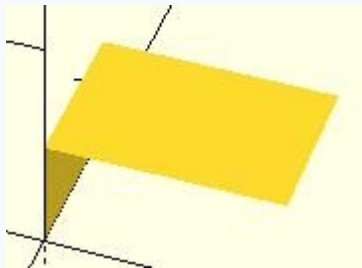
The first rule eliminates polyhedra like two cubes with a common edge and not watertight models; the second excludes polyhedra like two cubes with a common vertex.

When viewed from the outside, the points describing each face must be in the same order. OpenSCAD prefers CW, and provides a mechanism for detecting CCW. When the thrown together view (F12) is used with F5, CCW faces are shown in pink. Reorder the points for incorrect faces. Rotate the object to view all faces. The pink view can be turned off with F10.

OpenSCAD allows, temporarily, commenting out part of the face descriptions so that only the remaining faces are displayed. Use `//` to comment out the rest of the line. Use `/*` and `*/` to start and end a comment block. This can be part of a line or extend over several lines. Viewing only part of the faces can be helpful in determining the right points for an individual face. Note that a solid is not shown, only the faces. If using F12, all faces have one pink side. Commenting some faces helps also to show any internal face.

```
CubeFaces = [ /* [0,1,2,3], // bottom [4,5,1,0], // front */
[7,6,5,4], // top /* [5,6,2,1], // right [6,7,3,2], // back */
[7,4,0,3]]; // left
```

After defining a polyhedron, its preview may seem correct. The polyhedron alone may even render fine. However, to be sure it is a valid manifold and that it can generate a valid STL file, union it with any cube and render it (F6). If the polyhedron disappears, it means that it is not correct. Revise the winding order of all faces and the two rules stated above.



example 1 showing only 2 faces

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)

Mis-ordered faces

Mis-ordered faces

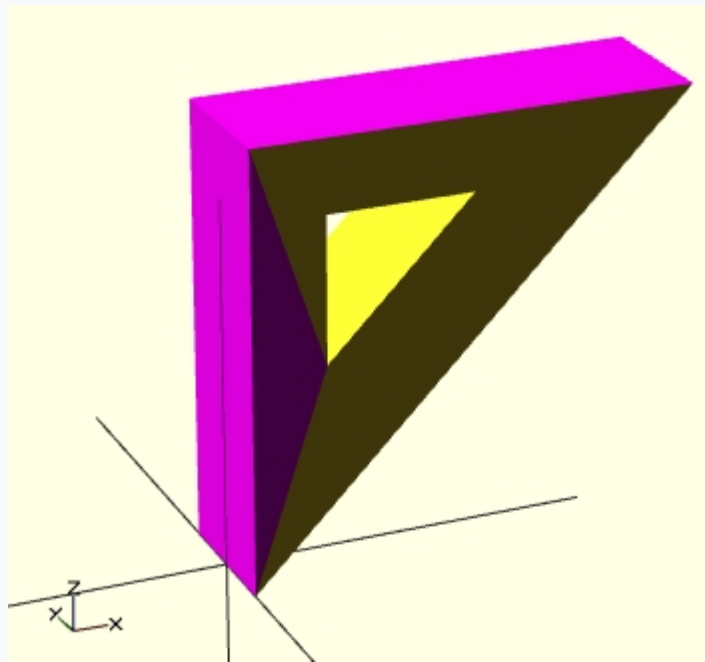
Example 4 a more complex polyhedron with mis-ordered faces

When you select 'Thrown together' from the view menu and **compile** the design (**not** compile and render!) the preview shows the mis-oriented polygons highlighted. Unfortunately this highlighting is not possible in the OpenCSG preview mode because it would interfere with the way the OpenCSG preview mode is implemented.)

Below you can see the code and the picture of such a problematic polyhedron, the bad polygons (faces or compositions of faces) are in pink.

```
// Bad polyhedron
polyhedron
(points = [
```

```
[0, -10, 60], [0, 10, 60], [0, 10, 0], [0, -10, 0], [60, -10, 60], [60,
10, 60], [10, -10, 50], [10, 10, 50], [10, 10, 30], [10, -10, 30], [30, -
10, 50], [30, 10, 50]
], faces = [
[0,2,3], [0,1,2], [0,4,5], [0,5,1], [5,4,2], [2,4,3],
[6,8,9], [6,7,8], [6,10,11], [6,11,7], [10,8,11],
[10,9,8], [0,3,9], [9,0,6], [10,6, 0], [0,4,10],
[3,9,10], [3,10,4], [1,7,11], [1,11,5], [1,7,8], [1,8,2], [2,8,11],
[2,11,5]
]
);
```



Polyhedron with badly oriented polygons

A correct polyhedron would be the following:

```
polyhedron
(points = [
[0, -10, 60], [0, 10, 60], [0, 10, 0], [0, -10, 0], [60, -10, 60], [60,
10, 60], [10, -10, 50], [10, 10, 50], [10, 10, 30], [10, -10, 30], [30, -
10, 50], [30, 10, 50]
], faces = [
[0,3,2], [0,2,1], [4,0,5], [5,0,1], [5,2,4], [4,2,3],
[6,8,9], [6,7,8], [6,10,11], [6,11,7], [10,8,11],
[10,9,8], [3,0,9], [9,0,6], [10,6, 0], [0,4,10],
[3,9,10], [3,10,4], [1,7,11], [1,11,5], [1,8,7], [2,8,1], [8,2,11],
[5,11,2]
]
);
```

Point repetitions in a polyhedron point list

Point repetitions in a polyhedron point list

Beginner's tip

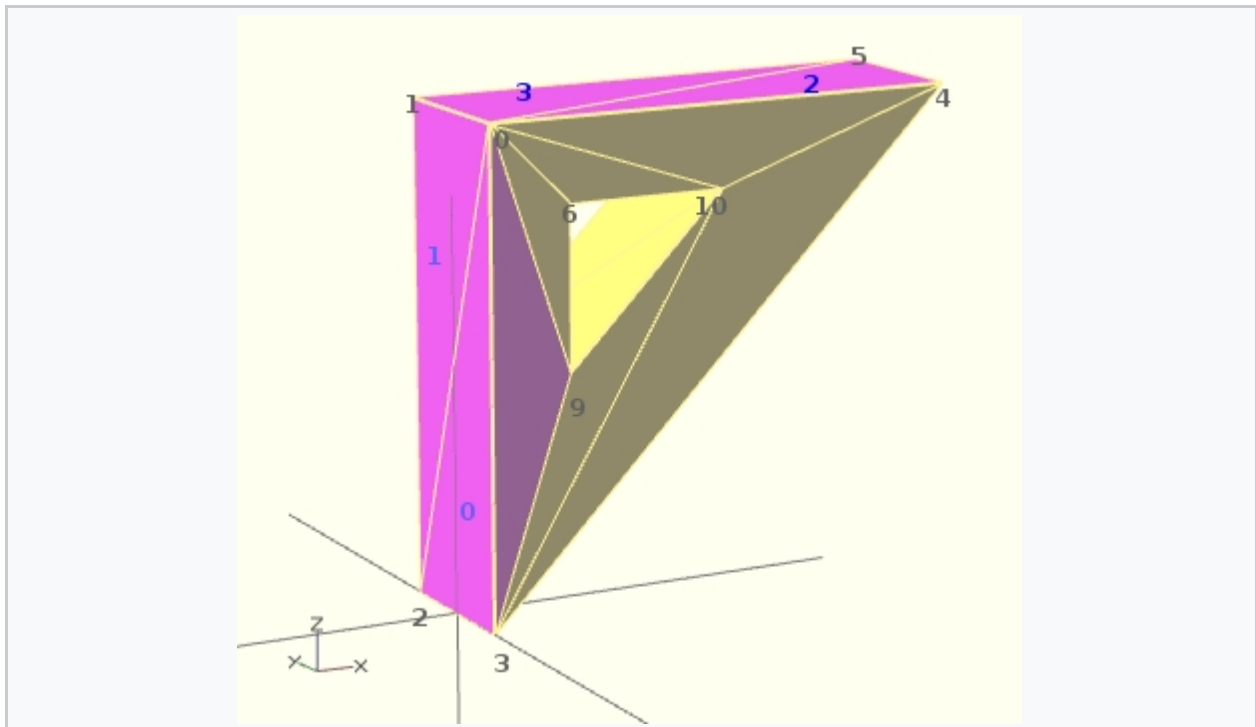
If you don't really understand "orientation", try to identify the mis-oriented pink faces and then invert the sequence of the references to the points/vectors until you get it right. E.g. in the above example, the third triangle ($[0,4,5]$) was wrong and we fixed it as $[4,0,5]$. Remember that a face list is a circular list. In addition, you may select "Show Edges" from the "View Menu", print a screen capture and number both the points and the faces. In our example, the points are annotated in black and the faces in blue. Turn the object around and make a second copy from the back if needed. This way you can keep track.

Clockwise Technique

Orientation is determined by clockwise circular indexing. This means that if you're looking at the triangle (in this case $[4,0,5]$) from the outside you'll see that the path is clockwise around the center of the face. The winding order $[4,0,5]$ is clockwise and therefore good. The winding order $[0,4,5]$ is counter-clockwise and therefore bad. Likewise, any other clockwise order of $[4,0,5]$ works: $[5,4,0]$ & $[0,5,4]$ are good too. If you use the clockwise technique, you'll always have your faces outside (outside of OpenSCAD, other programs do use counter-clockwise as the outside though).

Think of it as a Left Hand Rule:

If you place your left hand on the face with your fingers curled in the direction of the order of the points, your thumb should point outward. If your thumb points inward, you need to reverse the winding order.



Polyhedron with badly oriented polygons

Succinct description of a 'Polyhedron'

```
* Points define all of the points/vertices in the shape. * Faces
is a list of flat polygons that connect up the points/vertices.
```

Each point, in the point list, is defined with a 3-tuple x,y,z position specification. Points in the

point list are automatically enumerated starting from zero for use in the faces list (0,1,2,3,... etc).

Each face, in the faces list, is defined by selecting 3 or more of the points (using the point order number) out of the point list.

e.g. faces=[[0,1,2]] defines a triangle from the first point (points are zero referenced) to the second point and then to the third point.

When looking at any face from the outside, the face must list all points in a clockwise order.

Point repetitions in a polyhedron point list[\[edit\]](#)

The point list of the polyhedron definition may have repetitions. When two or more points have the same coordinates they are considered the same polyhedron vertex. So, the following polyhedron:

```
points = [[ 0, 0, 0], [10, 0, 0], [ 0,10, 0],
[ 0, 0, 0], [10, 0, 0], [ 0,10, 0],
[ 0,10, 0], [10, 0, 0], [ 0, 0,10],
[ 0, 0, 0], [ 0, 0,10], [10, 0, 0],
[ 0, 0, 0], [ 0,10, 0], [ 0, 0,10]];
polyhedron(points, [[0,1,2], [3,4,5], [6,7,8], [9,10,11], [12,13,14]]);
```

define the same tetrahedron as:

```
points = [[0,0,0], [0,10,0], [10,0,0], [0,0,10]];
polyhedron(points, [[0,2,1], [0,1,3], [1,2,3], [0,3,2]]);
```

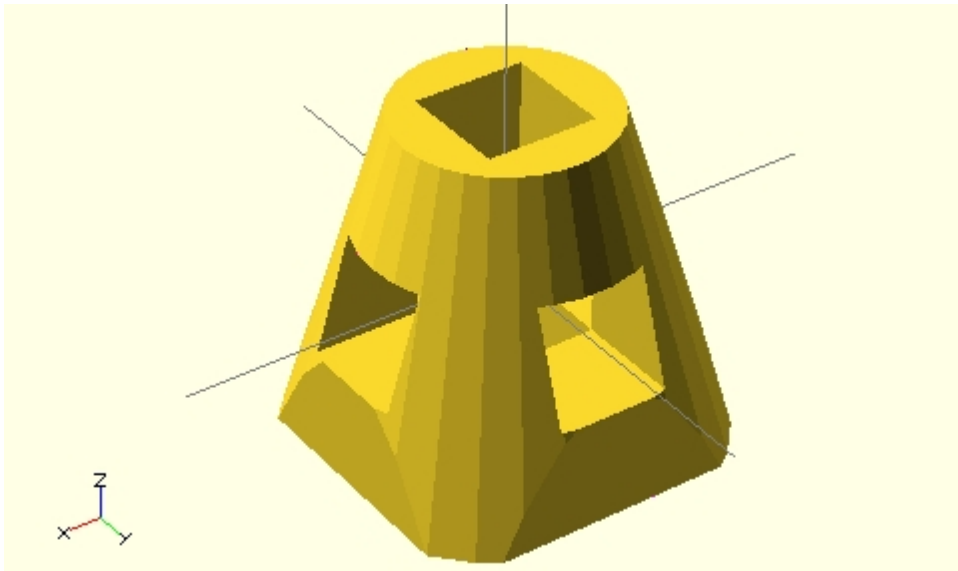
Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

3D to 2D Projection

3D to 2D Projection

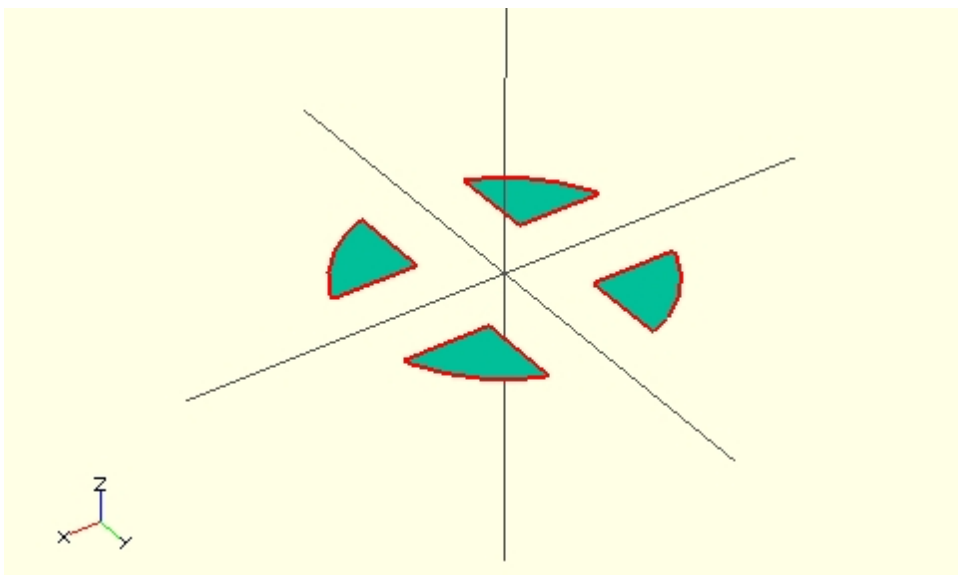
Using the `projection()` function, you can create 2d drawings from 3d models, and export them to the dxf format. It works by projecting a 3D model to the (x,y) plane, with z at 0. If `cut=true`, only points with z=0 are considered (effectively cutting the object), with `cut=false` (*the default*), points above and below the plane are considered as well (creating a proper projection).

Example: Consider example002.scad, that comes with OpenSCAD.



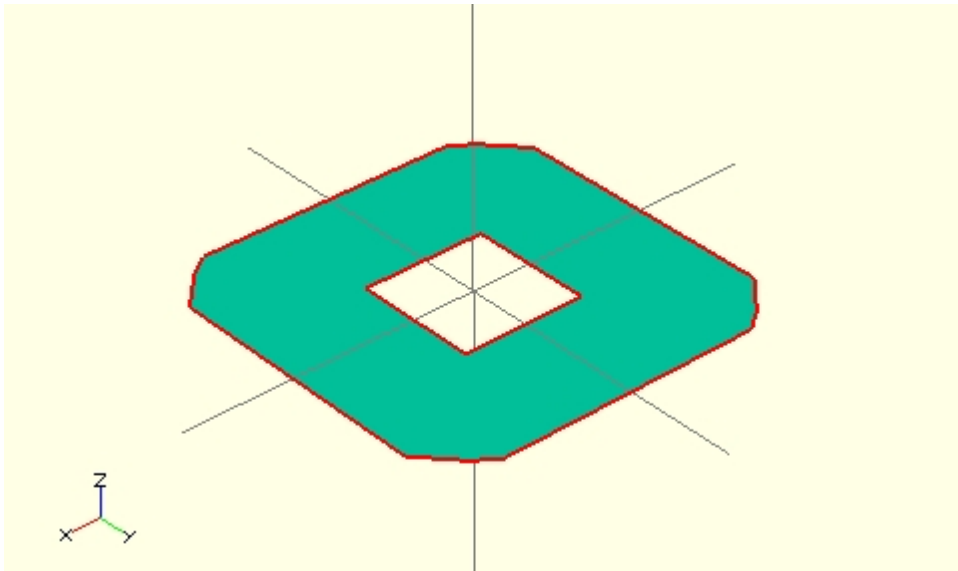
Then you can do a 'cut' projection, which gives you the 'slice' of the x-y plane with $z=0$.

```
projection(cut = true) example002();
```



You can also do an 'ordinary' projection, which gives a sort of 'shadow' of the object onto the xy plane.

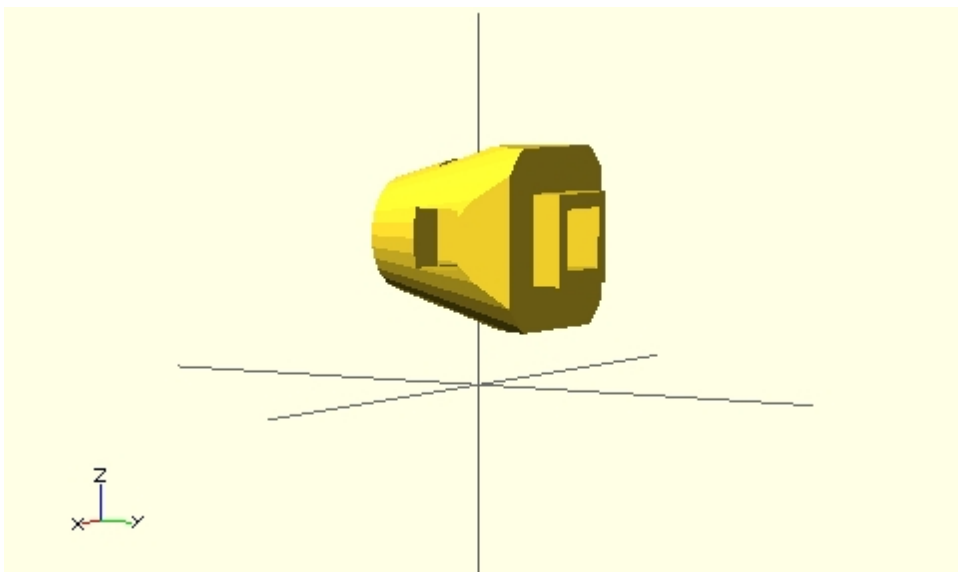
```
projection(cut = false) example002();
```



Another Example

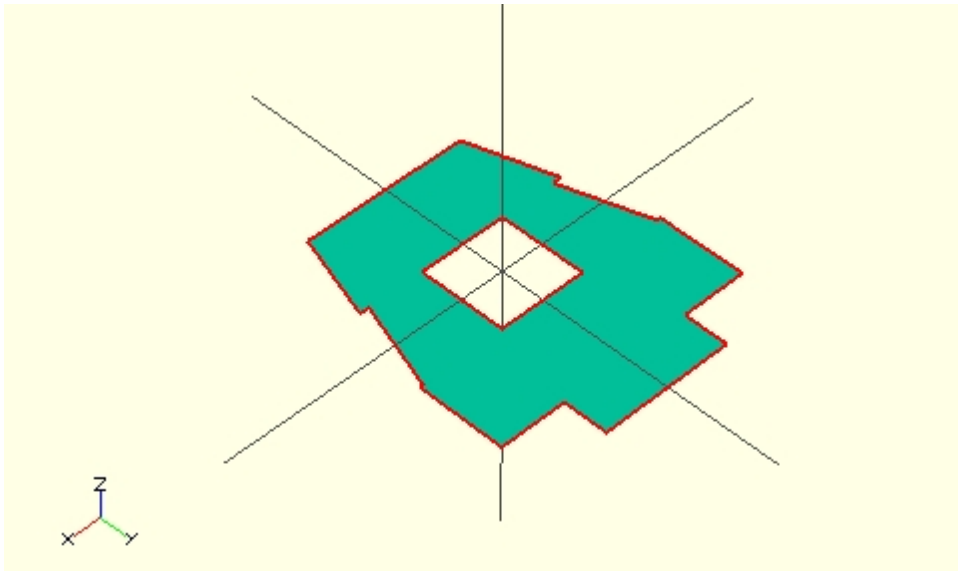
You can also use projection to get a 'side view' of an object. Let's take example002, and move it up, out of the X-Y plane, and rotate it:

```
translate([0,0,25]) rotate([90,0,0]) example002();
```



Now we can get a side view with projection()

```
projection() translate([0,0,25]) rotate([90,0,0]) example002();
```



Links:

- [example021.scad](#) from Clifford Wolf's site.
- [More complicated example](#) from Giles Bathgate's blog

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

2D Objects

2D Objects

All 2D primitives can be transformed with 3D transformations. Usually used as part of a 3D extrusion. Although infinitely thin, they are rendered with a 1 thickness.

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

square

square

Creates a square or rectangle in the first quadrant. When center is true the square is centered on the origin. Argument names are optional if given in the order shown here.

```
square(size = [x, y], center = true/false); square(size = x ,
center = true/false);
```

parameters:

size

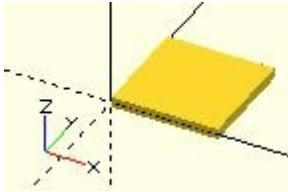
single value, square with both sides this length
 2 value array [x,y], rectangle with dimensions x and y

center

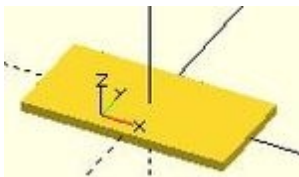
false (default), 1st (positive) quadrant, one corner at (0,0)
true, square is centered at (0,0)


```
default values: square(); yields: square(size = [1, 1], center = false);
```

examples:



```
equivalent scripts for this example square(size = 10); square(10);
square([10,10]); . square(10,false); square([10,10],false);
square([10,10],center=false); square(size = [10, 10], center =
false); square(center = false,size = [10, 10] );
```



```
equivalent scripts for this example square([20,10],true);
a=[20,10];square(a,true);
```

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

circle

circle

Creates a circle at the origin. All parameters, except **r**, **must** be named.

```
circle(r=radius | d=diameter);
```

Parameters

r : circle radius. **r** name is the only one optional with circle.

circle resolution is based on size, using \$fa or \$fs.

For a small, high resolution circle you can make a large circle, then scale it down, or you could set \$fn or other special variables. Note: These examples exceed the resolution of a 3d printer as well as of the display screen.

```
scale([1/100, 1/100, 1/100]) circle(200); // create a high
resolution circle with a radius of 2. circle(2, $fn=50); //
Another way.
```

d : circle diameter (only available in versions later than 2014.03).

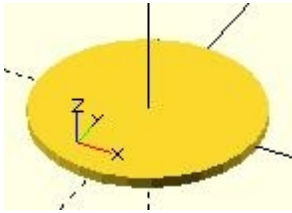
\$fa : minimum angle (in degrees) of each fragment.

\$fs : minimum circumferential length of each fragment.

\$fn : **fixed** number of fragments in 360 degrees. Values of 3 or more override \$fa and \$fs

\$fa, \$fs and \$fn must be named. [click here for more details](#),.

```
defaults: circle(); yields: circle($fn = 0, $fa = 12, $fs = 2, r = 1);
```



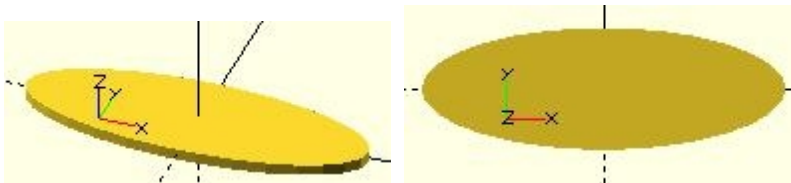
```
equivalent scripts for this example circle(10); circle(r=10);  
circle(d=20); circle(d=2+9*2);
```

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)

ellipse

ellipse

An ellipse can be created from a circle by using either `scale()` or `resize()` to make the x and y dimensions unequal. See [OpenSCAD User Manual/Transformations](#)



```
equivalent scripts for this example resize([30,10])circle(d=20);  
scale([1.5,.5])circle(d=20);
```

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

regular polygon

regular polygon

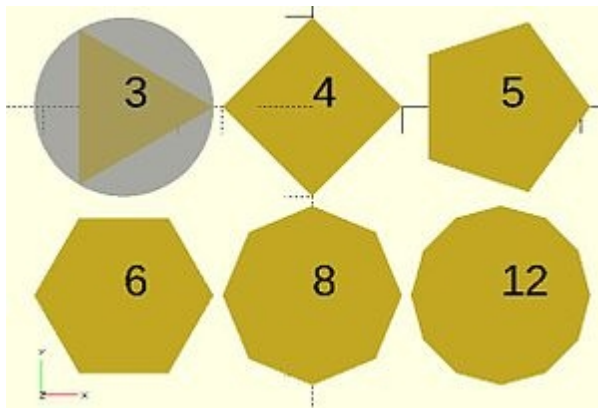
A regular polygon of 3 or more sides can be created by using `circle()` with `$fn` set to the number of sides. The following two pieces of code are equivalent.

```
circle(r=1, $fn=4);
```

```
module regular_polygon(order = 4, r=1){ angles=[ for (i =  
[0:order-1]) i*(360/order) ]; coords=[ for (th=angles) [r*cos(th),  
r*sin(th)] ]; polygon(coords); } regular_polygon();
```

These result in the following shapes, where the polygon is inscribed within the circle with all sides (and angles) equal. One corner points to the positive x direction. For irregular shapes see

the polygon primitive below.



```
script for these examples translate([-42, 0]){circle(20,$fn=3);%
circle(20,$fn=90);} translate([ 0, 0]) circle(20,$fn=4);
translate([ 42, 0]) circle(20,$fn=5); translate([-42,-42])
circle(20,$fn=6); translate([ 0,-42]) circle(20,$fn=8);
translate([ 42,-42]) circle(20,$fn=12);
```

```
color("black"){ translate([-42, 0,1])text("3",7,,center);
translate([ 0, 0,1])text("4",7,,center); translate([ 42, 0,1])
text("5",7,,center); translate([-42,-42,1])text("6",7,,center);
translate([ 0,-42,1])text("8",7,,center); translate([ 42,-42,1])
text("12",7,,center); }
```

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

polygon

polygon

Creates a multiple sided shape from a list of x,y coordinates. A polygon is the most powerful 2D object. It can create anything that circle and squares can, as well as much more. This includes irregular shapes with both concave and convex edges. In addition it can place holes within that shape.

```
polygon(points = [ [x, y], ... ], paths = [ [p1, p2, p3...], ...],
convexity = N);
```

Parameters

points

The list of x,y points of the polygon. : A vector of 2 element vectors.

Note: points are indexed from 0 to n-1.

paths

default

If no path is specified, all points are used in the order listed.

single vector

The order to traverse the points. Uses indices from 0 to n-1. May be in a different order and use all or part, of the points listed.

multiple vectors

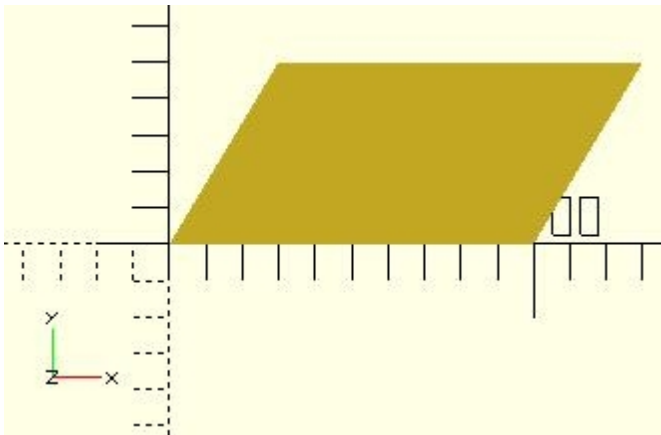
Creates primary and secondary shapes. Secondary shapes are subtracted from the primary shape (like difference). Secondary shapes may be wholly or partially within the primary shape.

A closed shape is created by returning from the last point specified to the first.

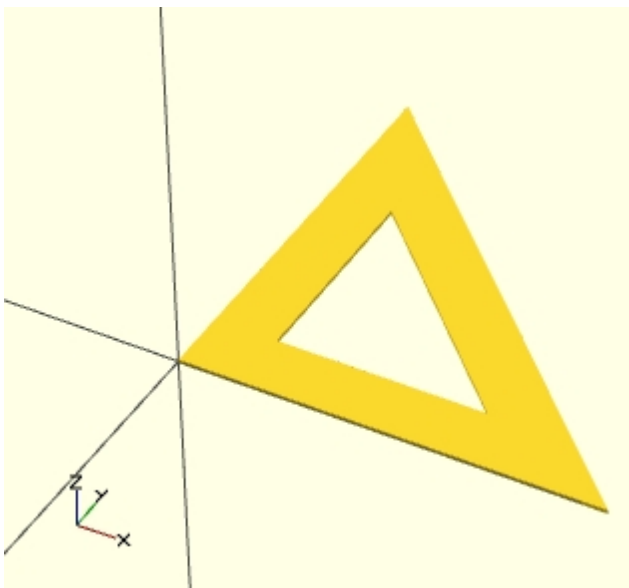
convexity

Integer number of "inward" curves, ie. expected path crossings of an arbitrary line through the polygon. See below.

```
defaults: polygon(); yields: polygon(points = undef, paths =
undef, convexity = 1);
```

Example no holes

```
equivalent scripts for this example polygon(points=[[0,0],[100,0],
[130,50],[30,50]]); polygon([[0,0],[100,0],[130,50],[30,50]],
paths=[[0,1,2,3]]); polygon([[0,0],[100,0],[130,50],[30,50]],
[[3,2,1,0]]); polygon([[0,0],[100,0],[130,50],[30,50]],
[[1,0,3,2]]); a=[[0,0],[100,0],[130,50],[30,50]]; b=[[3,0,1,2]];
polygon(a); polygon(a,b); polygon(a,[[2,3,0,1,2]]);
```

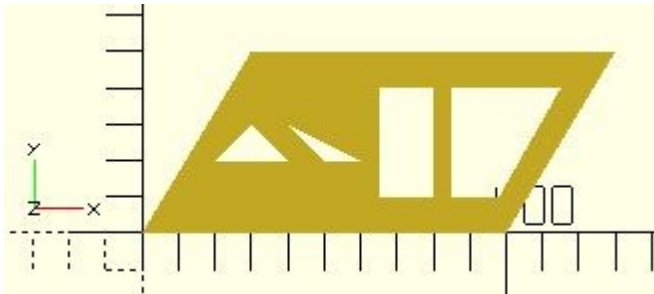
Example one hole

```
equivalent scripts for this example polygon(points=[[0,0],[100,0],
[0,100],[10,10],[80,10],[10,80]], paths=[[0,1,2],
[3,4,5]],convexity=10); triangle_points =[[0,0],[100,0],[0,100],
[10,10],[80,10],[10,80]]; triangle_paths =[[0,1,2],[3,4,5]];
polygon(triangle_points,triangle_paths,10);
```

The 1st path vector, [0,1,2], selects the points, [0,0],[100,0],[0,100], for the primary shape. The 2nd path vector, [3,4,5], selects the points, [10,10],[80,10],[10,80], for the secondary shape. The secondary shape is subtracted from the primary (think `difference()`). Since the secondary is wholly within the primary, it leaves a shape with a hole.

Example multi hole

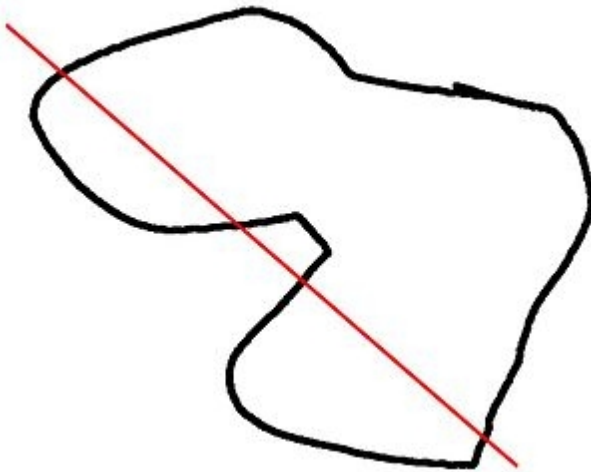
[Note: Requires version **2015.03**] (for use of `concat()`)



```
//example polygon with multiple holes a0 = [[0,0],[100,0],
[130,50],[30,50]]; // main b0 = [1,0,3,2]; a1 = [[20,20],[40,20],
[30,30]]; // hole 1 b1 = [4,5,6]; a2 = [[50,20],[60,20],
[40,30]]; // hole 2 b2 = [7,8,9]; a3 = [[65,10],[80,10],[80,40],
[65,40]]; // hole 3 b3 = [10,11,12,13]; a4 = [[98,10],[115,40],
[85,40],[85,10]]; // hole 4 b4 = [14,15,16,17]; a = concat
(a0,a1,a2,a3,a4); b = [b0,b1,b2,b3,b4]; polygon(a,b); //alternate
polygon(a,[b0,b1,b2,b3,b4]);
```

convexity

The convexity parameter specifies the maximum number of front sides (back sides) a ray intersecting the object might penetrate. This parameter is needed only for correct display of the object in OpenCSG preview mode and has no effect on the polyhedron rendering.



This image shows a 2D shape with a convexity of 4, as the ray indicated in red crosses the 2D shape a maximum of 4 times. The convexity of a 3D shape would be determined in a similar way. Setting it to 10 should work fine for most cases.

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

Text

Text

The `text` module creates text as a 2D geometric object, using fonts installed on the local system or provided as separate font file.

Parameters

text

String. The text to generate.

size

Decimal. The generated text has an ascent (height above the baseline) of approximately the given value. Default is 10. Different fonts can vary somewhat and may not fill the size specified exactly, typically they render slightly smaller.

font

String. The name of the font that should be used. This is not the name of the font file, but the logical font name (internally handled by the fontconfig library). This can also include a style parameter, see below. A list of installed fonts & styles can be obtained using the font list dialog (Help -> Font List).

halign

String. The horizontal alignment for the text. Possible values are "left", "center" and "right". Default is "left".

valign

String. The vertical alignment for the text. Possible values are "top", "center", "baseline" and "bottom". Default is "baseline".

spacing

Decimal. Factor to increase/decrease the character spacing. The default value of 1 results in the normal spacing for the font, giving a value greater than 1 causes the letters to be spaced further

apart.

direction

String. Direction of the text flow. Possible values are "ltr" (left-to-right), "rtl" (right-to-left), "ttb" (top-to-bottom) and "btt" (bottom-to-top). Default is "ltr".

language

String. The language of the text. Default is "en".

script

String. The script of the text. Default is "latin".

\$fn

used for subdividing the curved path segments provided by freetype

Example

```
text ( "OpenSCAD" );
```

Note

To allow specification of particular **Unicode** characters you can specify them in a string with the following escape codes;

\x03 - single hex character (only allowed values are 01h - 7fh)

\u0123 - unicode char with 4 hexadecimal digits (note: Lowercase)

\U012345 - unicode char with 6 hexadecimal digits (note: Uppercase)

Example

```
t="\u20AC10 \u263A"; // 10 euro and a smileie
```



Example 1: Result.

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

Using Fonts & Styles

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Alignment

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

Vertical Alignment

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Horizontal Alignment

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

3D Text

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

3D to 2D Projection

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

2D to 3D Projection

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

Linear Extrude

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

Usage

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Twist

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Center

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

Mesh Refinement

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

Scale

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

Rotate Extrude

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Parameters

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

Usage

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Examples

Created with the Personal Edition of HelpNDoc: [Free EBook and documentation generator](#)

Mesh Refinement

Created with the Personal Edition of HelpNDoc: [Easily create PDF Help documents](#)

Extruding a Polygon

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Description of extrude Parameters

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

Extrude parameters for all extrusion modes

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

Extrude parameters for linear extrusion only

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

Transforms

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Basic concept

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

Advanced concept

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

scale

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

resize

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

rotate

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

Rotation rule Help

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

translate

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

mirror

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

Function signature:

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

Examples

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

multimatrix

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

More?

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

color

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

function signature

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

Example

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

Example 2

Created with the Personal Edition of HelpNDoc: [Free Kindle producer](#)

offset

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

minkowski

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

hull

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

Combining transformations

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

Boolean Combinations

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

Boolean overview

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

2D examples

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

3D examples

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

union

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

difference

Created with the Personal Edition of HelpNDoc: [Free EPub and documentation generator](#)

difference with multiple children

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

intersection

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

render

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

Other Functions and Operators

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

Condition and Iterator functions

Created with the Personal Edition of HelpNDoc: [Free Kindle producer](#)

For loop

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

Intersection For loop

Created with the Personal Edition of HelpNDoc: [Free EPub and documentation generator](#)

if statement

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

else if

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

Conditional ?

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

Recursive function calls

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

Assign Statement

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Let Statement

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Mathematical Operators

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

Scalar Arithmetical Operators

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

Relational Operators

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

Logical Operators

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

Conditional Operator

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

Vector-Number Operator

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

Vector Operators

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

Vector Dot-Product Operator

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

Matrix Multiplication

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

Mathematical Functions

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

Trigonometric Functions

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

cos

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

sin

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

tan

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

acos

Created with the Personal Edition of HelpNDoc: [Free EBook and documentation generator](#)

asin

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

atan

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

atan2

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Other Mathematical Functions

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

abs

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

ceil

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

concat

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

cross

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

exp

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

floor

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

in

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

len

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

let

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

log

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

lookup

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

max

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

min

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

norm

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

pow

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

rands

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

round

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

sign

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

sqrt

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

Infinities and NaNs

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

String Functions

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

str

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

chr

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

ord

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

Also see search()

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

List Comprehensions

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

Basic Syntax

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

Multiple generator expressions

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

for

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

each

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

if

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

if/else

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

let

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

Nested loops

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

Advanced Examples

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

Generating vertices for a polygon

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Flattening a nested vector

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

Sorting a vector

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

Selecting elements of a vector

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Concatenating two vectors

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Other Language Features

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Special Variables

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

\$fa, \$fs and \$fn

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

\$t

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

\$vpr, \$vpt and \$vpd

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

\$preview

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

Echo Statements

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Usage examples

Created with the Personal Edition of HelpNDoc: [Easily create PDF Help documents](#)

Rounding examples

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Small and large Numbers

Created with the Personal Edition of HelpNDoc: [Easily create PDF Help documents](#)

HTML

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

Echo Function

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Render

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Surface

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

Text file format

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Images

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

Examples

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

Search

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

Search Usage

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

Search Arguments

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Search Usage Examples

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Index values return as list

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

Search on different column; return index values

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Search on list of values

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Search on list of strings

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Getting the right result

Created with the Personal Edition of HelpNDoc: [Free EPub and documentation generator](#)

OpenSCAD Version

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

parent_module(n) and \$parent_modules

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

assert

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

Example

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

Checking parameters

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Adding message

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

Using assertions in functions

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

User Defined Functions and Modules

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Introduction

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Scope

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

Functions

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

Recursive Functions

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

Function Literals

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

Modules

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

Object modules

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Operator Modules

Created with the Personal Edition of HelpNDoc: [Easily create PDF Help documents](#)

Children

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

Further Module Examples

Created with the Personal Edition of HelpNDoc: [Easily create PDF Help documents](#)

Recursive Modules

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

Overwriting built-in modules

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

Overwriting built functions

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Debugging Aids

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

Advanced Concept

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

Background Modifier

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Debug Modifier

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Root Modifier

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Disable Modifier

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

Echo Statements

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

External Libraries and code files

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

Use and Include

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

Directory separators

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

Variables

Created with the Personal Edition of HelpNDoc: [Free Kindle producer](#)

Scope of Variables

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

Overwriting variables

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Example "Ring-Library"

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

Nested Include and Use

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

Import

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

Parameters

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

Convexity

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Notes

Created with the Personal Edition of HelpNDoc: [Free EPub and documentation generator](#)

Import DXF

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Import STL

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

surface

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Parameters

Created with the Personal Edition of HelpNDoc: [Easily create eBooks](#)

Text file Format

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

Images

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

Examples

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

MCAD Library

MCAD Library

This library contains components commonly used in designing and mocking up mechanical designs. It is currently unfinished and you can expect some API changes, however many things are already working.

This library was created by various authors as named in the individual files' comments. All the files are

licensed under the LGPL 2.1 (see <http://creativecommons.org/licenses/LGPL/2.1/> or the included file lgpl-2.1.txt), some of them allow distribution under more permissive terms (as described in the files' comments).

Usage

You can import these files in your scripts with use <MCAD/filename.scad>, where 'filename' is one of the files listed below like 'motors' or 'servos'. Some files include useful constants which will be available with include <MCAD/filename.scad>, which should be safe to use on all included files (ie. no top level code should create geometry).

(There is a bug/feature that prevents including constants from files that "include" other files - see the openscad mailing list archives for more details. Since the maintainers aren't very responsive, may have to work around this somehow)

If you host your project in git, you can do git submodule add URL PATH in your repo to import this library as a git submodule for easy usage. Then you need to do a git submodule update --init after cloning. When you want to update the submodule, do cd PATH; git checkout master; git pull. See git help submodule for more info.

Currently Provided Tools:

```
regular_shapes.scad
    regular polygons, ie. 2D
    regular polyhedrons, ie. 3D
involute_gears.scad (http://www.thingiverse.com/thing:3575):
    gear()
    bevel_gear()
    bevel_gear_pair()
gears.scad (Old version):
    gear(number_of_teeth, circular_pitch OR diametrial_pitch, pressure_angle OPTIONAL,
clearance OPTIONAL)
motors.scad:
    stepper_motor_mount(nema_standard, slide_distance OPTIONAL, mochup OPTIONAL)
```

Tools (alpha and beta quality):

```
nuts_and_bolts.scad: for creating metric and imperial bolt/nut holes
bearing.scad: standard/custom bearings
screw.scad: screws and augers
materials.scad: color definitions for different materials
stepper.scad: NEMA standard stepper outlines
servos.scad: servo outlines
boxes.scad: box with rounded corners
triangles.scad: simple triangles
3d_triangle.scad: more advanced triangles
```

Very generally useful functions and constants:

```
math.scad: general math functions
constants.scad: mathematical constants
curves.scad: mathematical functions defining curves
units.scad: easy metric units
utilities.scad: geometric funtions and misc. useful stuff
teardrop.scad (http://www.thingiverse.com/thing:3457): parametric teardrop module
shapes.scad: DEPRECATED simple shapes by Catarina Mota
polyholes.scad: holes that should come out well when printed
```

Other:

alphabet_block.scad
 bitmap.scad
 letter_necklace.scad
 name_tag.scad
 height_map.scad
 trochoids.scad
 libtriangles.scad
 layouts.scad
 transformations.scad
 2Dshapes.scad
 gridbeam.scad
 fonts.scad
 unregular_shapes.scad
 metric_fasteners.scad
 lego_compatibility.scad
 multiply.scad
 hardware.scad

External utils that generate and process openscad code:

openscad_testing.py: testing code, see below
 openscad_utils.py: code for scraping function names etc.

Created with the Personal Edition of HelpNDoc: [Free Kindle producer](#)

regular_shapes.scad

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

2D regular shapes

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

regular_polygon(sides, radius)

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

n-gons 2D shapes

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

triangle(radius)

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

pentagon(radius)

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

hexagon(radius)

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

heptagon(radius)

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

octagon(radius)

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

nonagon(radius)

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

decagon(radius)

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

hendecagon(radius)

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

dodecagon(radius)

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

ring(inside_diameter, thickness)

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

ellipse(width, height)

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

egg_outline(width, thickness)

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

3D regular shapes

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

cone

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

oval_prism

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

oval_tube

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

cylinder_tube

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

triangle_prism

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

triangle_tube

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

pentagon_prism

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

pentagon_tube

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

hexagon_prism

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

hexagon_tube

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

heptagon_prism

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

heptagon_tube

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

octagon_prism

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

octagon_tube

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

nonagon_prism

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

`decagon_prism`

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

`hendecagon_prism`

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

`dodecagon_prism`

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

`torus`

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

`torus2`

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

`oval_torus`

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

`triangle_pyramid`

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

`square_pyramid`

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

`egg`

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

`involute_gears.scad`

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

`bevel_gear_pair()`

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

`bevel_gear()`

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

gear()

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

Tests

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

test_gears()

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

test_meshing_double_helix()

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

test_bevel_gear()

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

test_bevel_gear_pair()

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

test_backlash()

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

dotSCAD Library

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

2D modules

Created with the Personal Edition of HelpNDoc: [Free EBook and documentation generator](#)

arc

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

pie

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

rounded_square

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

line2d

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

polyline2d

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

hull_polyline2d

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

hexagons

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

polytransversals

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

multi_line_text

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

voronoi2d

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

3D modules

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

rounded_cube

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

rounded_cylinder

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

crystal_ball

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

line3d

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

polyline3d

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

hull_polyline3d

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

function_grapher

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

sweep

Created with the Personal Edition of HelpNDoc: [Free EBook and documentation generator](#)

loft

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

starburst

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

voronoi3d

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Transformations

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

along_width

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

hollow_out

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

bend

Created with the Personal Edition of HelpNDoc: [Free Kindle producer](#)

shear

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

2D functions

Created with the Personal Edition of HelpNDoc: [Easily create PDF Help documents](#)

in_shape

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

bijection_offset

Created with the Personal Edition of HelpNDoc: [Full-featured Documentation generator](#)

trim_shape

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

triangulate

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

contours

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

2D/3D functions

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

cross_sections

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

paths2sections

Created with the Personal Edition of HelpNDoc: [Easily create eBooks](#)

path_scaling_sections

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

bezier_surface

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

bezier_smooth

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

midpt_smooth

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

in_polyline

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

Paths

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

arc_path

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

bspline_curve

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

bezier_curve

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

helix

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

golden_spiral

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

archimedean_spiral

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

sphere_spiral

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

torus_knot

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Extrusion

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

box_extrude

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

ellipse_extrude

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

stereographic_extrude

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

rounded_extrude

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

bend_extrude

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

2D Shape

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

shape_taiwan

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

shape_arc

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

shape_pie

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

shape_circle

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

shape_ellipse

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

shape_square

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

shape_trapezium

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

shape_cyclicpolygon

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

shape_pentagram

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

shape_starburst

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

shape_superformula

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

shape_glue2circles

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

shape_path_extend

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

2D Shape extrusions

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

path_extrude

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

ring_extrude

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

helix_extrude

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

golden_spiral_extrude

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

archimedean_spiral_extrude

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

sphere_spiral_extrude

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Utils

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

util/sub_str

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

util/split_str

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

util/parse_number

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

util/reverse

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

util/slice

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

util/sort

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

util/rand

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

util/fibseq

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

util/bsearch

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

util/has

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

util/dedup

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

util/flat

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Matrix

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

matrix/m_cumulate

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

matrix/m_translation

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

matrix/m_rotation

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

matrix/m_scaling

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)

matrix/m_mirror

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

matrix/m_shearing

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

Point Transformation

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

ptf/ptf_rotate

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

ptf/ptf_x_twist

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

ptf/ptf_y_twist

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

ptf/ptf_circle

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

ptf/ptf_bend

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

ptf/ptf_ring

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

ptf/ptf_sphere

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

ptf/ptf_torus

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

Turtle

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

turtle/turtle2d

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

turtle/turtle3d

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

turtle/t2d

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

turtle/t3d

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Pixel

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

pixel/px_line

Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

pixel/px_polyline

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

pixel/px_circle

Created with the Personal Edition of HelpNDoc: [Free EBook and documentation generator](#)

pixel/px_cylinder

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

pixel/px_sphere

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

pixel/px_polygon

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

pixel/px_from

Created with the Personal Edition of HelpNDoc: [Free iPhone documentation generator](#)

pixel/px_ascii

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)

pixel/px_gray

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Part

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

part/connector_jpg

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

part/cone

Created with the Personal Edition of HelpNDoc: [Generate EPub eBooks with ease](#)

part/join_T

Created with the Personal Edition of HelpNDoc: [Easily create EBooks](#)

Surface

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

surface/sf_square

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

surface/sf_bend

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

surface/sf_ring

Created with the Personal Edition of HelpNDoc: [Free EBook and documentation generator](#)

surface/sf_sphere

Created with the Personal Edition of HelpNDoc: [Free Web Help generator](#)

surface/sf_torus

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

surface/sf_solidity

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

Noise

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

noise/nz_perlin1

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

noise/nz_perlin1s

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

noise/nz_perlin2

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

noise/nz_perlin2s

Created with the Personal Edition of HelpNDoc: [Easily create eBooks](#)

noise/nz_perlin3

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

noise/nz_perlin3s

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

noise/nz_worley2

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

noise/nz_worley2s

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

noise/nz_worley3

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

noise/nz_worley3s

Created with the Personal Edition of HelpNDoc: [Free CHM Help documentation generator](#)

noise/nz_cell

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

BOSL Library

Created with the Personal Edition of HelpNDoc: [Generate Kindle eBooks with ease](#)

Commonly Used

Commonly Used

The most commonly used transformations, manipulations, and shortcuts.

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

transforms.scad

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

Translations

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

move()

move()

use<transforms.scad>

Usage:

- move([x], [y], [z]) ...

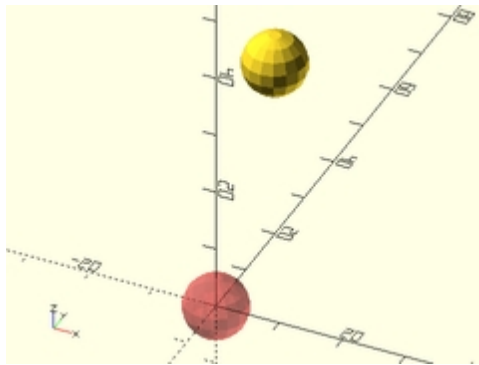
- `move([x, y, z]) ...`

Description: Moves/translated children.

Argument	What it does
x	X axis translation.
y	Y axis translation.
z	Z axis translation.

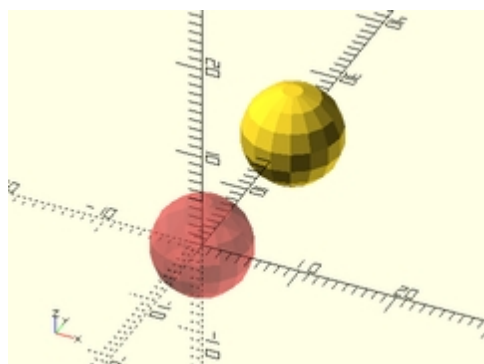
Example 1:

```
#sphere(d = 10);
move([0, 20, 30]) sphere(d = 10);
```



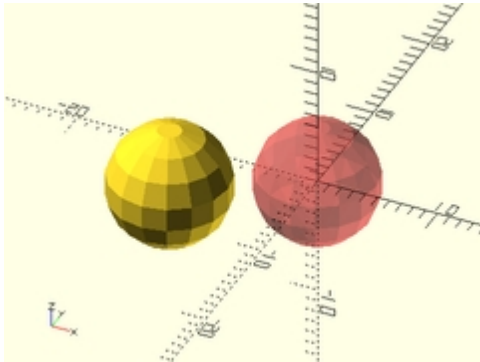
Example 2:

```
#sphere(d = 10);
move(y = 20) sphere(d = 10);
```



Example 3:

```
#sphere(d = 10);
move(x = -10, y = -5) sphere(d = 10);
```



Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

xmove()

xmove()

use<transforms.scad>

Usage:

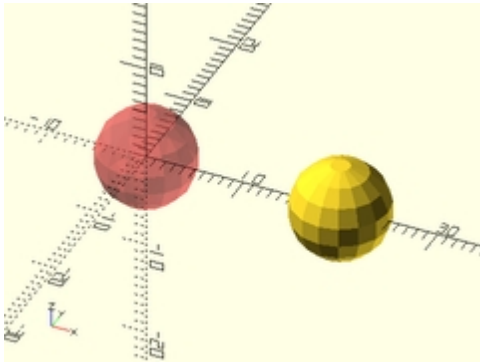
- xmove(x) ...

Description: Moves/translates children the given amount along the X axis.

Argument	What it does
x	Amount to move right along the X axis. Negative values move left.

Example:

```
#sphere(d = 10);
xmove(20) sphere(d = 10);
```



Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

`ymove()`

Created with the Personal Edition of HelpNDoc: [Easy CHM and documentation editor](#)

`zmove()`

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

`left()`

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

`right()`

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

`fwd() / forward()`

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)

`back()`

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

`down()`

Created with the Personal Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

`up()`

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

`shapes.scad`

Created with the Personal Edition of HelpNDoc: [Free EPub and documentation generator](#)

`masks.scad`

Created with the Personal Edition of HelpNDoc: [Free HTML Help documentation generator](#)

threading.scad

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

paths.scad

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

beziers.scad

Created with the Personal Edition of HelpNDoc: [Easy EBook and documentation generator](#)

Standard Parts

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

involute_gears.scad

Created with the Personal Edition of HelpNDoc: [Produce electronic books easily](#)

joiners.scad

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

sliders.scad

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

metric_screws.scad

Created with the Personal Edition of HelpNDoc: [Create cross-platform Qt Help files](#)

linear_bearings.scad

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

nema_steppers.scad

Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

phillips_drive.scad

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

torx_drive.scad

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

wiring_scad

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

Miscellaneous

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

constants.scad

Created with the Personal Edition of HelpNDoc: [Free Qt Help documentation generator](#)

math.scad

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

convex_hull.scad

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

quaternions.scad

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

triangulation.scad

Created with the Personal Edition of HelpNDoc: [Create iPhone web-based documentation](#)

debug.scad

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Nut Job Library

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)
