omdl

v0.5

Generated by Doxygen 1.8.9.1

Sat Feb 4 2017 00:28:43

Contents

1 omdl

A collection of documented design primitives for `OpenSCAD` inspired by *MCAD*.

1.1 What is omdl?

It is an OpenSCAD mechanical design library (omdl) that aims to offer open-source high-level design primitives with coherent documentation generated by Doxygen using openscad-amu.

With Doxygen, the code documentation is written within the code itself, and is thus easy to keep current. Moreover, it provides a standard way to both write and present OpenSCAD script documentation, compilable to multiple common output formats.

With omdl, all library primitives are *parametric* with minimal, mostly zero, global variable dependencies and all library API's include Doxygen markups that describe its parameters, behavior, and use. Validation scripts are used to check that the core building blocks work as expected across evolving OpenSCAD versions. These validation are performed automatically when rebuilding the documentation.

See the validation section for the results generated with this documentation.

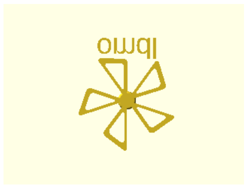
1.2 Quick Start

The library components may be `used` or `included` as expected.

Example:

```
include <shapes2de.scad>;
include <shapes3d.scad>;

$fn = 36;

frame = triangle_vp2vl( [ [30,0], [0,40], [30,40] ] );
core  = 2 * frame / 3;
vrnd  = [1, 2, 4];

cone( h=20, r=10, vr=2 );
rotate([0, 0, 360/20])
st_radial_copy( n=5, angle=true )
  etriangle_vl_c( vs=frame, vc=core, vr=vrnd, h=10 );

translate([0, -50,0])
linear_extrude(height=10)
text( text="omdl", size=20, halign="center", valign="center" );
```

Table 1: Example Result

Bottom	Diagonal	Right	Top
			

1.3 Using omdl

To use omdl the library files must be copied to one of the OpenSCAD Library Locations. This can be done manually, as described in the OpenSCAD documentation, or can be automated using openscad-amu.

The ladder is recommended and has several advantages. When using openscad-amu, the library reference documentation is installed together with the library source code. This reference documentation is also added to a browsable data-table of installed libraries, which facilitates design reference searches. Moreover, with openscad-amu installed, one can rebuild the omdl reference manual or develop documentation for new OpenSCAD designs.

Library release files are made available in the source repository in a sub-directory called *snapshots*.

1.3.1 Recommended install method

More information on installing openscad-amu can be found published on Thingiverse and in the GitHib openscad-amu repository where the source is maintained.

A build script exists for *Linux* and *Cygwin* (pull requests for *macos* are welcome). If wget is not available, here is a downloadable link to the bootstrap script.

```
$ mkdir tmp && cd tmp

$ wget https://raw.githubusercontent.com/royasutton/openscad-amu/master/snapshots/bootstrap.{bash,conf} .
$ chmod +x bootstrap.bash

$ ./bootstrap.bash --yes --install

$ openscad-seam -v -V
```

If the last step reports the tool build version, then the install most likely completed successfully and the temporary directory created in the first step may be removed when desired.

Now the documentation for omdl can be compiled and installed with a single command. First download the source from Thingiverse or clone the source repository and install as follows:

```
$ git clone https://github.com/royasutton/omdl.git
$ cd omdl

$ make install
```

The library and documentation should now have been installed to the OpenSCAD *built-in* library location along with the reference manual that can be views with a web browser.

Have a look in:

- **Linux:** $HOME/.local/share/OpenSCAD/libraries
- **Windows:** My Documents\OpenSCAD\libraries

Now you may include the desired library primitives in your project as follows, replacing the version number as needed (multiple version of a library may coexists):

```
include <omdl-v0.4/shapes2de.scad>;
include <omdl-v0.4/shapes3d.scad>;
...
```

1.4 Contributing

omdl uses git for development tracking, and is hosted on GitHub following the usual practice of forking and submitting pull requests to the source repository.

As it is released under the GNU Lesser General Public License, any file you change should bear your copyright notice alongside the original authors' copyright notices typically located at the top of each file.

Ideas, requests, comments, contributions, and constructive criticism are welcome.

1.5 Contact and Support

In case you have any questions or would like to make feature requests, you can contact the maintainer of the project or file an `issue`.

2 Validation

- Primitives Validation
- Computations Validation

2.1 Primitives Validation

- Variable Tests
- Vector Operations

2.1.1 Variable Tests

- Validation Script (group1)
- Validation Results (group1)
- Validation Script (group2)
- Validation Results (group2)

2.1.1.1 Validation Script (group1)

```
include <primitives.scad>;
use <table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "An odd integer",             1],
  ["t03", "An small even integer",      10],
  ["t04", "A large integer",            100000000],
  ["t05", "A small decimal (epsilon)",  eps],
  ["t06", "The max number",             number_max],
  ["t07", "The min number",             number_min],
  ["t08", "The max number^2",           number_max *
number_max],
  ["t09", "The invalid number nan",     0 / 0],
  ["t10", "The boolean true",           true],
  ["t11", "The boolean false",          false],
  ["t12", "A character string",         "a"],
  ["t13", "A string",                   "This is a longer string"],
    ["t14", "The empty string",          empty_str],
    ["t15", "The empty vector",          empty_v],
    ["t16", "A 1-tuple vector of undef", [undef]],
    ["t17", "A 1-tuple vector",          [10]],
    ["t18", "A 3-tuple vector",          [1, 2, 3]],
    ["t19", "A vector of vectors",       [[1,2,3], [4,5,6], [7,8,9]]],
    ["t20", "A shorthand range",         [0:9]],
    ["t21", "A range",                   [0:0.5:9]]
  ];

  test_ids = table_get_row_ids( test_r );

  // expected columns: ("id" + one column for each test)
  good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

  // expected rows: ("golden" test results), use 's' to skip test
  t = true;   // shortcuts
  f = false;
  u = undef;
  s = -1;     // skip test

  good_r =
  [ // function     01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
    ["is_defined",  f, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t],
    ["not_defined", t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["is_empty",    f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, f, f, f, f, f, f],
    ["is_scalar",   t, t, t, t, t, t, t, t, t, t, t, f, f, f, f, f, f, f, f, s, s],
    ["is_iterable", f, f, f, f, f, f, f, f, f, f, f, t, t, t, t, t, t, t, t, s, s],
    ["is_string",   f, f, f, f, f, f, f, f, f, f, f, t, t, t, f, f, f, f, f, f, f],
    ["is_vector",   f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, t, t, t, s, s],
    ["is_boolean",  f, f, f, f, f, f, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f],
    ["is_integer",  f, t, t, t, f, t, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["is_decimal",  f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["is_number",   f, t, t, t, t, t, t, t, t, f, f, f, f, f, f, f, f, f, f, f, f],
    ["is_range",    f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t],
    ["is_nan",      f, f, f, f, f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f],
    ["is_inf",      f, f, f, f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["is_even",     s, f, t, t, f, t, t, s, s, s, s, s, s, s, s, s, s, s, s, s, s],
    ["is_odd",      s, t, f, f, f, f, f, s, s, s, s, s, s, s, s, s, s, s, s, s, s]
  ];

  // sanity-test tables
  table_check( test_r, test_c, false );
  table_check( good_r, good_c, false );

  // validate helper function and module
  function get_value( vid ) = table_get(test_r, test_c, vid, "tv");
  module run_test( fname, fresult, vid )
  {
    value_text = table_get(test_r, test_c, vid, "td");
    pass_value = table_get(good_r, good_c, fname, vid);

    test_pass = validate( cv=fresult, t=pass_value, pf=true );
    test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult,
pass_value );


    if ( pass_value != s )
    {
      if ( !test_pass )
        log_warn( str(vid, "(", value_text, ") ", test_text) );
      else if ( show_passing )
        log_info( str(vid, " ", test_text) );
    }
    else if ( show_skipped )
      log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
  }

  // Indirect function calls would be very useful here!!!
  for (vid=test_ids) run_test( "is_defined", is_defined(get_value(vid)), vid );
  for (vid=test_ids) run_test( "not_defined", not_defined(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_empty", is_empty(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_scalar", is_scalar(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_iterable", is_iterable(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_string", is_string(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_vector", is_vector(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_boolean", is_boolean(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_integer", is_integer(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_decimal", is_decimal(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_number", is_number(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_range", is_range(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_nan", is_nan(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_inf", is_inf(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_even", is_even(get_value(vid)), vid );
  for (vid=test_ids) run_test( "is_odd", is_odd(get_value(vid)), vid );

  // end-of-tests
```

2.1.1.2 Validation Results (group1)

```
ECHO: "OpenSCAD Version [2016, 12, 21]"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_defined(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_defined(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_defined(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_defined(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_defined(0.01)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_defined(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_defined(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_defined(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_defined(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_defined(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_defined(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_defined(a)=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_defined(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_defined()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_defined([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_defined([undef])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_defined([10])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_defined([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_defined([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_defined([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_defined([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'not_defined(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'not_defined(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'not_defined(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'not_defined(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'not_defined(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'not_defined(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'not_defined(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'not_defined(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'not_defined(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'not_defined(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'not_defined(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'not_defined(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'not_defined(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'not_defined()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'not_defined([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'not_defined([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'not_defined([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'not_defined([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'not_defined([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'not_defined([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'not_defined([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_empty(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_empty(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_empty(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_empty(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_empty(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_empty(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_empty(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_empty(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_empty(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_empty(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_empty(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_empty(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_empty(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_empty()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_empty([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_empty([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_empty([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_empty([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_empty([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_empty([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_empty([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_scalar(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_scalar(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_scalar(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_scalar(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_scalar(0.01)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_scalar(1e+308)=true'"
```

```
ECHO: "[ INFO ] run_test(); t07 passed: 'is_scalar(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_scalar(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_scalar(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_scalar(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_scalar(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_scalar(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_scalar(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_scalar()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_scalar([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_scalar([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_scalar([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_scalar([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_scalar([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_scalar(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_scalar(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_iterable(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_iterable(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_iterable(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_iterable(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_iterable(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_iterable(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_iterable(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_iterable(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_iterable(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_iterable(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_iterable(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_iterable(a)=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_iterable(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_iterable()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_iterable([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_iterable([undef])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_iterable([10])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_iterable([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_iterable([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=true'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_iterable(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_iterable(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_string(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_string(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_string(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_string(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_string(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_string(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_string(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_string(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_string(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_string(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_string(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_string(a)=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_string(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_string()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_string([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_string([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_string([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_string([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_string([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_string([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_string([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_vector(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_vector(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_vector(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_vector(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_vector(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_vector(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_vector(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_vector(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_vector(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_vector(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_vector(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_vector(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_vector(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_vector()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_vector([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_vector([undef])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_vector([10])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_vector([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_vector([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=true'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_vector(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_vector(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_boolean(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_boolean(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_boolean(10)=false'"
```

```
ECHO: "[ INFO ] run_test(); t04 passed: 'is_boolean(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_boolean(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_boolean(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_boolean(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_boolean(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_boolean(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_boolean(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_boolean(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_boolean(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_boolean(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_boolean()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_boolean([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_boolean([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_boolean([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_boolean([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_boolean([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_boolean([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_boolean([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_integer(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_integer(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_integer(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_integer(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_integer(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_integer(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_integer(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_integer(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_integer(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_integer(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_integer(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_integer(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_integer(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_integer()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_integer([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_integer([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_integer([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_integer([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_integer([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_integer([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_integer([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_decimal(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_decimal(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_decimal(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_decimal(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_decimal(0.01)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_decimal(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_decimal(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_decimal(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_decimal(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_decimal(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_decimal(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_decimal(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_decimal(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_decimal()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_decimal([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_decimal([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_decimal([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_decimal([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_decimal([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_decimal([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_decimal([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_number(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_number(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_number(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_number(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_number(0.01)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_number(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_number(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_number(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_number(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_number(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_number(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_number(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_number(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_number()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_number([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_number([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_number([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_number([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_number([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_number([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_number([0 : 0.5 : 9])=false'"
```

```
ECHO: "[ INFO ] run_test(); t01 passed: ’is_range(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’is_range(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’is_range(10)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’is_range(1e+08)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’is_range(0.01)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’is_range(1e+308)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’is_range(-1e+308)=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’is_range(inf)=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’is_range(nan)=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’is_range(true)=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’is_range(false)=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’is_range(a)=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’is_range(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’is_range()=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’is_range([])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’is_range([undef])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’is_range([10])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’is_range([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’is_range([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’is_range([0 : 1 : 9])=true’"
ECHO: "[ INFO ] run_test(); t21 passed: ’is_range([0 : 0.5 : 9])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’is_nan(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’is_nan(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’is_nan(10)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’is_nan(1e+08)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’is_nan(0.01)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’is_nan(1e+308)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’is_nan(-1e+308)=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’is_nan(inf)=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’is_nan(nan)=true’"
ECHO: "[ INFO ] run_test(); t10 passed: ’is_nan(true)=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’is_nan(false)=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’is_nan(a)=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’is_nan(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’is_nan()=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’is_nan([])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’is_nan([undef])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’is_nan([10])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’is_nan([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’is_nan([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’is_nan([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’is_nan([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’is_inf(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’is_inf(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’is_inf(10)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’is_inf(1e+08)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’is_inf(0.01)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’is_inf(1e+308)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’is_inf(-1e+308)=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’is_inf(inf)=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’is_inf(nan)=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’is_inf(true)=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’is_inf(false)=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’is_inf(a)=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’is_inf(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’is_inf()=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’is_inf([])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’is_inf([undef])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’is_inf([10])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’is_inf([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’is_inf([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’is_inf([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’is_inf([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t01 *skip*: ’is_even(The undefined value)’"
ECHO: "[ INFO ] run_test(); t02 passed: ’is_even(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’is_even(10)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’is_even(1e+08)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’is_even(0.01)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’is_even(1e+308)=true’"
ECHO: "[ INFO ] run_test(); t07 passed: ’is_even(-1e+308)=true’"
ECHO: "[ INFO ] run_test(); t08 *skip*: ’is_even(The max number^2)’"
ECHO: "[ INFO ] run_test(); t09 *skip*: ’is_even(The invalid number nan)’"
ECHO: "[ INFO ] run_test(); t10 *skip*: ’is_even(The boolean true)’"
ECHO: "[ INFO ] run_test(); t11 *skip*: ’is_even(The boolean false)’"
ECHO: "[ INFO ] run_test(); t12 *skip*: ’is_even(A character string)’"
ECHO: "[ INFO ] run_test(); t13 *skip*: ’is_even(A string)’"
ECHO: "[ INFO ] run_test(); t14 *skip*: ’is_even(The empty string)’"
ECHO: "[ INFO ] run_test(); t15 *skip*: ’is_even(The empty vector)’"
ECHO: "[ INFO ] run_test(); t16 *skip*: ’is_even(A 1-tuple vector of undef)’"
ECHO: "[ INFO ] run_test(); t17 *skip*: ’is_even(A 1-tuple vector)’"
ECHO: "[ INFO ] run_test(); t18 *skip*: ’is_even(A 3-tuple vector)’"
```

```
ECHO: "[ INFO ] run_test(); t19 *skip*: 'is_even(A vector of vectors)'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_even(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_even(A range)'"
ECHO: "[ INFO ] run_test(); t01 *skip*: 'is_odd(The undefined value)'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_odd(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_odd(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_odd(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_odd(0.01)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_odd(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_odd(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'is_odd(The max number^2)'"
ECHO: "[ INFO ] run_test(); t09 *skip*: 'is_odd(The invalid number nan)'"
ECHO: "[ INFO ] run_test(); t10 *skip*: 'is_odd(The boolean true)'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'is_odd(The boolean false)'"
ECHO: "[ INFO ] run_test(); t12 *skip*: 'is_odd(A character string)'"
ECHO: "[ INFO ] run_test(); t13 *skip*: 'is_odd(A string)'"
ECHO: "[ INFO ] run_test(); t14 *skip*: 'is_odd(The empty string)'"
ECHO: "[ INFO ] run_test(); t15 *skip*: 'is_odd(The empty vector)'"
ECHO: "[ INFO ] run_test(); t16 *skip*: 'is_odd(A 1-tuple vector of undef)'"
ECHO: "[ INFO ] run_test(); t17 *skip*: 'is_odd(A 1-tuple vector)'"
ECHO: "[ INFO ] run_test(); t18 *skip*: 'is_odd(A 3-tuple vector)'"
ECHO: "[ INFO ] run_test(); t19 *skip*: 'is_odd(A vector of vectors)'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_odd(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_odd(A range)'"
```

2.1.1.3 Validation Script (group2)

```
include <primitives.scad>;
use <table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "An odd integer",             1],
  ["t03", "The boolean true",           true],
  ["t04", "The boolean false",          false],
  ["t05", "A character string",         "a"],
  ["t06", "A string",                   "This is a longer string"],
  ["t07", "The empty string",           empty_str],
  ["t08", "The empty vector",           empty_v],
  ["t09", "A shorthand range",          [0:9]],
  ["t10", "A range",                    [0:0.5:9]],
  ["t11", "Test vector 01",             [undef]],
  ["t12", "Test vector 02",             [1]],
  ["t13", "Test vector 03",             [1, 2, 3]],
  ["t14", "Test vector 04",             [[1], [2], [3], [4], [5]]],
  ["t15", "Test vector 05",             [[1,2], [2,3]]],
  ["t16", "Test vector 06",             [[1,2], [2,3], [4,5], "ab"]],
  ["t17", "Test vector 07",             [[1,2,3], [4,5,6], [7,8,9], ["a", "b", "c"]]],
  ["t18", "Test vector 08",             [1, 2, 3, undef]],
  ["t19", "Test vector 09",             [undef, undef, undef, undef]],
  ["t20", "Test vector 10",             [[undef], [undef], [undef]]],
  ["t21", "Test vector 11",             [true, true, true, false]],
  ["t22", "Test vector 12",             [true, false, false, false, false]],
  ["t23", "Test vector 13",             [true, true, true, true]]
];

test_ids = table_get_row_ids( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
t = true;   // shortcuts
```

```
  f = false;
  u = undef;
  s = -1;     // skip test

  good_r =
  [ // function         01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    ["all_equal_T",      f, f, t, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t],
    ["all_equal_F",      f, f, f, t, f, f, t, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["all_equal_U",      t, f, f, f, f, f, t, t, f, f, t, f, f, f, f, f, f, f, t, f, f, f, f],
    ["any_equal_T",      f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, t],
    ["any_equal_F",      f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, f],
    ["any_equal_U",      t, f, f, f, f, f, f, f, f, f, t, f, f, f, f, f, f, t, t, f, f, f, f],
    ["all_defined",      f, t, t, t, t, t, t, t, t, t, f, t, t, t, t, t, t, f, f, t, t, t, t],
    ["any_undefined",    t, f, f, f, f, f, f, f, f, f, t, f, f, f, f, f, f, t, t, f, f, f, f],
    ["all_scalars",      u, t, t, t, f, f, s, s, s, s, t, t, t, f, f, f, f, t, t, f, t, t, t],
    ["all_vectors",      u, f, f, f, f, f, t, t, f, f, f, f, f, t, t, f, t, f, f, t, f, f, f],
    ["all_strings",      u, f, f, f, t, t, t, s, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["all_numbers",      u, t, f, f, f, f, s, s, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f],
    ["all_len_1",        u, f, f, f, t, t, s, s, f, f, f, f, f, t, f, f, f, f, f, t, f, f, f],
    ["all_len_2",        u, f, f, f, f, f, s, s, f, f, f, f, f, f, t, t, f, f, f, f, f, f, f],
    ["all_len_3",        u, f, f, f, f, f, s, s, f, f, f, f, f, f, f, f, t, f, f, f, f, f, f],
    ["almost_equal_AA",  t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t],
    ["almost_equal_T",   f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["almost_equal_F",   f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["almost_equal_U",   t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
    ["compare_AA",       t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t]
  ];

  // sanity-test tables
  table_check( test_r, test_c, false );
  table_check( good_r, good_c, false );

  // validate helper function and module
  function get_value( vid ) = table_get(test_r, test_c, vid, "tv");
  module run_test( fname, fresult, vid )
  {
    value_text = table_get(test_r, test_c, vid, "td");
    pass_value = table_get(good_r, good_c, fname, vid);

    test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
    test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals
", pass_value );

    if ( pass_value != s )
    {
      if ( !test_pass )
        log_warn( str(vid, "(", value_text, ") ", test_text) );
      else if ( show_passing )
        log_info( str(vid, " ", test_text) );
    }
    else if ( show_skipped )
      log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
  }

  // Indirect function calls would be very useful here!!!
  for (vid=test_ids) run_test( "all_equal_T", all_equal(get_value(vid),t), vid );
  for (vid=test_ids) run_test( "all_equal_F", all_equal(get_value(vid),f), vid );
  for (vid=test_ids) run_test( "all_equal_U", all_equal(get_value(vid),u), vid );
  for (vid=test_ids) run_test( "any_equal_T", any_equal(get_value(vid),t), vid );
  for (vid=test_ids) run_test( "any_equal_F", any_equal(get_value(vid),f), vid );
  for (vid=test_ids) run_test( "any_equal_U", any_equal(get_value(vid),u), vid );
  for (vid=test_ids) run_test( "all_defined", all_defined(get_value(vid)), vid );
  for (vid=test_ids) run_test( "any_undefined", any_undefined(get_value(vid)), vid );
  for (vid=test_ids) run_test( "all_scalars", all_scalars(get_value(vid)), vid );
  for (vid=test_ids) run_test( "all_vectors", all_vectors(get_value(vid)), vid );
  for (vid=test_ids) run_test( "all_strings", all_strings(get_value(vid)), vid );
  for (vid=test_ids) run_test( "all_numbers", all_numbers(get_value(vid)), vid );
  for (vid=test_ids) run_test( "all_len_1", all_len(get_value(vid),1), vid );
  for (vid=test_ids) run_test( "all_len_2", all_len(get_value(vid),2), vid );
  for (vid=test_ids) run_test( "all_len_3", all_len(get_value(vid),3), vid );
  for (vid=test_ids) run_test( "almost_equal_AA", almost_equal(get_value(vid),get_value(
vid)), vid );
  for (vid=test_ids) run_test( "almost_equal_T", almost_equal(get_value(vid),t), vid );
  for (vid=test_ids) run_test( "almost_equal_F", almost_equal(get_value(vid),f), vid );
  for (vid=test_ids) run_test( "almost_equal_U", almost_equal(get_value(vid),u), vid );
  for (vid=test_ids) run_test( "compare_AA", compare(get_value(vid),get_value(vid)) == 0, vid
);

  // end-of-tests
```

2.1.1.4 Validation Results (group2)

```
ECHO: "OpenSCAD Version [2016, 12, 21]"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_equal_T(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_equal_T(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_equal_T(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_equal_T(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_equal_T(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_equal_T(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’all_equal_T()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’all_equal_T([])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_equal_T([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_equal_T([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_equal_T([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_equal_T([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_equal_T([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_equal_T([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_equal_T([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_equal_T([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_equal_T([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_equal_T([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_equal_T([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_equal_T([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_equal_T([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_equal_T([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_equal_T([true, true, true, true])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_equal_F(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_equal_F(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_equal_F(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_equal_F(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_equal_F(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_equal_F(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’all_equal_F()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’all_equal_F([])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_equal_F([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_equal_F([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_equal_F([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_equal_F([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_equal_F([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_equal_F([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_equal_F([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_equal_F([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_equal_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_equal_F([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_equal_F([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_equal_F([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_equal_F([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_equal_F([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_equal_F([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_equal_U(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_equal_U(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_equal_U(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_equal_U(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_equal_U(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_equal_U(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’all_equal_U()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’all_equal_U([])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_equal_U([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_equal_U([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_equal_U([undef])=true’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_equal_U([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_equal_U([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_equal_U([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_equal_U([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_equal_U([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_equal_U([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_equal_U([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_equal_U([undef, undef, undef, undef])=true’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_equal_U([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_equal_U([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_equal_U([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_equal_U([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’any_equal_T(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’any_equal_T(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’any_equal_T(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’any_equal_T(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’any_equal_T(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’any_equal_T(This is a longer string)=false’"
```

```
ECHO: "[ INFO ] run_test(); t07 passed: ’any_equal_T()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’any_equal_T([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’any_equal_T([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’any_equal_T([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’any_equal_T([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’any_equal_T([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’any_equal_T([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’any_equal_T([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’any_equal_T([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’any_equal_T([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’any_equal_T([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’any_equal_T([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’any_equal_T([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’any_equal_T([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’any_equal_T([true, true, true, true, false])=true’"
ECHO: "[ INFO ] run_test(); t22 passed: ’any_equal_T([true, false, false, false, false])=true’"
ECHO: "[ INFO ] run_test(); t23 passed: ’any_equal_T([true, true, true, true])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’any_equal_F(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’any_equal_F(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’any_equal_F(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’any_equal_F(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’any_equal_F(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’any_equal_F(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’any_equal_F()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’any_equal_F([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’any_equal_F([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’any_equal_F([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’any_equal_F([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’any_equal_F([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’any_equal_F([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’any_equal_F([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’any_equal_F([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’any_equal_F([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’any_equal_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’any_equal_F([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’any_equal_F([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’any_equal_F([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’any_equal_F([true, true, true, true, false])=true’"
ECHO: "[ INFO ] run_test(); t22 passed: ’any_equal_F([true, false, false, false, false])=true’"
ECHO: "[ INFO ] run_test(); t23 passed: ’any_equal_F([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’any_equal_U(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’any_equal_U(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’any_equal_U(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’any_equal_U(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’any_equal_U(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’any_equal_U(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’any_equal_U()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’any_equal_U([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’any_equal_U([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’any_equal_U([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’any_equal_U([undef])=true’"
ECHO: "[ INFO ] run_test(); t12 passed: ’any_equal_U([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’any_equal_U([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’any_equal_U([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’any_equal_U([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’any_equal_U([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’any_equal_U([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’any_equal_U([1, 2, 3, undef])=true’"
ECHO: "[ INFO ] run_test(); t19 passed: ’any_equal_U([undef, undef, undef, undef])=true’"
ECHO: "[ INFO ] run_test(); t20 passed: ’any_equal_U([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’any_equal_U([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’any_equal_U([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’any_equal_U([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_defined(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_defined(1)=true’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_defined(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_defined(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_defined(a)=true’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_defined(This is a longer string)=true’"
ECHO: "[ INFO ] run_test(); t07 passed: ’all_defined()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’all_defined([])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_defined([0 : 1 : 9])=true’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_defined([0 : 0.5 : 9])=true’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_defined([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_defined([1])=true’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_defined([1, 2, 3])=true’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_defined([[1], [2], [3], [4], [5]])=true’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_defined([[1, 2], [2, 3]])=true’"
```

```
ECHO: "[ INFO ] run_test(); t16 passed: ’all_defined([[1, 2], [2, 3], [4, 5], "ab"])=true’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_defined([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=true’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_defined([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_defined([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_defined([[undef], [undef], [undef]])=true’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_defined([true, true, true, true, false])=true’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_defined([true, false, false, false, false])=true’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_defined([true, true, true, true])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’any_undefined(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’any_undefined(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’any_undefined(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’any_undefined(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’any_undefined(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’any_undefined(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’any_undefined()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’any_undefined([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’any_undefined([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’any_undefined([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’any_undefined([undef])=true’"
ECHO: "[ INFO ] run_test(); t12 passed: ’any_undefined([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’any_undefined([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’any_undefined([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’any_undefined([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’any_undefined([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’any_undefined([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’any_undefined([1, 2, 3, undef])=true’"
ECHO: "[ INFO ] run_test(); t19 passed: ’any_undefined([undef, undef, undef, undef])=true’"
ECHO: "[ INFO ] run_test(); t20 passed: ’any_undefined([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’any_undefined([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’any_undefined([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’any_undefined([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_scalars(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_scalars(1)=true’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_scalars(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_scalars(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_scalars(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_scalars(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 *skip*: ’all_scalars(The empty string)’"
ECHO: "[ INFO ] run_test(); t08 *skip*: ’all_scalars(The empty vector)’"
ECHO: "[ INFO ] run_test(); t09 *skip*: ’all_scalars(A shorthand range)’"
ECHO: "[ INFO ] run_test(); t10 *skip*: ’all_scalars(A range)’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_scalars([undef])=true’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_scalars([1])=true’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_scalars([1, 2, 3])=true’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_scalars([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_scalars([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_scalars([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_scalars([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_scalars([1, 2, 3, undef])=true’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_scalars([undef, undef, undef, undef])=true’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_scalars([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_scalars([true, true, true, true, false])=true’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_scalars([true, false, false, false, false])=true’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_scalars([true, true, true, true])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_vectors(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_vectors(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_vectors(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_vectors(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_vectors(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_vectors(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’all_vectors()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’all_vectors([])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_vectors([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_vectors([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_vectors([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_vectors([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_vectors([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_vectors([[1], [2], [3], [4], [5]])=true’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_vectors([[1, 2], [2, 3]])=true’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_vectors([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_vectors([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=true’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_vectors([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_vectors([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_vectors([[undef], [undef], [undef]])=true’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_vectors([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_vectors([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_vectors([true, true, true, true])=false’"
```

```
ECHO: "[ INFO ] run_test(); t01 passed: ’all_strings(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_strings(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_strings(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_strings(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_strings(a)=true’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_strings(This is a longer string)=true’"
ECHO: "[ INFO ] run_test(); t07 passed: ’all_strings()=true’"
ECHO: "[ INFO ] run_test(); t08 *skip*: ’all_strings(The empty vector)’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_strings([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_strings([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_strings([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_strings([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_strings([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_strings([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_strings([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_strings([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_strings([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_strings([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_strings([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_strings([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_strings([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_strings([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_strings([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_numbers(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_numbers(1)=true’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_numbers(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_numbers(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_numbers(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_numbers(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 *skip*: ’all_numbers(The empty string)’"
ECHO: "[ INFO ] run_test(); t08 *skip*: ’all_numbers(The empty vector)’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_numbers([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_numbers([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_numbers([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_numbers([1])=true’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_numbers([1, 2, 3])=true’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_numbers([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_numbers([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_numbers([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_numbers([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_numbers([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_numbers([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_numbers([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_numbers([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_numbers([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_numbers([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_len_1(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_len_1(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_len_1(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_len_1(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_len_1(a)=true’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_len_1(This is a longer string)=true’"
ECHO: "[ INFO ] run_test(); t07 *skip*: ’all_len_1(The empty string)’"
ECHO: "[ INFO ] run_test(); t08 *skip*: ’all_len_1(The empty vector)’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_len_1([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’all_len_1([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’all_len_1([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’all_len_1([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’all_len_1([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’all_len_1([[1], [2], [3], [4], [5]])=true’"
ECHO: "[ INFO ] run_test(); t15 passed: ’all_len_1([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’all_len_1([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’all_len_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’all_len_1([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’all_len_1([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’all_len_1([[undef], [undef], [undef]])=true’"
ECHO: "[ INFO ] run_test(); t21 passed: ’all_len_1([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’all_len_1([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’all_len_1([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’all_len_2(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’all_len_2(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’all_len_2(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’all_len_2(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’all_len_2(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’all_len_2(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 *skip*: ’all_len_2(The empty string)’"
ECHO: "[ INFO ] run_test(); t08 *skip*: ’all_len_2(The empty vector)’"
ECHO: "[ INFO ] run_test(); t09 passed: ’all_len_2([0 : 1 : 9])=false’"
```

```
ECHO: "[ INFO ] run_test(); t10 passed: 'all_len_2([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_len_2([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_len_2([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_len_2([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_len_2([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_len_2([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_len_2([[1, 2], [2, 3], [4, 5], "ab"])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_len_2([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_len_2([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_len_2([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_len_2([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_len_2([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_len_2([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_len_2([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_len_3(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_len_3(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_len_3(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_len_3(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_len_3(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_len_3(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'all_len_3(The empty string)'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_len_3(The empty vector)'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_len_3([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_len_3([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_len_3([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_len_3([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_len_3([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_len_3([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_len_3([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_len_3([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_len_3([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_len_3([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_len_3([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_len_3([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_len_3([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_len_3([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_len_3([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'almost_equal_AA(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'almost_equal_AA(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'almost_equal_AA(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'almost_equal_AA(false)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'almost_equal_AA(a)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'almost_equal_AA(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'almost_equal_AA()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'almost_equal_AA([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'almost_equal_AA([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'almost_equal_AA([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'almost_equal_AA([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'almost_equal_AA([1])=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'almost_equal_AA([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'almost_equal_AA([[1], [2], [3], [4], [5]])=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'almost_equal_AA([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'almost_equal_AA([[1, 2], [2, 3], [4, 5], "ab"])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'almost_equal_AA([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'almost_equal_AA([1, 2, 3, undef])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'almost_equal_AA([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'almost_equal_AA([[undef], [undef], [undef]])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'almost_equal_AA([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'almost_equal_AA([true, false, false, false, false])=true'"
ECHO: "[ INFO ] run_test(); t23 passed: 'almost_equal_AA([true, true, true, true])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'almost_equal_T(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'almost_equal_T(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'almost_equal_T(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'almost_equal_T(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'almost_equal_T(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'almost_equal_T(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'almost_equal_T()=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'almost_equal_T([])=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'almost_equal_T([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'almost_equal_T([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'almost_equal_T([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'almost_equal_T([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'almost_equal_T([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'almost_equal_T([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'almost_equal_T([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'almost_equal_T([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'almost_equal_T([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false'"
```

```
ECHO: "[ INFO ] run_test(); t18 passed: ’almost_equal_T([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’almost_equal_T([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’almost_equal_T([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’almost_equal_T([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’almost_equal_T([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_T([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’almost_equal_F(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’almost_equal_F(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’almost_equal_F(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’almost_equal_F(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’almost_equal_F(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’almost_equal_F(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’almost_equal_F()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’almost_equal_F([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’almost_equal_F([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’almost_equal_F([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’almost_equal_F([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’almost_equal_F([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’almost_equal_F([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’almost_equal_F([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’almost_equal_F([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’almost_equal_F([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’almost_equal_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’almost_equal_F([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’almost_equal_F([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’almost_equal_F([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’almost_equal_F([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’almost_equal_F([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_F([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’almost_equal_U(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’almost_equal_U(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’almost_equal_U(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’almost_equal_U(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’almost_equal_U(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’almost_equal_U(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’almost_equal_U()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’almost_equal_U([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’almost_equal_U([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’almost_equal_U([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’almost_equal_U([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’almost_equal_U([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’almost_equal_U([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’almost_equal_U([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’almost_equal_U([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’almost_equal_U([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’almost_equal_U([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’almost_equal_U([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’almost_equal_U([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’almost_equal_U([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’almost_equal_U([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’almost_equal_U([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_U([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’compare_AA(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’compare_AA(1)=true’"
ECHO: "[ INFO ] run_test(); t03 passed: ’compare_AA(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’compare_AA(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’compare_AA(a)=true’"
ECHO: "[ INFO ] run_test(); t06 passed: ’compare_AA(This is a longer string)=true’"
ECHO: "[ INFO ] run_test(); t07 passed: ’compare_AA()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’compare_AA([])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’compare_AA([0 : 1 : 9])=true’"
ECHO: "[ INFO ] run_test(); t10 passed: ’compare_AA([0 : 0.5 : 9])=true’"
ECHO: "[ INFO ] run_test(); t11 passed: ’compare_AA([undef])=true’"
ECHO: "[ INFO ] run_test(); t12 passed: ’compare_AA([1])=true’"
ECHO: "[ INFO ] run_test(); t13 passed: ’compare_AA([1, 2, 3])=true’"
ECHO: "[ INFO ] run_test(); t14 passed: ’compare_AA([[1], [2], [3], [4], [5]])=true’"
ECHO: "[ INFO ] run_test(); t15 passed: ’compare_AA([[1, 2], [2, 3]])=true’"
ECHO: "[ INFO ] run_test(); t16 passed: ’compare_AA([[1, 2], [2, 3], [4, 5], "ab"])=true’"
ECHO: "[ INFO ] run_test(); t17 passed: ’compare_AA([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=true’"
ECHO: "[ INFO ] run_test(); t18 passed: ’compare_AA([1, 2, 3, undef])=true’"
ECHO: "[ INFO ] run_test(); t19 passed: ’compare_AA([undef, undef, undef, undef])=true’"
ECHO: "[ INFO ] run_test(); t20 passed: ’compare_AA([[undef], [undef], [undef]])=true’"
ECHO: "[ INFO ] run_test(); t21 passed: ’compare_AA([true, true, true, true, false])=true’"
ECHO: "[ INFO ] run_test(); t22 passed: ’compare_AA([true, false, false, false, false])=true’"
ECHO: "[ INFO ] run_test(); t23 passed: ’compare_AA([true, true, true, true])=true’"
```

2.1.2 Vector Operations

- Validation Script
- Validation Results

2.1.2.1 Validation Script

```
include <primitives.scad>;
use <table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "The empty vector",           empty_v],
  ["t03", "A range",                    [0:0.5:9]],
  ["t04", "A string",                   "A string"],
  ["t05", "Test vector 01",             ["orange","apple","grape","banana"]],
  ["t06", "Test vector 02",             ["b","a","n","a","n","a","s"]],
  ["t07", "Test vector 03",             [undef]],
  ["t08", "Test vector 04",             [[1,2],[2,3]]],
  ["t09", "Test vector 05",             ["ab",[1,2],[2,3],[4,5]]],
  ["t10", "Test vector 06",             [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]]],
  ["t11", "Vector of integers 0 to 15", [for (i=[0:15]) i]]
];

test_ids = table_get_row_ids( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["consts",
    empty_v,                                        // t01
    empty_v,                                        // t02
    empty_v,                                        // t03
    empty_v,                                        // t04
    empty_v,                                        // t05
    empty_v,                                        // t06
    empty_v,                                        // t07
    empty_v,                                        // t08
    empty_v,                                        // t09
    empty_v,                                        // t10
    empty_v                                         // t11
  ],
  ["vstr",
    undef,                                          // t01
    empty_str,                                      // t02
    "[0 : 0.5 : 9]",                                // t03
    "A string",                                     // t04
    "orangeapplegrapebanana",                       // t05
    "bananas",                                      // t06
    "undef",                                        // t07
    "[1, 2][2, 3]",                                 // t08
    "ab[1, 2][2, 3][4, 5]",                         // t09
    "[1, 2, 3][4, 5, 6][7, 8, 9][\"a\", \"b\", \"c\"]", // t10
    "0123456789101112131415"                        // t11
  ],
  ["sum",
    undef,                                               // t01
    0,                                                   // t02
    85.5,                                                // t03
    undef,                                               // t04
    undef,                                               // t05
    undef,                                               // t06
    undef,                                               // t07
    [3,5],                                               // t08
    undef,                                               // t09
    [undef,undef,undef],                                 // t10
    120                                                  // t11
  ],
  ["find_12",
    empty_v,                                             // t01
    empty_v,                                             // t02
    empty_v,                                             // t03
    empty_v,                                             // t04
    empty_v,                                             // t05
    empty_v,                                             // t06
    empty_v,                                             // t07
    [0],                                                 // t08
    [1],                                                 // t09
    empty_v,                                             // t10
    empty_v                                              // t11
  ],
  ["count_S1",
    0,                                                   // t01
    0,                                                   // t02
    0,                                                   // t03
    0,                                                   // t04
    0,                                                   // t05
    0,                                                   // t06
    0,                                                   // t07
    1,                                                   // t08
    1,                                                   // t09
    1,                                                   // t10
    1                                                    // t11
  ],
  ["exists_S1",
    false,                                               // t01
    false,                                               // t02
    false,                                               // t03
    false,                                               // t04
    false,                                               // t05
    false,                                               // t06
    false,                                               // t07
    true,                                                // t08
    true,                                                // t09
    true,                                                // t10
    true                                                 // t11
  ],
  ["defined_or_D",
    "default",                                           // t01
    empty_v,                                             // t02
    [0:0.5:9],                                           // t03
    "A string",                                          // 04
    ["orange","apple","grape","banana"],                 // t05
    ["b","a","n","a","n","a","s"],                       // t06
    [undef],                                             // t07
    [[1,2],[2,3]],                                       // t08
    ["ab",[1,2],[2,3],[4,5]],                            // t09
    [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],             // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]              // t11
  ],
  ["edefined_or_DE3",
    "default",                                           // t01
    "default",                                           // t02
    "default",                                           // t03
    "t",                                                 // t04
    "banana",                                            // t05
    "a",                                                 // t06
    "default",                                           // t07
    "default",                                           // t08
    [4,5],                                               // t09
    ["a","b","c"],                                       // t10
    3                                                    // t11
  ],
  ["first",
    undef,                                               // t01
    undef,                                               // t02
```

```
    undef,                                              // t03
    "A",                                                // t04
    "orange",                                           // t05
    "b",                                                // t06
    undef,                                              // t07
    [1,2],                                              // t08
    "ab",                                               // t09
    [1,2,3],                                            // t10
    0                                                   // t11
  ],
  ["second",
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    " ",                                                // t04
    "apple",                                            // t05
    "a",                                                // t06
    undef,                                              // t07
    [2,3],                                              // t08
    [1,2],                                              // t09
    [4,5,6],                                            // t10
    1                                                   // t11
  ],
  ["last",
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    "g",                                                // t04
    "banana",                                           // t05
    "s",                                                // t06
    undef,                                              // t07
    [2,3],                                              // t08
    [4,5],                                              // t09
    ["a","b","c"],                                      // t10
    15                                                  // t11
  ],
  ["nfirst_1",
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    ["A"],                                              // t04
    ["orange"],                                         // t05
    ["b"],                                              // t06
    [undef],                                            // t07
    [[1,2]],                                            // t08
    ["ab"],                                             // t09
    [[1,2,3]],                                          // t10
    [0]                                                 // t11
  ],
  ["nlast_1",
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    ["g"],                                              // t04
    ["banana"],                                         // t05
    ["s"],                                              // t06
    [undef],                                            // t07
    [[2,3]],                                            // t08
    [[4,5]],                                            // t09
    [["a","b","c"]],                                    // t10
    [15]                                                // t11
  ],
  ["nhead_1",
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    ["A"," ","s","t","r","i","n"],                      // t04
    ["orange","apple","grape"],                         // t05
    ["b","a","n","a","n","a"],                          // t06
    empty_v,                                            // t07
    [[1,2]],                                            // t08
    ["ab",[1,2],[2,3]],                                 // t09
    [[1,2,3],[4,5,6],[7,8,9]],                          // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]                // t11
  ],
  ["ntail_1",
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    [" ","s","t","r","i","n","g"],                      // t04
    ["apple","grape","banana"],                         // t05
```

```
      ["a","n","a","n","a","s"],                      // t06
      empty_v,                                        // t07
      [[2,3]],                                        // t08
      [[1,2],[2,3],[4,5]],                            // t09
      [[4,5,6],[7,8,9],["a","b","c"]],                // t10
      [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]           // t11
    ],
    ["rselect_02",
      undef,                                          // t01
      empty_v,                                        // t02
      undef,                                          // t03
      ["A"," ","s"],                                  // t04
      ["orange","apple","grape"],                     // t05
      ["b","a","n"],                                  // t06
      undef,                                          // t07
      undef,                                          // t08
      ["ab",[1,2],[2,3]],                             // t09
      [[1,2,3],[4,5,6],[7,8,9]],                      // t10
      [0,1,2]                                         // t11
    ],
    ["eselect_F",
      undef,                                          // t01
      empty_v,                                        // t02
      undef,                                          // t03
      ["A"," ","s","t","r","i","n","g"],              // t04
      ["o","a","g","b"],                              // t05
      ["b","a","n","a","n","a","s"],                  // t06
      [undef],                                        // t07
      [1,2],                                          // t08
      ["a",1,2,4],                                    // t09
      [1,4,7,"a"],                                    // t10
      skip                                            // t11
    ],
    ["eselect_L",
      undef,                                          // t01
      empty_v,                                        // t02
      undef,                                          // t03
      ["A"," ","s","t","r","i","n","g"],              // t04
      ["e","e","e","a"],                              // t05
      ["b","a","n","a","n","a","s"],                  // t06
      [undef],                                        // t07
      [2,3],                                          // t08
      ["b",2,3,5],                                    // t09
      [3,6,9,"c"],                                    // t10
      skip                                            // t11
    ],
    ["eselect_1",
      undef,                                          // t01
      empty_v,                                        // t02
      undef,                                          // t03
      skip,                                           // t04
      ["r","p","r","a"],                              // t05
      skip,                                           // t06
      [undef],                                        // t07
      [2,3],                                          // t08
      ["b",2,3,5],                                    // t09
      [2,5,8,"b"],                                    // t10
      skip                                            // t11
    ],
    ["smerge",
      undef,                                          // t01
      empty_v,                                        // t02
      [[0:0.5:9]],                                    // t03
      ["A string"],                                   // t04
      ["orange","apple","grape","banana"],            // t05
      ["b","a","n","a","n","a","s"],                  // t06
      [undef],                                        // t07
      [1,2,2,3],                                      // t08
      ["ab",1,2,2,3,4,5],                             // t09
      [1,2,3,4,5,6,7,8,9,"a","b","c"],                // t10
      [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]         // t11
    ],
    ["pmerge",
      undef,                                          // t01
      empty_v,                                        // t02
      undef,                                          // t03
      ["A string"],                                   // t04
      [
        ["o","a","g","b"],["r","p","r","a"],
        ["a","p","a","n"],["n","l","p","a"],
        ["g","e","e","n"]
      ],                                                   // t05
      [["b","a","n","a","n","a","s"]],                     // t06
      undef,                                               // t07
      [[1,2],[2,3]],                                       // t08
      [["a",1,2,4],["b",2,3,5]],                           // t09
      [[1,4,7,"a"],[2,5,8,"b"],[3,6,9,"c"]],               // t10
      undef                                                // t11
    ],
    ["reverse",
      undef,                                               // t01
      empty_v,                                             // t02
      undef,                                               // t03
      ["g","n","i","r","t","s"," ","A"],                   // t04
      ["banana","grape","apple","orange"],                 // t05
      ["s","a","n","a","n","a","b"],                       // t06
      [undef],                                             // t07
      [[2,3],[1,2]],                                       // t08
      [[4,5],[2,3],[1,2],"ab"],                            // t09
      [["a","b","c"],[7,8,9],[4,5,6],[1,2,3]],             // t10
      [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]              // t11
    ],
    ["qsort",
      undef,                                               // t01
      empty_v,                                             // t02
      undef,                                               // t03
      undef,                                               // t04
      ["apple","banana","grape","orange"],                 // t05
      ["a","a","a","b","n","n","s"],                       // t06
      undef,                                               // t07
      undef,                                               // t08
      undef,                                               // t09
      undef,                                               // t10
      [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]              // t11
    ],
    ["qsort2_HR",
      undef,                                               // t01
      empty_v,                                             // t02
      undef,                                               // t03
      undef,                                               // t04
      ["orange","grape","banana","apple"],                 // t05
      ["s","n","n","b","a","a","a"],                       // t06
      [undef],                                             // t07
      [[3,2],[2,1]],                                       // t08
      [[5,4],[3,2],[2,1],"ab"],                            // t09
      [["c","b","a"],[9,8,7],[6,5,4],[3,2,1]],             // t10
      [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]              // t11
    ],
    ["strip",
      undef,                                               // t01
      empty_v,                                             // t02
      undef,                                               // t03
      ["A"," ","s","t","r","i","n","g"],                   // t04
      ["orange","apple","grape","banana"],                 // t05
      ["b","a","n","a","n","a","s"],                       // t06
      [undef],                                             // t07
      [[1,2],[2,3]],                                       // t08
      ["ab",[1,2],[2,3],[4,5]],                            // t09
      [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],             // t10
      [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]              // t11
    ],
    ["eappend_T0",
      undef,                                               // t01
      [[0]],                                               // t02
      undef,                                               // t03
      [
        ["A",0],[" ",0],["s",0],["t",0],
        ["r",0],["i",0],["n",0],["g",0]
      ],                                                   // t04
      [
        ["orange",0],["apple",0],
        ["grape",0],["banana",0]
      ],                                                   // t05
      [
        ["b",0],["a",0],["n",0],["a",0],
        ["n",0],["a",0],["s",0]
      ],                                                   // t06
      [[undef,0]],                                         // t07
      [[1,2,0],[2,3,0]],                                   // t08
      [["ab",0],[1,2,0],[2,3,0],[4,5,0]],                  // t09
      [[1,2,3,0],[4,5,6,0],[7,8,9,0],["a","b","c",0]],     // t10
      [
        [0,0],[1,0],[2,0],[3,0],[4,0],[5,0],
        [6,0],[7,0],[8,0],[9,0],[10,0],[11,0],
        [12,0],[13,0],[14,0],[15,0]
      ]                                                    // t11
    ],
    ["insert_T0",
      undef,                                               // t01
      undef,                                               // t02
      undef,                                               // t03
      undef,                                               // t04
      ["orange",0,"apple","grape","banana"],               // t05
      ["b","a","n","a","n","a",0,"s"],                     // t06
      undef,                                               // t07
      [[1,2],0,[2,3]],                                     // t08
      ["ab",[1,2],0,[2,3],[4,5]],                          // t09
      undef,                                               // t10
      [0,1,2,3,4,0,5,6,7,8,9,10,11,12,13,14,15]            // t11
    ],
    ["delete_T0",
      undef,                                               // t01
      empty_v,                                             // t02
      undef,                                               // t03
      ["A"," ","s","t","r","i","n","g"],                   // t04
      ["orange","grape","banana"],                         // t05
      ["b","a","n","a","n","a"],                           // t06
      [undef],                                             // t07
      [[1,2]],                                             // t08
      ["ab",[1,2],[4,5]],                                  // t09
      [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],             // t10
      [0,1,2,3,4,6,7,8,9,10,11,12,13,14,15]                // t11
    ],
    ["unique",
      undef,                                               // t01
      empty_v,                                             // t02
      undef,                                               // t03
      ["A"," ","s","t","r","i","n","g"],                   // t04
      ["orange","apple","grape","banana"],                 // t05
      ["b","a","n","s"],                                   // t06
      [undef],                                             // t07
      [[1,2],[2,3]],                                       // t08
      ["ab",[1,2],[2,3],[4,5]],                            // t09
      [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],             // t10
      [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]              // t11
    ]
  ];

  // sanity-test tables
  table_check( test_r, test_c, false );
  table_check( good_r, good_c, false );

  // validate helper function and module
  function get_value( vid ) = table_get(test_r, test_c, vid, "tv");
  module run_test( fname, fresult, vid )
  {
    value_text = table_get(test_r, test_c, vid, "td");
    pass_value = table_get(good_r, good_c, fname, vid);

    test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
    test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals",
   pass_value );

    if ( pass_value != skip )
    {
      if ( !test_pass )
        log_warn( str(vid, "(", value_text, ") ", test_text) );
      else if ( show_passing )
        log_info( str(vid, " ", test_text) );
    }
    else if ( show_skipped )
      log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
  }

  // Indirect function calls would be very useful here!!!

  // create / convert
  for (vid=test_ids) run_test( "consts", consts(get_value(vid)), vid );
  for (vid=test_ids) run_test( "vstr", vstr(get_value(vid)), vid );
  for (vid=test_ids) run_test( "sum", sum(get_value(vid)), vid );

  // query
  for (vid=test_ids) run_test( "find_12", find([1,2],get_value(vid)), vid );
```

```
for (vid=test_ids) run_test( "count_S1", count(1,get_value(vid),true), vid );
for (vid=test_ids) run_test( "exists_S1", exists(1,get_value(vid),true), vid );

// select
for (vid=test_ids) run_test( "defined_or_D", defined_or(get_value(vid),"default"), vid );
for (vid=test_ids) run_test( "edefined_or_DE3", edefined_or(get_value(vid),3,"default"),
vid );
for (vid=test_ids) run_test( "first", first(get_value(vid)), vid );
for (vid=test_ids) run_test( "second", second(get_value(vid)), vid );
for (vid=test_ids) run_test( "last", last(get_value(vid)), vid );
for (vid=test_ids) run_test( "nfirst_1", nfirst(get_value(vid),n=1), vid );
for (vid=test_ids) run_test( "nlast_1", nlast(get_value(vid),n=1), vid );
for (vid=test_ids) run_test( "nhead_1", nhead(get_value(vid),n=1), vid );
for (vid=test_ids) run_test( "ntail_1", ntail(get_value(vid),n=1), vid );
for (vid=test_ids) run_test( "rselect_02", rselect(get_value(vid),i=[0:2]), vid );
for (vid=test_ids) run_test( "eselect_F", eselect(get_value(vid),f=true), vid );
for (vid=test_ids) run_test( "eselect_L", eselect(get_value(vid),l=true), vid );
for (vid=test_ids) run_test( "eselect_1", eselect(get_value(vid),i=1), vid );
// not tested: ciselect()
// not tested: cmvselect()

// reorder
for (vid=test_ids) run_test( "smerge", smerge(get_value(vid)), vid );
for (vid=test_ids) run_test( "pmerge", pmerge(get_value(vid)), vid );
for (vid=test_ids) run_test( "reverse", reverse(get_value(vid)), vid );
for (vid=test_ids) run_test( "qsort", qsort(get_value(vid)), vid );
for (vid=test_ids) run_test( "qsort2_HR", qsort2(get_value(vid), d=5, r=true), vid );

// grow / reduce
for (vid=test_ids) run_test( "strip", strip(get_value(vid)), vid );
for (vid=test_ids) run_test( "eappend_T0", eappend(0,get_value(vid)), vid );
for (vid=test_ids) run_test( "insert_T0", insert(0,get_value(vid),mv=["x","r","apple","s",[2,3]
,5]), vid );
for (vid=test_ids) run_test( "delete_T0", delete(get_value(vid),mv=["x","r","apple","s",[2,3],5]),
vid );
for (vid=test_ids) run_test( "unique", unique(get_value(vid)), vid );

// end-of-tests
```

2.1.2.2 Validation Results

```
ECHO: "OpenSCAD Version [2016, 12, 21]"
ECHO: "[ INFO ] run_test(); t01 passed: ’consts(undef)=[]’"
ECHO: "[ INFO ] run_test(); t02 passed: ’consts([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’consts([0 : 0.5 : 9])=[]’"
ECHO: "[ INFO ] run_test(); t04 passed: ’consts(A string)=[]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’consts(["orange", "apple", "grape", "banana"])=[]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’consts(["b", "a", "n", "a", "n", "a", "s"])=[]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’consts([undef])=[]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’consts([[1, 2], [2, 3]])=[]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’consts(["ab", [1, 2], [2, 3], [4, 5]])=[]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’consts([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’consts([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
      15])=[]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’vstr(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’vstr([])=’"
ECHO: "[ INFO ] run_test(); t03 passed: ’vstr([0 : 0.5 : 9])=[0 : 0.5 : 9]’"
ECHO: "[ INFO ] run_test(); t04 passed: ’vstr(A string)=A string’"
ECHO: "[ INFO ] run_test(); t05 passed: ’vstr(["orange", "apple", "grape",
      "banana"])=orangeapplegrapebanana’"
ECHO: "[ INFO ] run_test(); t06 passed: ’vstr(["b", "a", "n", "a", "n", "a", "s"])=bananas’"
ECHO: "[ INFO ] run_test(); t07 passed: ’vstr([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’vstr([[1, 2], [2, 3]])=[1, 2][2, 3]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’vstr(["ab", [1, 2], [2, 3], [4, 5]])=ab[1, 2][2, 3][4, 5]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’vstr([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 2,
      3][4, 5, 6][7, 8, 9]["a", "b", "c"]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’vstr([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
      15])=0123456789101112131415’"
ECHO: "[ INFO ] run_test(); t01 passed: ’sum(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’sum([])=0’"
ECHO: "[ INFO ] run_test(); t03 passed: ’sum([0 : 0.5 : 9])=85.5’"
ECHO: "[ INFO ] run_test(); t04 passed: ’sum(A string)=undef’"
ECHO: "[ INFO ] run_test(); t05 passed: ’sum(["orange", "apple", "grape", "banana"])=undef’"
ECHO: "[ INFO ] run_test(); t06 passed: ’sum(["b", "a", "n", "a", "n", "a", "s"])=undef’"
ECHO: "[ INFO ] run_test(); t07 passed: ’sum([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’sum([[1, 2], [2, 3]])=[3, 5]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’sum(["ab", [1, 2], [2, 3], [4, 5]])=undef’"
ECHO: "[ INFO ] run_test(); t10 passed: ’sum([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[undef,
      undef, undef]’"
```

```
ECHO: "[ INFO ] run_test(); t11 passed: ’sum([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=120’"
ECHO: "[ INFO ] run_test(); t01 passed: ’find_12(undef)=[]’"
ECHO: "[ INFO ] run_test(); t02 passed: ’find_12([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’find_12([0 : 0.5 : 9])=[]’"
ECHO: "[ INFO ] run_test(); t04 passed: ’find_12(A string)=[]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’find_12(["orange", "apple", "grape", "banana"])=[]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’find_12(["b", "a", "n", "a", "n", "a", "s"])=[]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’find_12([undef])=[]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’find_12([[1, 2], [2, 3]])=[0]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’find_12(["ab", [1, 2], [2, 3], [4, 5]])=[1]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’find_12([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’find_12([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
       15])=[]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’count_S1(undef)=0’"
ECHO: "[ INFO ] run_test(); t02 passed: ’count_S1([])=0’"
ECHO: "[ INFO ] run_test(); t03 passed: ’count_S1([0 : 0.5 : 9])=0’"
ECHO: "[ INFO ] run_test(); t04 passed: ’count_S1(A string)=0’"
ECHO: "[ INFO ] run_test(); t05 passed: ’count_S1(["orange", "apple", "grape", "banana"])=0’"
ECHO: "[ INFO ] run_test(); t06 passed: ’count_S1(["b", "a", "n", "a", "n", "a", "s"])=0’"
ECHO: "[ INFO ] run_test(); t07 passed: ’count_S1([undef])=0’"
ECHO: "[ INFO ] run_test(); t08 passed: ’count_S1([[1, 2], [2, 3]])=1’"
ECHO: "[ INFO ] run_test(); t09 passed: ’count_S1(["ab", [1, 2], [2, 3], [4, 5]])=1’"
ECHO: "[ INFO ] run_test(); t10 passed: ’count_S1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=1’"
ECHO: "[ INFO ] run_test(); t11 passed: ’count_S1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
       15])=1’"
ECHO: "[ INFO ] run_test(); t01 passed: ’exists_S1(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’exists_S1([])=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’exists_S1([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’exists_S1(A string)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’exists_S1(["orange", "apple", "grape", "banana"])=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’exists_S1(["b", "a", "n", "a", "n", "a", "s"])=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’exists_S1([undef])=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’exists_S1([[1, 2], [2, 3]])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’exists_S1(["ab", [1, 2], [2, 3], [4, 5]])=true’"
ECHO: "[ INFO ] run_test(); t10 passed: ’exists_S1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
       "c"]])=true’"
ECHO: "[ INFO ] run_test(); t11 passed: ’exists_S1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
       15])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’defined_or_D(undef)=default’"
ECHO: "[ INFO ] run_test(); t02 passed: ’defined_or_D([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’defined_or_D([0 : 0.5 : 9])=[0 : 0.5 : 9]’"
ECHO: "[ INFO ] run_test(); t04 passed: ’defined_or_D(A string)=A string’"
ECHO: "[ INFO ] run_test(); t05 passed: ’defined_or_D(["orange", "apple", "grape", "banana"])=["orange",
       "apple", "grape", "banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’defined_or_D(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n",
       "a", "n", "a", "s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’defined_or_D([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’defined_or_D([[1, 2], [2, 3]])=[[1, 2], [2, 3]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’defined_or_D(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2,
       3], [4, 5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’defined_or_D([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
       "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’defined_or_D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
       15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’edefined_or_DE3(undef)=default’"
ECHO: "[ INFO ] run_test(); t02 passed: ’edefined_or_DE3([])=default’"
ECHO: "[ INFO ] run_test(); t03 passed: ’edefined_or_DE3([0 : 0.5 : 9])=default’"
ECHO: "[ INFO ] run_test(); t04 passed: ’edefined_or_DE3(A string)=t’"
ECHO: "[ INFO ] run_test(); t05 passed: ’edefined_or_DE3(["orange", "apple", "grape", "banana"])=banana’"
ECHO: "[ INFO ] run_test(); t06 passed: ’edefined_or_DE3(["b", "a", "n", "a", "n", "a", "s"])=a’"
ECHO: "[ INFO ] run_test(); t07 passed: ’edefined_or_DE3([undef])=default’"
ECHO: "[ INFO ] run_test(); t08 passed: ’edefined_or_DE3([[1, 2], [2, 3]])=default’"
ECHO: "[ INFO ] run_test(); t09 passed: ’edefined_or_DE3(["ab", [1, 2], [2, 3], [4, 5]])=[4, 5]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’edefined_or_DE3([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
       "c"]])=["a", "b", "c"]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’edefined_or_DE3([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
       15])=3’"
ECHO: "[ INFO ] run_test(); t01 passed: ’first(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’first([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’first([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’first(A string)=A’"
ECHO: "[ INFO ] run_test(); t05 passed: ’first(["orange", "apple", "grape", "banana"])=orange’"
ECHO: "[ INFO ] run_test(); t06 passed: ’first(["b", "a", "n", "a", "n", "a", "s"])=b’"
ECHO: "[ INFO ] run_test(); t07 passed: ’first([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’first([[1, 2], [2, 3]])=[1, 2]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’first(["ab", [1, 2], [2, 3], [4, 5]])=ab’"
ECHO: "[ INFO ] run_test(); t10 passed: ’first([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 2,
       3]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’first([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=0’"
ECHO: "[ INFO ] run_test(); t01 passed: ’second(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’second([])=undef’"
```

```
ECHO: "[ INFO ] run_test(); t03 passed: ’second([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’second(A string)= ’"
ECHO: "[ INFO ] run_test(); t05 passed: ’second(["orange", "apple", "grape", "banana"])=apple’"
ECHO: "[ INFO ] run_test(); t06 passed: ’second(["b", "a", "n", "a", "n", "a", "s"])=a’"
ECHO: "[ INFO ] run_test(); t07 passed: ’second([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’second([[1, 2], [2, 3]])=[2, 3]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’second(["ab", [1, 2], [2, 3], [4, 5]])=[1, 2]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’second([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[4, 5, 6]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’second([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=1’"
ECHO: "[ INFO ] run_test(); t01 passed: ’last(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’last([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’last([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’last(A string)=g’"
ECHO: "[ INFO ] run_test(); t05 passed: ’last(["orange", "apple", "grape", "banana"])=banana’"
ECHO: "[ INFO ] run_test(); t06 passed: ’last(["b", "a", "n", "a", "n", "a", "s"])=s’"
ECHO: "[ INFO ] run_test(); t07 passed: ’last([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’last([[1, 2], [2, 3]])=[2, 3]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’last(["ab", [1, 2], [2, 3], [4, 5]])=[4, 5]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’last([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=["a", "b", "c"]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’last([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=15’"
ECHO: "[ INFO ] run_test(); t01 passed: ’nfirst_1(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’nfirst_1([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’nfirst_1([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’nfirst_1(A string)=["A"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’nfirst_1(["orange", "apple", "grape", "banana"])=["orange"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’nfirst_1(["b", "a", "n", "a", "n", "a", "s"])=["b"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’nfirst_1([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’nfirst_1([[1, 2], [2, 3]])=[[1, 2]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’nfirst_1(["ab", [1, 2], [2, 3], [4, 5]])=["ab"]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’nfirst_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’nfirst_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’nlast_1(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’nlast_1([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’nlast_1([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’nlast_1(A string)=["g"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’nlast_1(["orange", "apple", "grape", "banana"])=["banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’nlast_1(["b", "a", "n", "a", "n", "a", "s"])=["s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’nlast_1([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’nlast_1([[1, 2], [2, 3]])=[[2, 3]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’nlast_1(["ab", [1, 2], [2, 3], [4, 5]])=[[4, 5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’nlast_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[["a", "b", "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’nlast_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’nhead_1(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’nhead_1([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’nhead_1([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’nhead_1(A string)=["A", " ", "s", "t", "r", "i", "n"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’nhead_1(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’nhead_1(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’nhead_1([undef])=[]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’nhead_1([[1, 2], [2, 3]])=[[1, 2]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’nhead_1(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’nhead_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’nhead_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’ntail_1(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’ntail_1([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’ntail_1([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’ntail_1(A string)=[" ", "s", "t", "r", "i", "n", "g"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’ntail_1(["orange", "apple", "grape", "banana"])=["apple", "grape", "banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’ntail_1(["b", "a", "n", "a", "n", "a", "s"])=["a", "n", "a", "n", "a", "s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’ntail_1([undef])=[]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’ntail_1([[1, 2], [2, 3]])=[[2, 3]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’ntail_1(["ab", [1, 2], [2, 3], [4, 5]])=[[1, 2], [2, 3], [4, 5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’ntail_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[4, 5, 6], [7, 8, 9], ["a", "b", "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’ntail_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’rselect_02(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’rselect_02([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’rselect_02([0 : 0.5 : 9])=undef’"
```

```
ECHO: "[ INFO ] run_test(); t04 passed: 'rselect_02(A string)=["A", " ", "s"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'rselect_02(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'rselect_02(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'rselect_02([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'rselect_02([[1, 2], [2, 3]])=undef'"
ECHO: "[ INFO ] run_test(); t09 passed: 'rselect_02(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'rselect_02([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'rselect_02([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'eselect_F(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eselect_F([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eselect_F([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'eselect_F(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eselect_F(["orange", "apple", "grape", "banana"])=["o", "a", "g", "b"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'eselect_F(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eselect_F([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eselect_F([[1, 2], [2, 3]])=[1, 2]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eselect_F(["ab", [1, 2], [2, 3], [4, 5]])=["a", 1, 2, 4]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eselect_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 4, 7, "a"]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'eselect_F(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'eselect_L(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eselect_L([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eselect_L([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'eselect_L(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eselect_L(["orange", "apple", "grape", "banana"])=["e", "e", "e", "a"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'eselect_L(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eselect_L([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eselect_L([[1, 2], [2, 3]])=[2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eselect_L(["ab", [1, 2], [2, 3], [4, 5]])=["b", 2, 3, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eselect_L([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[3, 6, 9, "c"]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'eselect_L(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'eselect_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eselect_1([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eselect_1([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 *skip*: 'eselect_1(A string)'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eselect_1(["orange", "apple", "grape", "banana"])=["r", "p", "r", "a"]'"
ECHO: "[ INFO ] run_test(); t06 *skip*: 'eselect_1(Test vector 02)'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eselect_1([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eselect_1([[1, 2], [2, 3]])=[2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eselect_1(["ab", [1, 2], [2, 3], [4, 5]])=["b", 2, 3, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eselect_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[2, 5, 8, "b"]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'eselect_1(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'smerge(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'smerge([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'smerge([0 : 0.5 : 9])=[[0 : 0.5 : 9]]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'smerge(A string)=["A string"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'smerge(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'smerge(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'smerge([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'smerge([[1, 2], [2, 3]])=[1, 2, 2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'smerge(["ab", [1, 2], [2, 3], [4, 5]])=["ab", 1, 2, 2, 3, 4, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'smerge([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 2, 3, 4, 5, 6, 7, 8, 9, "a", "b", "c"]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'smerge([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'pmerge(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'pmerge([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'pmerge([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'pmerge(A string)=["A string"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'pmerge(["orange", "apple", "grape", "banana"])=[["o", "a", "g", "b"], ["r", "p", "r", "a"], ["a", "p", "a", "n"], ["n", "l", "p", "a"], ["g", "e", "e", "n"]]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'pmerge(["b", "a", "n", "a", "n", "a", "s"])=[["b", "a", "n", "a", "n", "a", "s"]]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'pmerge([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'pmerge([[1, 2], [2, 3]])=[[1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'pmerge(["ab", [1, 2], [2, 3], [4, 5]])=[["a", 1, 2, 4], ["b", 2, 3, 5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'pmerge([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 4,
    7, "a"], [2, 5, 8, "b"], [3, 6, 9, "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’pmerge([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=undef’"
ECHO: "[ INFO ] run_test(); t01 passed: ’reverse(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’reverse([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’reverse([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’reverse(A string)=["g", "n", "i", "r", "t", "s", " ", "A"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’reverse(["orange", "apple", "grape", "banana"])=["banana",
    "grape", "apple", "orange"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’reverse(["b", "a", "n", "a", "n", "a", "s"])=["s", "a", "n", "a",
    "n", "a", "b"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’reverse([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’reverse([[1, 2], [2, 3]])=[[2, 3], [1, 2]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’reverse(["ab", [1, 2], [2, 3], [4, 5]])=[[4, 5], [2, 3], [1, 2],
    "ab"]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’reverse([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[["a",
    "b", "c"], [7, 8, 9], [4, 5, 6], [1, 2, 3]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’reverse([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’qsort(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’qsort([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’qsort([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’qsort(A string)=undef’"
ECHO: "[ INFO ] run_test(); t05 passed: ’qsort(["orange", "apple", "grape", "banana"])=["apple", "banana",
    "grape", "orange"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’qsort(["b", "a", "n", "a", "n", "a", "s"])=["a", "a", "a", "b",
    "n", "n", "s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’qsort([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’qsort([[1, 2], [2, 3]])=undef’"
ECHO: "[ INFO ] run_test(); t09 passed: ’qsort(["ab", [1, 2], [2, 3], [4, 5]])=undef’"
ECHO: "[ INFO ] run_test(); t10 passed: ’qsort([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=undef’"
ECHO: "[ INFO ] run_test(); t11 passed: ’qsort([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’qsort2_HR(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’qsort2_HR([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’qsort2_HR([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’qsort2_HR(A string)=undef’"
ECHO: "[ INFO ] run_test(); t05 passed: ’qsort2_HR(["orange", "apple", "grape", "banana"])=["orange",
    "grape", "banana", "apple"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’qsort2_HR(["b", "a", "n", "a", "n", "a", "s"])=["s", "n", "n",
    "b", "a", "a", "a"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’qsort2_HR([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’qsort2_HR([[1, 2], [2, 3]])=[[3, 2], [2, 1]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’qsort2_HR(["ab", [1, 2], [2, 3], [4, 5]])=[[5, 4], [3, 2], [2, 1],
    "ab"]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’qsort2_HR([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=[["c", "b", "a"], [9, 8, 7], [6, 5, 4], [3, 2, 1]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’qsort2_HR([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’strip(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’strip([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’strip([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’strip(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’strip(["orange", "apple", "grape", "banana"])=["orange", "apple",
    "grape", "banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’strip(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a",
    "n", "a", "s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’strip([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’strip([[1, 2], [2, 3]])=[[1, 2], [2, 3]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’strip(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3], [4,
    5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’strip([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2,
    3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’strip([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’eappend_T0(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’eappend_T0([])=[[0]]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’eappend_T0([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’eappend_T0(A string)=[["A", 0], [" ", 0], ["s", 0], ["t", 0],
    ["r", 0], ["i", 0], ["n", 0], ["g", 0]]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’eappend_T0(["orange", "apple", "grape", "banana"])=[["orange", 0],
    ["apple", 0], ["grape", 0], ["banana", 0]]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’eappend_T0(["b", "a", "n", "a", "n", "a", "s"])=[["b", 0], ["a",
    0], ["n", 0], ["a", 0], ["n", 0], ["a", 0], ["s", 0]]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’eappend_T0([undef])=[[undef, 0]]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’eappend_T0([[1, 2], [2, 3]])=[[1, 2, 0], [2, 3, 0]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’eappend_T0(["ab", [1, 2], [2, 3], [4, 5]])=[["ab", 0], [1, 2, 0],
    [2, 3, 0], [4, 5, 0]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’eappend_T0([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=[[1, 2, 3, 0], [4, 5, 6, 0], [7, 8, 9, 0], ["a", "b", "c", 0]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’eappend_T0([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15])=[[0, 0], [1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0], [7, 0], [8, 0], [9, 0], [10, 0], [11, 0], [12, 0], [13, 0], [14, 0], [15, 0]]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’insert_T0(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’insert_T0([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’insert_T0([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’insert_T0(A string)=undef’"
ECHO: "[ INFO ] run_test(); t05 passed: ’insert_T0(["orange", "apple", "grape", "banana"])=["orange", 0, "apple", "grape", "banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’insert_T0(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", 0, "s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’insert_T0([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’insert_T0([[1, 2], [2, 3]])=[[1, 2], 0, [2, 3]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’insert_T0(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], 0, [2, 3], [4, 5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’insert_T0([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=undef’"
ECHO: "[ INFO ] run_test(); t11 passed: ’insert_T0([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 0, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’delete_T0(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’delete_T0([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’delete_T0([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’delete_T0(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’delete_T0(["orange", "apple", "grape", "banana"])=["orange", "grape", "banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’delete_T0(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’delete_T0([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’delete_T0([[1, 2], [2, 3]])=[[1, 2]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’delete_T0(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [4, 5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’delete_T0([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’delete_T0([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’unique(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’unique([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’unique([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’unique(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’unique(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape", "banana"]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’unique(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "s"]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’unique([undef])=[undef]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’unique([[1, 2], [2, 3]])=[[1, 2], [2, 3]]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’unique(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3], [4, 5]]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’unique([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’unique([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]’"
```

2.2 Computations Validation

- Point, Vector and Plane
- Bitwise Operations

2.2.1 Point, Vector and Plane

- Validation Script
- Validation Results

2.2.1.1 Validation Script

```
include <math.scad>;
use <table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );
```

```
// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["fac", "Function argument count",    undef],
  ["crp", "Result precision",           undef],
  ["t01", "All undefined",              [undef,undef,undef,undef,undef,undef]],
  ["t02", "All empty vector",           [empty_v,empty_v,
empty_v,empty_v,empty_v,empty_v]],
  ["t03", "All scalars",                [60, 50, 40, 30, 20, 10]],
  ["t04", "All 1D vectors",             [[99], [58], [12], [42], [15], [1]]],
  ["t05", "All 2D vectors",             [
                                          [99,2], [58,16], [12,43],
                                          [42,13], [15,59], [1,85]
                                        ]],
  ["t06", "All 3D vectors",             [
                                          [199,20,55], [158,116,75], [12,43,90],
                                          [42,13,34], [15,59,45], [62,33,69]
                                        ]],
  ["t07", "All 4D vectors",             [
                                          [169,27,35,10], [178,016,25,20], [12,43,90,30],
                                          [42,13,34,60], [15,059,45,50], [62,33,69,40]
                                        ]],
  ["t08", "Orthogonal vectors",         [
                                          +x_axis3d_uv, +
y_axis3d_uv, +z_axis3d_uv,
                                          -x_axis3d_uv, -
y_axis3d_uv, -z_axis3d_uv,
                                        ]],
  ["t09", "Coplanar vectors",           [
                                          +x_axis3d_uv, +
y_axis3d_uv, [2,2,0],
                                          origin3d, origin3d,
origin3d,
                                        ]]
];

test_ids = table_get_row_ids( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 'skip' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["distance_pp",
    2,                                                  // fac
    4,                                                  // crp
    undef,                                              // t01
    undef,                                              // t02
    undef,                                              // t03
    41,                                                 // t04
    43.3244,                                            // t05
    106.2873,                                           // t06
    undef,                                              // t07
    1.4142,                                             // t08
    1.4142                                              // t09
  ],
  ["dot_vv",
    4,                                                  // fac
    4,                                                  // crp
    undef,                                              // t01
    undef,                                              // t02
    400,                                                // t03
    1392,                                               // t04
    1269,                                               // t05
    17888,                                              // t06
    22599,                                              // t07
    1,                                                  // t08
    -2                                                  // t09
  ],
    ["cross_vv",
      4,                                                    // fac
      4,                                                    // crp
      skip,                                                 // t01
      skip,                                                 // t02
      skip,                                                 // t03
      skip,                                                 // t04
      917,                                                  // t05
      [2662,-11727,21929],                                  // t06
      skip,                                                 // t07
      [1,-1,1],                                             // t08
      [0,0,-1]                                              // t09
    ],
    ["striple_vvv",
      6,                                                    // fac
      4,                                                    // crp
      skip,                                                 // t01
      skip,                                                 // t02
      skip,                                                 // t03
      skip,                                                 // t04
      [-75981,14663],                                       // t05
      199188,                                               // t06
      skip,                                                 // t07
      8,                                                    // t08
      0                                                     // t09
    ],
    ["angle_vv",
      4,                                                    // fac
      4,                                                    // crp
      undef,                                                // t01
      undef,                                                // t02
      undef,                                                // t03
      undef,                                                // t04
      35.8525,                                              // t05
      54.4261,                                              // t06
      undef,                                                // t07
      60,                                                   // t08
      153.4350                                              // t09
    ],
    ["angle_vvn",
      6,                                                    // fac
      4,                                                    // crp
      skip,                                                 // t01
      skip,                                                 // t02
      skip,                                                 // t03
      skip,                                                 // t04
      skip,                                                 // t05
      83.2771,                                              // t06 (verify)
      skip,                                                 // t07
      90,                                                   // t08
      0                                                     // t09
    ],
    ["unit_v",
      2,                                                    // fac
      4,                                                    // crp
      undef,                                                // t01
      undef,                                                // t02
      undef,                                                // t03
      [1],                                                  // t04
      [0.9464,-0.3231],                                     // t05
      [0.3857,-0.9032,-0.1882],                             // t06
      undef,                                                // t07
      [0.7071,-0.7071,0],                                   // t08
      [0.7071,-0.7071,0]                                    // t09
    ],
    ["are_coplanar_vvv",
      6,                                                    // fac
      4,                                                    // crp
      skip,                                                 // t01
      skip,                                                 // t02
      skip,                                                 // t03
      skip,                                                 // t04
      skip,                                                 // t05
      false,                                                // t06
      skip,                                                 // t07
      false,                                                // t08
      true                                                  // t09
    ]
  ];

  // sanity-test tables
```

```
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = table_get(test_r, test_c, vid, "tv");
function gv( vid, e ) = get_value( vid )[e];
module run( fname, vid )
{
  value_text = table_get(test_r, test_c, vid, "td");

  if ( table_get(good_r, good_c, fname, vid) != skip )
    children();
  else if ( show_skipped )
    log_info( str("*skip*: ", vid, " '", fname, "(", value_text, ")'") );
}
module test( fname, fresult, vid )
{
  value_text = table_get(test_r, test_c, vid, "td");
  fname_argc = table_get(good_r, good_c, fname, "fac");
  comp_prcsn = table_get(good_r, good_c, fname, "crp");
  pass_value = table_get(good_r, good_c, fname, vid);

  test_pass = validate(cv=fresult, t="almost", ev=pass_value, p=comp_prcsn, pf=true);
  farg_text = vstr(eappend(", ", rselect(get_value(vid), [0:fname_argc-1]), r=false
, j=false, l=false));
  test_text = validate(str(fname, "(", farg_text, ")=~", pass_value), fresult, "almost",
pass_value, comp_prcsn);

  if ( pass_value != skip )
  {
    if ( !test_pass )
      log_warn( str(vid, "(", value_text, ") ", test_text) );
    else if ( show_passing )
      log_info( str(vid, " ", test_text) );
  }
  else if ( show_skipped )
    log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
}

// Indirect function calls would be very useful here!!!
run_ids = delete( test_ids, mv=["fac", "crp"] );
for (vid=run_ids) run("distance_pp",vid) test( "distance_pp", distance_pp(gv(vid,0),gv(vid
,1),gv(vid,2),gv(vid,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("dot_vv",vid) test( "dot_vv", dot_vv(gv(vid,0),gv(vid,1),gv(vid,2),gv(vid
,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("cross_vv",vid) test( "cross_vv", cross_vv(gv(vid,0),gv(vid,1),gv(vid,2
),gv(vid,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("striple_vvv",vid) test( "striple_vvv", striple_vvv(gv(vid,0),gv(vid
,1),gv(vid,2),gv(vid,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("angle_vv",vid) test( "angle_vv", angle_vv(gv(vid,0),gv(vid,1),gv(vid,2
),gv(vid,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("angle_vvn",vid) test( "angle_vvn", angle_vvn(gv(vid,0),gv(vid,1),gv(
vid,2),gv(vid,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("unit_v",vid) test( "unit_v", unit_v(gv(vid,0),gv(vid,1),gv(vid,2),gv(vid
,3),gv(vid,4),gv(vid,5)), vid );
for (vid=run_ids) run("are_coplanar_vvv",vid) test( "are_coplanar_vvv",
are_coplanar_vvv(gv(vid,0),gv(vid,1),gv(vid,2),gv(vid,3),gv(vid,4),gv(vid,5)), vid );

// end-of-tests
```

2.2.1.2 Validation Results

```
ECHO: "OpenSCAD Version [2016, 12, 21]"
ECHO: "[ INFO ] run(): test(); t01 passed: 'distance_pp(undef, undef)=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'distance_pp([], [])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'distance_pp(60, 50)=~undef'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'distance_pp([99], [58])=~41'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'distance_pp([99, 2], [58, 16])=~43.3244'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'distance_pp([199, 20, 55], [158, 116, 75])=~106.287'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'distance_pp([169, 27, 35, 10], [178, 16, 25, 20])=~undef'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'distance_pp([1, 0, 0], [0, 1, 0])=~1.4142'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'distance_pp([1, 0, 0], [0, 1, 0])=~1.4142'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'dot_vv(undef, undef, undef, undef)=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'dot_vv([], [], [], [])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'dot_vv(60, 50, 40, 30)=~400'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'dot_vv([99], [58], [12], [42])=~1392'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'dot_vv([99, 2], [58, 16], [12, 43], [42, 13])=~1269'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'dot_vv([199, 20, 55], [158, 116, 75], [12, 43, 90], [42, 13,
   34])=~17888'"
```

```
ECHO: "[ INFO ] run(): test(); t07 passed: 'dot_vv([169, 27, 35, 10], [178, 16, 25, 20], [12, 43, 90, 30], [42, 13, 34, 60])=~22599'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'dot_vv([1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0])=~1'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'dot_vv([1, 0, 0], [0, 1, 0], [2, 2, 0], [0, 0, 0])=~-2'"
ECHO: "[ INFO ] run(); *skip*: t01 'cross_vv(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'cross_vv(All empty vector)'"
ECHO: "[ INFO ] run(); *skip*: t03 'cross_vv(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'cross_vv(All 1D vectors)'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'cross_vv([99, 2], [58, 16], [12, 43], [42, 13])=~917'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'cross_vv([199, 20, 55], [158, 116, 75], [12, 43, 90], [42, 13, 34])=~[2662, -11727, 21929]'"
ECHO: "[ INFO ] run(); *skip*: t07 'cross_vv(All 4D vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'cross_vv([1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0])=~[1, -1, 1]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'cross_vv([1, 0, 0], [0, 1, 0], [2, 2, 0], [0, 0, 0])=~[0, 0, -1]'"
ECHO: "[ INFO ] run(); *skip*: t01 'striple_vvv(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'striple_vvv(All empty vector)'"
ECHO: "[ INFO ] run(); *skip*: t03 'striple_vvv(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'striple_vvv(All 1D vectors)'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'striple_vvv([99, 2], [58, 16], [12, 43], [42, 13], [15, 59], [1, 85])=~[-75981, 14663]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'striple_vvv([199, 20, 55], [158, 116, 75], [12, 43, 90], [42, 13, 34], [15, 59, 45], [62, 33, 69])=~199188'"
ECHO: "[ INFO ] run(); *skip*: t07 'striple_vvv(All 4D vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'striple_vvv([1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0], [0, -1, 0], [0, 0, -1])=~8'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'striple_vvv([1, 0, 0], [0, 1, 0], [2, 2, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0])=~0'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'angle_vv(undef, undef, undef, undef)=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'angle_vv([], [], [], [])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'angle_vv(60, 50, 40, 30)=~undef'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'angle_vv([99], [58], [12], [42])=~undef'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'angle_vv([99, 2], [58, 16], [12, 43], [42, 13])=~35.8525'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'angle_vv([199, 20, 55], [158, 116, 75], [12, 43, 90], [42, 13, 34])=~54.4261'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'angle_vv([169, 27, 35, 10], [178, 16, 25, 20], [12, 43, 90, 30], [42, 13, 34, 60])=~undef'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'angle_vv([1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0])=~60'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'angle_vv([1, 0, 0], [0, 1, 0], [2, 2, 0], [0, 0, 0])=~153.435'"
ECHO: "[ INFO ] run(); *skip*: t01 'angle_vvn(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'angle_vvn(All empty vector)'"
ECHO: "[ INFO ] run(); *skip*: t03 'angle_vvn(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'angle_vvn(All 1D vectors)'"
ECHO: "[ INFO ] run(); *skip*: t05 'angle_vvn(All 2D vectors)'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'angle_vvn([199, 20, 55], [158, 116, 75], [12, 43, 90], [42, 13, 34], [15, 59, 45], [62, 33, 69])=~83.2771'"
ECHO: "[ INFO ] run(); *skip*: t07 'angle_vvn(All 4D vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'angle_vvn([1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0], [0, -1, 0], [0, 0, -1])=~90'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'angle_vvn([1, 0, 0], [0, 1, 0], [2, 2, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0])=~0'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'unit_v(undef, undef)=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'unit_v([], [])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'unit_v(60, 50)=~undef'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'unit_v([99], [58])=~[1]'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'unit_v([99, 2], [58, 16])=~[0.9464, -0.3231]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'unit_v([199, 20, 55], [158, 116, 75])=~[0.3857, -0.9032, -0.1882]'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'unit_v([169, 27, 35, 10], [178, 16, 25, 20])=~undef'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'unit_v([1, 0, 0], [0, 1, 0])=~[0.7071, -0.7071, 0]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'unit_v([1, 0, 0], [0, 1, 0])=~[0.7071, -0.7071, 0]'"
ECHO: "[ INFO ] run(); *skip*: t01 'are_coplanar_vvv(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'are_coplanar_vvv(All empty vector)'"
ECHO: "[ INFO ] run(); *skip*: t03 'are_coplanar_vvv(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'are_coplanar_vvv(All 1D vectors)'"
ECHO: "[ INFO ] run(); *skip*: t05 'are_coplanar_vvv(All 2D vectors)'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'are_coplanar_vvv([199, 20, 55], [158, 116, 75], [12, 43, 90], [42, 13, 34], [15, 59, 45], [62, 33, 69])=~false'"
ECHO: "[ INFO ] run(); *skip*: t07 'are_coplanar_vvv(All 4D vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'are_coplanar_vvv([1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0], [0, -1, 0], [0, 0, -1])=~false'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'are_coplanar_vvv([1, 0, 0], [0, 1, 0], [2, 2, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0])=~true'"
```

2.2.2 Bitwise Operations

- Validation Script

- Validation Results

2.2.2.1 Validation Script

```
include <math_bitwise.scad>;
use <table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["fac", "Function argument count",    undef],
  ["t01", "All undefined",              [undef,undef]],
  ["t02", "All empty vector",           [empty_v,empty_v]],
  ["t03", "test value 1",               [254, 0]],
  ["t04", "test value 2",               [254, 1]],
  ["t05", "test value 3",               [255, 0]],
  ["t06", "test value 4",               [0, 255]],
  ["t07", "test value 5",               [126, 63]],
  ["t08", "test value 6",               [25, 10]],
  ["t09", "test value 7",               [1024, 512]],
  ["t10", "test value 8",               [4253, 315]],
  ["t11", "test value 9",               [835, 769]],
  ["t12", "test value 10",              [856, 625]]
];

test_ids = table_get_row_ids( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 'skip' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["bitwise_is_equal_0", 2,
    false,                          // t01
    false,                          // t02
    true,                           // t03
    false,                          // t04
    false,                          // t05
    true,                           // t06
    true,                           // t07
    true,                           // t08
    true,                           // t09
    true,                           // t10
    true,                           // t11
    true                            // t12
  ],
  ["bitwise_is_equal_1", 2,
    false,                          // t01
    false,                          // t02
    false,                          // t03
    true,                           // t04
    true,                           // t05
    false,                          // t06
    false,                          // t07
    false,                          // t08
    false,                          // t09
    false,                          // t10
    false,                          // t11
    false                           // t12
  ],
  ["bitwise_i2v", 1,
    undef,                          // t01
    undef,                          // t02
    [1,1,1,1,1,1,1,0],              // t03
    [1,1,1,1,1,1,1,0],              // t04
    [1,1,1,1,1,1,1,1],              // t05
    [0],                            // t06
    [1,1,1,1,1,1,0],                // t07
    [1,1,0,0,1],                    // t08
    [1,0,0,0,0,0,0,0,0,0,0],        // t09
    [1,0,0,0,0,1,0,0,1,1,1,0,1],    // t10
    [1,1,0,1,0,0,0,0,1,1],          // t11
    [1,1,0,1,0,1,1,0,0,0]           // t12
  ],
  ["bitwise_i2v_v2i", 1,
    undef,                          // t01
    undef,                          // t02
    254,                            // t03
    254,                            // t04
    255,                            // t05
    0,                              // t06
    126,                            // t07
    25,                             // t08
    1024,                           // t09
    4253,                           // t10
    835,                            // t11
    856                             // t12
  ],
  ["bitwise_i2s", 1,
    undef,                          // t01
    undef,                          // t02
    "11111110",                     // t03
    "11111110",                     // t04
    "11111111",                     // t05
    "0",                            // t06
    "1111110",                      // t07
    "11001",                        // t08
    "10000000000",                  // t09
    "1000010011101",                // t10
    "1101000011",                   // t11
    "1101011000"                    // t12
  ],
  ["bitwise_i2s_s2i", 1,
    undef,                          // t01
    undef,                          // t02
    254,                            // t03
    254,                            // t04
    255,                            // t05
    0,                              // t06
    126,                            // t07
    25,                             // t08
    1024,                           // t09
    4253,                           // t10
    835,                            // t11
    856                             // t12
  ],
  ["bitwise_and", 2,
    undef,                          // t01
    undef,                          // t02
    0,                              // t03
    0,                              // t04
    0,                              // t05
    0,                              // t06
    62,                             // t07
    8,                              // t08
    0,                              // t09
    25,                             // t10
    769,                            // t11
    592                             // t12
  ],
  ["bitwise_or", 2,
    undef,                          // t01
    undef,                          // t02
    254,                            // t03
    255,                            // t04
    255,                            // t05
    255,                            // t06
    127,                            // t07
    27,                             // t08
    1536,                           // t09
    4543,                           // t10
    835,                            // t11
    889                             // t12
  ],
  ["bitwise_xor", 2,
    undef,                          // t01
    undef,                          // t02
    254,                            // t03
    255,                            // t04
    255,                            // t05
    255,                            // t06
    65,                             // t07
    19,                             // t08
    1536,                           // t09
    4518,                           // t10
    66,                             // t11
    297                             // t12
  ],
  ["bitwise_not", 1,
    undef,                          // t01
    undef,                          // t02
    1,                              // t03
    1,                              // t04
    0,                              // t05
    1,                              // t06
    1,                              // t07
    6,                              // t08
    1023,                           // t09
    3938,                           // t10
    188,                            // t11
    167                             // t12
  ],
  ["bitwise_lsh", 1,
    undef,                          // t01
    undef,                          // t02
    508,                            // t03
    508,                            // t04
    510,                            // t05
    0,                              // t06
    252,                            // t07
    50,                             // t08
    2048,                           // t09
    8506,                           // t10
    1670,                           // t11
    1712                            // t12
  ],
  ["bitwise_rsh", 1,
    undef,                          // t01
    undef,                          // t02
    127,                            // t03
    127,                            // t04
    127,                            // t05
    0,                              // t06
    63,                             // t07
    12,                             // t08
    512,                            // t09
    2126,                           // t10
    417,                            // t11
    428                             // t12
  ]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = table_get(test_r, test_c, vid, "tv");
function gv( vid, e ) = get_value( vid )[e];
module run( fname, vid )
{
  value_text = table_get(test_r, test_c, vid, "td");

  if ( table_get(good_r, good_c, fname, vid) != skip )
    children();
  else if ( show_skipped )
    log_info( str("*skip*: ", vid, " '", fname, "(", value_text, ")'") );
}
module test( fname, fresult, vid )
{
  value_text = table_get(test_r, test_c, vid, "td");
  fname_argc = table_get(good_r, good_c, fname, "fac");
  pass_value = table_get(good_r, good_c, fname, vid);
```

```
 test_pass = validate(cv=fresult, t="equals", ev=pass_value, pf=true);
 farg_text = vstr(eappend(", ", rselect(get_value(vid), [0:fname_argc-1]), r=false,
 j=false, l=false));
 test_text = validate(str(fname, "(", farg_text, ")=", pass_value), fresult, "equals",
 pass_value);

 if ( pass_value != skip )
 {
   if ( !test_pass )
     log_warn( str(vid, "(", value_text, ") ", test_text) );
   else if ( show_passing )
     log_info( str(vid, " ", test_text) );
 }
 else if ( show_skipped )
   log_info( str(vid, " *skip*: ’", fname, "(", value_text, ")’") );
}

// Indirect function calls would be very useful here!!!
run_ids = delete( test_ids, mv=["fac", "crp"] );
for (vid=run_ids) run("bitwise_is_equal_0",vid) test( "bitwise_is_equal_0",
  bitwise_is_equal(gv(vid,0),gv(vid,1),0), vid );
for (vid=run_ids) run("bitwise_is_equal_1",vid) test( "bitwise_is_equal_1",
  bitwise_is_equal(gv(vid,0),gv(vid,1),1), vid );
for (vid=run_ids) run("bitwise_i2v",vid) test( "bitwise_i2v", bitwise_i2v(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_i2v_v2i",vid) test( "bitwise_i2v_v2i",
  bitwise_v2i(bitwise_i2v(gv(vid,0))), vid );
for (vid=run_ids) run("bitwise_i2s",vid) test( "bitwise_i2s", bitwise_i2s(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_i2s_s2i",vid) test( "bitwise_i2s_s2i",
  bitwise_s2i(bitwise_i2s(gv(vid,0))), vid );
for (vid=run_ids) run("bitwise_and",vid) test( "bitwise_and", bitwise_and(gv(vid,0),gv(vid,1
  )), vid );
for (vid=run_ids) run("bitwise_or",vid) test( "bitwise_or", bitwise_or(gv(vid,0),gv(vid,1)),
  vid );
for (vid=run_ids) run("bitwise_xor",vid) test( "bitwise_xor", bitwise_xor(gv(vid,0),gv(vid,1
  )), vid );
for (vid=run_ids) run("bitwise_not",vid) test( "bitwise_not", bitwise_not(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_lsh",vid) test( "bitwise_lsh", bitwise_lsh(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_rsh",vid) test( "bitwise_rsh", bitwise_rsh(gv(vid,0)), vid );

// end-of-tests
```

2.2.2.2 Validation Results

```
ECHO: "OpenSCAD Version [2016, 12, 21]"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_is_equal_0(undef, undef)=false’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_is_equal_0([], [])=false’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_is_equal_0(254, 0)=true’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_is_equal_0(254, 1)=false’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_is_equal_0(255, 0)=false’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_is_equal_0(0, 255)=true’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_is_equal_0(126, 63)=true’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_is_equal_0(25, 10)=true’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_is_equal_0(1024, 512)=true’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_is_equal_0(4253, 315)=true’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_is_equal_0(835, 769)=true’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_is_equal_0(856, 625)=true’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_is_equal_1(undef, undef)=false’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_is_equal_1([], [])=false’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_is_equal_1(254, 0)=false’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_is_equal_1(254, 1)=true’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_is_equal_1(255, 0)=true’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_is_equal_1(0, 255)=false’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_is_equal_1(126, 63)=false’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_is_equal_1(25, 10)=false’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_is_equal_1(1024, 512)=false’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_is_equal_1(4253, 315)=false’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_is_equal_1(835, 769)=false’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_is_equal_1(856, 625)=false’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2v(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2v([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2v(254)=[1, 1, 1, 1, 1, 1, 1, 0]’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2v(254)=[1, 1, 1, 1, 1, 1, 1, 0]’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2v(255)=[1, 1, 1, 1, 1, 1, 1, 1]’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2v(0)=[0]’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2v(126)=[1, 1, 1, 1, 1, 1, 0]’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2v(25)=[1, 1, 0, 0, 1]’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2v(1024)=[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2v(4253)=[1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1]’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2v(835)=[1, 1, 0, 1, 0, 0, 0, 0, 1, 1]’"
```

```
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2v(856)=[1, 1, 0, 1, 0, 1, 1, 0, 0, 0]’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2v_v2i(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2v_v2i([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2v_v2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2v_v2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2v_v2i(255)=255’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2v_v2i(0)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2v_v2i(126)=126’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2v_v2i(25)=25’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2v_v2i(1024)=1024’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2v_v2i(4253)=4253’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2v_v2i(835)=835’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2v_v2i(856)=856’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2s(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2s([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2s(254)=11111110’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2s(254)=11111110’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2s(255)=11111111’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2s(0)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2s(126)=1111110’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2s(25)=11001’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2s(1024)=10000000000’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2s(4253)=1000010011101’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2s(835)=1101000011’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2s(856)=1101011000’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2s_s2i(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2s_s2i([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2s_s2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2s_s2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2s_s2i(255)=255’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2s_s2i(0)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2s_s2i(126)=126’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2s_s2i(25)=25’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2s_s2i(1024)=1024’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2s_s2i(4253)=4253’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2s_s2i(835)=835’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2s_s2i(856)=856’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_and(undef, undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_and([], [])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_and(254, 0)=0’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_and(254, 1)=0’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_and(255, 0)=0’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_and(0, 255)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_and(126, 63)=62’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_and(25, 10)=8’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_and(1024, 512)=0’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_and(4253, 315)=25’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_and(835, 769)=769’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_and(856, 625)=592’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_or(undef, undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_or([], [])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_or(254, 0)=254’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_or(254, 1)=255’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_or(255, 0)=255’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_or(0, 255)=255’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_or(126, 63)=127’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_or(25, 10)=27’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_or(1024, 512)=1536’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_or(4253, 315)=4543’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_or(835, 769)=835’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_or(856, 625)=889’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_xor(undef, undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_xor([], [])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_xor(254, 0)=254’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_xor(254, 1)=255’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_xor(255, 0)=255’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_xor(0, 255)=255’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_xor(126, 63)=65’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_xor(25, 10)=19’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_xor(1024, 512)=1536’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_xor(4253, 315)=4518’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_xor(835, 769)=66’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_xor(856, 625)=297’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_not(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_not([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_not(254)=1’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_not(254)=1’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_not(255)=0’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_not(0)=1’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_not(126)=1’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_not(25)=6’"
```

```
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_not(1024)=1023'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_not(4253)=3938'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_not(835)=188'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_not(856)=167'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_lsh(undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_lsh([])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_lsh(254)=508'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_lsh(254)=508'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_lsh(255)=510'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_lsh(0)=0'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_lsh(126)=252'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_lsh(25)=50'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_lsh(1024)=2048'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_lsh(4253)=8506'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_lsh(835)=1670'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_lsh(856)=1712'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_rsh(undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_rsh([])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_rsh(254)=127'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_rsh(254)=127'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_rsh(255)=127'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_rsh(0)=0'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_rsh(126)=63'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_rsh(25)=12'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_rsh(1024)=512'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_rsh(4253)=2126'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_rsh(835)=417'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_rsh(856)=428'"
```

3 Todo List

globalScope> Global pyramid_q (size, center=false)

Support vertex rounding radius.

File resolution.scad

Review model for accuracy.

File shapes3d.scad

Complete rounded cylinder.

globalScope> Global tetrahedron (size, center=false)

Support vertex rounding radius.

globalScope> Global triangle_ppp (v1, v2, v3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

Replace the hull() operation with calculated tangential intersection of the rounded vertexes.

Remove the all or nothing requirement for vertex rounding.

4 Module Index

4.1 Modules

Here is a list of all modules:

5 File Index

5.1 File List

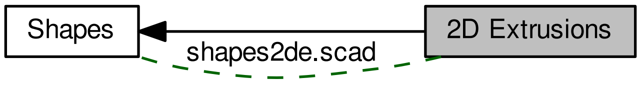
Here is a list of all documented files with brief descriptions:

6 Module Documentation

6.1 2D Extrusions

Extruded two dimensional geometric shapes.

Collaboration diagram for 2D Extrusions:



Files

- file shapes2de.scad

  Linearly extruded two-dimensional basic shapes.

Functions

- module erectangle (size, h, vr, vrm=0, center=false)

  An extruded rectangle with edge, fillet, and/or chamfer corners.
- module erectangle_c (size, core, h, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)

  An extruded rectangle with a removed rectangular core.
- module erhombus (size, h, vr, center=false)

  An extruded rhombus.
- module etriangle_ppp (v1, v2, v3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by three vertices.
- module etriangle_vp (v, h, vr, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a vector of its three vertices.
- module etriangle_lll (s1, s2, s3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by its three side lengths.
- module etriangle_vl (v, h, vr, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a vector of its three side lengths.
- module etriangle_vl_c (vs, vc, h, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false, center=false)

  A general triangle specified by its sides with a removed triangular core.
- module etriangle_lal (s1, a, s2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by two sides and the included angle.
- module etriangle_ala (a1, s, a2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a side and two adjacent angles.
- module etriangle_aal (a1, a2, s, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a side, one adjacent angle and the opposite angle.
- module etriangle_ll (x, y, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded right-angled triangle specified by its opposite and adjacent side lengths.
- module etriangle_la (x, y, aa, oa, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

An extruded right-angled triangle specified by a side length and an angle.

- module engon (n, r, h, vr, center=false)

  An extruded n-sided equiangular/equilateral regular polygon.
- module eellipse (size, h, center=false)

  An extruded ellipse.
- module eellipse_c (size, core, h, t, co, cr=0, center=false)

  An extruded ellipse with a removed elliptical core.
- module eellipse_s (size, h, a1=0, a2=0, center=false)

  An extruded ellipse sector.
- module eellipse_cs (size, core, h, t, a1=0, a2=0, co, cr=0, center=false)

  An extruded sector of an ellipse with a removed elliptical core.
- module estar2d (size, h, n=5, vr, center=false)

  An extruded two dimensional star.

6.1.1 Detailed Description

Extruded two dimensional geometric shapes.

6.1.2 Function Documentation

6.1.2.1 module eellipse (size , h , center = `false`)

An extruded ellipse.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

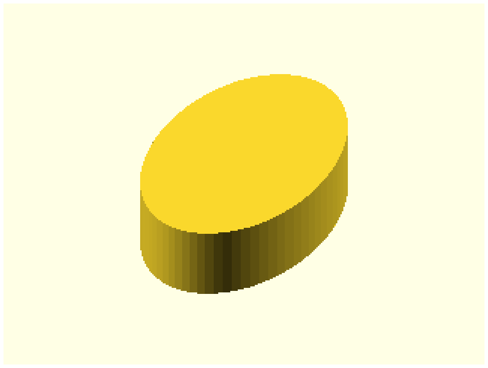


Figure 1: eellipse

```
eellipse( size=[25, 40], h=20, center=true );
```

Definition at line 767 of file shapes2de.scad.

6.1.2.2 module eellipse_c (size , core , h , t , co , cr = 0, center = false)

An extruded ellipse with a removed elliptical core.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
core	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
t	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

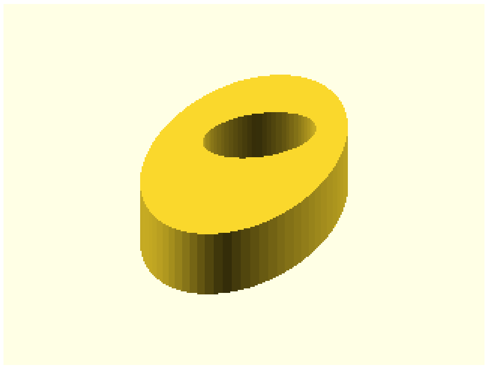


Figure 2: eellipse_c

```
eellipse_c( size=[25,40], core=[16,10], co=[0,10], cr=45, h=20, center=true );
```

Definition at line 807 of file shapes2de.scad.

6.1.2.3 module eellipse_cs (size , core , h , t , a1 = 0, a2 = 0, co , cr = 0, center = false)

An extruded sector of an ellipse with a removed elliptical core.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
core	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
t	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

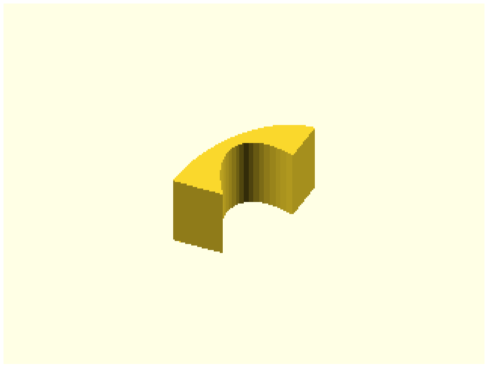


Figure 3: eellipse_cs

```
eellipse_cs( size=[25,40], t=[10,5], a1=90, a2=180, co=[10,0], cr=45, h=20, center=true );
```

Definition at line 887 of file shapes2de.scad.

6.1.2.4 module eellipse_s (size , h , a1 = 0, a2 = 0, center = false)

An extruded ellipse sector.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).

h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

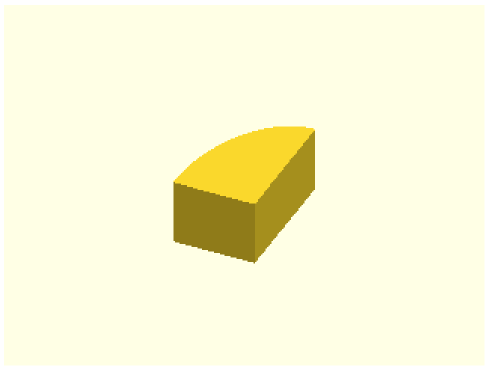


Figure 4: eellipse_s

```
eellipse_s( size=[25,40], h=20, a1=90, a2=180, center=true );
```

Definition at line 842 of file shapes2de.scad.

6.1.2.5 module engon (n , r , h , vr , center = `false`)

An extruded n-sided equiangular/equilateral regular polygon.

Parameters

n	<decimal> The number of sides.
r	<decimal> The ngon vertex radius.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The vertex rounding radius.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

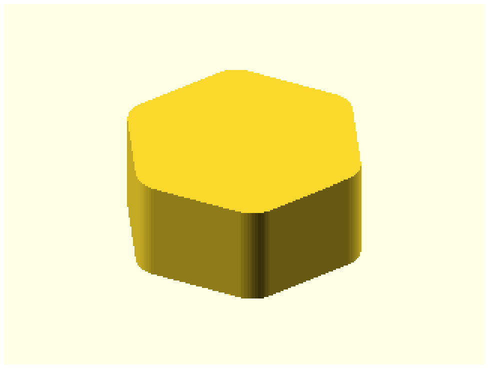


Figure 5: engon

```
engon( n=6, r=25, h=20, vr=6, center=true );
```

See `Wikipedia` for more information.

Definition at line 737 of file shapes2de.scad.

6.1.2.6 module erectangle (size , h , vr , vrm = `0`, center = `false`)

An extruded rectangle with edge, fillet, and/or chamfer corners.

Parameters

size	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<vector|decimal> The corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
vrm	<integer> The corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value **0** for *fillet* and **1** for *chamfer*.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

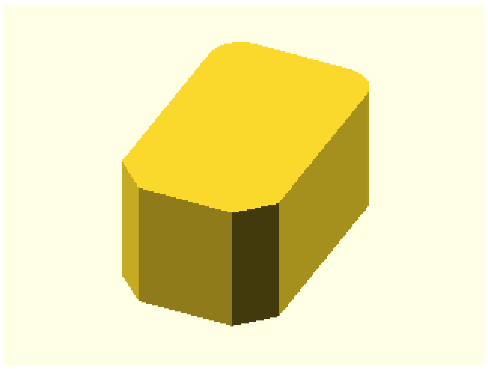


Figure 6: erectangle

```
erectangle( size=[25,40], vr=5, vrm=3, h=20, center=true );
```

Definition at line 110 of file shapes2de.scad.

6.1.2.7 module erectangle_c (size , core , h , t , co , cr = `0`, vr , vr1 , vr2 , vrm = `0`, vrm1 , vrm2 , center = `false`)

An extruded rectangle with a removed rectangular core.

Parameters

size	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
core	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
t	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<vector|decimal> The default corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
vr1	<vector|decimal> The outer corner rounding radius.
vr2	<vector|decimal> The core corner rounding radius.
vrm	<integer> The default corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value **0** for *fillet* and **1** for *chamfer*.
vrm1	<integer> The outer corner radius mode.
vrm2	<integer> The core corner radius mode.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.

- `size = core + t`; when `t` and `core` are given.

Example

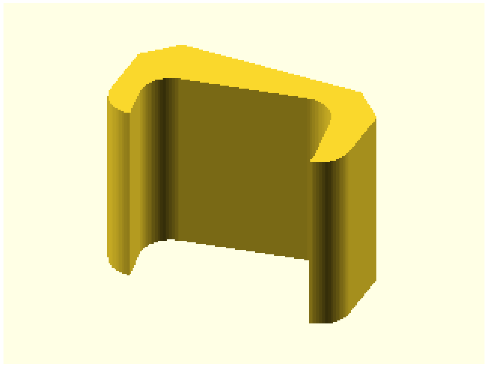


Figure 7: erectangle_c

```
erectangle_c( size=[40,20], t=[10,1], co=[0,-6], cr=10, vr=5, vrm1=12, h=30, center=true );
```

Definition at line 164 of file shapes2de.scad.

6.1.2.8 module erhombus (size , h , vr , center = `false`)

An extruded rhombus.

Parameters

size	<vector|decimal> A vector [w, h] of decimals or a single decimal for (w=h).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<vector|decimal> The corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

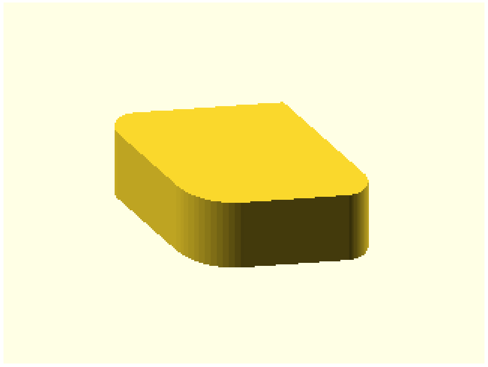


Figure 8: erhombus

```
erhombus( size=[40,25], h=10, vr=[3,0,3,9], center=true );
```

Definition at line 213 of file shapes2de.scad.

6.1.2.9 module estar2d (size , h , n = 5, vr , center = false)

An extruded two dimensional star.

Parameters

size	<vector|decimal> A vector [l, w] of decimals or a single decimal for (size=l=2∗w).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
n	<decimal> The number of points.
vr	<vector|decimal> The vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

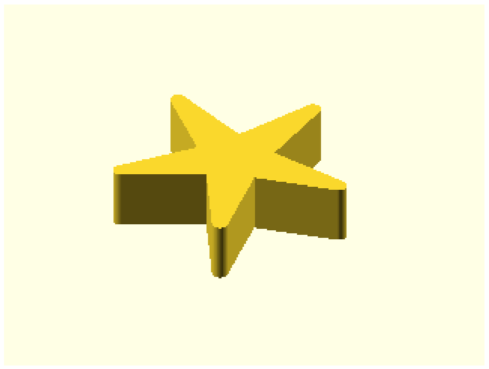


Figure 9: estar2d

```
estar2d( size=[40, 15], h=15, n=5, vr=2, center=true );
```

Definition at line 926 of file shapes2de.scad.

6.1.2.10 module etriangle_aal (a1 , a2 , s , h , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false, center = false)

An extruded general triangle specified by a side, one adjacent angle and the opposite angle.

Parameters

a1	<decimal> The opposite angle 1 in degrees.

a2	<decimal> The adjacent angle 2 in degrees.
s	<decimal> The side length.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
x	<decimal> The side to draw on the positive x-axis (`x=1` for `s`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

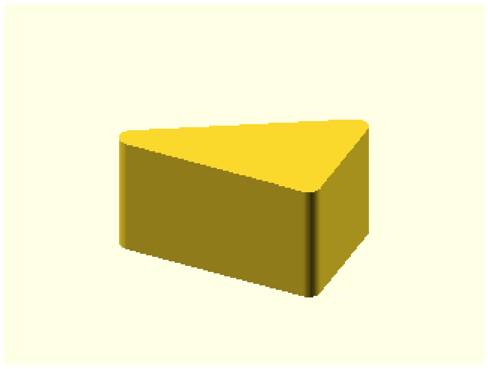


Figure 10: etriangle_aal

```
etriangle_aal( a1=60, a2=30, s=40, h=20, vr=2, centroid=true, center=true );
```

Definition at line 589 of file shapes2de.scad.

6.1.2.11 module etriangle_ala (a1 , s , a2 , h , x = `1`, vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by a side and two adjacent angles.

Parameters

a1	<decimal> The adjacent angle 1 in degrees.
s	<decimal> The side length adjacent to the angles.
a2	<decimal> The adjacent angle 2 in degrees.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.

x	<decimal> The side to draw on the positive x-axis (`x=1` for `s`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

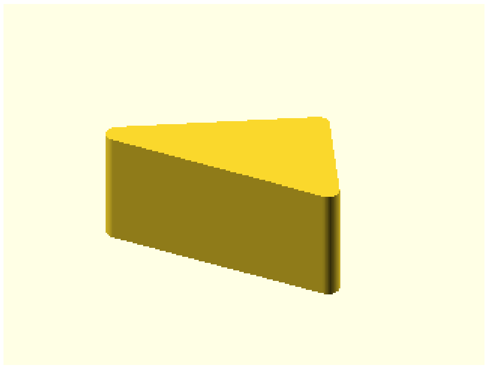


Figure 11: etriangle_ala

```
etriangle_ala( a1=30, s=50, a2=60, h=20, vr=2, centroid=true, center=true );
```

Definition at line 537 of file shapes2de.scad.

6.1.2.12 module etriangle_la (x , y , aa , oa , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded right-angled triangle specified by a side length and an angle.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
aa	<decimal> The adjacent angle in degrees.
oa	<decimal> The opposite angle in degrees.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.

v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

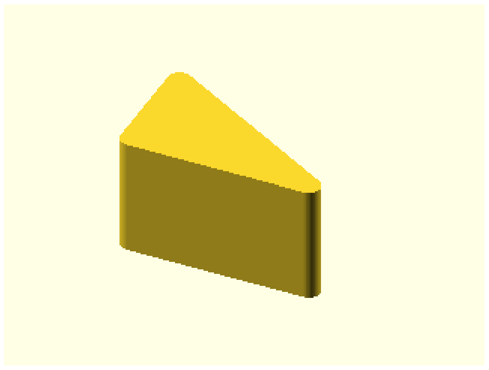


Figure 12: etriangle_la

```
etriangle_la( x=40, aa=30, h=20, vr=2, centroid=true, center=true );
```

Note

When both `x` and `y` are given, both triangles are rendered.
When both `aa` and `oa` are given, `aa` is used.

Definition at line 690 of file shapes2de.scad.

6.1.2.13 module etriangle_lal (s1 , a , s2 , h , x = `1`, vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by two sides and the included angle.

Parameters

s1	<decimal> The length of the side 1.
a	<decimal> The included angle in degrees.
s2	<decimal> The length of the side 2.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
x	<decimal> The side to draw on the positive x-axis (`x=1` for `s1`).
vr	<decimal> The default vertex rounding radius.

v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

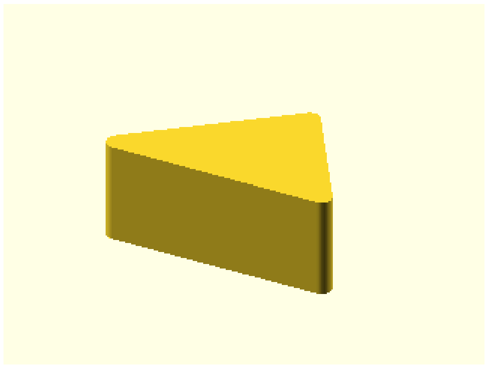


Figure 13: etriangle_lal

```
etriangle_lal( s1=50, a=60, s2=30, h=20, vr=2, centroid=true, center=true );
```

Definition at line 485 of file shapes2de.scad.

6.1.2.14 module etriangle_ll (x , y , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded right-angled triangle specified by its opposite and adjacent side lengths.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

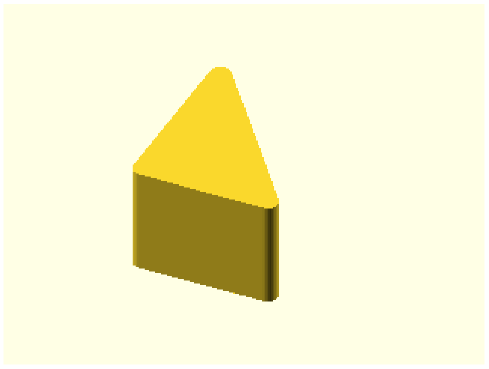


Figure 14: etriangle_ll

```
etriangle_ll( x=30, y=40, h=20, vr=2, centroid=true, center=true );
```

Definition at line 638 of file shapes2de.scad.

6.1.2.15 module etriangle_lll (s1 , s2 , s3 , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by its three side lengths.

Parameters

s1	<decimal> The length of the side 1 (along the x-axis).
s2	<decimal> The length of the side 2.
s3	<decimal> The length of the side 3.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

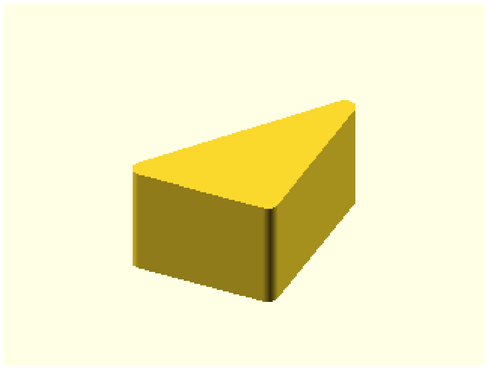


Figure 15: etriangle_lll

```
etriangle_lll( s1=30, s2=40, s3=50, h=20, vr=2, centroid=true, center=true );
```

Definition at line 338 of file shapes2de.scad.

6.1.2.16 module etriangle_ppp (v1 , v2 , v3 , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by three vertices.

Parameters

v1	<vector> A vector [x, y] for vertex 1.
v2	<vector> A vector [x, y] for vertex 2.
v3	<vector> A vector [x, y] for vertex 3.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

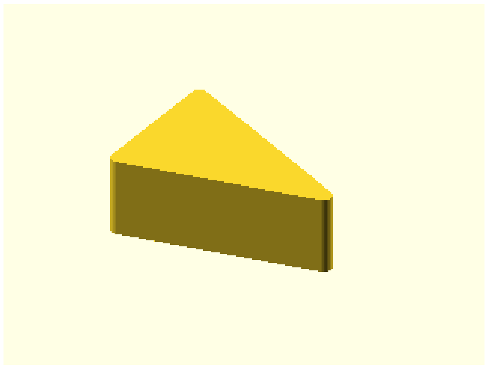


Figure 16: etriangle_ppp

```
etriangle_ppp( v1=[0,0], v2=[5,25], v3=[40,5], h=20, vr=2, centroid=true, center=true );
```

Definition at line 250 of file shapes2de.scad.

6.1.2.17 module etriangle_vl (v , h , vr , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by a vector of its three side lengths.

Parameters

v	<vector> A vector [s1, s2, s3] of decimals.
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<vector|decimal> The vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

```
t = triangle_lll2vp( 3, 4, 5 );
s = triangle_vp2vl( t );
etriangle_vl( v=s, h=5, vr=2 );
```

Definition at line 387 of file shapes2de.scad.

6.1.2.18 module etriangle_vl_c (vs , vc , h , co , cr = `0`, vr , vr1 , vr2 , centroid = `false`, incenter = `false`, center = `false`)

A general triangle specified by its sides with a removed triangular core.

Parameters

vs	<vector|decimal> The size. A vector [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
vc	<vector|decimal> The core. A vector [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<vector|decimal> The default vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
vr1	<vector|decimal> The outer vertex rounding radius.
vr2	<vector|decimal> The core vertex rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

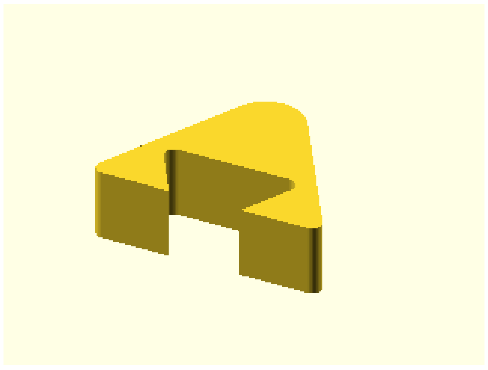


Figure 17: etriangle_vl_c

```
etriangle_vl_c(vs=50, vc=30, h=15, co=[0,-10], cr=180, vr=[2,2,8], centroid=true, center=true);
```

Note

The outer and inner triangles centroids are aligned prior to the core removal.

Definition at line 433 of file shapes2de.scad.

6.1.2.19 module etriangle_vp (v , h , vr , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by a vector of its three vertices.

Parameters

v	<vector> A vector [v1, v2, v3] of vectors [x, y].
h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.
vr	<vector|decimal> The vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

st_linear_extrude_scale for a description on specifying `h`.

Example

```
t = triangle_lll2vp( 30, 40, 50 );
r = [2, 4, 6];
etriangle_vp( v=t, h=5, vr=r );
```

Definition at line 299 of file shapes2de.scad.

6.2 2D Shapes

Two dimensional geometric shapes.

Collaboration diagram for 2D Shapes:

2D Shapes
shapes2d.scad
Shapes

Files

- file shapes2d.scad

  Two-dimensional basic shapes.

Functions

- module rectangle (size, vr, vrm=0, center=false)

  A rectangle with edge, fillet, and/or chamfer corners.
- module rectangle_c (size, core, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)

  A rectangle with a removed rectangular core.
- module rhombus (size, vr, center=false)

  A rhombus.
- module triangle_ppp (v1, v2, v3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by three vertices.
- module triangle_vp (v, vr, centroid=false, incenter=false)

  A general triangle specified by a vector of its three vertices.
- module triangle_lll (s1, s2, s3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by its three side lengths.
- module triangle_vl (v, vr, centroid=false, incenter=false)

  A general triangle specified by a vector of its three side lengths.
- module triangle_vl_c (vs, vc, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false)

  A general triangle specified by its sides with a removed triangular core.
- module triangle_lal (s1, a, s2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by two sides and the included angle.
- module triangle_ala (a1, s, a2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by a side and two adjacent angles.
- module triangle_aal (a1, a2, s, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by a side, one adjacent angle and the opposite angle.
- module triangle_ll (x, y, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A right-angled triangle specified by its opposite and adjacent side lengths.
- module triangle_la (x, y, aa, oa, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A right-angled triangle specified by a side length and an angle.

- module ngon (n, r, vr)

  An n-sided equiangular/equilateral regular polygon.
- module ellipse (size)

  An ellipse.
- module ellipse_c (size, core, t, co, cr=0)

  An ellipse with a removed elliptical core.
- module ellipse_s (size, a1=0, a2=0)

  An ellipse sector.
- module ellipse_cs (size, core, t, a1=0, a2=0, co, cr=0)

  A sector of an ellipse with a removed elliptical core.
- module star2d (size, n=5, vr)

  A two dimensional star.

6.2.1 Detailed Description

Two dimensional geometric shapes.

6.2.2 Function Documentation

6.2.2.1 module ellipse (size)

An ellipse.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).

Example

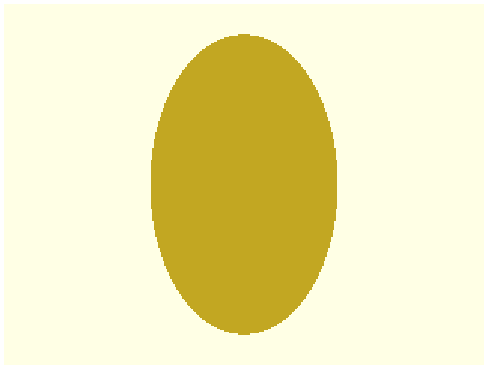


Figure 18: ellipse

```
ellipse( size=[25, 40] );
```

Definition at line 1088 of file shapes2d.scad.

6.2.2.2 module ellipse_c (size , core , t , co , cr = 0)

An ellipse with a removed elliptical core.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
core	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
t	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

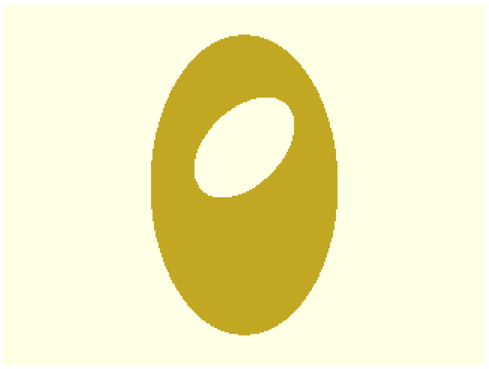


Figure 19: ellipse_c

```
ellipse_c( size=[25,40], core=[16,10], co=[0,10], cr=45 );
```

Definition at line 1129 of file shapes2d.scad.

6.2.2.3 module ellipse_cs (size , core , t , a1 = 0, a2 = 0, co , cr = 0)

A sector of an ellipse with a removed elliptical core.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
core	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
t	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

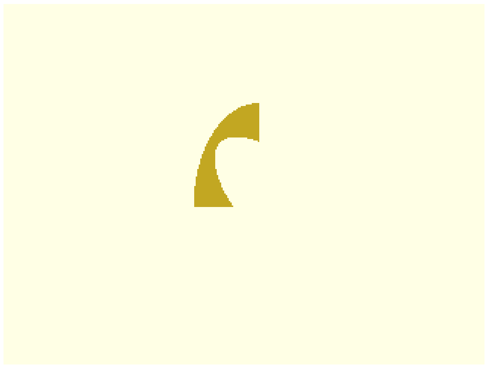


Figure 20: ellipse_cs

```
ellipse_cs( size=[25,40], t=[10,5], a1=90, a2=180, co=[10,0], cr=45);
```

Definition at line 1239 of file shapes2d.scad.

6.2.2.4 module ellipse_s (size , a1 = 0, a2 = 0)

An ellipse sector.

Parameters

size	<vector|decimal> A vector [rx, ry] of decimals or a single decimal for (rx=ry).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.

Example

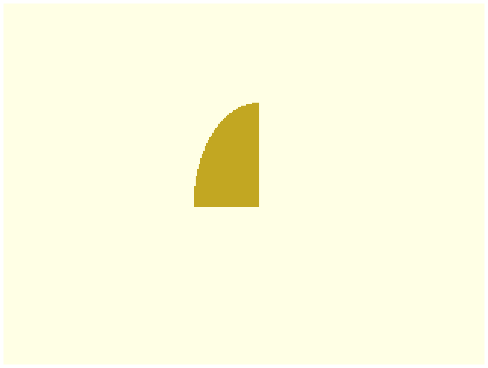


Figure 21: ellipse_s

```
ellipse_s( size=[25,40], a1=90, a2=180 );
```

Definition at line 1171 of file shapes2d.scad.

6.2.2.5 module ngon (n , r , vr)

An n-sided equiangular/equilateral regular polygon.

Parameters

n	<decimal> The number of sides.
r	<decimal> The ngon vertex radius.
vr	<decimal> The vertex rounding radius.

Example

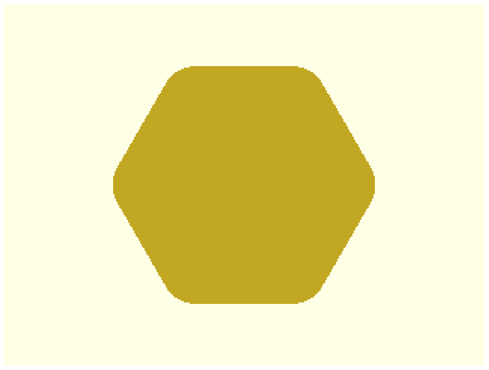


Figure 22: ngon

```
ngon( n=6, r=25, vr=6 );
```

See Wikipedia for more information.

Definition at line 1054 of file shapes2d.scad.

6.2.2.6 module rectangle (size , vr , vrm = 0, center = false)

A rectangle with edge, fillet, and/or chamfer corners.

Parameters

size	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
vr	<vector|decimal> The corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
vrm	<integer> The corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value **0** for *fillet* and **1** for *chamfer*.
center	<boolean> Center about origin.

Example

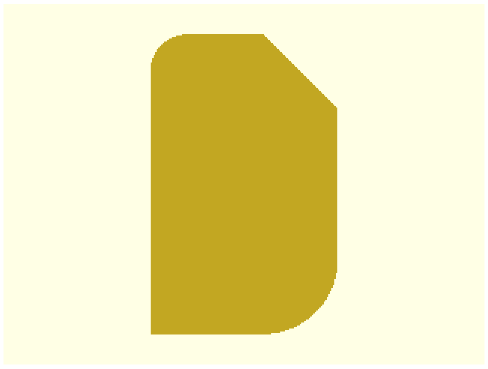


Figure 23: rectangle

```
rectangle( size=[25,40], vr=[0,10,10,5], vrm=4, center=true );
```

Note

A corner *fillet* replaces an edge with a quarter circle of radius `vr`, inset `[vr, vr]` from the corner vertex.
A corner *chamfer* replaces an edge with an isosceles right triangle with side lengths equal to the corresponding corner rounding radius `vr`. Therefore the chamfer length will be `vr*sqrt(2)` at 45 degree angles.

Definition at line 112 of file shapes2d.scad.

6.2.2.7 module rectangle_c (size , core , t , co , cr = 0, vr , vr1 , vr2 , vrm = 0, vrm1 , vrm2 , center = false)

A rectangle with a removed rectangular core.

Parameters

size	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
core	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
t	<vector|decimal> A vector [x, y] of decimals or a single decimal for (x=y).
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<vector|decimal> The default corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
vr1	<vector|decimal> The outer corner rounding radius.
vr2	<vector|decimal> The core corner rounding radius.
vrm	<integer> The default corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value **0** for *fillet* and **1** for *chamfer*.
vrm1	<integer> The outer corner radius mode.
vrm2	<integer> The core corner radius mode.
center	<boolean> Center about origin.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

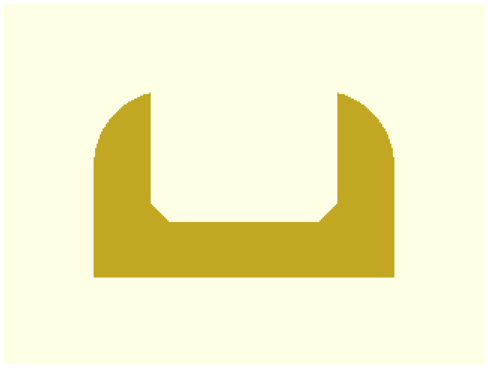


Figure 24: rectangle_c

```
rectangle_c( size=[40,25], t=[15,5], vr1=[0,0,10,10], vr2=2.5, vrm2=3, co=[0,5], center=true );
```

Definition at line 232 of file shapes2d.scad.

6.2.2.8 module rhombus (size , vr , center = `false`)

A rhombus.

Parameters

size	<vector|decimal> A vector [w, h] of decimals or a single decimal for (w=h).
vr	<vector|decimal> The corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
center	<boolean> Center about origin.

Example

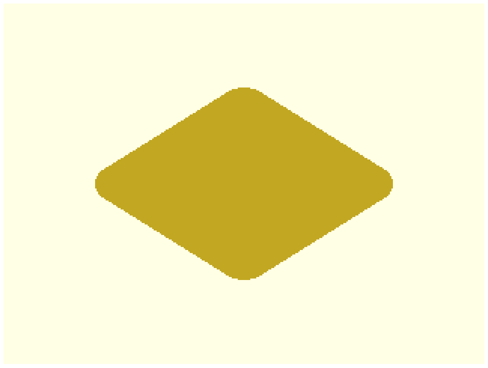


Figure 25: rhombus

```
rhombus( size=[40,25], vr=[2,4,2,4], center=true );
```

See `Wikipedia` for more information.

Definition at line 297 of file shapes2d.scad.

6.2.2.9 module star2d (size , n = 5, vr)

A two dimensional star.

Parameters

size	<vector|decimal> A vector [l, w] of decimals or a single decimal for (size=l=2∗w).
n	<decimal> The number of points.
vr	<vector|decimal> The vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).

Example

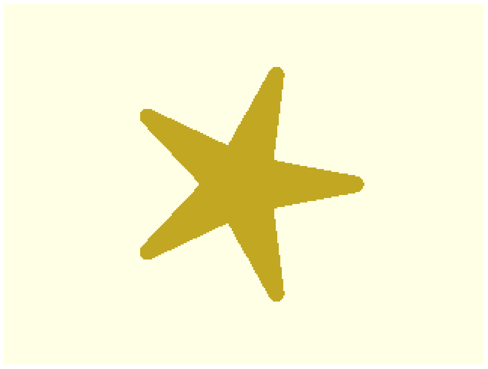


Figure 26: star2d

```
star2d( size=[40, 15], n=5, vr=2 );
```

Definition at line 1285 of file shapes2d.scad.

6.2.2.10 module triangle_aal (a1 , a2 , s , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false)

A general triangle specified by a side, one adjacent angle and the opposite angle.

Parameters

a1	<decimal> The opposite angle 1 in degrees.
a2	<decimal> The adjacent angle 2 in degrees.
s	<decimal> The side length.
x	<decimal> The side to draw on the positive x-axis (`x=1` for `s`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

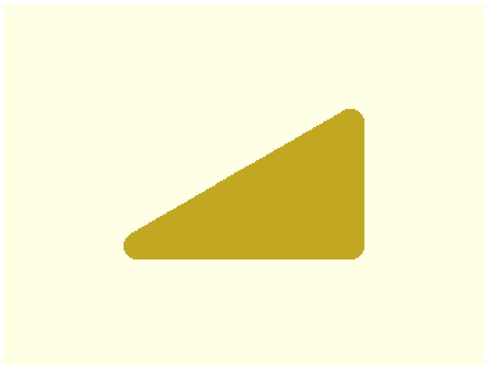


Figure 27: triangle_aal

```
triangle_aal( a1=60, a2=30, s=40, vr=2, centroid=true );
```

See `Wikipedia` for more information.

Definition at line 882 of file shapes2d.scad.

6.2.2.11 module triangle_ala (a1 , s , a2 , x = `1`, vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A general triangle specified by a side and two adjacent angles.

Parameters

a1	<decimal> The adjacent angle 1 in degrees.
s	<decimal> The side length adjacent to the angles.
a2	<decimal> The adjacent angle 2 in degrees.
x	<decimal> The side to draw on the positive x-axis (`x=1` for `s`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

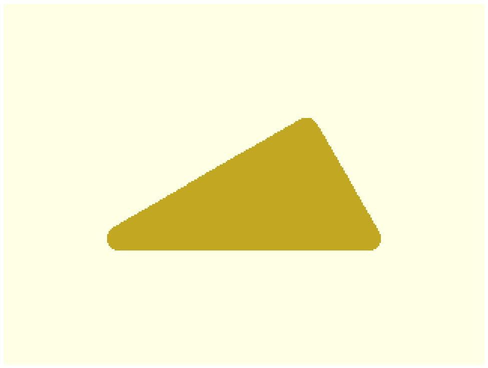


Figure 28: triangle_ala

```
triangle_ala( a1=30, s=50, a2=60, vr=2, centroid=true );
```

See Wikipedia for more information.

Definition at line 799 of file shapes2d.scad.

6.2.2.12 module triangle_la (x , y , aa , oa , vr , v1r , v2r , v3r , centroid = false, incenter = false)

A right-angled triangle specified by a side length and an angle.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
aa	<decimal> The adjacent angle in degrees.
oa	<decimal> The opposite angle in degrees.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

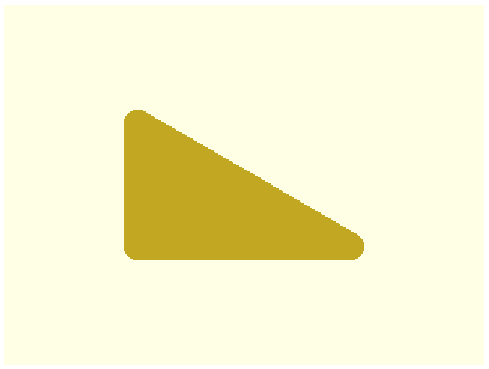


Figure 29: triangle_la

```
triangle_la( x=40, aa=30, vr=2, centroid=true );
```

Note

When both x and y are given, both triangles are rendered.
When both aa and oa are given, aa is used.

Definition at line 1002 of file shapes2d.scad.

6.2.2.13 module triangle_lal (s1 , a , s2 , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false)

A general triangle specified by two sides and the included angle.

Parameters

s1	<decimal> The length of the side 1.
a	<decimal> The included angle in degrees.
s2	<decimal> The length of the side 2.
x	<decimal> The side to draw on the positive x-axis (`x=1` for `s1`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

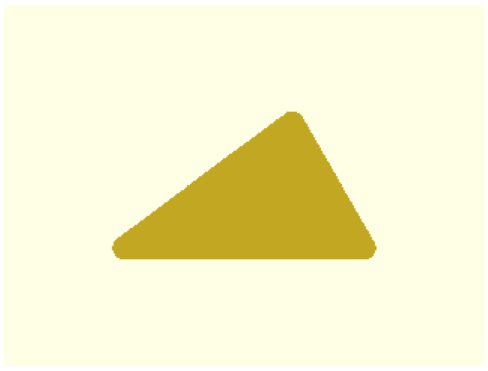


Figure 30: triangle_lal

```
triangle_lal( s1=50, a=60, s2=30, vr=2, centroid=true );
```

See `Wikipedia` for more information.

Definition at line 730 of file shapes2d.scad.

6.2.2.14 module triangle_ll (x , y , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A right-angled triangle specified by its opposite and adjacent side lengths.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

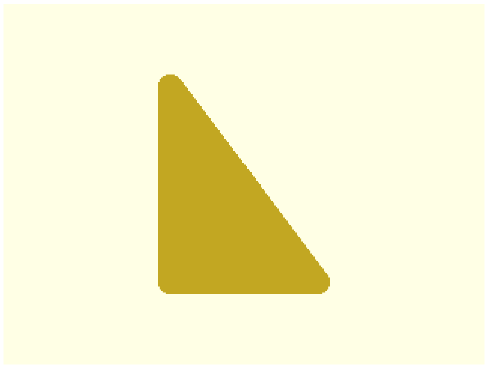


Figure 31: triangle_ll

```
triangle_ll( x=30, y=40, vr=2, centroid=true );
```

Definition at line 959 of file shapes2d.scad.

6.2.2.15 module triangle_lll (s1 , s2 , s3 , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A general triangle specified by its three side lengths.

Parameters

s1	<decimal> The length of the side 1 (along the x-axis).
s2	<decimal> The length of the side 2.
s3	<decimal> The length of the side 3.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

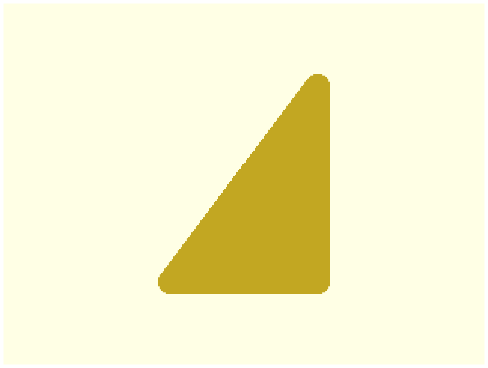


Figure 32: triangle_lll

```
triangle_lll( s1=30, s2=40, s3=50, vr=2, centroid=true );
```

See `Wikipedia` for more information.

Definition at line 524 of file shapes2d.scad.

6.2.2.16 module triangle_ppp (v1 , v2 , v3 , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A general triangle specified by three vertices.

Parameters

v1	<vector> A vector [x, y] for vertex 1.
v2	<vector> A vector [x, y] for vertex 2.
v3	<vector> A vector [x, y] for vertex 3.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

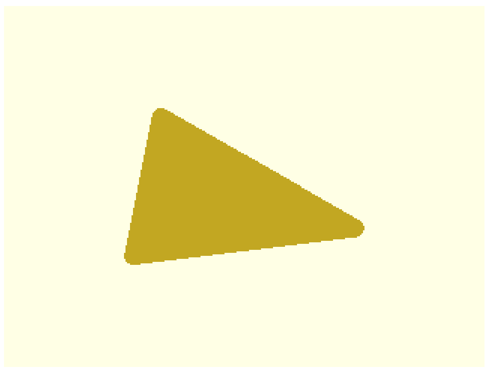


Figure 33: triangle_ppp

```
triangle_ppp( v1=[0,0], v2=[5,25], v3=[40,5], vr=2, centroid=true );
```

Warning

Currently, in order to round any vertex, all must be given a rounding radius, either via `vr` or individually.

Todo Replace the hull() operation with calculated tangential intersection of the rounded vertexes.

Remove the all or nothing requirement for vertex rounding.

Definition at line 398 of file shapes2d.scad.

6.2.2.17 module triangle_vl (v , vr , centroid = `false`, incenter = `false`)

A general triangle specified by a vector of its three side lengths.

Parameters

v	<vector> A vector [s1, s2, s3] of decimals.
vr	<vector|decimal> The vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

```
t = triangle_lll2vp( 3, 4, 5 );
s = triangle_vp2vl( t );
triangle_vl( v=s, vr=2, centroid=true );
```

Definition at line 584 of file shapes2d.scad.

6.2.2.18 module triangle_vl_c (vs , vc , co , cr = 0, vr , vr1 , vr2 , centroid = false, incenter = false)

A general triangle specified by its sides with a removed triangular core.

Parameters

vs	<vector|decimal> The size. A vector [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
vc	<vector|decimal> The core. A vector [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<vector|decimal> The default vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
vr1	<vector|decimal> The outer vertex rounding radius.
vr2	<vector|decimal> The core vertex rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

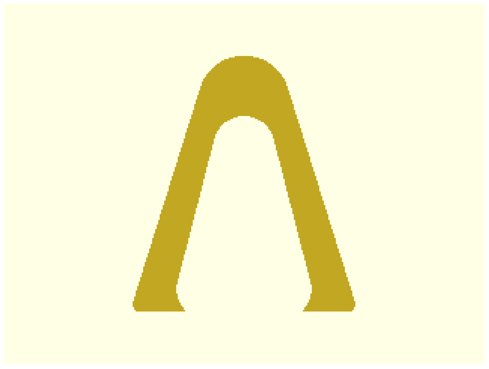


Figure 34: triangle_vl_c

```
triangle_vl_c( vs=[30,50,50], vc=[20,40,40], co=[0,-4], vr1=[1,1,6], vr2=4, centroid=true );
```

Note

The outer and inner triangles centroids are aligned prior to the core removal.

Definition at line 638 of file shapes2d.scad.

6.2.2.19 module triangle_vp (v , vr , centroid = `false`, incenter = `false`)

A general triangle specified by a vector of its three vertices.

Parameters

v	<vector> A vector [v1, v2, v3] of vectors [x, y].
vr	<vector|decimal> The vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

```
t = triangle_lll2vp( 30, 40, 50 );
r = [2, 4, 6];
triangle_vp( v=t, vr=r  );
```

Definition at line 474 of file shapes2d.scad.

6.3 3D Shapes

Three dimensional geometric shapes.

Collaboration diagram for 3D Shapes:

Files

- file shapes3d.scad

  Three-dimensional basic shapes.

Functions

- module cone (r, h, d, vr, vr1, vr2)

  A cone.
- module cuboid (size, vr, vrm=0, center=false)

  A cuboid with edge, fillet, or chamfer corners.
- module ellipsoid (size)

  An ellipsoid.
- module ellipsoid_s (size, a1=0, a2=0)

  A sector of an ellipsoid.
- module tetrahedron (size, center=false)

  A pyramid with trilateral base formed by four equilateral triangles.
- module pyramid_q (size, center=false)

  A pyramid with quadrilateral base.
- module star3d (size, n=5, half=false)

  A three dimensional star.
- module torus_rp (size, core, r, l, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, pa=0, ra=360, m=255, center=false, profile=false)

  A rectangular cross-sectional profile revolved about the z-axis.
- module torus_tp (size, core, r, l, co, cr=0, vr, vr1, vr2, pa=0, ra=360, m=255, centroid=false, incenter=false, profile=false,)

  A triangular cross-sectional profile revolved about the z-axis.
- module torus_ep (size, core, r, l, t, a1=0, a2=0, co, cr=0, pa=0, ra=360, m=255, profile=false)

  An elliptical cross-sectional profile revolved about the z-axis.

6.3.1 Detailed Description

Three dimensional geometric shapes.

6.3.2 Function Documentation

6.3.2.1 module cone (r , h , d , vr , vr1 , vr2)

A cone.

Parameters

r	<decimal> The base radius.
h	<decimal> The height.
d	<decimal> The base diameter.
vr	<decimal> The default corner rounding radius.
vr1	<decimal> The base corner rounding radius.
vr2	<decimal> The point corner rounding radius.

Example

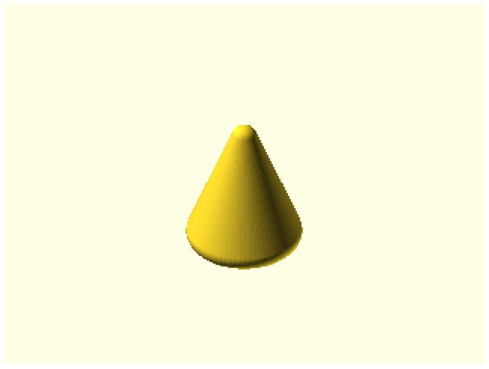


Figure 35: cone

```
cone( h=25, r=10, vr=2 );
```

Definition at line 103 of file shapes3d.scad.

6.3.2.2 module cuboid (size , vr , vrm = 0, center = false)

A cuboid with edge, fillet, or chamfer corners.

Parameters

size	<vector|decimal> A vector [x, y, z] of decimals or a single decimal for (x=y=z).
vr	<decimal> The rounding radius.
vrm	<integer> The radius mode. A 2-bit encoded integer that indicates edge and vertex finish. *B0* controls edge and *B1* controls vertex.
center	<boolean> Center about origin.

Example

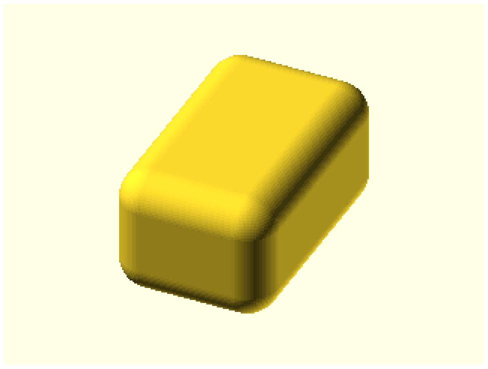


Figure 36: cuboid

```
cuboid( size=[25,40,20], vr=5, center=true );
```

vrm	B1	B0	Description
0	0	0	*fillet* edges with *fillet* vertexes
1	0	1	*chamfer* edges with *sphere* vertexes
2	1	0	*fillet* edges with *chamfer* vertexes
3	1	1	*chamfer* edges with *chamfer* vertexes

Note

Using *fillet* replaces all edges with a quarter circle of radius `vr`, inset `[vr, vr]` from the each edge.
Using *chamfer* replaces all edges with isosceles right triangles with side lengths equal to the corner rounding radius `vr`. Therefore the chamfer length will be `vr*sqrt(2)` at 45 degree angles.

Definition at line 168 of file shapes3d.scad.

6.3.2.3 module ellipsoid (size)

An ellipsoid.

Parameters

size	<vector|decimal> A vector [w, h] of decimals or a single decimal for (w=h).

Example

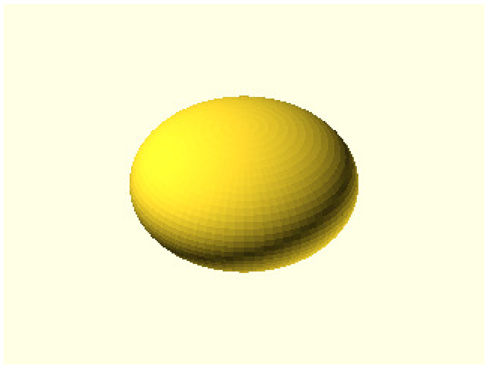


Figure 37: ellipsoid

```
ellipsoid( size=[40,25] );
```

Definition at line 253 of file shapes3d.scad.

6.3.2.4 module ellipsoid_s (size , a1 = 0, a2 = 0)

A sector of an ellipsoid.

Parameters

size	<vector|decimal> A vector [w, h] of decimals or a single decimal for (w=h).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.

Example

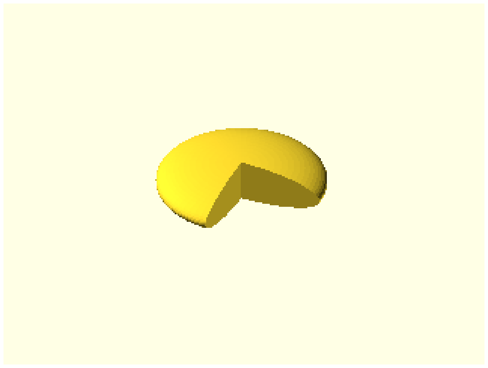


Figure 38: ellipsoid_s

```
ellipsoid_s( size=[60,15], a1=0, a2=270 );
```

Definition at line 285 of file shapes3d.scad.

6.3.2.5 module pyramid_q (size , center = false)

A pyramid with quadrilateral base.

Parameters

size	<vector|decimal> A vector [x, y, z] of decimals or a single decimal for (x=y=z).
center	<boolean> Center about origin.

Example

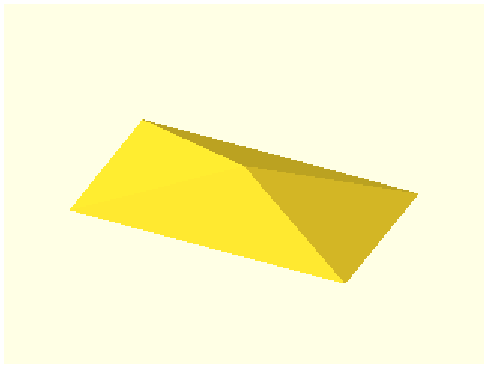


Figure 39: pyramid_q

```
pyramid_q( size=[35,20,5], center=true );
```

Todo Support vertex rounding radius.

Definition at line 387 of file shapes3d.scad.

6.3.2.6 module star3d (size , n = 5, half = false)

A three dimensional star.

Parameters

size	<vector|decimal> A vector [l, w, h] of decimals or a single decimal for (size=l=2∗w=4∗h).
n	<decimal> The number of points.
half	<boolean> Render upper half only.

Example

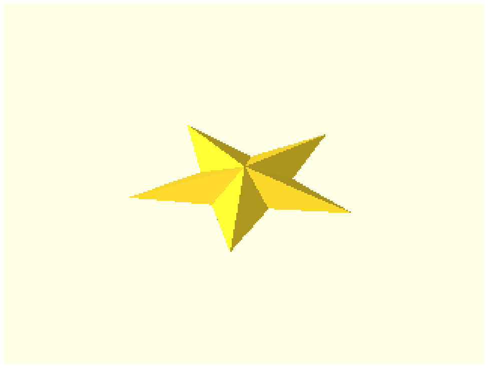


Figure 40: star3d

```
star3d( size=40, n=5, half=true );
```

Definition at line 433 of file shapes3d.scad.

6.3.2.7 module tetrahedron (size , center = `false`)

A pyramid with trilateral base formed by four equilateral triangles.

Parameters

size	<decimal> The face radius.
center	<boolean> Center about origin.

Example

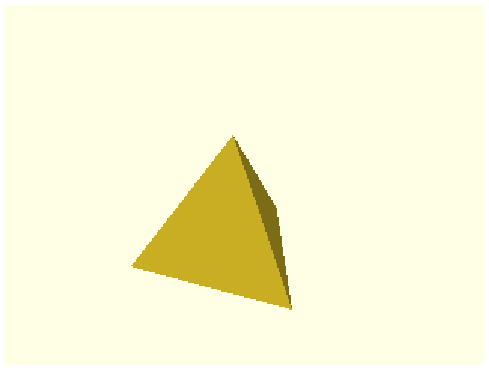


Figure 41: tetrahedron

```
tetrahedron( size=20, center=true );
```

Todo Support vertex rounding radius.

Definition at line 343 of file shapes3d.scad.

6.3.2.8 module torus_ep (size , core , r , l , t , a1 = `0`, a2 = `0`, co , cr = `0`, pa = `0`, ra = `360`, m = `255`, profile = `false`)

An elliptical cross-sectional profile revolved about the z-axis.

Parameters

size	<vector|decimal> The profile size. A vector [x, y] of decimals or a single decimal for (x=y).
core	<vector|decimal> The profile core. A vector [x, y] of decimals or a single decimal for (x=y).
r	<decimal> The rotation radius.
l	<vector|decimal> The elongation length. A vector [x, y] of decimals or a single decimal for (x=y)
t	<vector|decimal> The profile thickness. A vector [x, y] of decimals or a single decimal for (x=y).
a1	<decimal> The profile start angle in degrees.
a2	<decimal> The profile stop angle in degrees.
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
pa	<decimal> The profile pitch angle in degrees.

ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render.
profile	<boolean> Show profile only (do not extrude).

See also

st_rotate_extrude_elongate for description of extrude parameters.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

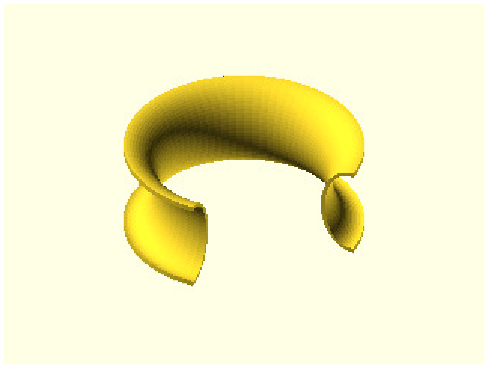


Figure 42: torus_ep

```
torus_ep( size=[20,15], t=[2,4], r=50, a1=0, a2=180, pa=90, ra=270, co=[0,2] );
```

Definition at line 654 of file shapes3d.scad.

6.3.2.9 module torus_rp (size , core , r , l , t , co , cr = 0, vr , vr1 , vr2 , vrm = 0, vrm1 , vrm2 , pa = 0, ra = 360, m = 255, center = false, profile = false)

A rectangular cross-sectional profile revolved about the z-axis.

Parameters

size	<vector|decimal> The profile size. A vector [x, y] of decimals or a single decimal for (x=y).
core	<vector|decimal> The profile core. A vector [x, y] of decimals or a single decimal for (x=y).
r	<decimal> The rotation radius.
l	<vector|decimal> The elongation length. A vector [x, y] of decimals or a single decimal for (x=y)
t	<vector|decimal> The profile thickness. A vector [x, y] of decimals or a single decimal for (x=y).
co	<vector> Core offset. A vector [x, y] of decimals.

cr	<decimal> Core z-rotation.
vr	<vector|decimal> The profile default corner rounding radius. A vector [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
vr1	<vector|decimal> The profile outer corner rounding radius.
vr2	<vector|decimal> The profile core corner rounding radius.
vrm	<integer> The default corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value **0** for *fillet* and **1** for *chamfer*.
vrm1	<integer> The outer corner radius mode.
vrm2	<integer> The core corner radius mode.
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render.
center	<boolean> Rotate about profile center.
profile	<boolean> Show profile only (do not extrude).

See also

st_rotate_extrude_elongate for description of extrude parameters.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

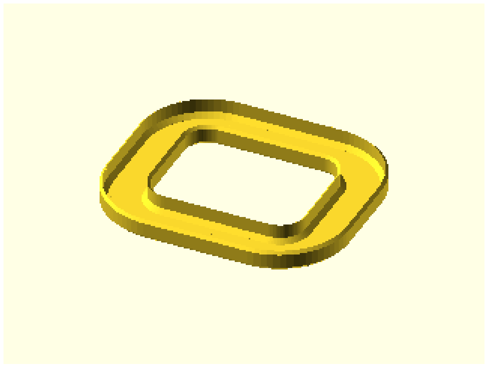


Figure 43: torus_rp

```
torus_rp( size=[40,20], core=[35,20], r=40, l=[90,60], co=[0,2.5], vr=4, vrm=15, center=true );
```

Definition at line 516 of file shapes3d.scad.

6.3.2.10 module torus_tp (size , core , r , l , co , cr = 0, vr , vr1 , vr2 , pa = 0, ra = 360, m = 255, centroid = false, incenter = false, profile = false)

A triangular cross-sectional profile revolved about the z-axis.

Parameters

size	<vector|decimal> The size. A vector [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
core	<vector|decimal> The core. A vector [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
r	<decimal> The rotation radius.
l	<vector|decimal> The elongation length. A vector [x, y] of decimals or a single decimal for (x=y)
co	<vector> Core offset. A vector [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<vector|decimal> The default vertex rounding radius. A vector [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
vr1	<vector|decimal> The outer vertex rounding radius.
vr2	<vector|decimal> The core vertex rounding radius.
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render.
centroid	<boolean> Rotate about profile centroid.
incenter	<boolean> Rotate about profile incenter.
profile	<boolean> Show profile only (do not extrude).

See also

st_rotate_extrude_elongate for description of extrude parameters.

Example

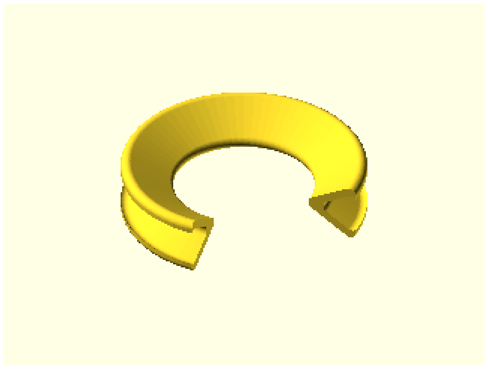


Figure 44: torus_tp

```
torus_tp( size=40, core=30, r=60, co=[0,-4], vr=4, pa=90, ra=270, centroid=true );
```

Note

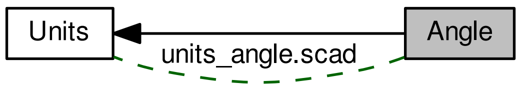
The outer and inner triangles centroids are aligned prior to the core removal.

Definition at line 587 of file shapes3d.scad.

6.4 Angle

Angle units and conversions.

Collaboration diagram for Angle:



Files

- file units_angle.scad

    Angle units and conversions.

Functions

- function unit_angle_name (units=base_unit_angle)

    Return the name of the given angle `unit` identifier.
- function convert_angle (angle, from=base_unit_angle, to=base_unit_angle)

    Convert the `angle` from `from` units to `to` units.

Variables

- base_unit_angle = "d"

    <string> Base unit for angle measurements.

6.4.1 Detailed Description

Angle units and conversions.

These functions allow for angles to be specified with units. Angles specified with units are independent of (base_unit↵_angle). There are also unit conversion functions for converting from one unit to another.

The table below enumerates the supported unit identifiers and their descriptions.

units id	description
r	radian
d	degree
dms	degree, minute, second

Example

```
include <units_angle.scad>;

base_unit_angle = "d";
```

```
    // get base unit name
    un = unit_angle_name();

    // absolute angle measurements in base unit.
    c1 = convert_angle(pi/6, "r");
    c2 = convert_angle(pi/4, "r");
    c3 = convert_angle(180, "d");
    c4 = convert_angle([30, 15, 50], "dms");

    // convert between units.
    c5 = convert_angle([30, 15, 50], from="dms", to="r");
    c6 = convert_angle(0.528205, from="r", to="dms");
```

Result (base_angle_length = **r**):

```
ECHO: un = "radian"
ECHO: c1 = 0.523599
ECHO: c2 = 0.785398
ECHO: c3 = 3.14159
ECHO: c4 = 0.528205
ECHO: c5 = 0.528205
ECHO: c6 = [30, 15, 50.102]
```

Result (base_angle_length = **d**):

```
ECHO: un = "degree"
ECHO: c1 = 30
ECHO: c2 = 45
ECHO: c3 = 180
ECHO: c4 = 30.2639
ECHO: c5 = 0.528205
ECHO: c6 = [30, 15, 50.102]
```

Result (base_angle_length = **dms**):

```
ECHO: un = "degree, minute, second"
ECHO: c1 = [29, 59, 60]
ECHO: c2 = [45, 0, 0]
ECHO: c3 = [180, 0, 0]
ECHO: c4 = [30, 15, 50]
ECHO: c5 = 0.528205
ECHO: c6 = [30, 15, 50.102]
```

6.4.2 Function Documentation

6.4.2.1 function convert_angle (angle , from = base_unit_angle, to = base_unit_angle)

Convert the `angle` from `from` units to `to` units.

Parameters

angle	<decimal|vector> An angle to convert (dms angles are 3-tuple vector [d, m, s]).
from	<string> The units of the angle to be converted.
to	<string> A units to which the angle should be converted.

Returns

<decimal|vector> The conversion result (dms angles are 3-tuple vector [d, m, s]). Returns **'undef'** for identifiers that are not defined.

6.4.2.2 function unit_angle_name (units = base_unit_angle)

Return the name of the given angle `unit` identifier.

Parameters

units	<string> An angle unit identifier.

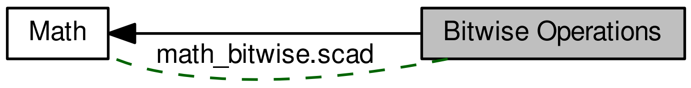
Returns

<string> The units name for the given angle unit identifier. Returns **'undef'** for identifiers that are not defined.

6.5 Bitwise Operations

Bitwise binary (base-two) operations.

Collaboration diagram for Bitwise Operations:



Files

- file math_bitwise.scad

  Mathematical bitwise binary (base-two) functions.

Functions

- function bitwise_is_equal (v, b, t=1)

  Test if a base-two bit position of an integer value equals a test bit.
- function bitwise_i2v (v, w=1, bv=1)

  Encode an integer value as a base-two vector of bits.
- function bitwise_v2i (v)

  Decode a base-two vector of bits to an integer value.
- function bitwise_i2s (v, w=1)

  Encode an integer value as a base-two string of bits.
- function bitwise_s2i (v)

  Decode a base-two string of bits to an integer value.
- function bitwise_and (v1, v2, bv=1)

  Base-two bitwise AND operation for integers.
- function bitwise_or (v1, v2, bv=1)

  Base-two bitwise OR operation for integers.
- function bitwise_xor (v1, v2, bv=1)

  Base-two bitwise XOR operation for integers.
- function bitwise_not (v, w=1, bv=1)

  Base-two bitwise NOT operation for an integer.
- function bitwise_lsh (v, s=1, bm=1, bv=1)

  Base-two bitwise left-shift operation for an integer.
- function bitwise_rsh (v, s=1)

  Base-two bitwise right-shift operation for an integer.

6.5.1 Detailed Description

Bitwise binary (base-two) operations.

See Wikipedia binary `numbers` and `operations` for more information.

See validation results.

6.5.2 Function Documentation

6.5.2.1 function bitwise_and (v1 , v2 , bv = `1`)

Base-two bitwise AND operation for integers.

Parameters

v1	<integer> An integer value.
v2	<integer> An integer value.
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise AND of `v1` and `v2`. Returns **undef** when `v1` or `v2` is not an integer.

6.5.2.2 function bitwise_i2s (v , w = `1`)

Encode an integer value as a base-two string of bits.

Parameters

v	<integer> An integer value.
w	<integer> The minimum bit width.

Returns

<string> of bits base-two encoding of the integer value. Returns **undef** when `v` or `w` is not an integer.

6.5.2.3 function bitwise_i2v (v , w = `1`, bv = `1`)

Encode an integer value as a base-two vector of bits.

Parameters

v	<integer> An integer value.
w	<integer> The minimum bit width.
bv	(an internal recursion loop variable).

Returns

<vector> of bits base-two encoding of the integer value. Returns **undef** when `v` or `w` is not an integer.

6.5.2.4 function bitwise_is_equal (v , b , t = `1`)

Test if a base-two bit position of an integer value equals a test bit.

Parameters

v	<integer> An integer value.
b	<integer> A base-two bit position.
t	<bit> The bit test value [0|1].

Returns

<boolean> **true** when the base-two bit position of the integer value equals `t`, otherwise returns **false**.

6.5.2.5 function bitwise_lsh (v , s = `1`, bm = `1`, bv = `1`)

Base-two bitwise left-shift operation for an integer.

Parameters

v	<integer> An integer value.
s	<integer> The number of bits to shift.
bm	(an internal recursion loop variable).
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise left-shift of `v` by `s` bits. Returns **undef** when `v` or `s` is not an integer.

6.5.2.6 function bitwise_not (v , w = `1`, bv = `1`)

Base-two bitwise NOT operation for an integer.

Parameters

v	<integer> An integer value.
w	<integer> The minimum bit width.
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise NOT of `v`. Returns **undef** when `v` is not an integer.

6.5.2.7 function bitwise_or (v1 , v2 , bv = `1`)

Base-two bitwise OR operation for integers.

Parameters

v1	<integer> An integer value.
v2	<integer> An integer value.
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise OR of `v1` and `v2`. Returns **undef** when `v1` or `v2` is not an integer.

6.5.2.8 function bitwise_rsh (v , s = `1`)

Base-two bitwise right-shift operation for an integer.

Parameters

v	<integer> An integer value.
s	<integer> The number of bits to shift.

Returns

<integer> result of the base-two bitwise right-shift of `v` by `s` bits. Returns **undef** when `v` or `s` is not an integer.

6.5.2.9 function bitwise_s2i (v)

Decode a base-two string of bits to an integer value.

Parameters

v	<string> A value encoded as a base-two string of bits.

Returns

<integer> value encoding of the base-two string of bits. Returns **undef** when `v` is not a string of bit values.

6.5.2.10 function bitwise_v2i (v)

Decode a base-two vector of bits to an integer value.

Parameters

v	<vector> A value encoded as a base-two vector of bits.

Returns

<integer> value encoding of the base-two vector of bits. Returns **undef** when `v` is not a vector of bit values.

6.5.2.11 function bitwise_xor (v1 , v2 , bv = `1`)

Base-two bitwise XOR operation for integers.

Parameters

v1	<integer> An integer value.
v2	<integer> An integer value.
bv	(an internal recursion loop variable).

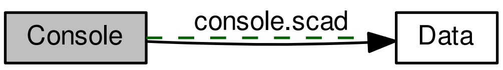
Returns

<integer> result of the base-two bitwise XOR of `v1` and `v2`. Returns **undef** when `v1` or `v2` is not an integer.

6.6 Console

Console message logging.

Collaboration diagram for Console:



Files

- file console.scad

    Message logging functions.

Functions

- module log_echo (m)

    Output message to console.
- module log_debug (m)

    Output diagnostic message to console.
- module log_info (m)

    Output information message to console.
- module log_warn (m)

    Output warning message to console.
- module log_error (m)

    Output error message to console.

6.6.1 Detailed Description

Console message logging.

Example

```
use <console.scad>;

$log_debug = true;
message = "console log message";

// general
log_echo( message );

// debugging
log_debug( message );
log_debug( message, $log_debug = false );

// information
log_info( message );

// warning
```

```
    log_warn( message );

    // error
    log_error( message );
```

Result

```
ECHO: "console log message"
ECHO: "[ DEBUG ] root(); console log message"
ECHO: "[ INFO ] root(); console log message"
ECHO:
ECHO: "root()"
ECHO: "###################################"
ECHO: "# [ WARNING ] console log message #"
ECHO: "###################################"
ECHO:
ECHO: "root()"
ECHO: "###################################"
ECHO: "###################################"
ECHO: "##                               ##"
ECHO: "## [ ERROR ] console log message ##"
ECHO: "##                               ##"
ECHO: "###################################"
ECHO: "###################################"
```

6.6.2 Function Documentation

6.6.2.1 module log_debug (m)

Output diagnostic message to console.

Parameters

m	<string> An output message.

When `$log_debug` == `true`, message is written to the console. When `false`, output is not generated.

Definition at line 77 of file console.scad.

6.6.2.2 module log_echo (m)

Output message to console.

Parameters

m	<string> An output message.

Definition at line 61 of file console.scad.

6.6.2.3 module log_error (m)

Output error message to console.

Parameters

m	<string> An output message.

Output an error message to the console. Ideally, rendering should halt and the script should exit. However, no suitable abort function exists. To alert of the critical error, the error message is also rendered graphically.

Definition at line 138 of file console.scad.

6.6.2.4 module log_info (m)

Output information message to console.

Parameters

m	<string> An output message.

Definition at line 94 of file console.scad.

6.6.2.5 module log_warn (m)

Output warning message to console.

Parameters

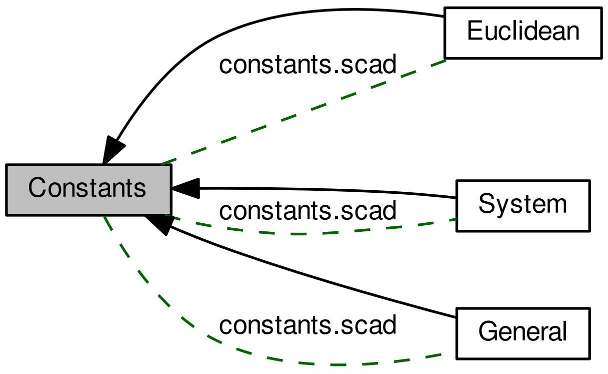
m	<string> An output message.

Definition at line 109 of file console.scad.

6.7 Constants

General design constants.

Collaboration diagram for Constants:



Modules

- Euclidean

  Euclidean 2D/3D space mapping.
- General

  General design constants.
- System

  System/Program limits.

Files

- file constants.scad

  Mechanical design constants.

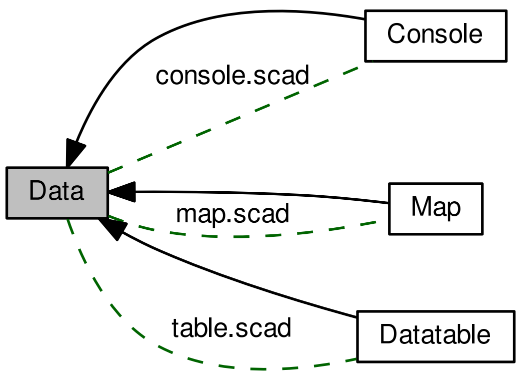
6.7.1 Detailed Description

General design constants.

6.8 Data

Data values and reference.

Collaboration diagram for Data:



Modules

- Console

  Console message logging.
- Datatable

  Data table encoding and lookup.
- Map

  Mapped data access via key-value pairs.

Files

- file console.scad

  Message logging functions.
- file map.scad

  Mapped key-value pair data access.
- file table.scad

  Data table encoding and lookup.

6.8.1 Detailed Description

Data values and reference.

6.9 Datatable

Data table encoding and lookup.

Collaboration diagram for Datatable:

Data
table.scad
Datatable

Files

- file table.scad

  Data table encoding and lookup.

Functions

- function table_get_row_idx (rows, row_id)

  Get the index for a table row identifier.
- function table_get_row (rows, row_id)

  Get the row for a table row identifier.
- function table_get_col_idx (cols, col_id)

  Get the index for a table column identifier.
- function table_get_col (cols, col_id)

  Get the column for a table column identifier.
- function table_get (rows, cols, row_id, col_id)

  Get the value for a table row and column identifier.
- function table_get_row_cols (rows, cols, col_id)

  Form a vector from the specified column of each table row.
- function table_get_row_ids (rows)

  Form a vector of each table row identifier.
- function table_exists (rows, cols, row_id, col_id)

  Test the existence of a table row and column identifier.
- function table_size (rows, cols)

  Get the size of a table.
- module table_check (rows, cols, verbose=false)

  Perform some basic validation/checks on a table.
- module table_dump (rows, cols, rows_sel, cols_sel, number=true)

  Dump a table to the console.
- function table_copy (rows, cols, rows_sel, cols_sel)

  Create a copy of select rows and columns of a table.
- function table_sum (rows, cols, rows_sel, cols_sel)

  Sum select rows and columns of a table.

6.9.1 Detailed Description

Data table encoding and lookup.

Example

```
use     <table.scad>;

base_unit_length = "mm";

table_cols =
[ // id,  description
  ["id",  "row identifier"],
  ["ht",  "head type [r|h|s]"],
  ["td",  "thread diameter"],
  ["tl",  "thread length"],
  ["hd",  "head diameter"],
  ["hl",  "head length"],
  ["nd",  "hex nut flat-to-flat width"],
  ["nl",  "hex nut length"]
];

table_rows =
[ //     id,  ht,     td,     tl,   hd,    hl,    nd,                  nl
  ["m3r08r", "r",  3.000,   8.00, 5.50, 3.000,  5.50, convert_length(1.00, "in")],
  ["m3r14r", "r",  3.000,  14.00, 5.50, 3.000,  5.50, convert_length(1.25, "in")],
  ["m3r16r", "r",  3.000,  16.00, 5.50, 3.000,  5.50, convert_length(1.50, "in")],
  ["m3r20r", "r",  3.000,  20.00, 5.50, 3.000,  5.50, convert_length(1.75, "in")]
];

table_check( table_rows, table_cols, true );
table_dump( table_rows, table_cols );

m3r16r_tl = table_get( table_rows, table_cols, "m3r16r", "tl" );

if ( table_exists( cols=table_cols, col_id="nl" ) )
  echo ( "metric 'nl' available" );

table_ids = table_get_row_ids( table_rows );
table_cols_tl = table_get_row_cols( table_rows, table_cols, "tl" );

echo ( table_ids=table_ids );
echo ( table_cols_tl=table_cols_tl );

tnew = table_copy( table_rows, table_cols, cols_sel=["tl", "nl"] );
tsum = table_sum( table_rows, table_cols, cols_sel=["tl", "nl"] );

echo ( m3r16r_tl=m3r16r_tl );
echo ( tnew=tnew );
echo ( tsum=tsum );
```

Result

```
ECHO: "[ INFO ] table_check(); begin table check"
ECHO: "[ INFO ] table_check(); row identifier found at column zero."
ECHO: "[ INFO ] table_check(); checking row column counts."
ECHO: "[ INFO ] table_check(); checking for repeat column identifiers."
ECHO: "[ INFO ] table_check(); checking for repeat row identifiers."
ECHO: "[ INFO ] table_check(); table size: 4 rows by 8 columns."
ECHO: "[ INFO ] table_check(); end table check"
ECHO: ""
ECHO: "row: 0"
ECHO: "[m3r08r] [id] (row identifier)             = [m3r08r]"
ECHO: "[m3r08r] [ht] (head type [r|h|s])          = [r]"
ECHO: "[m3r08r] [td] (thread diameter)            = [3]"
ECHO: "[m3r08r] [tl] (thread length)              = [8]"
ECHO: "[m3r08r] [hd] (head diameter)              = [5.5]"
ECHO: "[m3r08r] [hl] (head length)                = [3]"
ECHO: "[m3r08r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r08r] [nl] (hex nut length)             = [25.4]"
ECHO: ""
ECHO: "row: 1"
ECHO: "[m3r14r] [id] (row identifier)             = [m3r14r]"
ECHO: "[m3r14r] [ht] (head type [r|h|s])          = [r]"
ECHO: "[m3r14r] [td] (thread diameter)            = [3]"
ECHO: "[m3r14r] [tl] (thread length)              = [14]"
```

```
ECHO: "[m3r14r] [hd] (head diameter)               = [5.5]"
ECHO: "[m3r14r] [hl] (head length)                 = [3]"
ECHO: "[m3r14r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r14r] [nl] (hex nut length)              = [31.75]"
ECHO: ""
ECHO: "row: 2"
ECHO: "[m3r16r] [id] (row identifier)              = [m3r16r]"
ECHO: "[m3r16r] [ht] (head type [r|h|s])           = [r]"
ECHO: "[m3r16r] [td] (thread diameter)             = [3]"
ECHO: "[m3r16r] [tl] (thread length)               = [16]"
ECHO: "[m3r16r] [hd] (head diameter)               = [5.5]"
ECHO: "[m3r16r] [hl] (head length)                 = [3]"
ECHO: "[m3r16r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r16r] [nl] (hex nut length)              = [38.1]"
ECHO: ""
ECHO: "row: 3"
ECHO: "[m3r20r] [id] (row identifier)              = [m3r20r]"
ECHO: "[m3r20r] [ht] (head type [r|h|s])           = [r]"
ECHO: "[m3r20r] [td] (thread diameter)             = [3]"
ECHO: "[m3r20r] [tl] (thread length)               = [20]"
ECHO: "[m3r20r] [hd] (head diameter)               = [5.5]"
ECHO: "[m3r20r] [hl] (head length)                 = [3]"
ECHO: "[m3r20r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r20r] [nl] (hex nut length)              = [44.45]"
ECHO: ""
ECHO: "table size: 4 rows by 8 columns."
ECHO: "metric 'nl' available"
ECHO: table_ids = ["m3r08r", "m3r14r", "m3r16r", "m3r20r"]
ECHO: table_cols_tl = [8, 14, 16, 20]
ECHO: m3r16r_tl = 16
ECHO: tnew = [[8, 25.4], [14, 31.75], [16, 38.1], [20, 44.45]]
ECHO: tsum = [58, 139.7]
```

6.9.2 Function Documentation

6.9.2.1 module table_check (rows , cols , verbose = `false`)

Perform some basic validation/checks on a table.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
verbose	<boolean> Be verbose during check.

Check that: (1) the first table column identifier is 'id'. (2) Make sure that each row has the same number of columns as defined in the columns vector. (3) Make sure that there are no repeating column identifiers. (4) Make sure that there are no repeating row identifiers.

Definition at line 254 of file table.scad.

6.9.2.2 function table_copy (rows , cols , rows_sel , cols_sel)

Create a copy of select rows and columns of a table.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
rows_sel	<1d-vector> A n-tuple vector of row identifier to select.
cols_sel	<1d-vector> A n-tuple vector of column identifier to select.

Returns

<2d-vector> The selected rows and columns of the table.

6.9.2.3 module table_dump (rows , cols , rows_sel , cols_sel , number = `true`)

Dump a table to the console.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
rows_sel	<1d-vector> A n-tuple vector of row identifier to select.
cols_sel	<1d-vector> A n-tuple vector of column identifier to select.
number	<boolean> Number the table rows.

Output each table row to the console. To output only select rows and columns, assign the desired identifiers to `rows↵_sel` and `cols_sel`. For example to output only the column identifiers 'c1' and 'c2', assign `cols_sel = ["c1", "c2"]`.

Definition at line 337 of file table.scad.

6.9.2.4 function table_exists (rows , cols , row_id , col_id)

Test the existence of a table row and column identifier.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
row_id	<string> The row identifier string to locate.
col_id	<string> The column identifier string to locate.

Returns

true if the row and column identifier exists, otherwise returns **false**.

6.9.2.5 function table_get (rows , cols , row_id , col_id)

Get the value for a table row and column identifier.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
row_id	<string> The row identifier string to locate.
col_id	<string> The column identifier string to locate.

Returns

<decimal\|string> The value at the located `row_id` and `col_id`. If it does not exists, returns **undef**.

6.9.2.6 function table_get_col (cols , col_id)

Get the column for a table column identifier.

Parameters

cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
col_id	<string> The column identifier string to locate.

Returns

<vector> The column where the row identifier is located. If the identifier does not exists, returns **undef**.

6.9.2.7 function table_get_col_idx (cols , col_id)

Get the index for a table column identifier.

Parameters

cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
col_id	<string> The column identifier string to locate.

Returns

<decimal> The column index where the identifier is located. If the identifier does not exists, returns **empty_v**.

6.9.2.8 function table_get_row (rows , row_id)

Get the row for a table row identifier.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
row_id	<string> The row identifier string to locate.

Returns

<vector> The row where the row identifier is located. If the identifier does not exists, returns **undef**.

6.9.2.9 function table_get_row_cols (rows , cols , col_id)

Form a vector from the specified column of each table row.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
col_id	<string> The column identifier string.

Returns

<vector> The vector formed by selecting the `col_id` for each row in the table. If column does not exists, returns **undef**.

6.9.2.10 function table_get_row_ids (rows)

Form a vector of each table row identifier.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.

Returns

<vector> The vector of table row identifiers. If column `"id"` does not exists, returns **undef**.

Note

This functions assumes the first element of each table row to be the row identifier, as enforced by the table_check(). As an alternative, the function table_get_row_cols(), of the form table_get_row_cols(rows, cols, "id"), may be used without this assumption.

6.9.2.11 function table_get_row_idx (rows , row_id)

Get the index for a table row identifier.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
row_id	<string> The row identifier string to locate.

Returns

<decimal> The row index where the identifier is located. If the identifier does not exists, returns **empty_v**.

6.9.2.12 function table_size (rows , cols)

Get the size of a table.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.

Returns

<decimal> The table size.

The size is reported as: (1) The number of rows when only the `rows` parameter is specified. (2) The number of columns when only the `cols` parameter is specified. (3) The (rows $*$ columns) when both parameters are specified.

6.9.2.13 function table_sum (rows , cols , rows_sel , cols_sel)

Sum select rows and columns of a table.

Parameters

rows	<2d-vector> A two dimensional vector (r-tuple x c-tuple) containing the table rows.
cols	<2d-vector> A two dimensional vector (c-tuple x 1-tuple) containing the table columns.
rows_sel	<1d-vector> A vector n-tuple of row identifier to select.
cols_sel	<1d-vector> A vector n-tuple of column identifier to select.

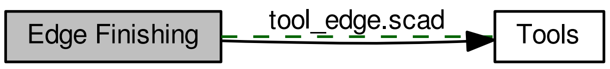
Returns

<1d-vector> The sum of the selected rows and columns of the table.

6.10 Edge Finishing

Shape edge finishing tools.

Collaboration diagram for Edge Finishing:



Files

- file tool_edge.scad

  Shape edge finishing tools.

Functions

- module edge_profile_r (r, p=0, f=1, a=90,)

  A 2D edge-finish profile specified by intersection radius.
- module edge_add_r (r, l=1, p=0, f=1, m=3, ba=45, a1=0, a2=90, center=false)

  A 3D edge-finish additive shape specified by intersection radius.

6.10.1 Detailed Description

Shape edge finishing tools.

6.10.2 Function Documentation

6.10.2.1 module edge_add_r (r , l = 1, p = 0, f = 1, m = 3, ba = 45, a1 = 0, a2 = 90, center = false)

A 3D edge-finish additive shape specified by intersection radius.

Parameters

r	<decimal> The radius length.
l	<decimal> The edge length.
p	<integer> The profile identifier.
f	<decimal> The mid-point offset factor.
m	<integer> The end finish mode: (B0: bottom, B1: top).
ba	<decimal> The end bevel angle.

a1	<decimal> The edge intersection start angle.
a2	<decimal> The edge intersection end angle.
center	<boolean> Center length about origin.

Example

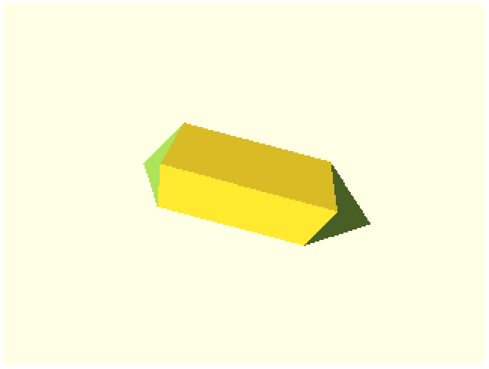


Figure 45: edge_add_r

```
rotate([90,-90,90]) edge_add_r( r=5, l=20, f=5/8, center=true );
```

m	B1	B0	Description
0	0	0	cut bottom (-z) and top (+z)
1	0	1	bevel bottom and cut top
2	1	0	cut bottom and bevel top
3	1	1	bevel bottom and top

See edge_profile_r() for description of available profiles.

Definition at line 204 of file tool_edge.scad.

6.10.2.2 module edge_profile_r (r , p = 0, f = 1, a = 90)

A 2D edge-finish profile specified by intersection radius.

Parameters

r	<decimal> The radius length.
p	<integer> The profile identifier.
f	<decimal> The mid-point offset factor.
a	<decimal> The sweep angle.

Example

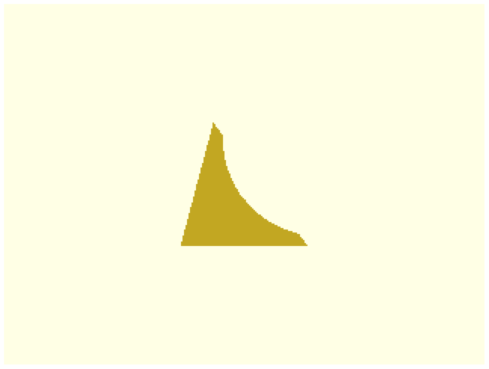


Figure 46: edge_profile_r

```
edge_profile_r( r=5, p=1, f=1+10/100, a=75 );
```

Profiles:

p	Description
0	Two segment bevel with mid inflection
1	A cove with cut-out offset
2	A quarter round with offset

Note

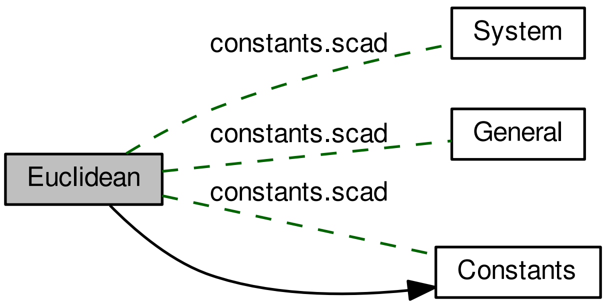
A offset factor greater than 1 moves the mid-point away from the profile edge-vertex. A factor less than 1 move it inwards towards the edge-vertex.

Definition at line 110 of file tool_edge.scad.

6.11 Euclidean

Euclidean 2D/3D space mapping.

Collaboration diagram for Euclidean:



Files

- file constants.scad

  Mechanical design constants.

Variables

- x_axis_vi = 0

  The vector index for the x-coordinate of a vector.
- y_axis_vi = 1

  The vector index for the y-coordinate of a vector.
- z_axis_vi = 2

  The vector index for the z-coordinate of a vector.
- origin2d = [0, 0]

  The origin coordinates in 2-dimensional Euclidean space.
- x_axis2d_uv = [1, 0]

  The unit vector of the positive x-axis in 2-dimensional Euclidean space.
- y_axis2d_uv = [0, 1]

  The unit vector of the positive y-axis in 2-dimensional Euclidean space.
- origin3d = [0, 0, 0]

  The origin coordinates in 3-dimensional Euclidean space.
- x_axis3d_uv = [1, 0, 0]

  The unit vector of the positive x-axis in 3-dimensional Euclidean space.
- y_axis3d_uv = [0, 1, 0]

  The unit vector of the positive y-axis in 3-dimensional Euclidean space.
- z_axis3d_uv = [0, 0, 1]

  The unit vector of the positive z-axis in 3-dimensional Euclidean space.

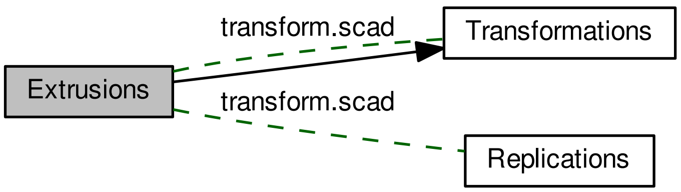
6.11.1 Detailed Description

Euclidean 2D/3D space mapping.

6.12 Extrusions

Shape Extrusions.

Collaboration diagram for Extrusions:



Files

- file transform.scad

  Shape transformation functions.

Functions

- module st_rotate_extrude (r, pa=0, ra=360, profile=false)

  Revolve the 2D shape about the z-axis.
- module st_rotate_extrude_elongate (r, l, pa=0, ra=360, m=255, profile=false)

  Revolve the 2D shape about the z-axis with linear elongation.
- module st_linear_extrude_scale (h, center=false)

  Linearly extrude 2D shape with extrusion upper and lower scaling.

6.12.1 Detailed Description

Shape Extrusions.

6.12.2 Function Documentation

6.12.2.1 module st_linear_extrude_scale (h , center = `false`)

Linearly extrude 2D shape with extrusion upper and lower scaling.

Parameters

h	<vector|decimal> A vector of decimals or a single decimal to specify simple extrusion height.

center	<boolean> Center extrusion about origin.

When `h` is a decimal, the shape is extruded linearly as normal. To scale the upper and lower slices of the extrusion, `h` must be assigned a vector with a minimum of three decimal values as described in the following table.

sym	h[n]	default	description
h	0		total extrusion height
n1	1		(+z) number of scaled extrusion slices
h1	2		(+z) extrusion scale percentage
x1	3	-h1	(+z) x-dimension scale percentage
y1	4	x1	(+z) y-dimension scale percentage
n2	5	n1	(-z) number of scaled extrusion slices
h2	6	h1	(-z) extrusion scale percentage
x2	7	x1	(-z) x-dimension scale percentage
y2	8	y1	(-z) y-dimension scale percentage

Example

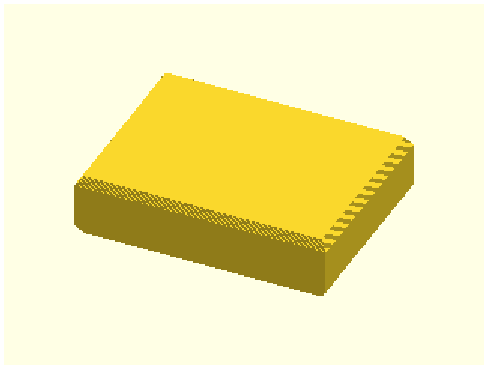


Figure 47: st_linear_extrude_scale

```
st_linear_extrude_scale( [5,10,15,-5], center=true ) square( [20,15], center=true );
```

Note

When symmetrical scaling is desired, shape must be centered about origin.

Definition at line 230 of file transform.scad.

6.12.2.2 module st_rotate_extrude (r , pa = 0, ra = 360, profile = false)

Revolve the 2D shape about the z-axis.

Parameters

r	<decimal> The rotation radius.
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
profile	<boolean> Show profile only (do not extrude).

Example

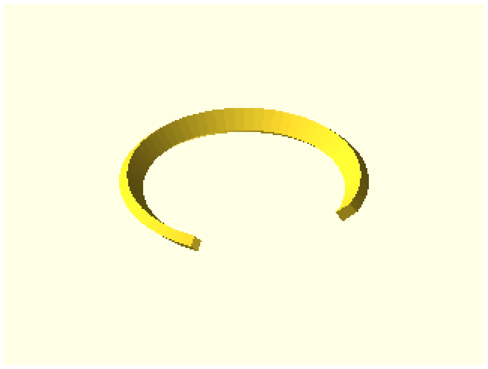


Figure 48: st_rotate_extrude

```
st_rotate_extrude( r=50, pa=45, ra=270 ) square( [10,5], center=true );
```

Definition at line 103 of file transform.scad.

6.12.2.3 module st_rotate_extrude_elongate (r , l , pa = 0, ra = 360, m = 255, profile = false)

Revolve the 2D shape about the z-axis with linear elongation.

Parameters

r	<decimal> The rotation radius.
l	<vector|decimal> The elongation length. A vector [x, y] of decimals or a single decimal for (x=y)
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render. Bit values **1** enables the corresponding section and bit values **0** are disabled. Sections are assigned to the bit position in counter-clockwise order.
profile	<boolean> Show profile only (do not extrude).

Example

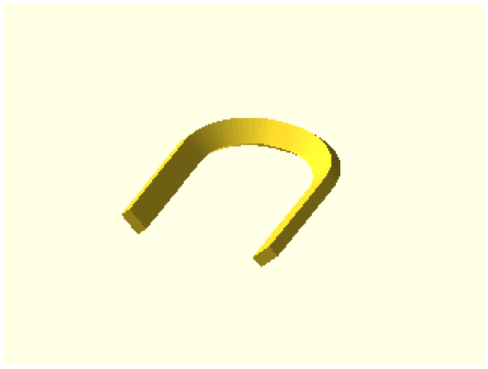


Figure 49: st_rotate_extrude_elongate

```
st_rotate_extrude_elongate( r=25, l=[5, 50], pa=45, m=31 ) square( [10,5], center=true );
```

Note

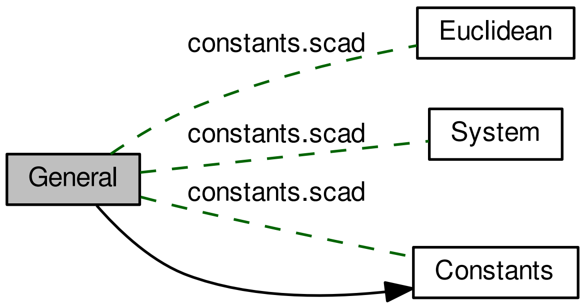
When elongating `(l > 0)`, `ra` is ignored. However, `m` may be used to control which complete revolution section to render.

Definition at line 139 of file transform.scad.

6.13 General

General design constants.

Collaboration diagram for General:



Files

- file constants.scad

  Mechanical design constants.

Variables

- eps = 0.01

  <decimal> Epsilon, small distance to deal with overlaping shapes
- pi = 3.1415926535897932384626433832795

  <decimal> The ratio of a circle's circumference to its diameter
- tau = 2∗pi

  <decimal> The ratio of a circle's circumference to its radius

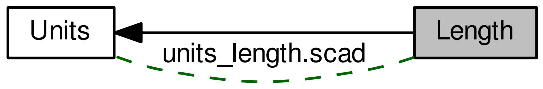
6.13.1 Detailed Description

General design constants.

6.14 Length

Length units and conversions.

Collaboration diagram for Length:



Files

- file units_length.scad

  Length units and conversions.

Functions

- function unit_length_name (units=base_unit_length, d=1, w=false)

  Return the name of the given `unit` identifier with dimension.
- function convert_length (value, from=base_unit_length, to=base_unit_length, d=1)

  Convert the `value` from `from` units to `to` units with dimensions.

Variables

- base_unit_length = "mm"

  <string> Base unit for length measurements.

6.14.1 Detailed Description

Length units and conversions.

These functions allow for lengths to be specified with units. Lengths specified with units are independent of (base_unit_length). There are also unit conversion functions for converting from one unit to another.

The table below enumerates the supported unit identifiers and their descriptions.

units id	description
pm	picometer
nm	nanometer
um	micrometer
mm	millimeter
cm	centimeter

dm	decimeter
m	meter
km	kilometer
thou, mil	thousandth of an inch
in	inch
ft	feet
yd	yard
mi	mile

Example

```
include <units_length.scad>;

base_unit_length = "mm";

// get base unit name
un = unit_length_name();

// absolute length measurements in base unit.
c1 = convert_length(1/8, "in");
c2 = convert_length(3.175, "mm");
c3 = convert_length(25, "mil");
c4 = convert_length(1, "ft", d=3);

// convert between units.
c5 = convert_length(10, from="mil", to="in");
c6 = convert_length(10, from="ft", to="mm");
```

Result (base_unit_length = **mm**):

```
ECHO: un = "millimeter"
ECHO: c1 = 3.175
ECHO: c2 = 3.175
ECHO: c3 = 0.635
ECHO: c4 = 2.83168e+07
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Result (base_unit_length = **cm**):

```
ECHO: un = "centimeter"
ECHO: c1 = 0.3175
ECHO: c2 = 0.3175
ECHO: c3 = 0.0635
ECHO: c4 = 28316.8
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Result (base_unit_length = **mil**):

```
ECHO: un = "thousandth"
ECHO: c1 = 125
ECHO: c2 = 125
ECHO: c3 = 25
ECHO: c4 = 1.728e+12
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Result (base_unit_length = **in**):

```
ECHO: un = "inch"
ECHO: c1 = 0.125
ECHO: c2 = 0.125
ECHO: c3 = 0.025
ECHO: c4 = 1728
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Example (equivalent lengths)

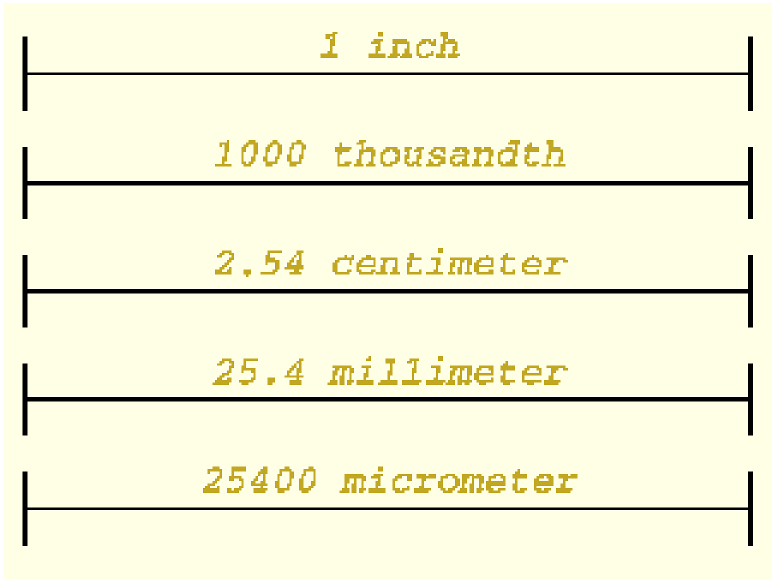


Figure 50: Unit Lengths

6.14.2 Function Documentation

6.14.2.1 function convert_length (value , from = **base_unit_length**, to = **base_unit_length**, d = `1`)

Convert the `value` from `from` units to `to` units with dimensions.

Parameters

value	<decimal> A value to convert.
from	<string> The units of the value to be converted.
to	<string> A units to which the value should be converted.
d	<decimal> The dimension set to one of [1|2|3].

Returns

<decimal> The conversion result. Returns **'undef'** for identifiers or dimensions that are not defined.

6.14.2.2 function unit_length_name (units = **base_unit_length**, d = `1`, w = `false`)

Return the name of the given `unit` identifier with dimension.

Parameters

w	<boolean> **true:** use word format, **false:** use symbol format.
units	<string> A length unit identifier.
d	<decimal> A dimension set to one of [1|2|3].

Returns

<string> The units name for the given length unit identifier with is specified dimension. Returns **'undef'** for identifiers or dimensions that are not defined.

6.15 Map

Mapped data access via key-value pairs.

Collaboration diagram for Map:

Map
map.scad
Data

Files

- file map.scad

  Mapped key-value pair data access.

Functions

- function map_get_idx (map, key)

  Return the index for the storage location of a map key-value pair.
- function map_exists (map, key)

  Test if a key exists in a map.
- function map_get (map, key)

  Get the value associated with a map key.
- function map_get_keys (map)

  Get a vector of the map entry identifier keys.
- function map_get_values (map)

  Get a vector of the map entry values.
- function map_size (map)

  Get the number of key-value pairs stored in a map.
- module map_check (map, verbose=false)

  Perform some basic validation/checks on a map.
- module map_dump (map, sort=true, number=true, p=3)

  Dump each map key-value pair to the console.

6.15.1 Detailed Description

Mapped data access via key-value pairs.

Example

```
use <map.scad>;

map =
[
  ["part1",        ["screw10", [10, 11, 13]]],
  ["part2",        ["screw12", [20, 21, 30]]],
  ["part3",        ["screw10", [10, 10, -12]]],
  ["config",       ["top", "front", "rear"]],
  ["version",      [21, 5, 0]],
  ["runid",        10]
];

map_check(map, true);

echo( str("is part0 = ", map_exists(map, "part0")) );
echo( str("is part1 = ", map_exists(map, "part1")) );

p1 = map_get(map, "part1");
echo( c=second(p1) );

keys=map_get_keys(map);
parts = delete(keys, mv=["config", "version", "runid"]);

for ( p = parts )
  echo
  (
    n=p,
    p=first(map_get(map, p)),
    l=second(map_get(map, p))
  );

map_dump(map);
```

Result

```
ECHO: "[ INFO ] map_check(); begin map check"
ECHO: "[ INFO ] map_check(); checking map format and keys."
ECHO: "[ INFO ] map_check(); map size: 6 entries."
ECHO: "[ INFO ] map_check(); end map check"
ECHO: "is part0 = false"
ECHO: "is part1 = true"
ECHO: c = [10, 11, 13]
ECHO: n = "part1", p = "screw10", l = [10, 11, 13]
ECHO: n = "part2", p = "screw12", l = [20, 21, 30]
ECHO: n = "part3", p = "screw10", l = [10, 10, -12]
ECHO: "003:   'config' = '["top", "front", "rear"]'"
ECHO: "000:    'part1' = '["screw10", [10, 11, 13]]'"
ECHO: "001:    'part2' = '["screw12", [20, 21, 30]]'"
ECHO: "002:    'part3' = '["screw10", [10, 10, -12]]'"
ECHO: "005:    'runid' = '10'"
ECHO: "004:  'version' = '[21, 5, 0]'"
ECHO: "map size: 6 entries."
```

6.15.2 Function Documentation

6.15.2.1 module map_check (map , verbose = `false`)

Perform some basic validation/checks on a map.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.
verbose	<boolean> Be verbose during check.

Check that: (1) each entry has key-value 2-tuple, (2) each key is a string, and (3) key identifiers are unique.

Definition at line 169 of file map.scad.

6.15.2.2 module map_dump (map , sort = `true`, number = `true`, p = `3`)

Dump each map key-value pair to the console.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.
sort	<boolean> Sort the output by key.
number	<boolean> Output index number.
p	<integer> Number of places for zero-padded numbering.

Definition at line 245 of file map.scad.

6.15.2.3 function map_exists (map , key)

Test if a key exists in a map.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.
key	<string> A map entry identifier.

Returns

<boolean> **true** when the key exists and **false** otherwise.

6.15.2.4 function map_get (map , key)

Get the value associated with a map key.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.
key	<string> A map entry identifier.

Returns

The map value associated with `key`. Returns **undef** if `key` does not exists.

6.15.2.5 function map_get_idx (map , key)

Return the index for the storage location of a map key-value pair.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.
key	<string> A map entry identifier.

Returns

<integer> The index of the value associated `key` in the map. Returns **undef** if `key` is not a string or does not exists.

6.15.2.6 function map_get_keys (map)

Get a vector of the map entry identifier keys.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.

Returns

<vector> A vector of keys that exist in the associative map.

Note

Uses function eselect to select the first column of the vector defining the map.

6.15.2.7 function map_get_values (map)

Get a vector of the map entry values.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.

Returns

<vector> A vector of values stored in the associative map.

Note

Uses function eselect to select the last column of the vector defining the map.

6.15.2.8 function map_size (map)

Get the number of key-value pairs stored in a map.

Parameters

map	<2d-vector> A two dimensional vector (2-tuple x n-tuple) containing an associative map with n elements.

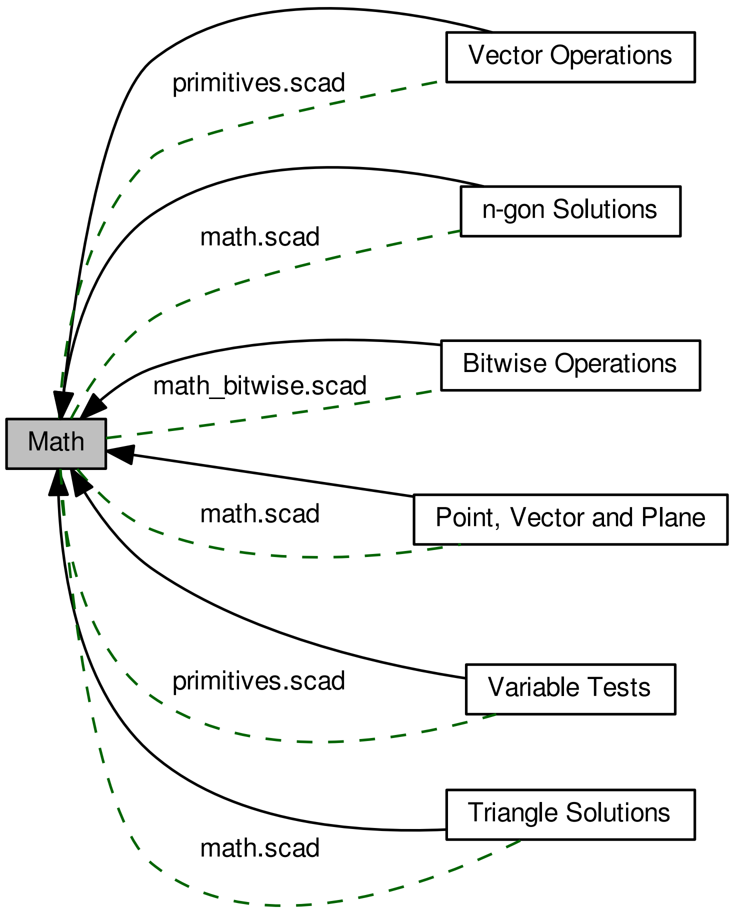
Returns

<integer> The number of key-value pairs stored in the map.

6.16 Math

Mathematical functions.

Collaboration diagram for Math:



Modules

- Bitwise Operations

  Bitwise binary (base-two) operations.
- Point, Vector and Plane

  Point, vector, and plane computations.
- Triangle Solutions

  Triangle computations.
- Variable Tests

  Variable property test primitives.

- Vector Operations

  Vector operation primitives.
- n-gon Solutions

  Regular n-sided polygon computations.

Files

- file primitives.scad

  Mathematical primitive functions.
- file math.scad

  Mathematical functions.
- file math_bitwise.scad

  Mathematical bitwise binary (base-two) functions.

6.16.1 Detailed Description

Mathematical functions.

6.17 Parts

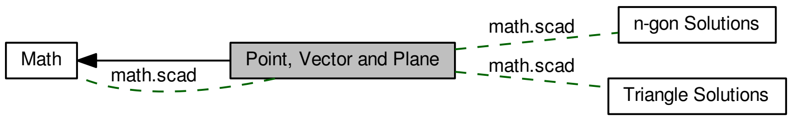
Parametric parts/components.

Parametric parts/components.

6.18 Point, Vector and Plane

Point, vector, and plane computations.

Collaboration diagram for Point, Vector and Plane:



Files

- file math.scad

  Mathematical functions.

Functions

- function distance_pp (p1, p2)

  Compute the distance between two points in a Euclidean 1, 2, or 3D-space.
- function dot_vv (v1t, v2t, v1i, v2i)

  Compute the dot product of two vectors.
- function cross_vv (v1t, v2t, v1i, v2i)

  Compute the cross product of two vectors in a Euclidean 3D-space (2D).
- function striple_vvv (v1t, v2t, v3t, v1i, v2i, v3i)

  Compute scalar triple product of two vectors in a Euclidean 3D-space.
- function angle_vv (v1t, v2t, v1i, v2i)

  Compute the angle between two vectors in a Euclidean 2 or 3D-space.
- function angle_vvn (v1t, v2t, nvt, v1i, v2i, nvi)

  Compute the angle between two vectors in a Euclidean 3D-space.
- function unit_v (vt, vi)

  Compute the normalized unit vector for a 1, 2, or 3 term vector.
- function are_coplanar_vvv (v1t, v2t, v3t, v1i, v2i, v3i)

  Test if three vectors are coplanar in Euclidean 3D-space.

6.18.1 Detailed Description

Point, vector, and plane computations.

See validation results.

6.18.2 Function Documentation

6.18.2.1 function angle_vv (v1t , v2t , v1i , v2i)

Compute the angle between two vectors in a Euclidean 2 or 3D-space.

Parameters

v1t	<vector> Vector 1 head. A 2 or 3-tuple of coordinates.
v2t	<vector> Vector 2 head. A 2 or 3-tuple of coordinates.
v1i	<vector> Vector 1 tail. A 2 or 3-tuple of coordinates.
v2i	<vector> Vector 2 tail. A 2 or 3-tuple of coordinates.

Returns

<decimal> The angle between the two vectors in degrees. Returns **'undef'** when vector coordinates do not have same number of terms or when the vectors do not intersect.

Each vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

Note

For 3D vectors, a normal vector is required to uniquely identify the perpendicular plane and axis of rotation for the two vectors. This function calculates the positive angle, and the plane and axis of rotation will be that which fits this assumed positive angle.

See also

angle_vvn().

6.18.2.2 function angle_vvn (v1t , v2t , nvt , v1i , v2i , nvi)

Compute the angle between two vectors in a Euclidean 3D-space.

Parameters

v1t	<vector> Vector 1 head. A 3-tuple of coordinates.
v2t	<vector> Vector 2 head. A 3-tuple of coordinates.
nvt	<vector> Normal vector head. A 3-tuple of coordinates.
v1i	<vector> Vector 1 tail. A 3-tuple of coordinates.
v2i	<vector> Vector 2 tail. A 3-tuple of coordinates.
nvi	<vector> Normal vector tail. A 3-tuple of coordinates.

Returns

<decimal> The angle between the two vectors in degrees. Returns **'undef'** when vector coordinates do not have same number of terms or when the vectors do not intersect.

Each vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

See also

angle_vv().

6.18.2.3 function are_coplanar_vvv (v1t , v2t , v3t , v1i , v2i , v3i)

Test if three vectors are coplanar in Euclidean 3D-space.

Parameters

v1t	<vector> Vector 1 head. A 3-tuple of coordinates.
v2t	<vector> Vector 2 head. A 3-tuple of coordinates.
v3t	<vector> Vector 3 head. A 3-tuple of coordinates.
v1i	<vector> Vector 1 tail. A 3-tuple of coordinates.
v2i	<vector> Vector 2 tail. A 3-tuple of coordinates.
v3i	<vector> Vector 3 tail. A 3-tuple of coordinates.

Returns

<boolean> **true** when all three vectors are coplanar, and **false** otherwise.

Each vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

See Wikipedia for more information.

Note

Coplanar vectors must all be within the same plane. However, this function can test if vectors are in a plane that is parallel to a coplanar plane by using non-zero vector tails.

6.18.2.4 function cross_vv (v1t , v2t , v1i , v2i)

Compute the cross product of two vectors in a Euclidean 3D-space (2D).

Parameters

v1t	<vector> Vector 1 head. A 2 or 3-tuple of coordinates.
v2t	<vector> Vector 2 head. A 2 or 3-tuple of coordinates.
v1i	<vector> Vector 1 tail. A 2 or 3-tuple of coordinates.
v2i	<vector> Vector 2 tail. A 2 or 3-tuple of coordinates.

Returns

<decimal> The cross product of the two vectors. Returns **'undef'** when vector coordinates do not have same number of terms, n.

Each vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

See Wikipedia cross and determinant for more information.

Note

Although the cross product of two vectors is defined only in 3D space, this function will return the 2x2 determinant for a 2D vector.

6.18.2.5 function distance_pp (p1 , p2)

Compute the distance between two points in a Euclidean 1, 2, or 3D-space.

Parameters

p1	<vector> A 1, 2, or 3-tuple of coordinates.
p2	<vector> A 1, 2, or 3-tuple of coordinates.

Returns

<decimal> The distance between the two points. Returns **'undef'** when x and y do not have same number of terms or for n-tuple where n>3.

When `p2` is not given, it is assumed to be at the origin.

6.18.2.6 function dot_vv (v1t , v2t , v1i , v2i)

Compute the dot product of two vectors.

Parameters

v1t	<vector> Vector 1 head. An n-tuple of coordinates.
v2t	<vector> Vector 2 head. An n-tuple of coordinates.
v1i	<vector> Vector 1 tail. An n-tuple of coordinates.
v2i	<vector> Vector 2 tail. An n-tuple of coordinates.

Returns

<decimal> The dot product of the two vectors. Returns **'undef'** when vector coordinates do not have same number of terms, n.

Each vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

See `Wikipedia` for more information.

6.18.2.7 function striple_vvv (v1t , v2t , v3t , v1i , v2i , v3i)

Compute scalar triple product of two vectors in a Euclidean 3D-space.

Parameters

v1t	<vector> Vector 1 head. A 2 or 3-tuple of coordinates.
v2t	<vector> Vector 2 head. A 2 or 3-tuple of coordinates.
v3t	<vector> Vector 3 head. A 2 or 3-tuple of coordinates.
v1i	<vector> Vector 1 tail. A 2 or 3-tuple of coordinates.
v2i	<vector> Vector 2 tail. A 2 or 3-tuple of coordinates.
v3i	<vector> Vector 3 tail. A 2 or 3-tuple of coordinates.

Returns

<decimal> The scalar triple product of the three vectors. Returns **'undef'** when vector coordinates do not have same number of terms, n.

Each vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

[v1, v2, v3] = v1 $*$ (v2 x v3)

See `Wikipedia` for more information.

Warning

For 2D vectors, this function produces a 2D *non-scalar* vector result. The cross produce function computes the 2x2 determinant of the 2D vectors `(v2 x v3)`, which is a scalar value, and this value is *multiplied* by `v1`, which results in a 2D vector.

6.18.2.8 function unit_v (vt , vi)

Compute the normalized unit vector for a 1, 2, or 3 term vector.

Parameters

vt	<vector> Vector head. A 1, 2, or 3-tuple of coordinates.
vi	<vector> Vector tail. A 1, 2, or 3-tuple of coordinates.

Returns

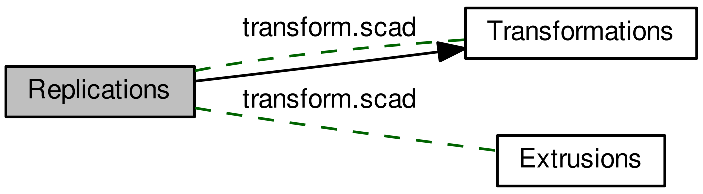
<vector> The vector normalized to its unit-vector. Returns **'undef'** when vector coordinates do not have same number of terms or for n-tuple where n>3.

The vector may be specified by both its head and tail coordinates. When specified by head coordinate only, the tail is assumed to be at origin.

6.19 Replications

Shape Replications and distribution.

Collaboration diagram for Replications:



Files

- file transform.scad

  Shape transformation functions.

Functions

- module st_radial_copy (n, r=1, angle=true, move=false)

  Distribute copies of a 2D or 3D shape equally about a z-axis radius.
- module st_cartesian_copy (grid, incr, copy=1, center=false)

  Distribute copies of 2D or 3D shapes about Cartesian grid.

6.19.1 Detailed Description

Shape Replications and distribution.

6.19.2 Function Documentation

6.19.2.1 module st_cartesian_copy (grid , incr , copy = 1, center = false)

Distribute copies of 2D or 3D shapes about Cartesian grid.

Parameters

grid	<vector|decimal> A vector [x, y, z] of decimals or a single decimal for (x=y=z).
incr	<vector|decimal> A vector [x, y, z] of decimals or a single decimal for (x=y=z).
copy	<decimal> Number of times to iterate over children.

center	<boolean> Center distribution about origin.

Example

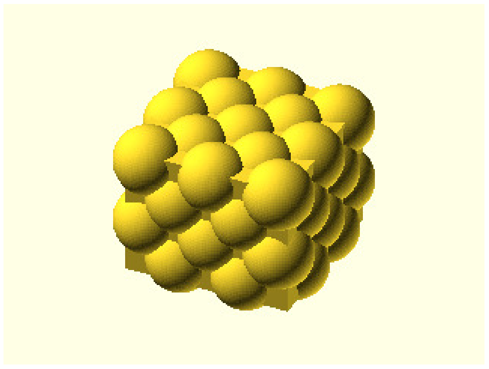


Figure 51: st_cartesian_copy

```
st_cartesian_copy( grid=[5,5,4], incr=10, copy=50, center=true ) {cube(10, center=true); sphere(10);}
```

Definition at line 352 of file transform.scad.

6.19.2.2 module st_radial_copy (n , r = 1, angle = true, move = false)

Distribute copies of a 2D or 3D shape equally about a z-axis radius.

Parameters

n	<decimal> The number of equally spaced radii.
r	<decimal> The shape move radius.
angle	<boolean> Rotate each copy about z-axis.
move	<boolean> Move each shape copy to radii coordinate.

Example

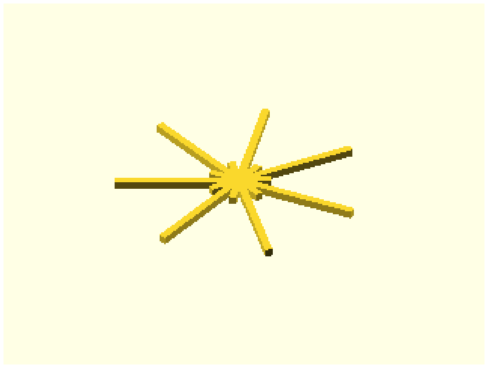


Figure 52: st_radial_copy

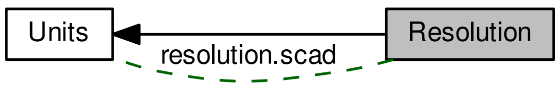
```
st_radial_copy( n=7, r=6, move=true ) square( [20,1], center=true );
```

Definition at line 322 of file transform.scad.

6.20 Resolution

Arch rendering resolution management.

Collaboration diagram for Resolution:



Files

- file resolution.scad

  Arc rendering resolution abstraction.

Functions

- function resolution_fn (radius)

  Return facets number for the given arc radius.
- function resolution_fs ()

  Return minimum facets size.
- function resolution_fa (radius)

  Return the minimum facets angle.
- function resolution_reduced ()

  Return the radius at which arc resolution will begin to degrade.
- module resolution_info (radius)

  Echo resolution information to the console for given radius.
- function resolution_facets (radius)

  Return facet count used to render a radius.
- function resolution_facetsv (radius)

  Return facet count used to render a radius as vector triple.

Variables

- $resolution_mode = "fast"

  <string> Global special variable that configures the arc resolution mode.
- $resolution_value = 0

  <decimal> Global special variable for modes that use custom resolutions.

6.20.1 Detailed Description

Arch rendering resolution management.

Functions, global variables, and configuration presets to provide a common mechanism for managing arc rendering resolution. Specifically, the number of fragments/facets with which arcs (circles, spheres, and cylinders, etc.) are rendered in OpenSCAD.

Example

```
include <resolution.scad>;

base_unit_length = "in";

// set resolution to 25 fpi
$resolution_mode  = "fpi";
$resolution_value = 25;

// use radius length of 1 inch
r = convert_length(1, "in");

$fs=resolution_fs();
$fa=resolution_fa( r );

resolution_info( r );

f = resolution_facets( r );
echo(str("for r = ", r, " ", unit_length_name(), ", facets = ", f));
```

Result (base_unit_length = **mm**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = millimeter"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 1.016"
ECHO: "resolution reduction at radius > 25.4 millimeter"
ECHO: "for radius = 25.4 millimeter facets limited to 158 by $fs=1.016 millimeter"
ECHO: "for r = 25.4 millimeter, facets = 158"
```

Result (base_unit_length = **cm**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = centimeter"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 0.1016"
ECHO: "resolution reduction at radius > 2.54 centimeter"
ECHO: "for radius = 2.54 centimeter facets limited to 158 by $fs=0.1016 centimeter"
ECHO: "for r = 2.54 centimeter, facets = 158"
```

Result (base_unit_length = **mil**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = thousandth"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 40"
ECHO: "resolution reduction at radius > 1000 thousandth"
ECHO: "for radius = 1000 thousandth facets limited to 158 by $fs=40 thousandth"
ECHO: "for r = 1000 thousandth, facets = 158"
```

Result (base_unit_length = **in**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = inch"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 0.04"
ECHO: "resolution reduction at radius > 1 inch"
ECHO: "for radius = 1 inch facets limited to 158 by $fs=0.04 inch"
ECHO: "for r = 1 inch, facets = 158"
```

6.20.2 Function Documentation

6.20.2.1 function resolution_fa (radius)

Return the minimum facets angle.

Parameters

radius	<decimal> An arc radius.

Returns

<decimal> Minimum facet angle to be assigned to $fa.

The return result of this function can be assigned to the OpenSCAD special variables `$fa` to render arcs.

6.20.2.2 function resolution_facets (radius)

Return facet count used to render a radius.

Parameters

radius	<decimal> An arc radius.

Returns

<decimal> The number of fragments/facets that will be used to render a radius given the current values for `$fn`, `$fa`, and `$fs`.

6.20.2.3 function resolution_facetsv (radius)

Return facet count used to render a radius as vector triple.

Parameters

radius	<decimal> An arc radius.

Returns

A vector triple: [**facets** <decimal>,**limiter** <string>,**value** <decimal>].

Where `facets` is the number of fragments/facets that will be used to render the `radius` given the current values for `$fn`, `$fa`, and `$fs`. `limiter` identifies the special variable that currently limits the facets, and `value` is the current value assigned to the limiter.

6.20.2.4 function resolution_fn (radius)

Return facets number for the given arc radius.

Parameters

radius	<decimal> An arc radius.

Returns

<decimal> The number of facets to be assigned to $fn.

The return result of this function can be assigned to the special variables `$fn` to render arcs according to the resolution mode set by $resolution_mode and $resolution_value.

The following table shows the modes that require $resolution_value to be set prior to specify the custom values used during resolution calculation.

$resolution_mode	$resolution_value sets	radius dependent
set	fixed value	no
upf	units per facet	yes
fpu	facets per unit	yes
fpi	facets per inch	yes

The following table has common resolution presets. Equivalent configuration can be obtained using resolution_mode and resolution_value as described in the preview table.

$resolution_mode	preset description	radius dependent
fast	fast rendering mode	no
low	low resolution	yes
medium	medium resolution	yes
high	high resolution	yes
50um	50 micron per facets	yes
100um	100 micron per facets	yes
200um	200 micron per facets	yes
300um	300 micron per facets	yes
400um	400 micron per facets	yes
500um	500 micron per facets	yes
50mil	50 thousandth per facets	yes
100mil	100 thousandth per facets	yes
200mil	200 thousandth per facets	yes
300mil	300 thousandth per facets	yes
400mil	400 thousandth per facets	yes
500mil	500 thousandth per facets	yes

6.20.2.5 function resolution_fs ( )

Return minimum facets size.

Returns

<decimal> Minimum facet size to be assigned to $fs.

The return result of this function can be assigned to the OpenSCAD special variables `$fs` to render arcs according to the resolution mode set by $resolution_mode and $resolution_value.

The following table shows the modes that require $resolution_value to be set prior to calling this function in order to specify the custom values used during resolution calculation.

$resolution_mode	$resolution_value sets	radius dependent
set	fixed value	no
upf	units per facet	no
fpu	facets per unit	no
fpi	facets per inch	no

The following table has common resolution presets. Equivalent configuration can be obtained using resolution_mode and resolution_value as described in the preview table.

$resolution_mode	preset description	radius dependent
fast	fast rendering mode	no
low	low resolution	no
medium	medium resolution	no

high	high resolution	no
50um	50 micron per facets	no
100um	100 micron per facets	no
200um	200 micron per facets	no
300um	300 micron per facets	no
400um	400 micron per facets	no
500um	500 micron per facets	no
50mil	50 thousandth per facets	no
100mil	100 thousandth per facets	no
200mil	200 thousandth per facets	no
300mil	300 thousandth per facets	no
400mil	400 thousandth per facets	no
500mil	500 thousandth per facets	no

6.20.2.6 module resolution_info (radius)

Echo resolution information to the console for given radius.

Parameters

radius	<decimal> An arc radius.

Definition at line 319 of file resolution.scad.

6.20.2.7 function resolution_reduced ( )

Return the radius at which arc resolution will begin to degrade.

Returns

<decimal> Transition radius where resolution reduction begins.

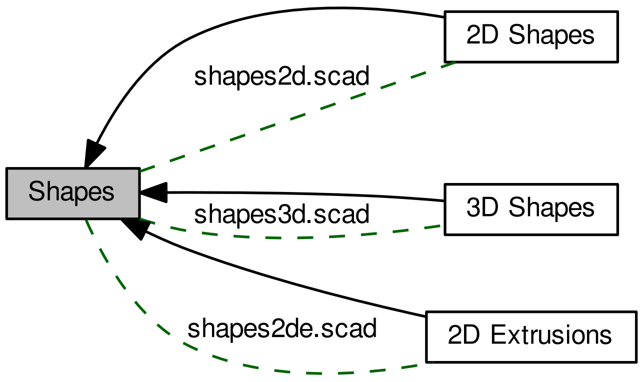
The OpenSCAD special variables `$fs` and `$fa` work together when `$fn=0`. For a given `$fs`, the fragment angle of a drawn arc gets smaller with increasing radius. In other words, the fragment angle is inversely proportional to the arc radius for a given fragment size. The special variable `$fa` enforces a minimum fragment angle limit and at some radius, the fragment angle would becomes smaller than this limit. At this point, OpenSCAD limits further reduction in the facet angle which forces the use of increased fragment size. This in effect begins the gradual reduction of arc resolution with increasing radius.

The return result of this function indicates the radius at which this enforced limiting begins. When `$fn` != 0, returns **'undef'**.

6.21 Shapes

2D and 3D shapes.

Collaboration diagram for Shapes:



Modules

- 2D Extrusions

  Extruded two dimensional geometric shapes.
- 2D Shapes

  Two dimensional geometric shapes.
- 3D Shapes

  Three dimensional geometric shapes.

Files

- file shapes2d.scad

  Two-dimensional basic shapes.
- file shapes2de.scad

  Linearly extruded two-dimensional basic shapes.
- file shapes3d.scad

  Three-dimensional basic shapes.

6.21.1 Detailed Description

2D and 3D shapes.

Table 2: 2D Shapes

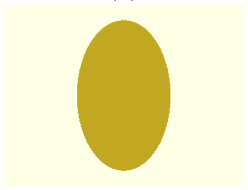
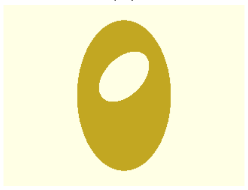
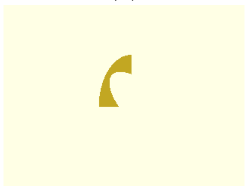
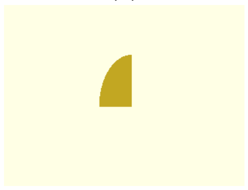
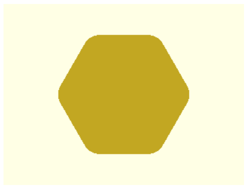
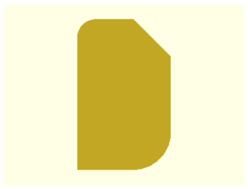
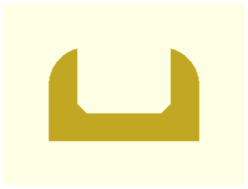
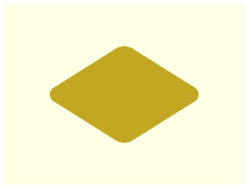
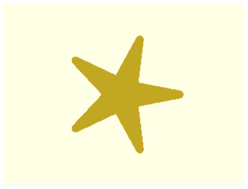
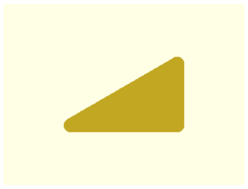
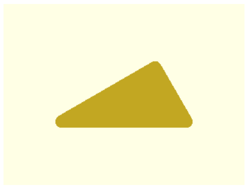
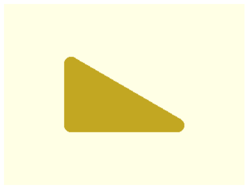
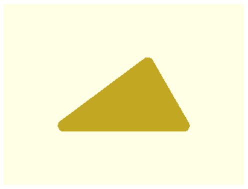
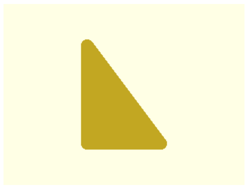
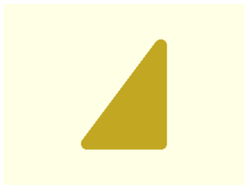
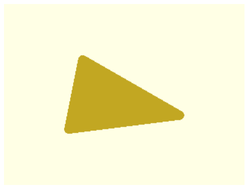
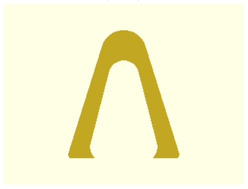
(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			
(9)	(10)	(11)	(12)
			
(13)	(14)	(15)	(16)
			
(17)			
			

Table 3: 2D Extrusions

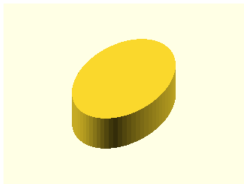
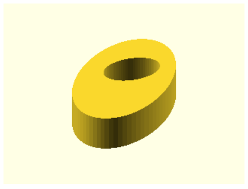
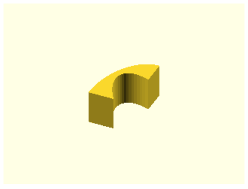
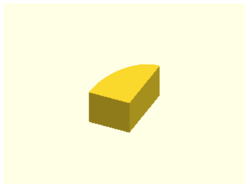
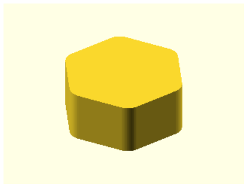
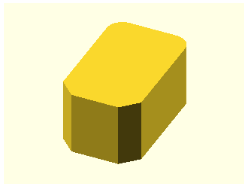
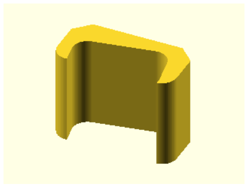
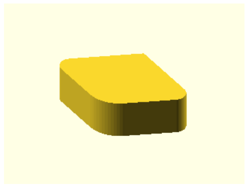
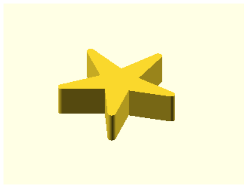
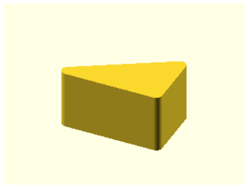
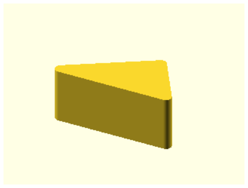
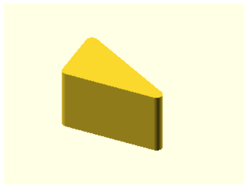
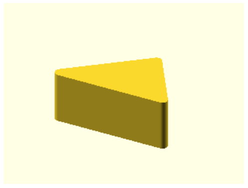
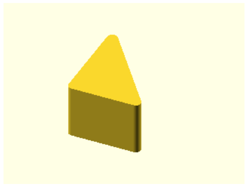
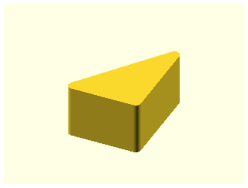
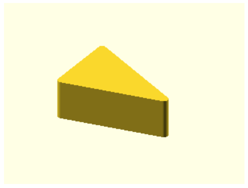
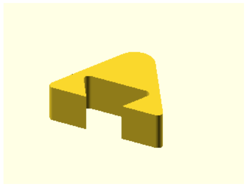
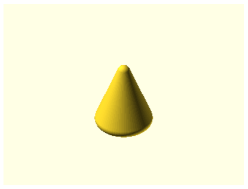
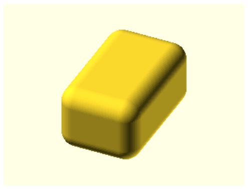
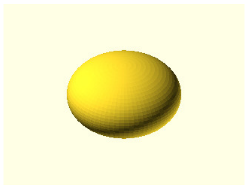
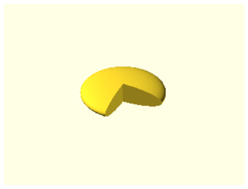
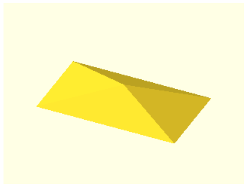
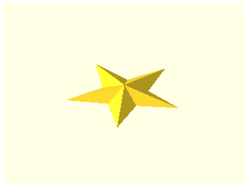
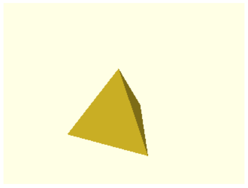
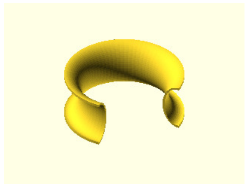
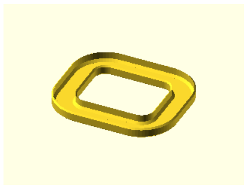
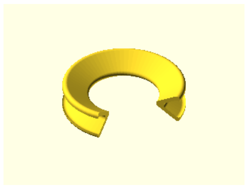
(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			
(9)	(10)	(11)	(12)
			
(13)	(14)	(15)	(16)
			
(17)			
			

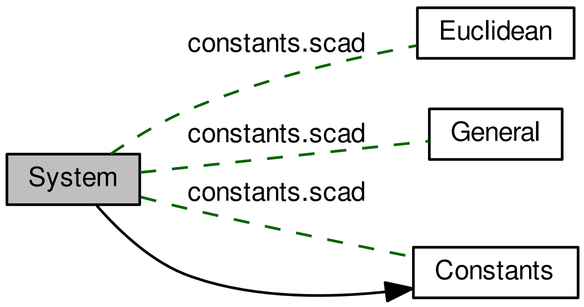
Table 4: 3D Shapes

(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			
(9)	(10)		
			

6.22 System

System/Program limits.

Collaboration diagram for System:



Files

- file constants.scad

  Mechanical design constants.

Variables

- number_max = 1e308

  The largest representable number in OpenSCAD scripts.
- number_min = -1e308

  The smallest representable number in OpenSCAD scripts.
- empty_str = ""

  A string with no content (the empty string).
- empty_v = [ ]

  A vector with no content (the empty vector).

6.22.1 Detailed Description

System/Program limits.

6.23 Tools

Design tools and techniques.

Collaboration diagram for Tools:

Tools
tool_edge.scad
Edge Finishing

Modules

- Edge Finishing

    Shape edge finishing tools.

Files

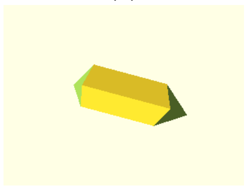
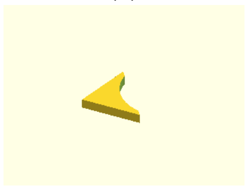
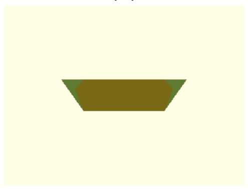
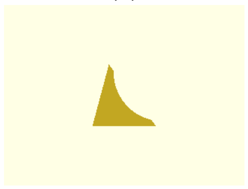
- file tool_edge.scad

    Shape edge finishing tools.

6.23.1 Detailed Description

Design tools and techniques.

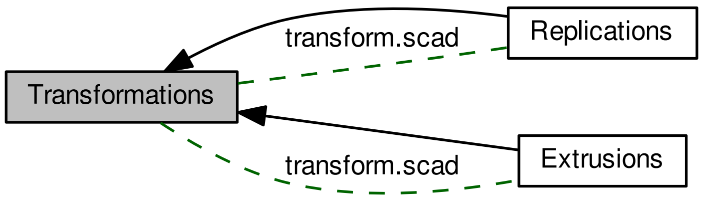
Table 5: Edge Finishing

(1)	(2)	(3)	(4)
			

6.24 Transformations

Shape transformations.

Collaboration diagram for Transformations:



Modules

- Extrusions

  Shape Extrusions.
- Replications

  Shape Replications and distribution.

Files

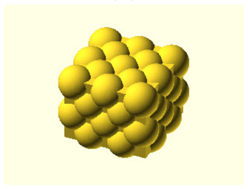
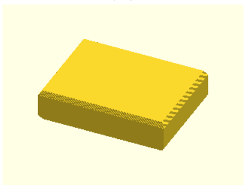
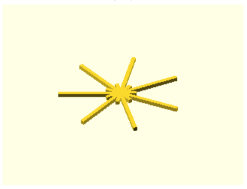
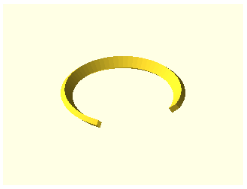
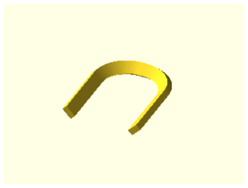
- file transform.scad

  Shape transformation functions.

6.24.1 Detailed Description

Shape transformations.

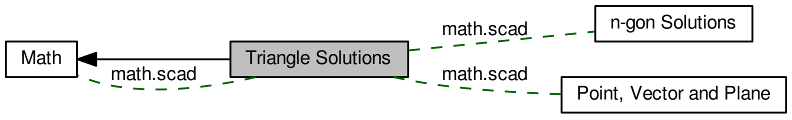
Table 6: Extrusions and Replications

(1)	(2)	(3)	(4)
			
(5)			
			

6.25 Triangle Solutions

Triangle computations.

Collaboration diagram for Triangle Solutions:



Files

- file math.scad

  Mathematical functions.

Functions

- function triangle_lll2vp (s1, s2, s3)

  Compute the vertices of a plane triangle given its side lengths.
- function triangle_vl2vp (v)

  Compute the vertices of a plane triangle given its side lengths.
- function triangle_ppp2vl (v1, v2, v3)

  Compute the side lengths of a triangle given its vertices.
- function triangle_vp2vl (v)

  Compute the side lengths of a triangle given its vertices.
- function triangle_centroid_ppp (v1, v2, v3)

  Compute the centroid (geometric center) of a triangle.
- function triangle_centroid_vp (v)

  Compute the centroid (geometric center) of a triangle.
- function triangle_incenter_ppp (v1, v2, v3)

  Compute the coordinate for the triangle's incircle.
- function triangle_incenter_vp (v)

  Compute the coordinate for the triangle's incircle.
- function triangle_inradius_ppp (v1, v2, v3)

  Compute the inradius of a triangle's incircle.
- function triangle_inradius_vp (v)

  Compute the inradius of a triangle's incircle.

6.25.1 Detailed Description

Triangle computations.

See `Wikipedia` for more information.

6.25.2 Function Documentation

6.25.2.1 function triangle_centroid_ppp (v1 , v2 , v3)

Compute the centroid (geometric center) of a triangle.

Parameters

v1	<vector> A vector [x, y] for vertex 1 coordinates.
v2	<vector> A vector [x, y] for vertex 2 coordinates.
v3	<vector> A vector [x, y] for vertex 3 coordinates.

Returns

<vector> A vector [x, y] of coordinates.

6.25.2.2 function triangle_centroid_vp (v)

Compute the centroid (geometric center) of a triangle.

Parameters

v	<vector> A vector [v1, v2, v3] of vectors [x, y] coordinates.

Returns

<vector> A vector [x, y] of coordinates.

6.25.2.3 function triangle_incenter_ppp (v1 , v2 , v3)

Compute the coordinate for the triangle's incircle.

Parameters

v1	<vector> A vector [x, y] for vertex 1 coordinates.
v2	<vector> A vector [x, y] for vertex 2 coordinates.
v3	<vector> A vector [x, y] for vertex 3 coordinates.

Returns

<vector> A vector [x, y] of coordinates.

The interior point for which distances to the sides of the triangle are equal.

6.25.2.4 function triangle_incenter_vp (v)

Compute the coordinate for the triangle's incircle.

Parameters

v	<vector> A vector [v1, v2, v3] of vectors [x, y] coordinates.

Returns

<vector> A vector [x, y] of coordinates.

The interior point for which distances to the sides of the triangle are equal.

6.25.2.5 function triangle_inradius_ppp (v1 , v2 , v3)

Compute the inradius of a triangle's incircle.

Parameters

v1	<vector> A vector [x, y] for vertex 1 coordinates.
v2	<vector> A vector [x, y] for vertex 2 coordinates.
v3	<vector> A vector [x, y] for vertex 3 coordinates.

Returns

<decimal> The incircle radius.

6.25.2.6 function triangle_inradius_vp (v)

Compute the inradius of a triangle's incircle.

Parameters

v	<vector> A vector [v1, v2, v3] of vectors [x, y] coordinates.

Returns

<decimal> The incircle radius.

6.25.2.7 function triangle_lll2vp (s1 , s2 , s3)

Compute the vertices of a plane triangle given its side lengths.

Parameters

s1	<decimal> The length of the side 1.
s2	<decimal> The length of the side 2.
s3	<decimal> The length of the side 3.

Returns

<vector> A vector [v1, v2, v3] of vectors [x, y] of coordinates.

Vertex `v1` at the origin. Geometry required that `s1` + `s2` is greater then `s3`. Coordinates `v3`**:**[x, y] will be **'nan'** when specified triangle does not exists.

Note

Side length `s1` is measured along the positive x-axis.
Sides are numbered counterclockwise.

6.25.2.8 function triangle_ppp2vl (v1 , v2 , v3)

Compute the side lengths of a triangle given its vertices.

Parameters

v1	<vector> A vector [x, y] for vertex 1 coordinates.

v2	<vector> A vector [x, y] for vertex 2 coordinates.
v3	<vector> A vector [x, y] for vertex 3 coordinates.

Returns

<vector> A vector [s1, s2, s3] of lengths.

Note

Vertices are numbered counterclockwise.

6.25.2.9 function triangle_vl2vp (v)

Compute the vertices of a plane triangle given its side lengths.

Parameters

v	<vector> of decimal side lengths.

Returns

<vector> A vector [v1, v2, v3] of vectors [x, y] of coordinates.

Vertex `vs`[0] at the origin. Geometry required that `vs`[0] + `vs`[1] is greater then `vs`[2]. Coordinates `v3:`[x, y] will be **'nan'** when specified triangle does not exists.

Note

Side length `vs`[0] is measured along the positive x-axis.
Sides are numbered counterclockwise.

6.25.2.10 function triangle_vp2vl (v)

Compute the side lengths of a triangle given its vertices.

Parameters

v	<vector> A vector [v1, v2, v3] of vectors [x, y] coordinates.

Returns

<vector> A vector [s1, s2, s3] of lengths.

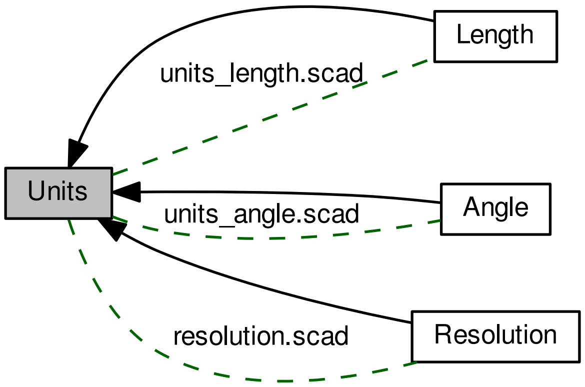
Note

Vertices are numbered counterclockwise.

6.26 Units

Units and unit conversions.

Collaboration diagram for Units:



Modules

- Angle

  Angle units and conversions.
- Length

  Length units and conversions.
- Resolution

  Arch rendering resolution management.

Files

- file units_angle.scad

  Angle units and conversions.
- file units_length.scad

  Length units and conversions.
- file resolution.scad

  Arc rendering resolution abstraction.

6.26.1 Detailed Description

Units and unit conversions.

6.27 Utilities

General utilities.

Files

- file utilities.scad

  Miscellaneous utilities.
- file validation.scad

  Result validation functions.

- function stack (b=0, t=0)

  Format the function call stack as a string.

- function validate (d, cv, t, ev, p=4, pf=false)

  Compare a computed test value with an known good result.

6.27.1 Detailed Description

General utilities.

6.27.2 Function Documentation

6.27.2.1 function stack (b = `0`, t = `0`)

Format the function call stack as a string.

Parameters

b	<decimal> The stack index bottom offset. Include function names above this offset.
t	<decimal> The stack index top offset. Include function names below this offset.

Returns

<string> A colon-separated list of functions names for the current function call stack.

Note

Returns **undef** when `b` is greater than the current number of function instances (ie: `bo > $parent_↩ modules-1`).
Returns the string "`root()`" when the function call stack is empty (ie: at the root of the script).

6.27.2.2 function validate (d , cv , t , ev , p = `4`, pf = `false`)

Compare a computed test value with an known good result.

Parameters

d	<string> A description.
cv	<value> A computed value to validate.
t	<string|boolean> The validation type.
ev	<value> The expected good value.
p	<number> A numerical precision for approximate comparisons.
pf	<boolean> Result reported as a pass or fail boolean value.

Returns

<string|boolean> Validation result indicating if the test passed or failed.

validation types	pass if (else fail)
"almost"	`cv` almost equals `ev`
"equals"	`cv` equals `ev`
"not"	`cv` not equal to `ev`
"true" | **true**	`cv` is **true**
"false" | **false**	`cv` is **false**

Example

```
use <validation.scad>;
use <console.scad>;

//
// function to validate
//
function f1( x ) = (x == undef) ? 1 : 2;

farg = undef;     // function test argument
 erv1 = 1;          // correct expected function result
 erv2 = 3;          // incorrect expected function result

 //
 // pass test example
 //
 pass_result = validate("test-a f1(farg)", f1(farg), "equals", erv1);

 if ( !validate(cv=f1(farg), t="equals", ev=erv1, pf=true) )
   log_warn( pass_result );
 else
   log_info( pass_result );

 //
 // fail test example
 //
 fail_result = validate("test-b f1(farg)", f1(farg), "equals", erv2);

 if ( !validate(cv=f1(farg), t="equals", ev=erv2, pf=true) )
   log_warn( fail_result );
 else
   log_info( fail_result );

 //
 // almost equal test example
 //
 tvae1 = [[90.001], [[45.009], true]];
 tvae2 = [[90.002], [[45.010], true]];

 log_info( validate("test-c", tvae1, "almost", tvae2, 3) );
 log_warn( validate("test-d", tvae1, "almost", tvae2, 4) );
```

Result

```
ECHO: "[ INFO ] root(); passed: 'test-a f1(farg)'"
ECHO:
ECHO: "root()"
```

```
ECHO: "##########################################################################"
ECHO: "# [ WARNING ] FAILED: 'test-b f1(farg)'.  Got '1'. Expected to equal '3' #"
ECHO: "##########################################################################"
ECHO: "[ INFO ] root(); passed: 'test-c'"
ECHO:
ECHO: "root()"
ECHO:
       "#########################################################################################################################
ECHO: "# [ WARNING ] FAILED: 'test-d'.  Got '[[90.001], [[45.009], true]]'. Expected to almost equal
       '[[90.002], [[45.01], true]]' to 4 digits #"
ECHO:
       "#########################################################################################################################
```

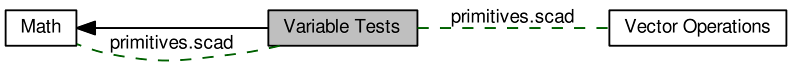
Note

When performing an almost equal validation type, the comparison precision is controlled by `p`. This specifies the number of digits of precision for each numerical comparison. A passing result indicates that `cv` equals `ev` to the number of decimal digits specified by `p`. The comparison is performed by the function almost_equal.

6.28 Variable Tests

Variable property test primitives.

Collaboration diagram for Variable Tests:



Files

- file primitives.scad

  Mathematical primitive functions.

Functions

- function is_defined (v)

  Test if a value is defined.
- function not_defined (v)

  Test if a value is not defined.
- function is_empty (v)

  Test if an iterable value is empty.
- function is_scalar (v)

  Test if a value is a single non-iterable value.
- function is_iterable (v)

  Test if a value has multiple parts and is iterable.
- function is_string (v)

  Test if a value is a string.
- function is_vector (v)

  Test if a value is a vector.
- function is_boolean (v)

  Test if a value is a boolean constant.
- function is_integer (v)

  Test if a value is an integer.
- function is_decimal (v)

  Test if a value is a decimal.
- function is_number (v)

  Test if a value is a number.
- function is_range (v)

  Test if a value is a range definition.
- function is_nan (v)

  Test if a numerical value is invalid.
- function is_inf (v)

Test if a numerical value is infinite.
- function is_even (v)

  Test if a numerical value is even.
- function is_odd (v)

  Test if a numerical value is odd.
- function all_equal (v, cv)

  Test if all elements of a value equal a comparison value.
- function any_equal (v, cv)

  Test if any element of a value equals a comparison value.
- function all_defined (v)

  Test if no element of a value is undefined.
- function any_undefined (v)

  Test if any element of a value is undefined.
- function all_scalars (v)

  Test if all elements of a value are scalars.
- function all_vectors (v)

  Test if all elements of a value are vectors.
- function all_strings (v)

  Test if all elements of a value are strings.
- function all_numbers (v)

  Test if all elements of a value are numbers.
- function all_len (v, l)

  Test if all elements of a value have a given length.
- function almost_equal (v1, v2, p=4)

  Test if all elements of two values are approximately equal.
- function compare (v1, v2, s=true)

  Compare any two values (may be iterable and/or of different types).

6.28.1 Detailed Description

Variable property test primitives.

See validation results group1 and group2.

6.28.2 Function Documentation

6.28.2.1 function all_defined (v)

Test if no element of a value is undefined.

Parameters

v	<value> A value or an iterable value.

Returns

<boolean> **true** when no element is undefined and **false** otherwise.

Warning

Always returns **true** when `v` is empty.

6.28.2.2 function all_equal (v , cv)

Test if all elements of a value equal a comparison value.

Parameters

v	<value> A value or an iterable value.
cv	<value> A comparison value.

Returns

<boolean> **true** when all elements equal the value `cv` and **false** otherwise.

Warning

Always returns **true** when `v` is empty.

6.28.2.3 function all_len (v , l)

Test if all elements of a value have a given length.

Parameters

v	<value> A value or an iterable value.
l	<integer> The length.

Returns

<boolean> **true** when all elements have length equal to `l` and **false** otherwise. Returns the value of `v` when it is not defined.

Warning

Always returns **true** when `v` is empty.

6.28.2.4 function all_numbers (v)

Test if all elements of a value are numbers.

Parameters

v	<value> A value or an iterable value.

Returns

<boolean> **true** when all elements are numerical values and **false** otherwise. Returns **true** when `v` is a single numerical value. Returns the value of `v` when it is not defined.

Warning

Always returns **true** when `v` is empty.

6.28.2.5 function all_scalars (v)

Test if all elements of a value are scalars.

Parameters

v	<value> A value or an iterable value.

Returns

<boolean> **true** when all elements are scalar values and **false** otherwise. Returns **true** when `v` is a single scalar value. Returns the value of `v` when it is not defined.

Warning

Always returns **true** when `v` is empty.

6.28.2.6 function all_strings (v)

Test if all elements of a value are strings.

Parameters

v	<value> A value or an iterable value.

Returns

<boolean> **true** when all elements are string values and **false** otherwise. Returns **true** when `v` is a single string value. Returns the value of `v` when it is not defined.

Warning

Always returns **true** when `v` is empty.

6.28.2.7 function all_vectors (v)

Test if all elements of a value are vectors.

Parameters

v	<value> A value or an iterable value.

Returns

<boolean> **true** when all elements are vector values and **false** otherwise. Returns **true** when `v` is a single vector value. Returns the value of `v` when it is not defined.

Warning

Always returns **true** when `v` is empty.

6.28.2.8 function almost_equal (v1 , v2 , p = 4)

Test if all elements of two values are approximately equal.

Parameters

v1	<value> A value or an iterable value 1.
v2	<value> A value or an iterable value 2.
p	<number> A numerical precision.

Returns

<boolean> **true** when all elements of each values are sufficiently equal and **false** otherwise. All numerical comparisons are performed with precision limited by `p`. All non-numeric comparisons test for exact equality.

Note

The parameter `p` indicated the number of digits of precision for each numerical comparison.

Warning

Always returns **true** when `v` is empty.

6.28.2.9 function any_equal (v , cv)

Test if any element of a value equals a comparison value.

Parameters

v	<value> A value or an iterable value.
cv	<value> A comparison value.

Returns

<boolean> **true** when any element equals the value `cv` and **false** otherwise.

Warning

Always returns **false** when `v` is empty.

6.28.2.10 function any_undefined (v)

Test if any element of a value is undefined.

Parameters

v	<value> A value or an iterable value.

Returns

<boolean> **true** when any element is undefined and **false** otherwise.

Warning

Always returns **false** when `v` is empty.

6.28.2.11 function compare (v1 , v2 , s = `true`)

Compare any two values (may be iterable and/or of different types).

Parameters

v1	<value> A value or an iterable value 1.
v2	<value> A value or an iterable value 2.
s	<boolean> Order ranges by their numerical sum.

Returns

<integer> **-1** when `(v2 < v1)`, **+1** when `(v2 > v1)`, and **0** when `(v2 == v1)`.

The following table summarizes how values are ordered.

order	type	s	intra-type ordering
1	**undef**		(singular)
2	number		numerical comparison
3	string		lexical comparison
4	boolean		**false** < **true**
5	vector		lengths then element-wise comparison
6	range	**true**	compare sum of range elements
6	range	**false**	lengths then element-wise comparison

Note

When comparing two vectors of equal length, the comparison continue element-by-element until an ordering can be determined. Two vectors are declared equal when all elements have been compared and no ordering has been determined.

Warning

The performance of element-wise comparisons of vectors degrades exponentially with vector size.
The sum of a range may quickly exceeded the intermediate variable storage capacity for long ranges.

6.28.2.12 function is_boolean (v)

Test if a value is a boolean constant.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is one of the predefined boolean constants `[true|false]` and **false** otherwise.

6.28.2.13 function is_decimal (v)

Test if a value is a decimal.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a decimal and **false** otherwise.

6.28.2.14 function is_defined (v)

Test if a value is defined.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is defined and **false** otherwise.

6.28.2.15 function is_empty (v)

Test if an iterable value is empty.

Parameters

v	<value> An iterable value.

Returns

<boolean> **true** when the iterable value has zero elements and **false** otherwise.

6.28.2.16 function is_even (v)

Test if a numerical value is even.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be *even* and **false** otherwise.

Note

The value must be valid and defined but may be positive or negative. Any value that is not an integer returns **false**.

6.28.2.17 function is_inf (v)

Test if a numerical value is infinite.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be **inf** (greater than the largest representable number) and **false** otherwise.

6.28.2.18 function is_integer (v)

Test if a value is an integer.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is an integer and **false** otherwise.

6.28.2.19 function is_iterable (v)

Test if a value has multiple parts and is iterable.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is an iterable multi-part value and **false** otherwise.

value is	defined result
undef	**false**
inf	**false**
nan	**false**
integer	**false**
decimal	**false**
boolean	**false**
string	**true**
vector	**true**
range	not defined

6.28.2.20 function is_nan (v)

Test if a numerical value is invalid.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be **nan** (Not A Number) and **false** otherwise.

6.28.2.21 function is_number (v)

Test if a value is a number.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a number and **false** otherwise.

Warning

Returns **true** even for numerical values that are considered infinite and invalid.

6.28.2.22 function is_odd (v)

Test if a numerical value is odd.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be *odd* and **false** otherwise.

Note

The value must be valid and defined but may be positive or negative. Any value that is not an integer returns **false**.

6.28.2.23 function is_range (v)

Test if a value is a range definition.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a range definition and **false** otherwise.

6.28.2.24 function is_scalar (v)

Test if a value is a single non-iterable value.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a single non-iterable value and **false** otherwise.

value is	defined result
undef	**true**
inf	**true**
nan	**true**
integer	**true**
decimal	**true**
boolean	**true**
string	**false**
vector	**false**
range	not defined

6.28.2.25 function is_string (v)

Test if a value is a string.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a string and **false** otherwise.

6.28.2.26 function is_vector (v)

Test if a value is a vector.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a vector and **false** otherwise.

6.28.2.27 function not_defined (v)

Test if a value is not defined.

Parameters

v	<value> A value.

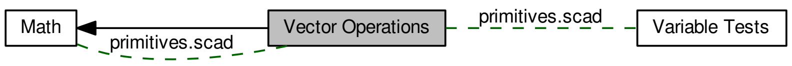
Returns

<boolean> **true** when the value is not defined and **false** otherwise.

6.29 Vector Operations

Vector operation primitives.

Collaboration diagram for Vector Operations:



Files

- file primitives.scad

  Mathematical primitive functions.

Functions

- function consts (l, v)

  Create a vector of constant elements.
- function vstr (v)

  Convert all vector elements to strings and concatenate.
- function sum (v, i1, i2)

  Compute the sum of a vector of numbers.
- function find (mv, v, c=1, i, i1=0, i2)

  Find the occurrences of a match value in an iterable value.
- function count (mv, v, s=true, i)

  Count all occurrences of a match value in an iterable value.
- function exists (mv, v, s=true, i)

  Check the existence of a match value in an iterable value.
- function defined_or (v, d)

  Return a defined or default value.
- function edefined_or (v, i, d)

  Return a defined vector element or default value.
- function first (v)

  Return the first element of an iterable value.
- function second (v)

  Return the second element of an iterable value.
- function last (v)

  Return the last element of an iterable value.
- function nfirst (v, n=1)

  Return a vector containing the first n elements of an iterable value.
- function nlast (v, n=1)

  Return a vector containing the last n elements of an iterable value.
- function nhead (v, n=1)

Return a vector containing all but the last n elements of an iterable value.

- function ntail (v, n=1)

    Return a vector containing all but the first n elements of an iterable value.
- function rselect (v, i)

    Select a range of elements from an iterable value.
- function eselect (v, f=true, l=false, i)

    Select an element from each iterable value.
- function ciselect (v, i)

    Case-like select a value from a vector of ordered options by index.
- function cmvselect (v, mv)

    Case-like select a value from a vector of identified options by match-value.
- function smerge (v, r=false)

    Serial-merge vectors of iterable values.
- function pmerge (v, j=true)

    Parallel-merge vectors of iterable values.
- function reverse (v)

    Reverse the elements of an iterable value.
- function qsort (v, r=false)

    Sort the numeric or string elements of a vector using quick sort.
- function qsort2 (v, d=0, r=false, s=true)

    Hierarchically sort all elements of a vector using quick sort.
- function strip (v, mv=empty_v)

    Strip all matching values from an iterable value.
- function eappend (nv, v, r=true, j=true, l=true)

    Append a value to each element of an iterable value.
- function insert (nv, v, i=0, mv, mi=0, s=true, si)

    Insert a new value into an iterable value.
- function delete (v, i, mv, mc=1, s=true, si)

    Delete elements from an iterable value.
- function unique (v)

    Return the unique elements of an iterable value.

6.29.1 Detailed Description

Vector operation primitives.

See validation results.

6.29.2 Function Documentation

6.29.2.1 function ciselect (v , i)

Case-like select a value from a vector of ordered options by index.

Parameters

v	<vector> A vector of values.
i	<integer> Element selection index.

Returns

<value> The value of the vector element at the specified index. Returns the default value when `i` does not map to an element of `v` or when `i` is undefined.

Behaves like a case statement for selecting values from a list of *ordered options*. The default value is: `last(v)`.

Example

```
ov = [ "value1", "value2", "default" ];

ciselect( ov );     // "default"
ciselect( ov, 4 );  // "default"
ciselect( ov, 0 );  // "value1"
```

6.29.2.2 function cmvselect (v , mv)

Case-like select a value from a vector of identified options by match-value.

Parameters

v	<vector> A two dimensional vector of one or more identified values [[identifier, value], ...].
mv	<value> Element selection match value.

Returns

<value> The value from the vector of identified elements with an identifier matching `mv`. Returns the default value when `mv` does not match any of the element identifiers of `v` or when `mv` is undefined.

Behaves like a case statement for selecting values from a list of *identified options*. The default value is↩ : `second(last(v))`.

Example

```
ov = [ [0,"value0"], ["a","value1"], ["b","value2"], ["c","default"] ];

cmvselect( ov );       // "default"
cmvselect( ov, "x" ); // "default"
cmvselect( ov, 0 );    // "value0"
cmvselect( ov, "b" ); // "value2"
```

6.29.2.3 function consts (l , v)

Create a vector of constant elements.

Parameters

l	<integer> The vector length.
v	<value> The element value.

Returns

<vector> With `l` copies of the element value `v`. Returns **empty_v** when `l` is not a number or if `(l < 1)`.

Note

When `v` is not specified, each element is assigned the value of its index position.

6.29.2.4 function count (mv , v , s = `true`, i)

Count all occurrences of a match value in an iterable value.

Parameters

mv	<value> A match value.
v	<value> An iterable value.
s	<boolean> Use search for element matching (**false** uses find).
i	<integer> The element column index to match.

Returns

<integer> The number of times `mv` occurs in `v`.

See find() for information on value matching.

6.29.2.5 function defined_or (v , d)

Return a defined or default value.

Parameters

v	<value> A value.
d	<value> A default value.

Returns

<value> `v` when it is defined or `d` otherwise.

6.29.2.6 function delete (v , i , mv , mc = `1`, s = `true`, si)

Delete elements from an iterable value.

Parameters

v	<value> An iterable value.
i	<range|vector|integer> Deletion Indexes.
mv	<vector|string|value> Match value candidates.
mc	<integer> A match count. For `(mc>=1)`, remove the first `mc` matches. For `(mc<=0)`, remove all matches.
s	<boolean> Use search for element matching (**false** uses find).
si	<integer> The element column index when matching.

Returns

<vector> `v` with all specified elements removed. Returns **undef** when `i` does not map to an element of `v`. Returns **undef** when `v` is not defined or is not iterable.

The elements to delete can be specified by an index position, a vector of index positions, an index range, an element match value, or a vector of element match values (when using search). When `mv` is a vector of match values, all values of `mv` that exists in `v` are candidates for deletion. For each matching candidate, `mc` indicates the quantity to remove. When more than one deletion criteria is specified, the order of precedence is: `mv`, `i`.

See find() for information on value matching.

6.29.2.7 function eappend (nv , v , r = `true`, j = `true`, l = `true`)

Append a value to each element of an iterable value.

Parameters

nv	<value> A new value to append.
v	<vector> A vector of values.
r	<boolean> Reduce vector element value before appending.
j	<boolean> Join each appendage as a vector.
l	<boolean> Append to last element.

Returns

<vector> With `nv` appended to each element of `v`. Returns **undef** when `v` is not defined or is not iterable.

Example

```
v1=[["a"], ["b"], ["c"], ["d"]];
v2=[1, 2, 3];

echo( eappend( v2, v1 ) );
echo( eappend( v2, v1, r=false ) );
echo( eappend( v2, v1, j=false, l=false ) );
```

Result

```
ECHO: [["a", 1, 2, 3], ["b", 1, 2, 3], ["c", 1, 2, 3], ["d", 1, 2, 3]]
ECHO: [[["a"], 1, 2, 3], [["b"], 1, 2, 3], [["c"], 1, 2, 3], [["d"], 1, 2, 3]]
ECHO: ["a", 1, 2, 3, "b", 1, 2, 3, "c", 1, 2, 3, "d"]
```

Note

Appending with reduction causes `nv` to be appended to the *elements* of each value of `v` that is a vector. Otherwise, `nv` is appended to the *vector* itself of each value of `v` that is a vector.

6.29.2.8 function edefined_or (v , i , d)

Return a defined vector element or default value.

Parameters

v	<vector> A vector.
i	<integer> An element index.
d	<value> A default value.

Returns

<value> `v[i]` when it is defined or `d` otherwise.

6.29.2.9 function eselect (v , f = `true`, l = `false`, i)

Select an element from each iterable value.

Parameters

v	<vector> A vector of iterable values.

f	<boolean> Select the first element.
l	<boolean> Select the last element.
i	<integer> Select a numeric element index position.

Returns

<vector> Containing the selected element of each iterable value of `v`. Returns **empty_v** when `v` is empty. Returns **undef** when `v` is not defined or is not iterable.

Note

When more than one selection criteria is specified, the order of precedence is: `i`, `l`, `f`.

6.29.2.10 function exists (mv , v , s = `true`, i)

Check the existence of a match value in an iterable value.

Parameters

mv	<value> A match value.
v	<value> An iterable value.
s	<boolean> Use search for element matching (**false** uses find).
i	<integer> The element column index to match.

Returns

<boolean> **true** when `mv` exists in `v` and **false** otherwise.

See find() for information on value matching.

6.29.2.11 function find (mv , v , c = `1`, i , i1 = `0`, i2)

Find the occurrences of a match value in an iterable value.

Parameters

mv	<value> A match value.
v	<value> An iterable value.
c	<integer> A match count. For `(c>=1)`, return the first `c` matches. For `(c<=0)`, return all matches.
i	<integer> The element column index to match.
i1	<integer> The element index where find begins (default: first).
i2	<integer> The element index where find ends (default: last).

Returns

<vector> Of indexes where elements match `mv`. Returns **empty_v** when no element of `v` matches `mv` or when `v` is not iterable.

The use-cases for find() and `search()` are summarized in the following tables.

Find:

mv / v	string	vector of scalars	vector of iterables
scalar		(a)	(b) see note 1
string	(c)		(b) see note 1
vector of scalars			(b) see note 1
vector of iterables			(b) see note 1

Search:

mv / v	string	vector of scalars	vector of iterables
scalar		(a)	(b)
string	(d)	invalid	(e) see note 2
vector of scalars		(f)	(g)
vector of iterables			(g)

Key:

- (a) Identify each element of `v` that equals `mv`.
- (b) Identify each element of `v` where `mv` equals the element at the specified column index, `i`, of each iterable value in `v`.
- (c) If, and only if, `mv` is a single character, identify each character in `v` that equals `mv`.
- (d) For each character of `mv`, identify where it exists in `v`. **empty_v** is returned for each character of `mv` absent from `v`.
- (e) For each character of `mv`, identify where it exists in `v` either as a numeric value or as a character at the specified column index, `i`. **empty_v** is returned for each character of `mv` absent from `v`.
- (f) For each scalar of `mv`, identify where it exists in `v`. **empty_v** is returned for each scalar of `mv` absent from `v`.
- (g) For each element of `mv`, identify where it equals the element at the specified column index, `i`, of each iterable value in `v`. **empty_v** is returned for each element of `mv` absent from `v` in the specified column index.

Note

1: When `i` is specified, that element column is compared. Otherwise, the entire element is compared. Functions find() and `search()` behave differently in this regard.
2: Invalid use combination when any element of `v` is a string. However, an element that is a vector of one or more strings is valid. In which case, only the first character of each string element is considered.

6.29.2.12 function first (v)

Return the first element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The first element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.13 function insert (nv , v , i = `0`, mv , mi = `0`, s = `true`, si)

Insert a new value into an iterable value.

Parameters

nv	<value> A new value to insert.
v	<value> An iterable value.
i	<integer> An insert position index.
mv	<vector|string|value> Match value candidates.
mi	<integer> A match index.
s	<boolean> Use search for element matching (**false** uses find).
si	<integer> The element column index when matching.

Returns

<vector> With `nv` inserted into `v` at the specified position. Returns **undef** when no value of `mv` exists in `v`. Returns **undef** when `(mi + 1)` exceeds the matched element count. Returns **undef** when `i` does not map to an element of `v`. Returns **undef** when `v` is not defined or is not iterable.

The insert position can be specified by an index, an element match value, or vector of potential match values (when using search). When multiple matches exists, `mi` indicates the insert position. When more than one insert position criteria is specified, the order of precedence is: `mv`, `i`.

See find() for information on value matching.

6.29.2.14 function last (v)

Return the last element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The last element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.15 function nfirst (v , n = `1`)

Return a vector containing the first n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> An element count.

Returns

<vector> Containing the first `n` elements of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.16 function nhead (v , n = `1`)

Return a vector containing all but the last n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> An element count.

Returns

<vector> Containing all but the last `n` elements of `v`. Returns **empty_v** when `v` contains fewer than `n` elements. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.17 function nlast (v , n = `1`)

Return a vector containing the last n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> An element count.

Returns

<vector> Containing the last `n` elements of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.18 function ntail (v , n = `1`)

Return a vector containing all but the first n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> An element count.

Returns

<vector> Containing all but the first n elements of `v`. Returns **empty_v** when `v` contains fewer than `n` elements. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.19 function pmerge (v , j = `true`)

Parallel-merge vectors of iterable values.

Parameters

v	<vector> A vector of iterable values.
j	<boolean> Join each merge as a vector.

Returns

<vector> Containing the parallel-wise element concatenation of each iterable value in `v`. Returns **empty_v** when any element value in `v` is empty. Returns **undef** when `v` is not defined or when any element value in `v` is not iterable.

Example

```
v1=["a", "b", "c", "d"];
v2=[1, 2, 3];

echo( pmerge( [v1, v2], true ) );
echo( pmerge( [v1, v2], false ) );
```

Result

```
ECHO: [["a", 1], ["b", 2], ["c", 3]]
ECHO: ["a", 1, "b", 2, "c", 3]
```

Note

The resulting vector length will be limited by the iterable value with the shortest length.
A string, although iterable, is treated as a merged unit.

6.29.2.20 function qsort (v , r = `false`)

Sort the numeric or string elements of a vector using quick sort.

Parameters

v	<vector> A vector of values.
r	<boolean> Reverse sort order.

Returns

<vector> With elements sorted in ascending order. Returns **undef** when `v` is not all strings or all numbers. Returns **undef** when `v` is not defined or is not a vector.

Warning

This implementation relies on the comparison operators '<' and '>' which expect the operands to be either two scalar numbers or two strings. Therefore, this function returns **undef** for vectors containing anything other than all scalar numbers or all strings.

See `Wikipedia` for more information.

6.29.2.21 function qsort2 (v , d = `0`, r = `false`, s = `true`)

Hierarchically sort all elements of a vector using quick sort.

Parameters

v	<vector> A vector of values.
d	<integer> Recursive sort depth.
r	<boolean> Reverse sort order.
s	<boolean> Order ranges by their numerical sum.

Returns

<vector> With all elements sorted in ascending order. Returns **undef** when `v` is not defined or is not a vector.

Elements are sorted using the compare function. See its documentation for a description of the parameter `s`. To recursively sort all elements, set `d` greater than, or equal to, the maximum level of hierarchy in `v`.

See `Wikipedia` for more information.

6.29.2.22 function reverse (v)

Reverse the elements of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<vector> Containing the elements of `v` in reversed order. Returns **empty_v** when `v` is empty. Returns **undef** when `v` is not defined or is not iterable.

6.29.2.23 function rselect (v , i)

Select a range of elements from an iterable value.

Parameters

v	<value> An iterable value.
i	<range|vector|integer> Index selection.

Returns

<vector> Containing the vector element indexes selected in `i`. Returns **undef** when `i` does not map to an element of `v`. Returns **empty_v** when `v` is empty. Returns **undef** when `v` is not defined or is not iterable.

6.29.2.24 function second (v)

Return the second element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The second element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

6.29.2.25 function smerge (v, r = `false`)

Serial-merge vectors of iterable values.

Parameters

v	<vector> A vector of iterable values.
r	<boolean> Recursively merge iterable elements.

Returns

<vector> Containing the serial-wise element concatenation of each element in `v`. Returns **empty_v** when `v` is empty. Returns **undef** when `v` is not defined.

Note

A string, although iterable, is treated as a merged unit.

6.29.2.26 function strip (v, mv = **empty_v**)

Strip all matching values from an iterable value.

Parameters

v	<vector> A vector of values.
mv	<value> A match value.

Returns

<vector> `v` with all elements equal to `mv` removed. Returns **undef** when `v` is not defined or is not iterable.

6.29.2.27 function sum (v, i1, i2)

Compute the sum of a vector of numbers.

Parameters

v	<range|vector> A vector of numerical values.
i1	<integer> The element index at which to begin summation (first when not specified).
i2	<integer> The element index at which to end summation (last when not specified).

Returns

<decimal> The summation of elements over the index range. Returns **v** when it is a scalar number. Returns 0 when `v` is empty. Returns **undef** when `v` is not defined or is not iterable and not a number.

6.29.2.28 function unique (v)

Return the unique elements of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<vector> Of unique elements of `v` with order preserved. Returns **undef** when `v` is not defined or is not iterable.

6.29.2.29 function vstr (v)

Convert all vector elements to strings and concatenate.

Parameters

v	<vector> A vector of values.

Returns

<string> Constructed by converting each element of the vector to a string and concatenating together. Returns **undef** when `v` is not defined.

Example

```
v1=["a", "b", "c", "d"];
v2=[1, 2, 3];

echo( vstr(concat(v1, v2)) );
```

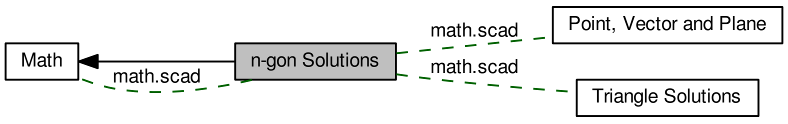
Result

```
ECHO: "abcd123"
```

6.30 n-gon Solutions

Regular n-sided polygon computations.

Collaboration diagram for n-gon Solutions:



Files

- file math.scad

  Mathematical functions.

Functions

- function ngon_vp (n, r, vr)

  Compute the vertices for an n-sided equiangular/equilateral regular polygon.

6.30.1 Detailed Description

Regular n-sided polygon computations.

6.30.2 Function Documentation

6.30.2.1 function ngon_vp (n , r , vr)

Compute the vertices for an n-sided equiangular/equilateral regular polygon.

Parameters

n	<decimal> The number of sides.
r	<decimal> The ngon vertex radius.
vr	<decimal> The vertex rounding radius.

Returns

<vector> A vector [v1, v2, ..., vn] of vectors [x, y] of coordinates.

Example

```
vr=5;

hull()
{
  for ( p = ngon_vp( r=20, n=5, vr=vr ) )
```

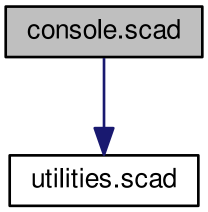
```
    translate( p )
    circle( r=vr );
}
```

7 File Documentation

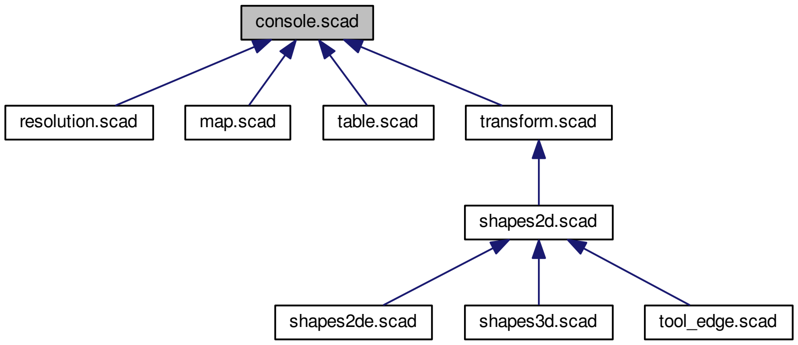
7.1 console.scad File Reference

Message logging functions.

`#include <utilities.scad>`
Include dependency graph for console.scad:



This graph shows which files directly or indirectly include this file:



Functions

- module log_echo (m)

  Output message to console.
- module log_debug (m)

  Output diagnostic message to console.
- module log_info (m)

  Output information message to console.
- module log_warn (m)

Output warning message to console.

- module log_error (m)

  Output error message to console.

7.1.1 Detailed Description

Message logging functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

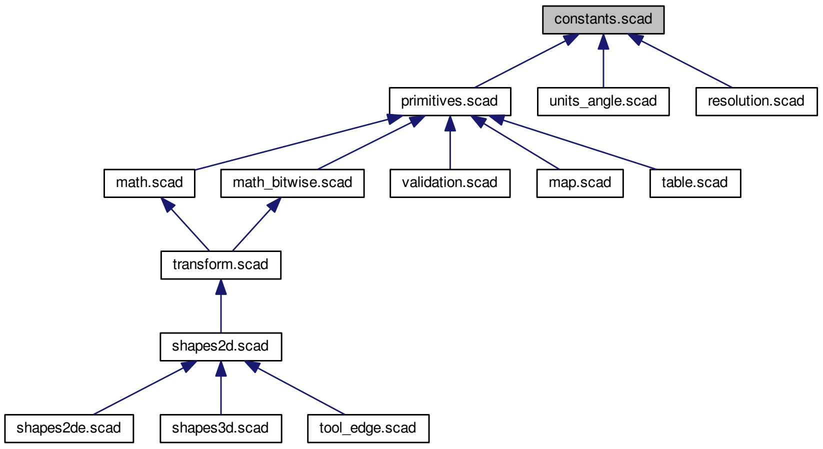
The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

7.2 constants.scad File Reference

Mechanical design constants.

This graph shows which files directly or indirectly include this file:



Variables

- eps = 0.01

  <decimal> Epsilon, small distance to deal with overlaping shapes
- pi = 3.14159265358979323846264338327950

  <decimal> The ratio of a circle's circumference to its diameter
- tau = 2∗pi

  <decimal> The ratio of a circle's circumference to its radius
- number_max = 1e308

  The largest representable number in OpenSCAD scripts.
- number_min = -1e308

  The smallest representable number in OpenSCAD scripts.
- empty_str = ""

  A string with no content (the empty string).
- empty_v = [ ]

  A vector with no content (the empty vector).
- x_axis_vi = 0

  The vector index for the x-coordinate of a vector.
- y_axis_vi = 1

  The vector index for the y-coordinate of a vector.
- z_axis_vi = 2

  The vector index for the z-coordinate of a vector.
- origin2d = [0, 0]

  The origin coordinates in 2-dimensional Euclidean space.
- x_axis2d_uv = [1, 0]

  The unit vector of the positive x-axis in 2-dimensional Euclidean space.
- y_axis2d_uv = [0, 1]

  The unit vector of the positive y-axis in 2-dimensional Euclidean space.

- origin3d = [0, 0, 0]

  The origin coordinates in 3-dimensional Euclidean space.
- x_axis3d_uv = [1, 0, 0]

  The unit vector of the positive x-axis in 3-dimensional Euclidean space.
- y_axis3d_uv = [0, 1, 0]

  The unit vector of the positive y-axis in 3-dimensional Euclidean space.
- z_axis3d_uv = [0, 0, 1]

  The unit vector of the positive z-axis in 3-dimensional Euclidean space.

7.2.1 Detailed Description

Mechanical design constants.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.↩
gnu.org/licenses/.

Note

Include this library file using the **include** statement.

7.3 mainpage.scad File Reference

Documentation main page.

7.3.1 Detailed Description

Documentation main page.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

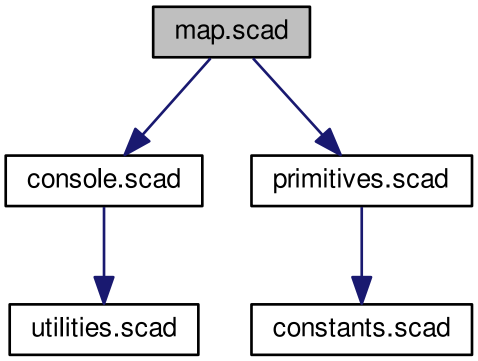
You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

7.4 map.scad File Reference

Mapped key-value pair data access.

```
#include <console.scad>
#include <primitives.scad>
```

Include dependency graph for map.scad:



Functions

- function map_get_idx (map, key)

    Return the index for the storage location of a map key-value pair.
- function map_exists (map, key)

    Test if a key exists in a map.
- function map_get (map, key)

    Get the value associated with a map key.
- function map_get_keys (map)

    Get a vector of the map entry identifier keys.
- function map_get_values (map)

    Get a vector of the map entry values.
- function map_size (map)

    Get the number of key-value pairs stored in a map.
- module map_check (map, verbose=false)

    Perform some basic validation/checks on a map.
- module map_dump (map, sort=true, number=true, p=3)

    Dump each map key-value pair to the console.

7.4.1 Detailed Description

Mapped key-value pair data access.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

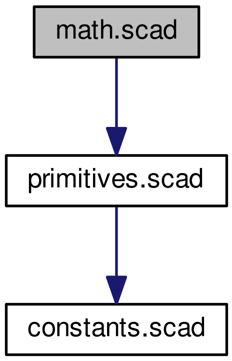
You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

Manage a collection of key-value pairs where keys are unique.

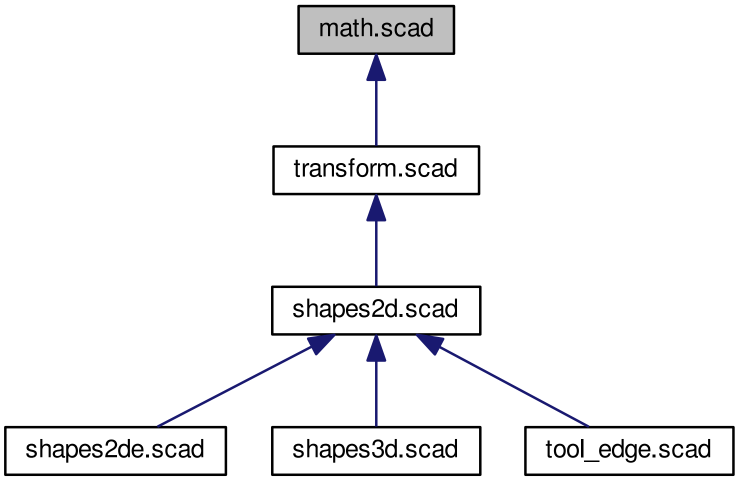
7.5 math.scad File Reference

Mathematical functions.

`#include <primitives.scad>`
Include dependency graph for math.scad:



This graph shows which files directly or indirectly include this file:



Functions

- function distance_pp (p1, p2)

  Compute the distance between two points in a Euclidean 1, 2, or 3D-space.
- function dot_vv (v1t, v2t, v1i, v2i)

  Compute the dot product of two vectors.
- function cross_vv (v1t, v2t, v1i, v2i)

  Compute the cross product of two vectors in a Euclidean 3D-space (2D).
- function striple_vvv (v1t, v2t, v3t, v1i, v2i, v3i)

  Compute scalar triple product of two vectors in a Euclidean 3D-space.
- function angle_vv (v1t, v2t, v1i, v2i)

  Compute the angle between two vectors in a Euclidean 2 or 3D-space.
- function angle_vvn (v1t, v2t, nvt, v1i, v2i, nvi)

  Compute the angle between two vectors in a Euclidean 3D-space.
- function unit_v (vt, vi)

  Compute the normalized unit vector for a 1, 2, or 3 term vector.
- function are_coplanar_vvv (v1t, v2t, v3t, v1i, v2i, v3i)

  Test if three vectors are coplanar in Euclidean 3D-space.
- function ngon_vp (n, r, vr)

  Compute the vertices for an n-sided equiangular/equilateral regular polygon.
- function triangle_lll2vp (s1, s2, s3)

  Compute the vertices of a plane triangle given its side lengths.
- function triangle_vl2vp (v)

  Compute the vertices of a plane triangle given its side lengths.
- function triangle_ppp2vl (v1, v2, v3)

  Compute the side lengths of a triangle given its vertices.
- function triangle_vp2vl (v)

  Compute the side lengths of a triangle given its vertices.
- function triangle_centroid_ppp (v1, v2, v3)

  Compute the centroid (geometric center) of a triangle.
- function triangle_centroid_vp (v)

  Compute the centroid (geometric center) of a triangle.
- function triangle_incenter_ppp (v1, v2, v3)

  Compute the coordinate for the triangle's incircle.
- function triangle_incenter_vp (v)

  Compute the coordinate for the triangle's incircle.
- function triangle_inradius_ppp (v1, v2, v3)

  Compute the inradius of a triangle's incircle.
- function triangle_inradius_vp (v)

  Compute the inradius of a triangle's incircle.

7.5.1 Detailed Description

Mathematical functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.↩
gnu.org/licenses/.

Note

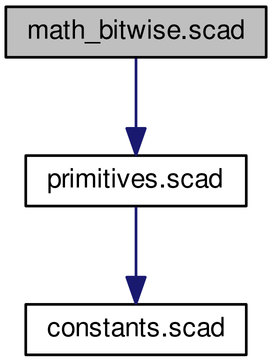
Include this library file using the **include** statement.

7.6 math_bitwise.scad File Reference

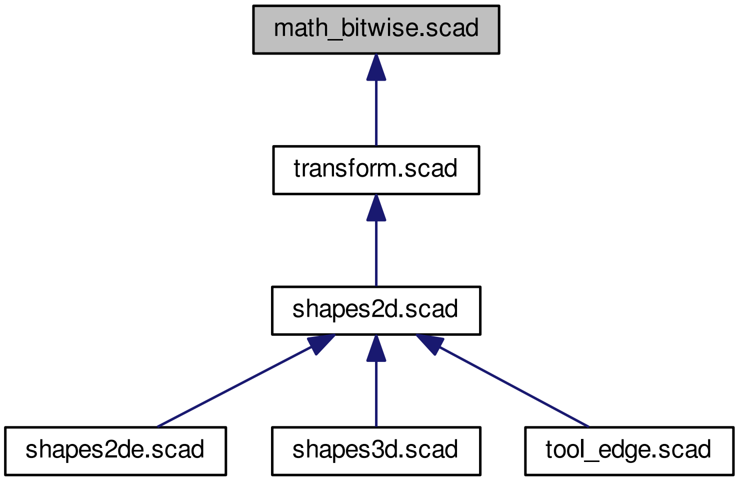
Mathematical bitwise binary (base-two) functions.

```
#include <primitives.scad>
```

Include dependency graph for math_bitwise.scad:



This graph shows which files directly or indirectly include this file:



Functions

- function bitwise_is_equal (v, b, t=1)

  Test if a base-two bit position of an integer value equals a test bit.
- function bitwise_i2v (v, w=1, bv=1)

  Encode an integer value as a base-two vector of bits.
- function bitwise_v2i (v)

  Decode a base-two vector of bits to an integer value.
- function bitwise_i2s (v, w=1)

  Encode an integer value as a base-two string of bits.
- function bitwise_s2i (v)

  Decode a base-two string of bits to an integer value.
- function bitwise_and (v1, v2, bv=1)

  Base-two bitwise AND operation for integers.
- function bitwise_or (v1, v2, bv=1)

  Base-two bitwise OR operation for integers.
- function bitwise_xor (v1, v2, bv=1)

  Base-two bitwise XOR operation for integers.
- function bitwise_not (v, w=1, bv=1)

  Base-two bitwise NOT operation for an integer.
- function bitwise_lsh (v, s=1, bm=1, bv=1)

  Base-two bitwise left-shift operation for an integer.
- function bitwise_rsh (v, s=1)

  Base-two bitwise right-shift operation for an integer.

7.6.1 Detailed Description

Mathematical bitwise binary (base-two) functions.

Author

Roy Allen Sutton

Date

2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

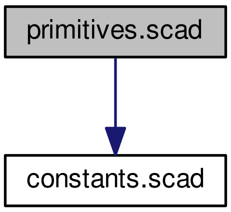
Note

Include this library file using the **include** statement.

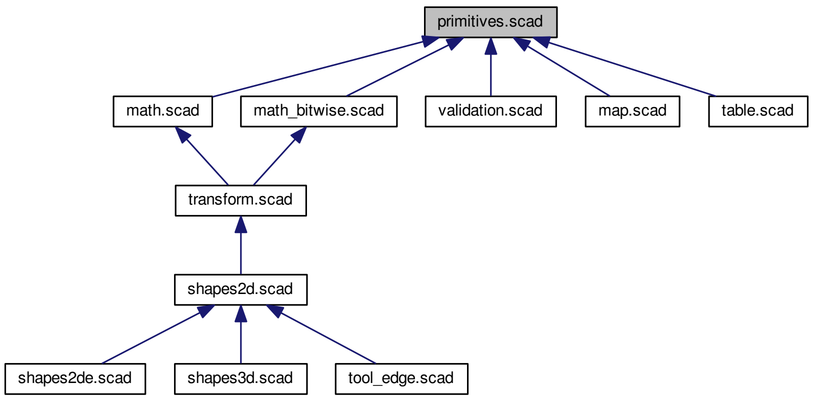
7.7 primitives.scad File Reference

Mathematical primitive functions.

```
#include <constants.scad>
```
Include dependency graph for primitives.scad:



This graph shows which files directly or indirectly include this file:



Functions

- function is_defined (v)

  Test if a value is defined.
- function not_defined (v)

  Test if a value is not defined.
- function is_empty (v)

  Test if an iterable value is empty.
- function is_scalar (v)

  Test if a value is a single non-iterable value.
- function is_iterable (v)

  Test if a value has multiple parts and is iterable.
- function is_string (v)

  Test if a value is a string.
- function is_vector (v)

  Test if a value is a vector.
- function is_boolean (v)

  Test if a value is a boolean constant.
- function is_integer (v)

  Test if a value is an integer.
- function is_decimal (v)

  Test if a value is a decimal.
- function is_number (v)

  Test if a value is a number.
- function is_range (v)

  Test if a value is a range definition.
- function is_nan (v)

  Test if a numerical value is invalid.
- function is_inf (v)

Test if a numerical value is infinite.
- function is_even (v)

  Test if a numerical value is even.
- function is_odd (v)

  Test if a numerical value is odd.
- function all_equal (v, cv)

  Test if all elements of a value equal a comparison value.
- function any_equal (v, cv)

  Test if any element of a value equals a comparison value.
- function all_defined (v)

  Test if no element of a value is undefined.
- function any_undefined (v)

  Test if any element of a value is undefined.
- function all_scalars (v)

  Test if all elements of a value are scalars.
- function all_vectors (v)

  Test if all elements of a value are vectors.
- function all_strings (v)

  Test if all elements of a value are strings.
- function all_numbers (v)

  Test if all elements of a value are numbers.
- function all_len (v, l)

  Test if all elements of a value have a given length.
- function almost_equal (v1, v2, p=4)

  Test if all elements of two values are approximately equal.
- function compare (v1, v2, s=true)

  Compare any two values (may be iterable and/or of different types).
- function consts (l, v)

  Create a vector of constant elements.
- function vstr (v)

  Convert all vector elements to strings and concatenate.
- function sum (v, i1, i2)

  Compute the sum of a vector of numbers.
- function find (mv, v, c=1, i, i1=0, i2)

  Find the occurrences of a match value in an iterable value.
- function count (mv, v, s=true, i)

  Count all occurrences of a match value in an iterable value.
- function exists (mv, v, s=true, i)

  Check the existence of a match value in an iterable value.
- function defined_or (v, d)

  Return a defined or default value.
- function edefined_or (v, i, d)

  Return a defined vector element or default value.
- function first (v)

  Return the first element of an iterable value.
- function second (v)

  Return the second element of an iterable value.

- function last (v)

  Return the last element of an iterable value.
- function nfirst (v, n=1)

  Return a vector containing the first n elements of an iterable value.
- function nlast (v, n=1)

  Return a vector containing the last n elements of an iterable value.
- function nhead (v, n=1)

  Return a vector containing all but the last n elements of an iterable value.
- function ntail (v, n=1)

  Return a vector containing all but the first n elements of an iterable value.
- function rselect (v, i)

  Select a range of elements from an iterable value.
- function eselect (v, f=true, l=false, i)

  Select an element from each iterable value.
- function ciselect (v, i)

  Case-like select a value from a vector of ordered options by index.
- function cmvselect (v, mv)

  Case-like select a value from a vector of identified options by match-value.
- function smerge (v, r=false)

  Serial-merge vectors of iterable values.
- function pmerge (v, j=true)

  Parallel-merge vectors of iterable values.
- function reverse (v)

  Reverse the elements of an iterable value.
- function qsort (v, r=false)

  Sort the numeric or string elements of a vector using quick sort.
- function qsort2 (v, d=0, r=false, s=true)

  Hierarchically sort all elements of a vector using quick sort.
- function strip (v, mv=empty_v)

  Strip all matching values from an iterable value.
- function eappend (nv, v, r=true, j=true, l=true)

  Append a value to each element of an iterable value.
- function insert (nv, v, i=0, mv, mi=0, s=true, si)

  Insert a new value into an iterable value.
- function delete (v, i, mv, mc=1, s=true, si)

  Delete elements from an iterable value.
- function unique (v)

  Return the unique elements of an iterable value.

7.7.1 Detailed Description

Mathematical primitive functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

Note

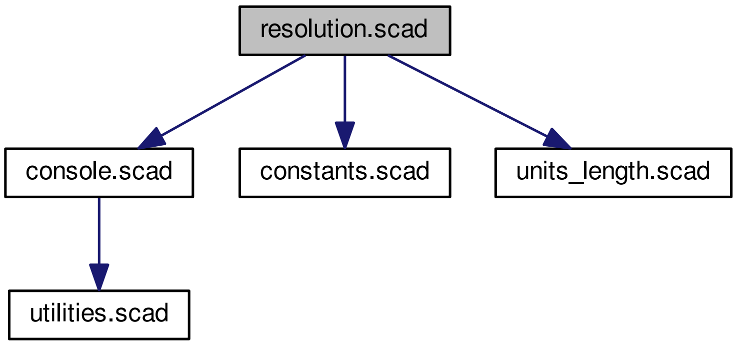
Include this library file using the **include** statement.

7.8 resolution.scad File Reference

Arc rendering resolution abstraction.

```
#include <console.scad>
#include <constants.scad>
#include <units_length.scad>
```

Include dependency graph for resolution.scad:



Functions

- function resolution_fn (radius)

Return facets number for the given arc radius.

- function resolution_fs ()

  Return minimum facets size.
- function resolution_fa (radius)

  Return the minimum facets angle.
- function resolution_reduced ()

  Return the radius at which arc resolution will begin to degrade.
- module resolution_info (radius)

  Echo resolution information to the console for given radius.
- function resolution_facets (radius)

  Return facet count used to render a radius.
- function resolution_facetsv (radius)

  Return facet count used to render a radius as vector triple.

Variables

- $resolution_mode = "fast"

  <string> Global special variable that configures the arc resolution mode.
- $resolution_value = 0

  <decimal> Global special variable for modes that use custom resolutions.

7.8.1 Detailed Description

Arc rendering resolution abstraction.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

Note

Include this library file using the **include** statement.

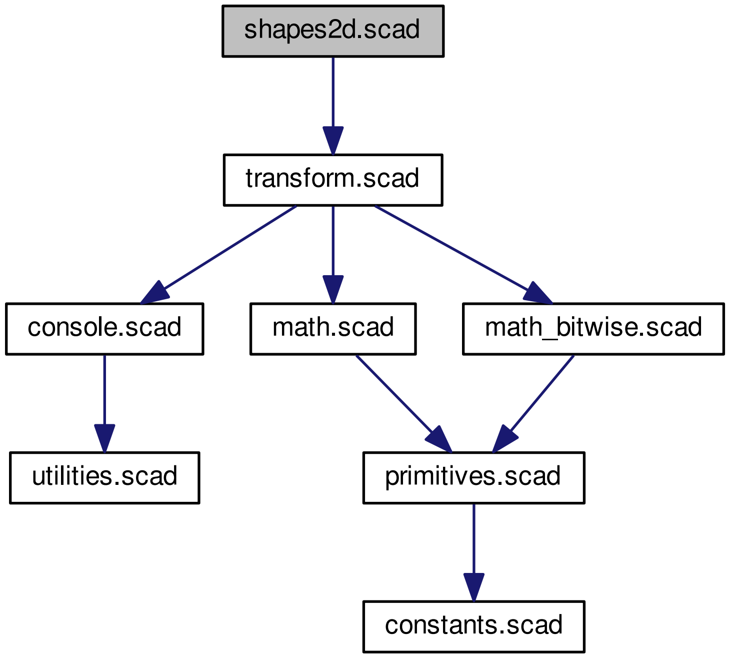
Todo Review model for accuracy.

7.9 shapes2d.scad File Reference

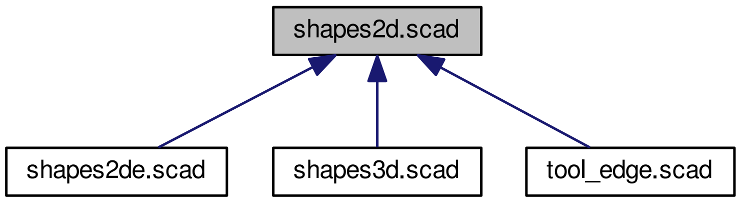
Two-dimensional basic shapes.

`#include <transform.scad>`

Include dependency graph for shapes2d.scad:



This graph shows which files directly or indirectly include this file:



Functions

- module rectangle (size, vr, vrm=0, center=false)

    A rectangle with edge, fillet, and/or chamfer corners.
- module rectangle_c (size, core, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)

    A rectangle with a removed rectangular core.
- module rhombus (size, vr, center=false)

    A rhombus.
- module triangle_ppp (v1, v2, v3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A general triangle specified by three vertices.
- module triangle_vp (v, vr, centroid=false, incenter=false)

    A general triangle specified by a vector of its three vertices.
- module triangle_lll (s1, s2, s3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A general triangle specified by its three side lengths.
- module triangle_vl (v, vr, centroid=false, incenter=false)

    A general triangle specified by a vector of its three side lengths.
- module triangle_vl_c (vs, vc, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false)

    A general triangle specified by its sides with a removed triangular core.
- module triangle_lal (s1, a, s2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A general triangle specified by two sides and the included angle.
- module triangle_ala (a1, s, a2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A general triangle specified by a side and two adjacent angles.
- module triangle_aal (a1, a2, s, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A general triangle specified by a side, one adjacent angle and the opposite angle.
- module triangle_ll (x, y, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A right-angled triangle specified by its opposite and adjacent side lengths.
- module triangle_la (x, y, aa, oa, vr, v1r, v2r, v3r, centroid=false, incenter=false)

    A right-angled triangle specified by a side length and an angle.
- module ngon (n, r, vr)

    An n-sided equiangular/equilateral regular polygon.
- module ellipse (size)

    An ellipse.
- module ellipse_c (size, core, t, co, cr=0)

    An ellipse with a removed elliptical core.
- module ellipse_s (size, a1=0, a2=0)

    An ellipse sector.
- module ellipse_cs (size, core, t, a1=0, a2=0, co, cr=0)

    A sector of an ellipse with a removed elliptical core.
- module star2d (size, n=5, vr)

    A two dimensional star.

7.9.1 Detailed Description

Two-dimensional basic shapes.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

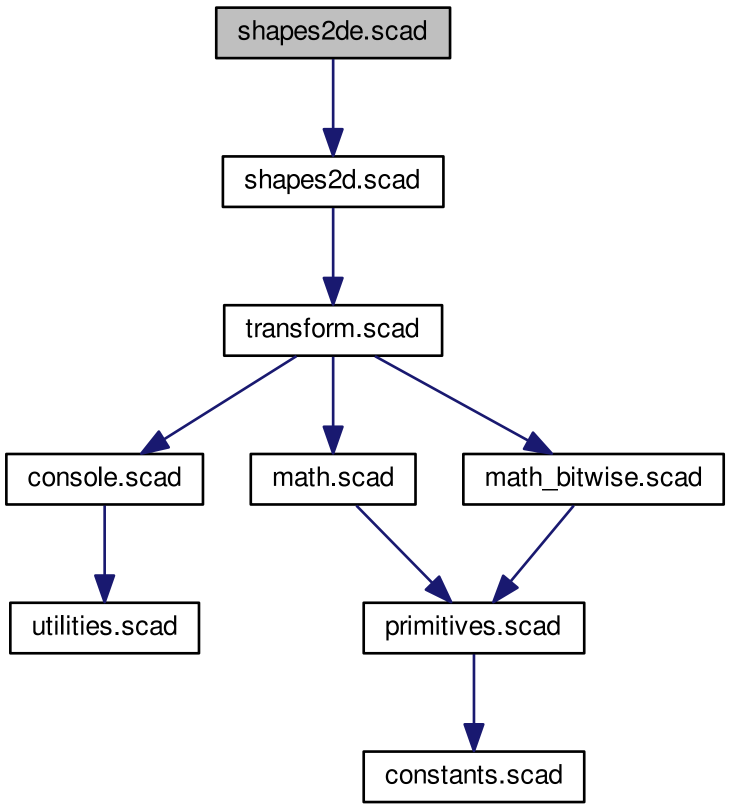
The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

7.10 shapes2de.scad File Reference

Linearly extruded two-dimensional basic shapes.

`#include <shapes2d.scad>`
Include dependency graph for shapes2de.scad:



Functions

- module erectangle (size, h, vr, vrm=0, center=false)
  An extruded rectangle with edge, fillet, and/or chamfer corners.
- module erectangle_c (size, core, h, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)
  An extruded rectangle with a removed rectangular core.
- module erhombus (size, h, vr, center=false)
  An extruded rhombus.
- module etriangle_ppp (v1, v2, v3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by three vertices.
- module etriangle_vp (v, h, vr, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by a vector of its three vertices.
- module etriangle_lll (s1, s2, s3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by its three side lengths.
- module etriangle_vl (v, h, vr, centroid=false, incenter=false, center=false)

An extruded general triangle specified by a vector of its three side lengths.

- module etriangle_vl_c (vs, vc, h, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false, center=false)

  A general triangle specified by its sides with a removed triangular core.
- module etriangle_lal (s1, a, s2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by two sides and the included angle.
- module etriangle_ala (a1, s, a2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a side and two adjacent angles.
- module etriangle_aal (a1, a2, s, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a side, one adjacent angle and the opposite angle.
- module etriangle_ll (x, y, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded right-angled triangle specified by its opposite and adjacent side lengths.
- module etriangle_la (x, y, aa, oa, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded right-angled triangle specified by a side length and an angle.
- module engon (n, r, h, vr, center=false)

  An extruded n-sided equiangular/equilateral regular polygon.
- module eellipse (size, h, center=false)

  An extruded ellipse.
- module eellipse_c (size, core, h, t, co, cr=0, center=false)

  An extruded ellipse with a removed elliptical core.
- module eellipse_s (size, h, a1=0, a2=0, center=false)

  An extruded ellipse sector.
- module eellipse_cs (size, core, h, t, a1=0, a2=0, co, cr=0, center=false)

  An extruded sector of an ellipse with a removed elliptical core.
- module estar2d (size, h, n=5, vr, center=false)

  An extruded two dimensional star.

7.10.1 Detailed Description

Linearly extruded two-dimensional basic shapes.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

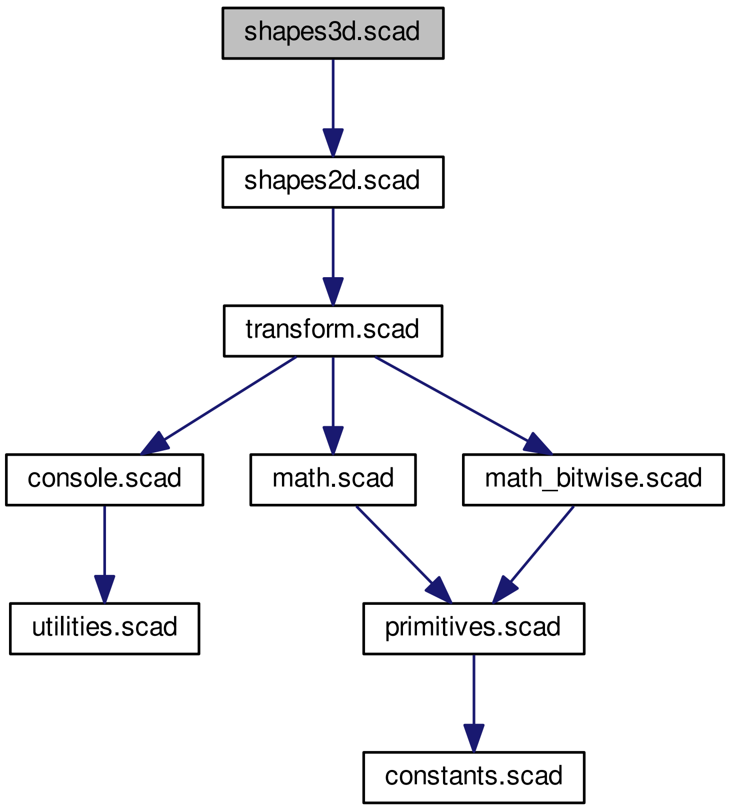
You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

7.11 shapes3d.scad File Reference

Three-dimensional basic shapes.

```
#include <shapes2d.scad>
```

Include dependency graph for shapes3d.scad:



Functions

- module cone (r, h, d, vr, vr1, vr2)

    A cone.
- module cuboid (size, vr, vrm=0, center=false)

    A cuboid with edge, fillet, or chamfer corners.
- module ellipsoid (size)

    An ellipsoid.
- module ellipsoid_s (size, a1=0, a2=0)

    A sector of an ellipsoid.
- module tetrahedron (size, center=false)

    A pyramid with trilateral base formed by four equilateral triangles.

- module pyramid_q (size, center=false)

  A pyramid with quadrilateral base.
- module star3d (size, n=5, half=false)

  A three dimensional star.
- module torus_rp (size, core, r, l, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, pa=0, ra=360, m=255, center=false, profile=false)

  A rectangular cross-sectional profile revolved about the z-axis.
- module torus_tp (size, core, r, l, co, cr=0, vr, vr1, vr2, pa=0, ra=360, m=255, centroid=false, incenter=false, profile=false,)

  A triangular cross-sectional profile revolved about the z-axis.
- module torus_ep (size, core, r, l, t, a1=0, a2=0, co, cr=0, pa=0, ra=360, m=255, profile=false)

  An elliptical cross-sectional profile revolved about the z-axis.

7.11.1 Detailed Description

Three-dimensional basic shapes.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

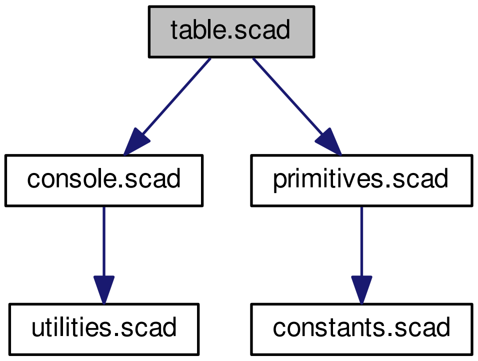
Todo Complete rounded cylinder.

7.12 table.scad File Reference

Data table encoding and lookup.

```
#include <console.scad>
#include <primitives.scad>
```

Include dependency graph for table.scad:



Functions

- function table_get_row_idx (rows, row_id)

  Get the index for a table row identifier.
- function table_get_row (rows, row_id)

  Get the row for a table row identifier.
- function table_get_col_idx (cols, col_id)

  Get the index for a table column identifier.
- function table_get_col (cols, col_id)

  Get the column for a table column identifier.
- function table_get (rows, cols, row_id, col_id)

  Get the value for a table row and column identifier.
- function table_get_row_cols (rows, cols, col_id)

  Form a vector from the specified column of each table row.
- function table_get_row_ids (rows)

  Form a vector of each table row identifier.
- function table_exists (rows, cols, row_id, col_id)

  Test the existence of a table row and column identifier.
- function table_size (rows, cols)

  Get the size of a table.
- module table_check (rows, cols, verbose=false)

  Perform some basic validation/checks on a table.
- module table_dump (rows, cols, rows_sel, cols_sel, number=true)

  Dump a table to the console.
- function table_copy (rows, cols, rows_sel, cols_sel)

  Create a copy of select rows and columns of a table.
- function table_sum (rows, cols, rows_sel, cols_sel)

  Sum select rows and columns of a table.

7.12.1 Detailed Description

Data table encoding and lookup.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

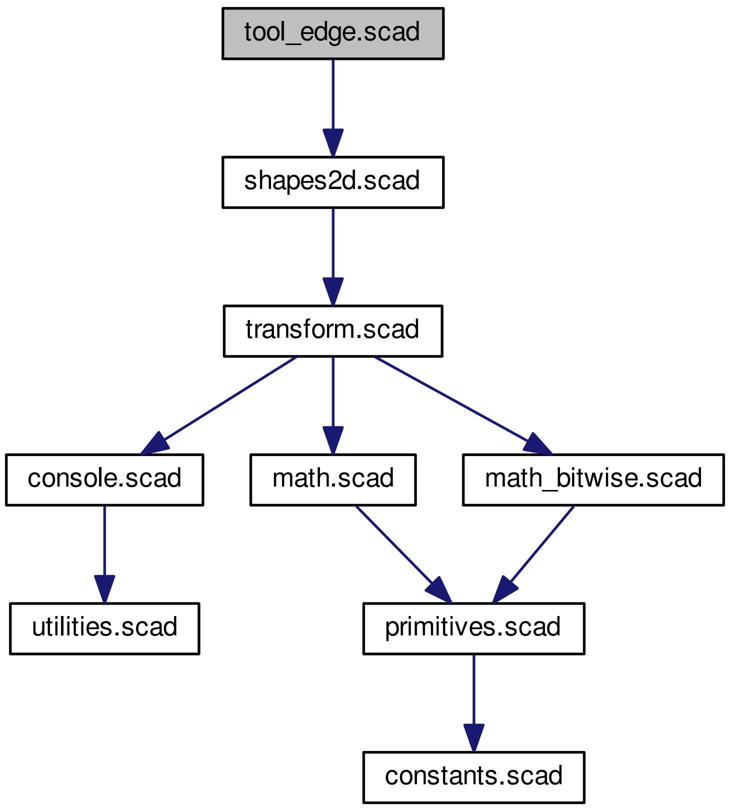
You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

7.13 tool_edge.scad File Reference

Shape edge finishing tools.

```
#include <shapes2d.scad>
```

Include dependency graph for tool_edge.scad:



Functions

- module edge_profile_r (r, p=0, f=1, a=90,)

    A 2D edge-finish profile specified by intersection radius.
- module edge_add_r (r, l=1, p=0, f=1, m=3, ba=45, a1=0, a2=90, center=false)

    A 3D edge-finish additive shape specified by intersection radius.

7.13.1 Detailed Description

Shape edge finishing tools.

Author

Roy Allen Sutton

Date

2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

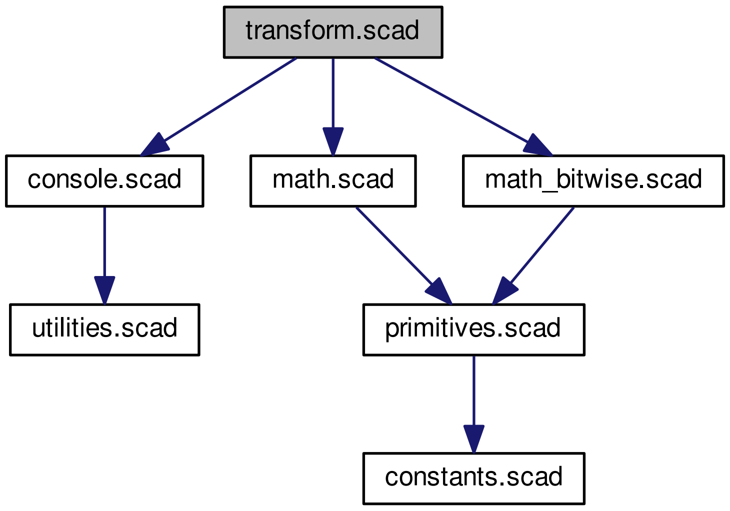
You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.↩ gnu.org/licenses/.

7.14 transform.scad File Reference

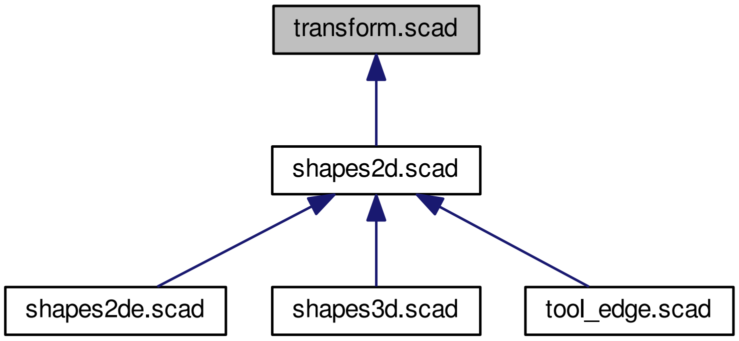
Shape transformation functions.

```
#include <console.scad>
#include <math.scad>
#include <math_bitwise.scad>
```

Include dependency graph for transform.scad:



This graph shows which files directly or indirectly include this file:



Functions

- module st_rotate_extrude (r, pa=0, ra=360, profile=false)

  Revolve the 2D shape about the z-axis.
- module st_rotate_extrude_elongate (r, l, pa=0, ra=360, m=255, profile=false)

  Revolve the 2D shape about the z-axis with linear elongation.
- module st_linear_extrude_scale (h, center=false)

  Linearly extrude 2D shape with extrusion upper and lower scaling.
- module st_radial_copy (n, r=1, angle=true, move=false)

  Distribute copies of a 2D or 3D shape equally about a z-axis radius.
- module st_cartesian_copy (grid, incr, copy=1, center=false)

  Distribute copies of 2D or 3D shapes about Cartesian grid.

7.14.1 Detailed Description

Shape transformation functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

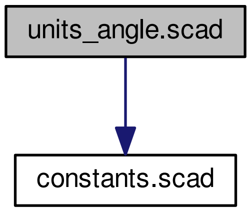
You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.↩ gnu.org/licenses/.

7.15 units_angle.scad File Reference

Angle units and conversions.

`#include <constants.scad>`

Include dependency graph for units_angle.scad:



Functions

- function unit_angle_name (units=base_unit_angle)

  Return the name of the given angle `unit` identifier.
- function convert_angle (angle, from=base_unit_angle, to=base_unit_angle)

  Convert the `angle` from `from` units to `to` units.

Variables

- base_unit_angle = "d"

  <string> Base unit for angle measurements.

7.15.1 Detailed Description

Angle units and conversions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

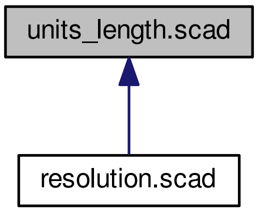
Note

Include this library file using the **include** statement.

7.16 units_length.scad File Reference

Length units and conversions.

This graph shows which files directly or indirectly include this file:



Functions

- function unit_length_name (units=base_unit_length, d=1, w=false)

  Return the name of the given `unit` identifier with dimension.
- function convert_length (value, from=base_unit_length, to=base_unit_length, d=1)

  Convert the `value` from `from` units to `to` units with dimensions.

Variables

- base_unit_length = "mm"

  <string> Base unit for length measurements.

7.16.1 Detailed Description

Length units and conversions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

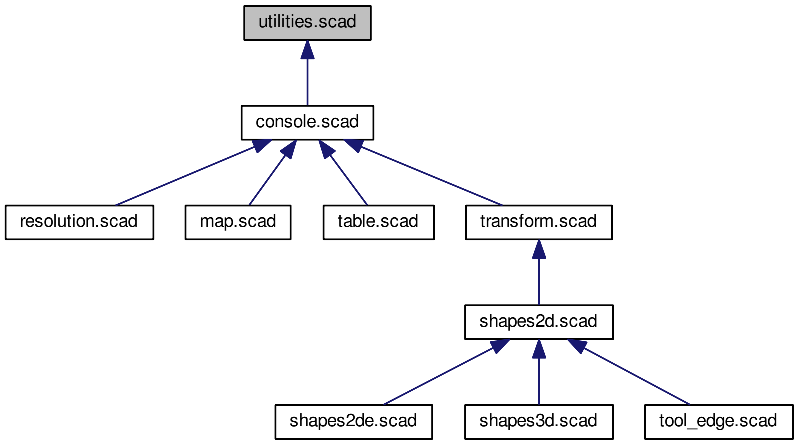
Note

Include this library file using the **include** statement.

7.17 utilities.scad File Reference

Miscellaneous utilities.

This graph shows which files directly or indirectly include this file:



Functions

- function stack (b=0, t=0)

  Format the function call stack as a string.

7.17.1 Detailed Description

Miscellaneous utilities.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

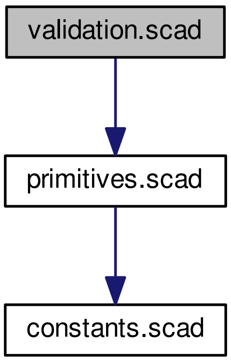
The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.↩ gnu.org/licenses/`.

7.18 validation.scad File Reference

Result validation functions.

`#include <primitives.scad>`
Include dependency graph for validation.scad:



Functions

- function validate (d, cv, t, ev, p=4, pf=false)

  Compare a computed test value with an known good result.

7.18.1 Detailed Description

Result validation functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

Index