SCaLE21x

Turn code into real-life stuff with OpenSCAD

Kyle Davis

Spot Callaway

About Kyle

- I work for AWS
  - This talk has nothing to do with my work
  - Talk to me at the AWS booth about Linux and Bottlerocket
- If I can code it, I will.
- CAD for less time
- 3D Printing for 4-ish years

About Tom "Spot" Callaway

- Principal Open Source Strategist, AWS
- 3D Printer & OpenSCAD for about 10 years

Your experience

- Raise your hands if you've:
    - Coding/programming:
        - Written code before
        - Written code for more than 2 years
        - Written code for more than 10 years
    - CAD:
        - Used CAD before
        - Used done parametric CAD work
        - Used OpenSCAD

Agenda:

- Where OpenSCAD fits
- Setup and Tour of OpenSCAD
- OpenSCAD as a language
- 2D Subsystem
- Geometric Booleans
- 3D Primitives
- Advanced Modelling

70 minutes of slides, 90 minutes of exercise, and a capstone project

CAD Landscape

- Commerical
  - Autodesk Fusion360
  - Dassault Systèmes SolidWorks
  - Trimble SketchUp
  - Autodesk TinkerCAD
- Open Source
  - FreeCAD *LGPL-2.0-or-later* (freecadweb.org)
    - Ondsel (ondsel.com)
  - OpenSCAD *GPL-2.0-or-later* (openscad.org)
  - Python SDF *MIT* (github.com/fogleman/sdf)

Design Process

- Direct Modelling
- Parametric Design
    - Antoni Gaudí

Flickr / stuart madeley / cc-by-nc-sa-20

Flickr / michael levine clark /c c-by-nc-nd-20

You're not drawing.

- Create a program that produces the design
- Specify the parameters of the program

The remote control

Change the spacing between buttons

- Design Modelling
    - Click each button
    - Change the width of the button by dragging
    - Adjust the spacing between each button
- Parametric Modelling
    - Change the button spacing variable

Draw an organic shape?

OpenSCAD Setup

- https://openscad.org
- Version 2021.01

OpenSCAD IDE Walk Though

OpenSCAD as a language

Why language first

- Common examples favour brevity
- Bad software
- Misplaced challenge

Types and variables

Declaring variables

- Simplest form: `[identifier]= [value];`
- Type inference by syntax
- No user-defined types

Type examples

Type	Example

Number	`y= 4;`
	`y= 4.25;`

Boolean	`smooth= true;`
	`smooth= false;`

String	`foo= "Bar";`

Vectors
'lists'
	`l= [10,20,30];`
	`l_n = [ [10,10], false, “:)” ];`

Range	`r= [0:10];`
	`r_skip= [0:2:10];`

Exercise 1 (5 Minutes)

Create five variables with compatible values, one for each OpenSCAD type.

Name them `my_<type>`.

Make sure it compiles

- Types are `number`, `boolean`, `string`, `vectors`, and `ranges`
- Strings use double quotes (`"`)
- Vectors & ranges start/end with square brackets (`[` / `]`)
- Vectors are delimited by commas (`,`)
- Ranges are delimited by colons (`:`)

Vector access

Access vector elements with zero-based bracket notation

Code:

```
v= ["a", "b", "c", [ 1, 2, 3] ];
echo(v[0]);
echo(v[2]);
echo(v[3][1]);
```

Console:

```
ECHO: "a"
ECHO: "c"
ECHO: 2
```

Cartesian vectors

Code:

```
v= [10, 20, 30];
echo(v.x);
echo(v.y);
echo(v.z);
```

Console:

```
ECHO: 10
ECHO: 20
ECHO: 30
```

⚠️ Scope ⚠️

Last declaration wins:

a variable keeps a single value during it's entire life time

🚫 iterative variables

This will not work 🤯

```
x= x + 1;
```

Scope Example

Code:

```
x= 5;
echo(x);
x= 6;
```

Console:

```
WARNING: x was assigned on line 1 but was overwritten in file exercises.scad, line 3
ECHO: 6
```

What creates a scope?

Code:

```
foo = 100;
if (true) {
    if (true) {
        bar= 200;
        echo(foo);
    }
    echo(bar);
}
```

Console:

```
ECHO: 100
WARNING: Ignoring unknown variable 'bar' in file exercises.scad, line 7
ECHO: undef
```

What about second declaration & scope?

Code:

```
foo = 100;
if (true) {
    foo = 10;
    bar = 500;
    echo(foo);
}
echo(foo);
echo(bar);
```

Console:

```
ECHO: 10
ECHO: 100
WARNING: Ignoring unknown variable 'bar' in file exercises.scad, line 8
ECHO: undef
```

Special Variables

- Start with `$`
- Built-in:
  - `$fn` (`$fs`/`$fa`)
  - `$preview`
  - `$t`
  - `$vpr`/ `$vpt` / `$vpf` / `$vpd`
- User-defined

Exercise 2 (5 Minutes)

Create a script (with no warnings on errors) that outputs:

```
ECHO: 1
ECHO: 2
ECHO: 3
```

You can use:

- variable declarations of `a_number = ...`
- `if (true) { ... }` blocks
- `echo(a_number);`

(with a minimal number of lines)

Exercise 2 Solution

```
a_number= 1;
echo(a_number);

if (true) {
    a_number= 2;
    echo(a_number);
}

if (true) {
    a_number= 3;
    echo(a_number);
}
```

(This is a bad example: more later!)

Operators & Expressions

Math

- Add: `two= 1 + 1;`
- Subtract: `three= 4 - 1;`
- Multiply: `four= 2 * 2;`
- Divide: `five= 10 / 2;`
- Modulo: `six= 100 % 94;`
- Exponent: `seven_ish= 2.64575 ^ 2;`
- Parens: `(` `)` for grouping

Relational

Evaluate to `true` or `false`

- less than `<`
- less or equal `<=`
- equal `==`
- not equal `!=`
- greater or equal `>=`
- greater than `>`

Logical & Conditional

- Evaluates two booleans and produces a boolean.
  - logical AND `&&`
  - logical OR `||`
- Evaluates a boolean and produces a boolean
  - logical NOT `!`
- Conditional `?`
  - *[relational comparison]* `?` *[when true]* `:` *[when false]*
  - Example: `a > 10 ? "bigger than 10" : "10 or smaller"`;

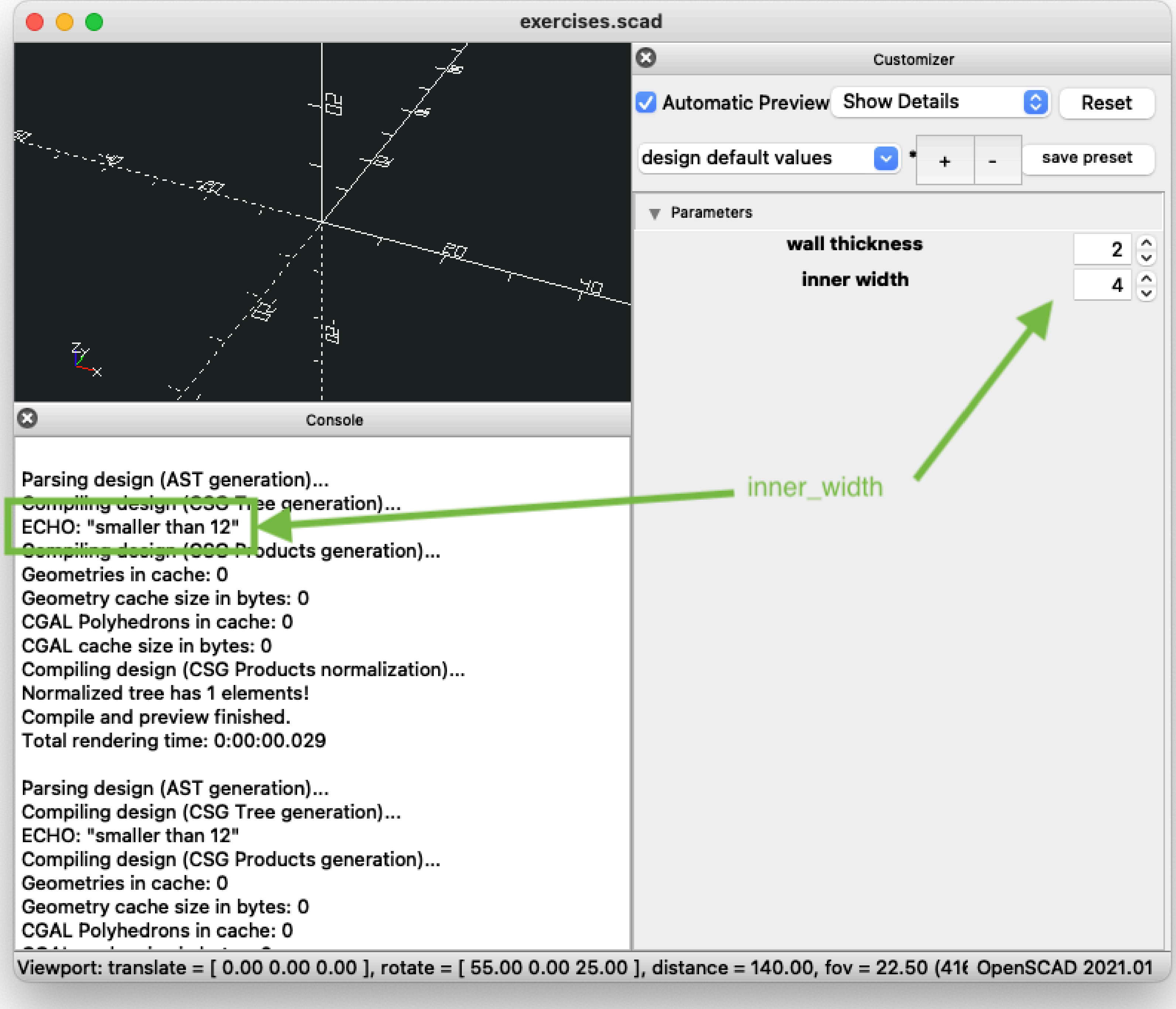
Exercise 3 (10 Minutes)

- Declare a variable named `wall_thickness` with a number value of your choice
- Declare a variable named `inner_width` with a number value of your choice
- Multiply `wall_thickness` by 2 and add `inner_width`
- Determine if the calculation from the previous step is greater than or equal to `12`
  - Use a conditional operator (`?`)
  - Use `echo` to write out a message describing the relationship of the calculation compared to `12`.

Exercise 3 Solution

```
wall_thickness= 2;
inner_width= 4;

echo(
    wall_thickness*2 + inner_width >= 12 ?
        "12 or bigger" :
        "smaller than 12"
);
```

Customizer 🪄



☑ Automatic Preview   Show Details

design default values *

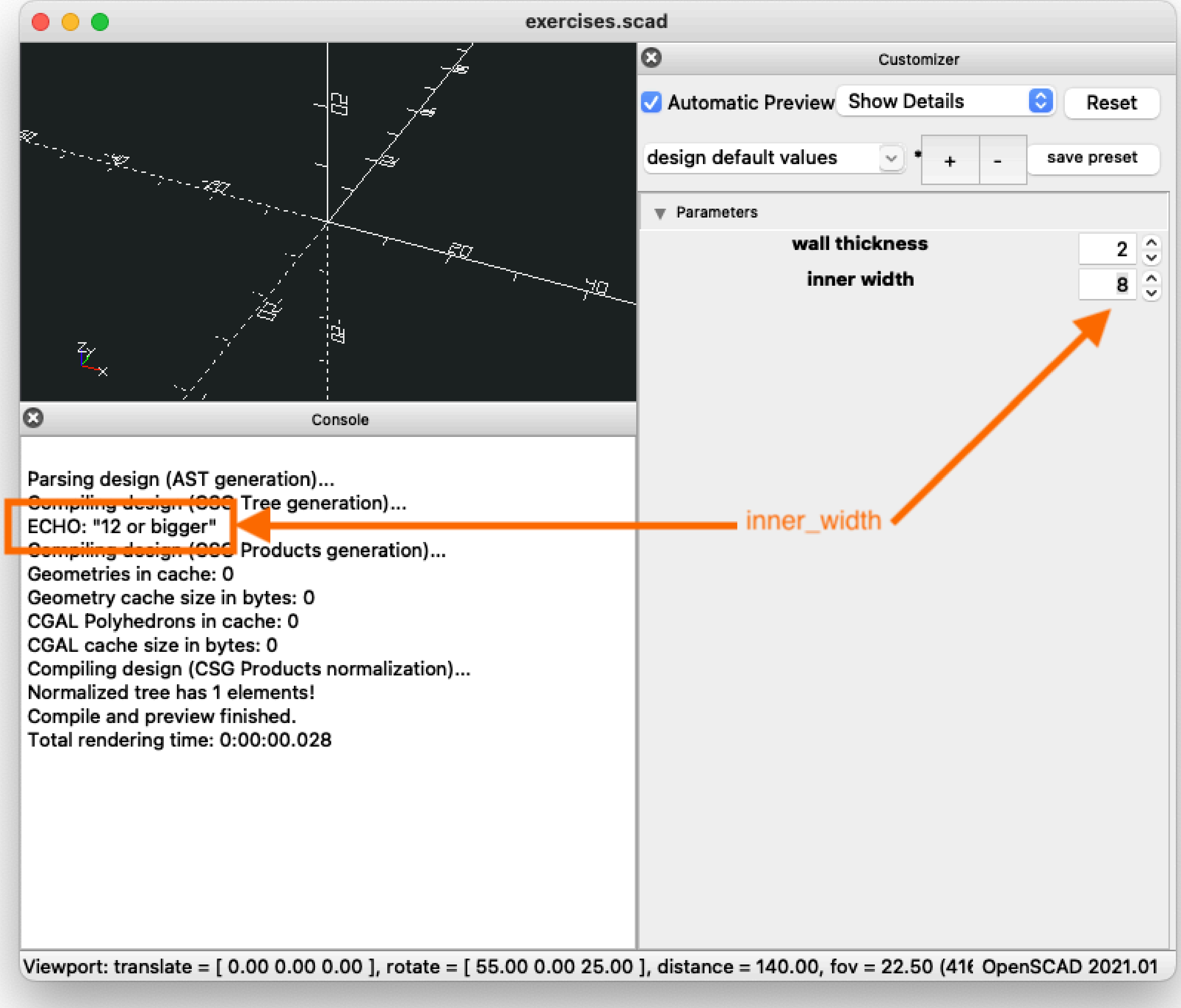
▼ Parameters

wall thickness 2

inner width 4

```
Parsing design (AST generation)...
Compiling design (CSG Tree generation)...
ECHO: "smaller than 12"
Compiling design (CSG Products generation)...
Geometries in cache: 0
Geometry cache size in bytes: 0
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized tree has 1 elements!
Compile and preview finished.
Total rendering time: 0:00:00.029

Parsing design (AST generation)...
Compiling design (CSG Tree generation)...
ECHO: "smaller than 12"
Compiling design (CSG Products generation)...
Geometries in cache: 0
Geometry cache size in bytes: 0
CGAL Polyhedrons in cache: 0
```

Viewport: translate = [0.00 0.00 0.00], rotate = [55.00 0.00 25.00], distance = 140.00, fov = 22.50 (41(OpenSCAD 2021.01



wall thickness 2

inner width 8

```
Parsing design (AST generation)...
Compiling design (CSG Tree generation)...
ECHO: "12 or bigger"
Compiling design (CSG Products generation)...
Geometries in cache: 0
Geometry cache size in bytes: 0
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized tree has 1 elements!
Compile and preview finished.
Total rendering time: 0:00:00.028
```

Viewport: translate = [0.00 0.00 0.00], rotate = [55.00 0.00 25.00], distance = 140.00, fov = 22.50 (41(OpenSCAD 2021.01

inner_width

Wait, what just happened?

- Interpolation of variables from the UI into the program
- Each change recalculates the the expressions
- Weird variable scope for the win!

From the command line

```
$ OpenSCAD -o - \
    --export-format asciistl \
    ./exercises.scad
ECHO: "smaller than 12"
Geometries in cache: 1
Geometry cache size in bytes: 0
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Total rendering time: 0:00:00.000
Current top level object is empty.
```

From the command line with customizer values

```
$ OpenSCAD -o - \
    --export-format asciistl \
    -D inner_width=10 \
    ./exercises.scad
ECHO: "12 or bigger"
Geometries in cache: 1
Geometry cache size in bytes: 0
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Total rendering time: 0:00:00.000
Current top level object is empty.
```

Exercise 4 (3 Minutes)

Try using the customizer for your script and see the output.

Modules & Functions

What's the difference?

- **Functions** & **Modules** accept zero or more parameters inside parents `(` / `)`
- **Functions** contain a single expressions and return a value.
  - Example: `function foo() = 1;`
- **Modules** produce geometry from zero or more children.
  - Example: `module foo();`

Parameters

- Used with `function` & `module`
- Always named when declared, but sometimes have default values.
  - Example: `module my_module(first, second);`
    Usage: `my_module(1,2);`
  - Example: `module my_module(first, second= 2);`
    Usage: `my_module(1);` or `my_module(1,3);`
- Parameters work like variables inside the scope of the expression or children.
  - Example: `function foo(bar= 5) = bar + 1;`

Functions

- Syntax: `function` *function_name* `(` *parameter(s)* `) =` *single expression* `;`
- Great for abstracting calculations.
- Everything must be in a single expression, no curly braces
- Can be declared before or after called.
- Can be redefined without warning nor error, last one wins.

Function Example

Code:

```
one = foo(1,2);

echo(one);

function foo(first, second) = [first, second];
function foo(first, second) = [second, first];
```

Console:

```
ECHO: [2, 1]
```

Functions with `let`

- Create variable-like abstractions in scope of a single function.
- Between the `=` and the start of the function's expression.
- Order is relevant and can be linearly dependent.
- Example: `function foo(a)= let(b= 1, c= b + 1) a + b + c;`

Exercise 5 (5 Minutes)

- Create a `function` that doubles a parameter then adds one.
  - E.g. pass in `3` and get a result of `7`
- Use `let` in the function to store the doubled parameter value.
- Pass in the values of `100` and `0.5` and `echo` the result.
- **Stretch goal:** Create a function that can double by default, but can optionally multiply by an arbitrary value.

Exercise 5 Solution

```
echo(mul_add_one(100)); // 201
echo(mul_add_one(0.5)); // 2
echo(mul_add_one(3, mul_by= 3)); // 10


function mul_add_one(n, mul_by= 2) =
    let(mul_ed = n * mul_by)
        mul_ed + 1;
```

Modules

- Syntax:
  `module` *module_name* `(` *parameter(s)* `)` *single child* `;`
- Syntax:
  `module` *module_name* `(` *parameter(s)* `) {` *expressions and/or multiple children* `}`
- Children are other modules.
- `echo` is a module!
  - (But it's a little special)
- Most modules are used to create/manipulate geometry.
- Modules **cannot** return anything.

Simple Module with Parameters

Code:

```
my_module("one");
my_module(first="one");
my_module(second="foo");

module my_module(first,second)
    echo(second);
```

Console:

```
ECHO: undef
ECHO: undef
ECHO: "foo"
```

Exercise 6 (5 Minutes)

- Create a `module` called `someone_says`
  - parameter called `name` that defaults to your name
  - parameter called `msg` without a default
- Invoke the module with the parameter `"hello"`
- When invoked, it should output:

```
ECHO: hello
ECHO: kyle
```

- Invoke the module and override the default for `name` with neighbours name and nothing set for `msg`
  - What happens?

Exercise 6 Solution

```
someone_says(msg= "hello");
// ECHO: hello
// ECHO: kyle
someone_says(name= "spot");
// ECHO: undef
// ECHO: spot

module someone_says(name= "kyle", msg) {
    echo(msg);
    echo(name);
}
```

Time to make geometry!

- Standard Library
  - 2D
  - 3D
- 2D geometry is represented as having a nominal `z`
  - only `x` and `y` are *real*
- Note: OpenSCAD is unit-less.

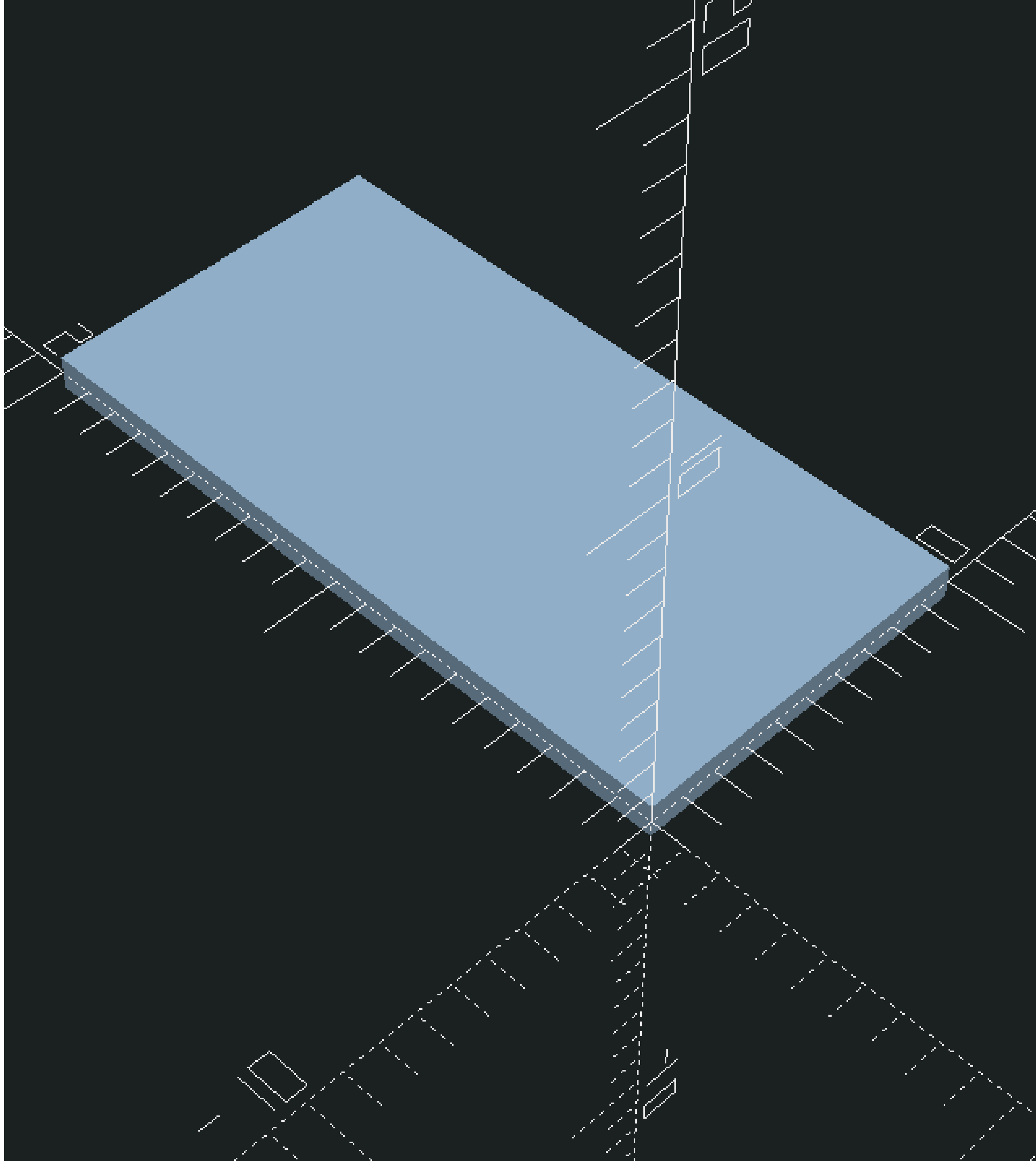
`square`

- Really a rectangle
- specify the `size` as a 2-element vector
    - remember the `[` and `]`!
- `center` optional
    - default: x and y in positive

`square` example

Code:

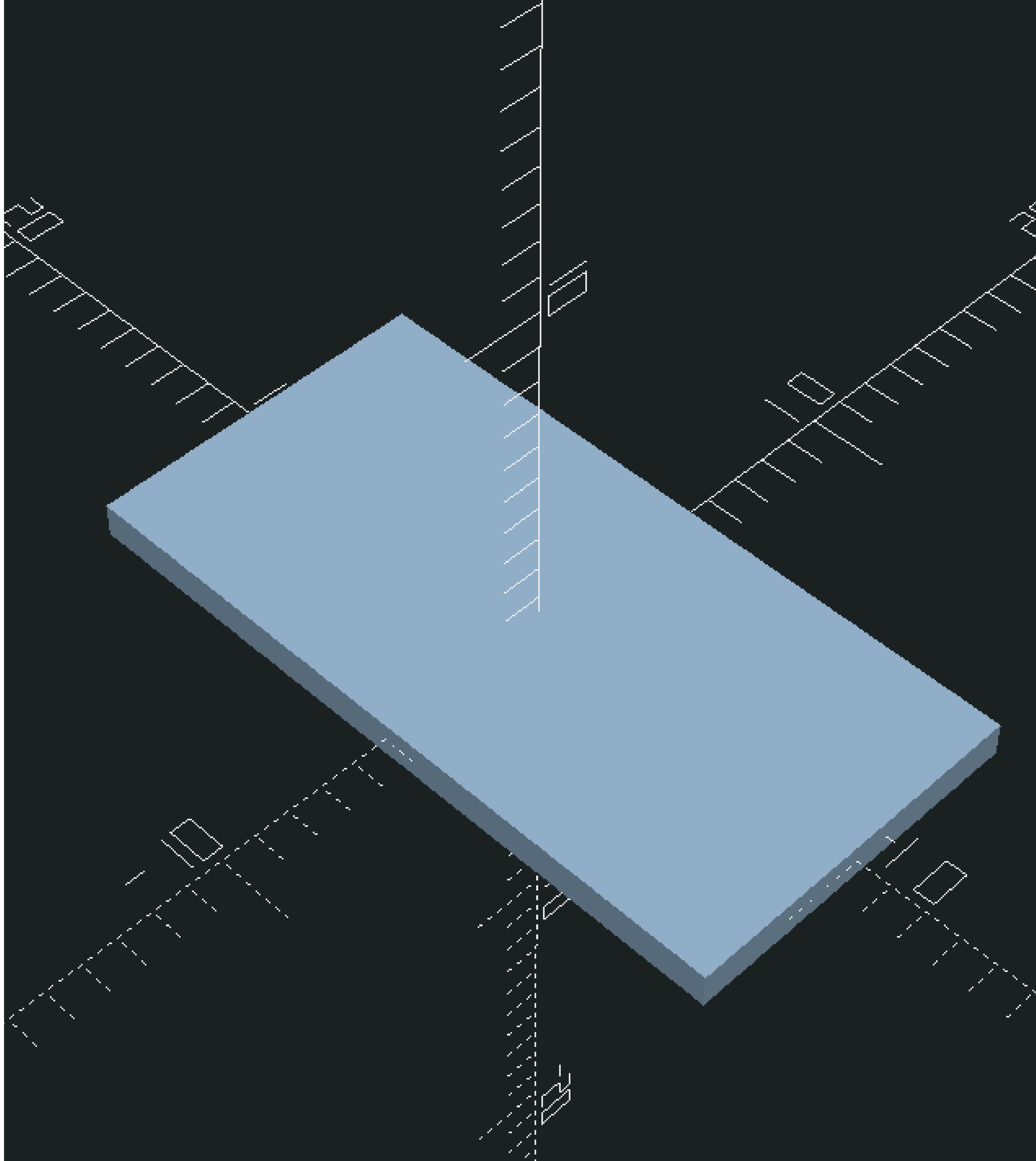
```
width= 10;
length= 20;

square([width, length]);
```



`square` example (center)

Code:

```
width= 10;
length= 20;

square(
    [width, length],
    center= true
);
```



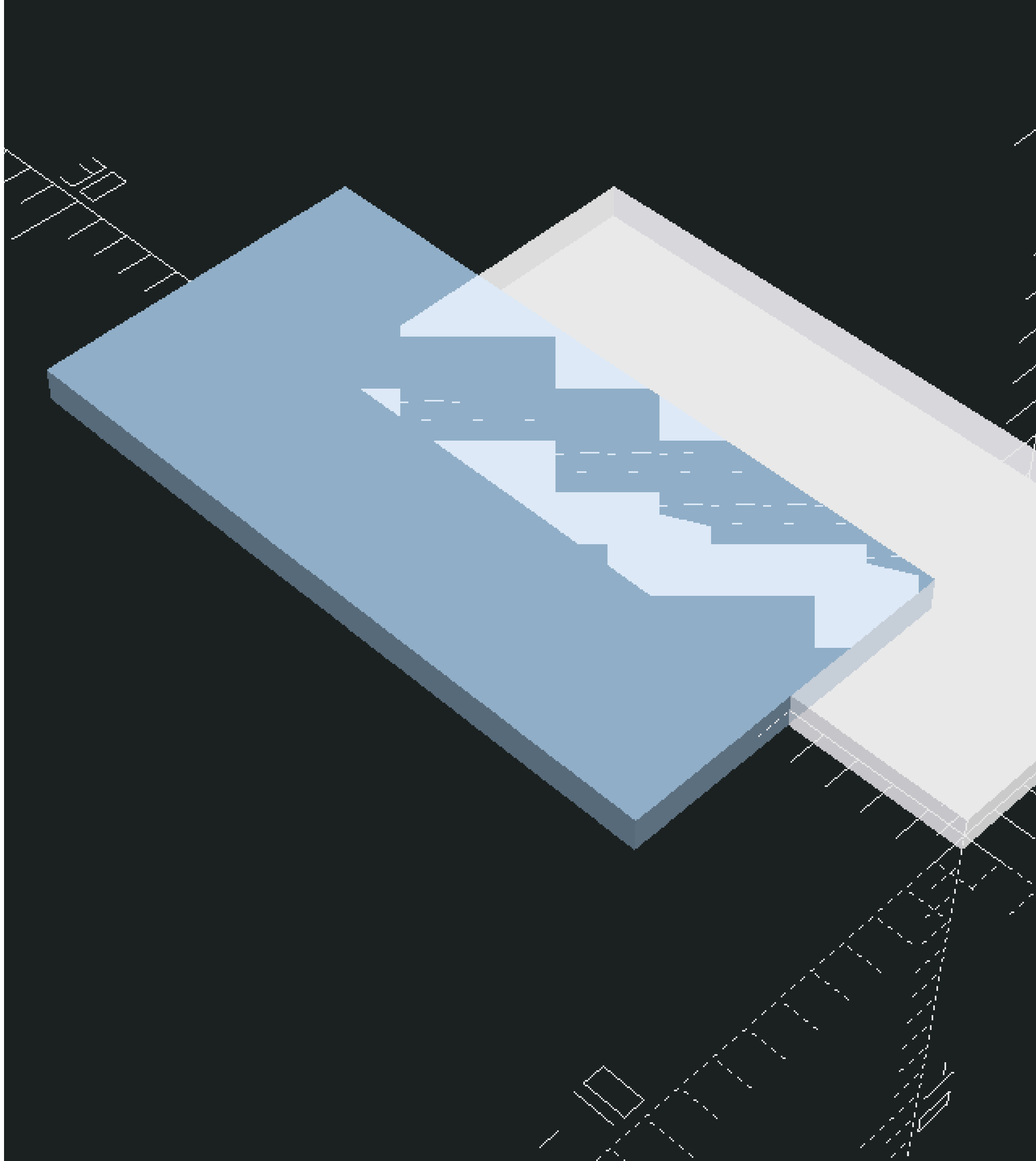
Moving with geometry with `translate`

- changes the position of child geometry along the `x`, `y` and `z` plane
- Accepts a 3-element vector for the position
- Can be nested

`translate` example

Code:

```
width= 10;
length= 20;

%my_square();

translate([-width / 2, length * 0.25])
    my_square();

module my_square()
    square([width, length]);
```



Exercise 7 (8 Minutes)

Start with:

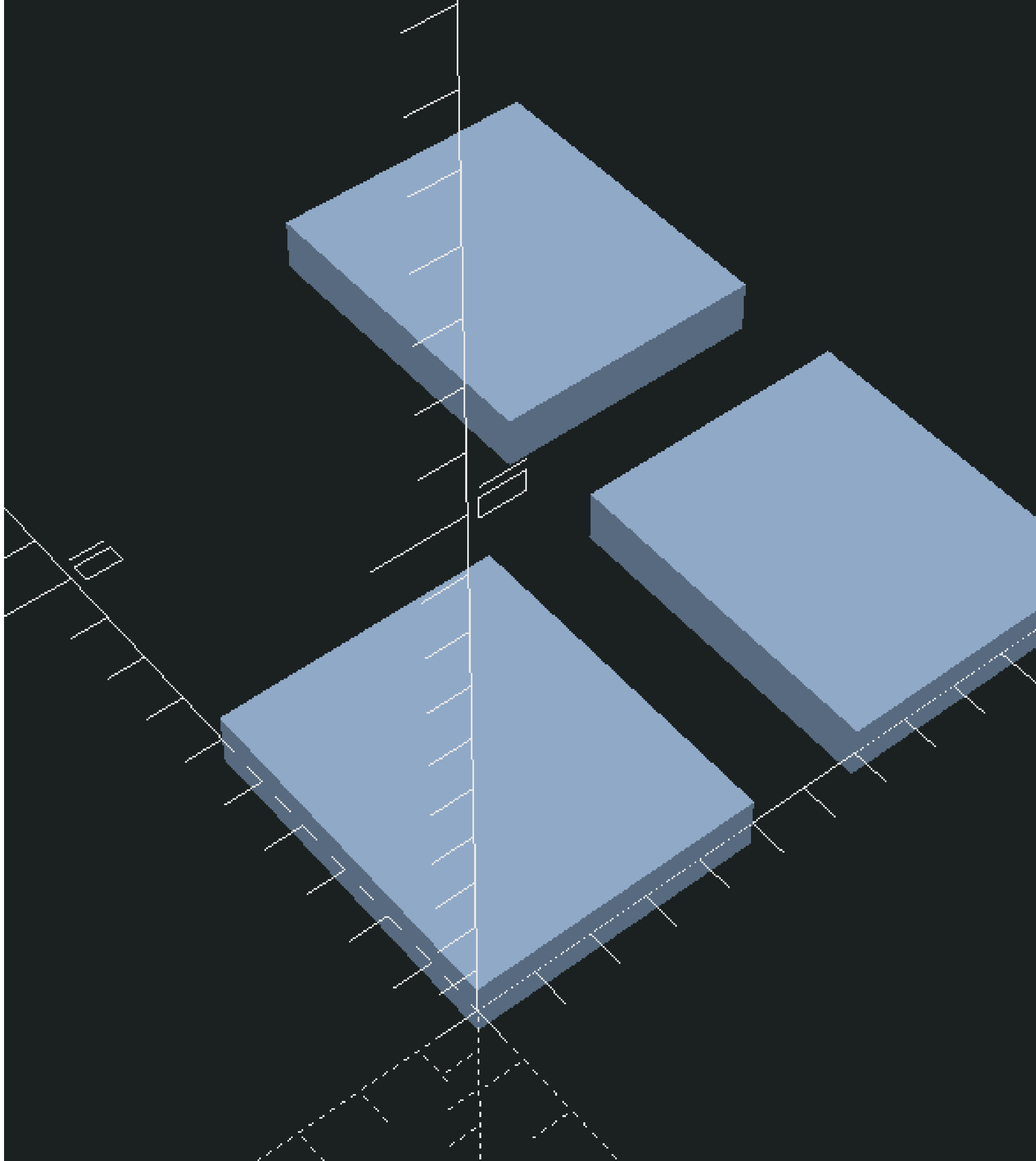
```
width= 5;
length= 6;
spacing= 2;
```

Create a module named `my_square`

Use `translate` position the 2 of the 3 rectangles

The spacing between the rectangles should be calculated with the variables

Stretch: Manipulate `width` / `length` / `spacing` with the customizer.



Exercise 7 Solution

```
width= 5;
length= 6;
spacing= 2;

my_square();

translate([width + spacing, 0])
    my_square();

translate([width + spacing, length + spacing])
    my_square();

/*Alternately:
translate([width + spacing, 0]) {
    my_square();
    translate([0, length + spacing])
        my_square();
}*/

module my_square()
    square([width, length]);
```

`for`

- `for (` *iterator variable* `=` *range* `)` *child*
- `for (` *iterator variable* `=` *range* `) {` *children* `}`
- Remember the `range` type?
  - `[` *start* `:` *end* `]`
  - `[` *start* `:` *increment* `:` *end* `]`

`for` example

```
for (i= [0 : 2])
    echo(i);
```

```
ECHO: 0
ECHO: 1
ECHO: 2
```

`for` example (increment)

```
for (i= [0 : 3 : 6])
    echo(i);
```

```
ECHO: 0
ECHO: 3
ECHO: 6
```

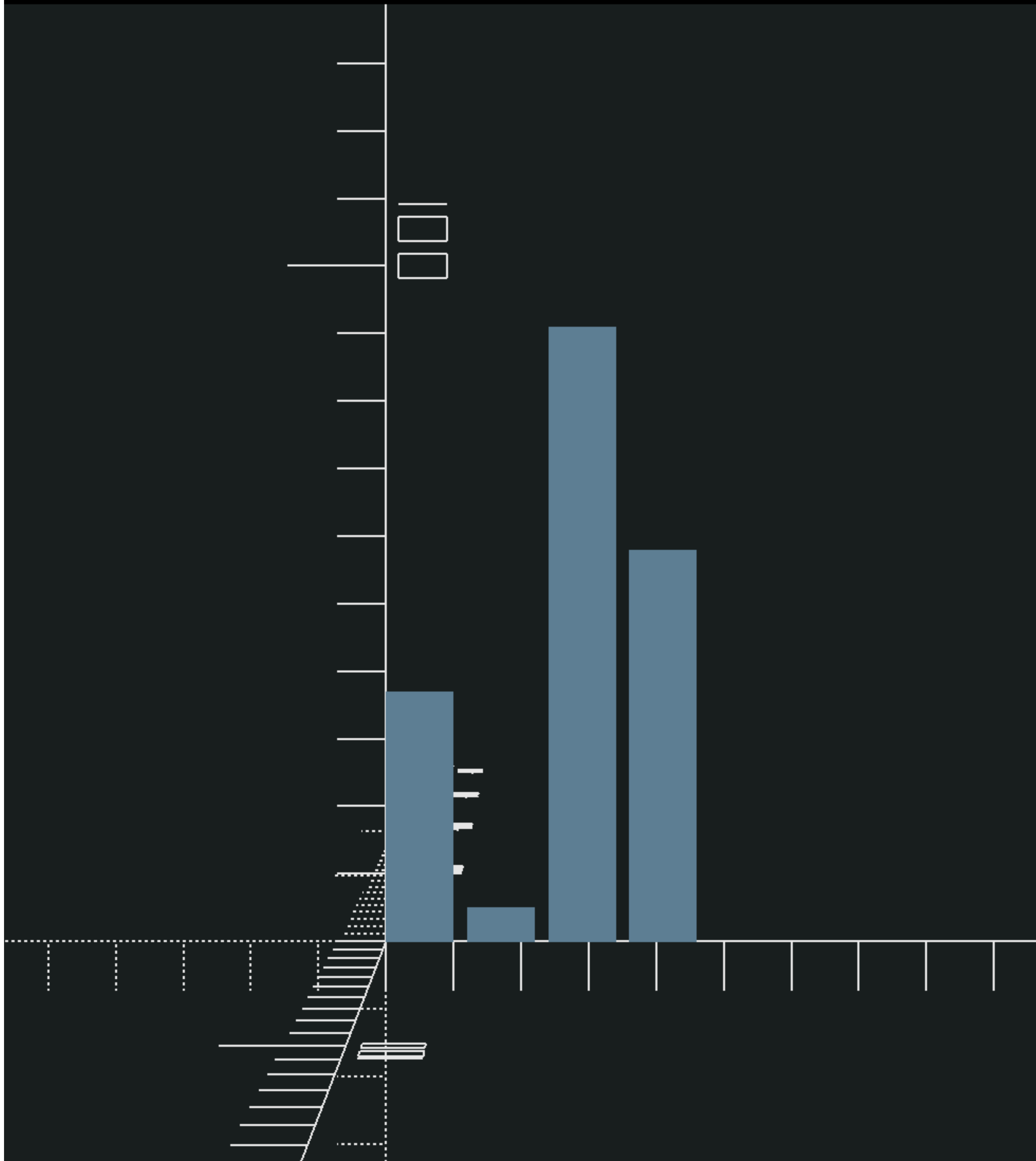
`for` and `len`

- `len` is a function that returns the length of a string or a vector.
- Useful for iterating over a vector.
- Remember *end* in a range is inclusive, so subtract 1 from `len` in `for`
- Example:

  ```
  my_v= ["a", "b", "c"];
  for (i= [0 : len(my_v)-1]) ...
  ```

Exercise 8
(12 minutes)

- Create a bar chart using `for`, `square`, and `translate`
- Start with `data= [37, 5, 91, 58];`
- Each bar should be 10 wide and the spacing between each bar is 2.
- **Stretch**: Change one element of `data` to a negative and make your bar chart render correctly.



Exercise 8 Solution

```
bar_width= 10;
bar_spacing= 2;
data= [37, 5, 91, 58];

for(i = [0 : len(data)-1])
    translate([(bar_width + bar_spacing) * i, 0])
        square([bar_width, data[i]]);
```

```
for(i = [0 : len(data)-1])
    translate([
        (bar_width + bar_spacing) * i,
        (data[i] > 0 ? 0 : data[i])
    ])
        square([bar_width, abs(data[i])]);
```

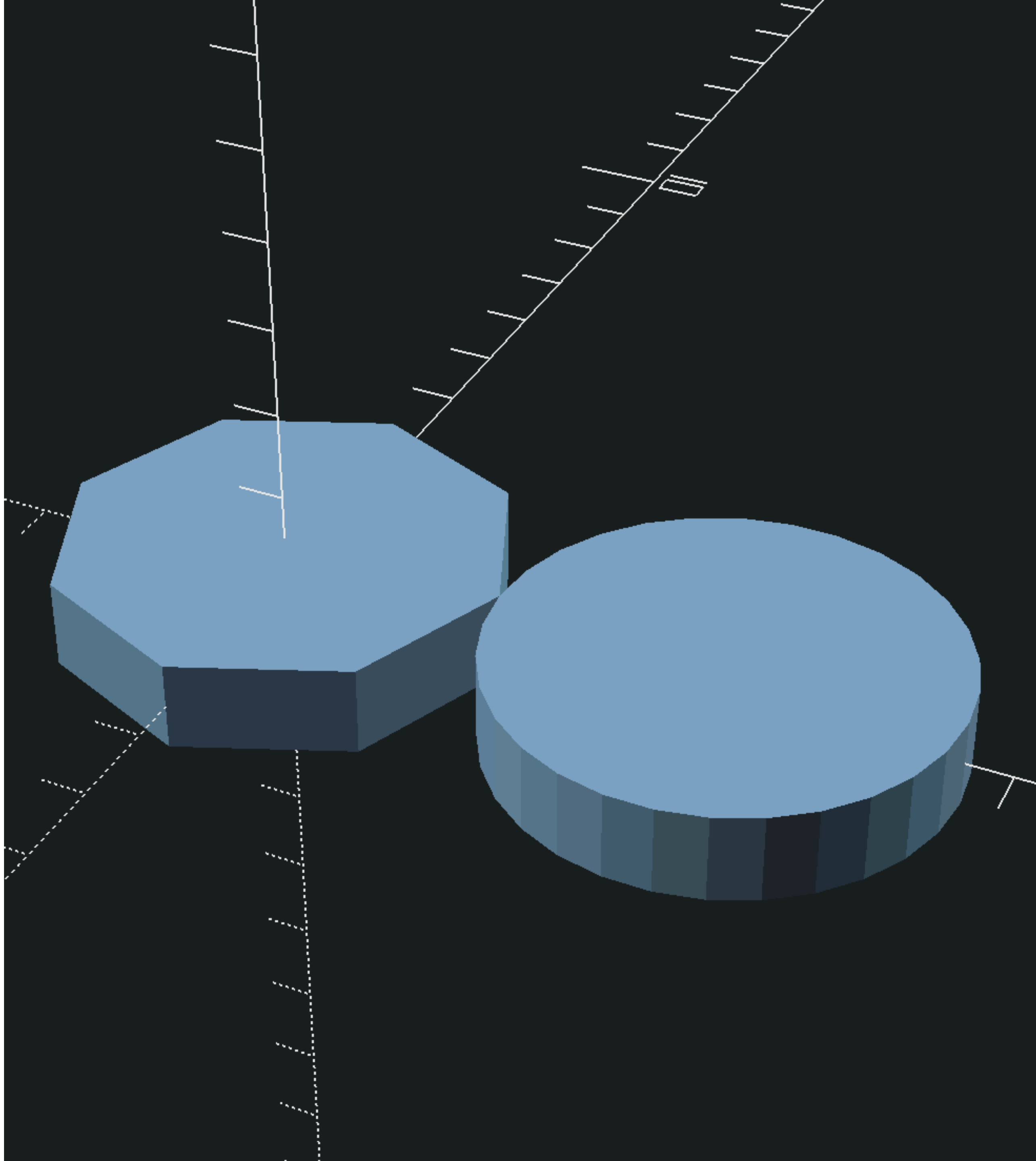
`circle`

- A 2D circle
- Always positioned on center point
- size by diameter (`d`) or radius (`r`)
- Special variable `$fn` for facets (smoothness)

`circle` example

Code:

```
d= 5;

circle(d= d);
translate([d, 0])
    circle(r= d/2, $fn= 30);
```



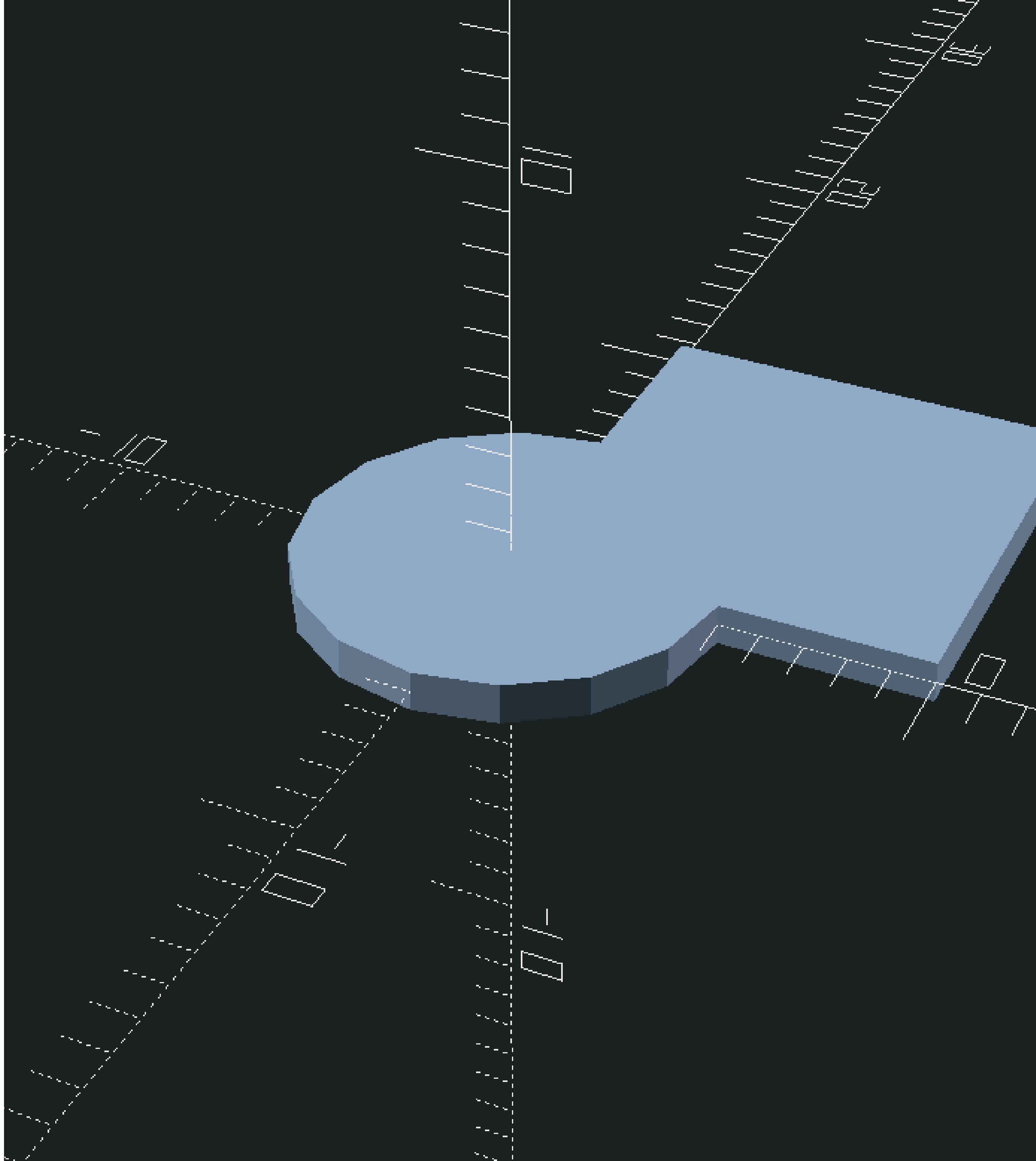
Constructive Solid Geometry

- Meshes vs "Holding Water"
- Complex shapes by combining simple objects with boolean operations
    - Unions
    - Differences
    - Intersections
- You can nest shapes and boolean operations to create any shape!

Unions

- Mostly implicit
- Explicit is occasionally useful

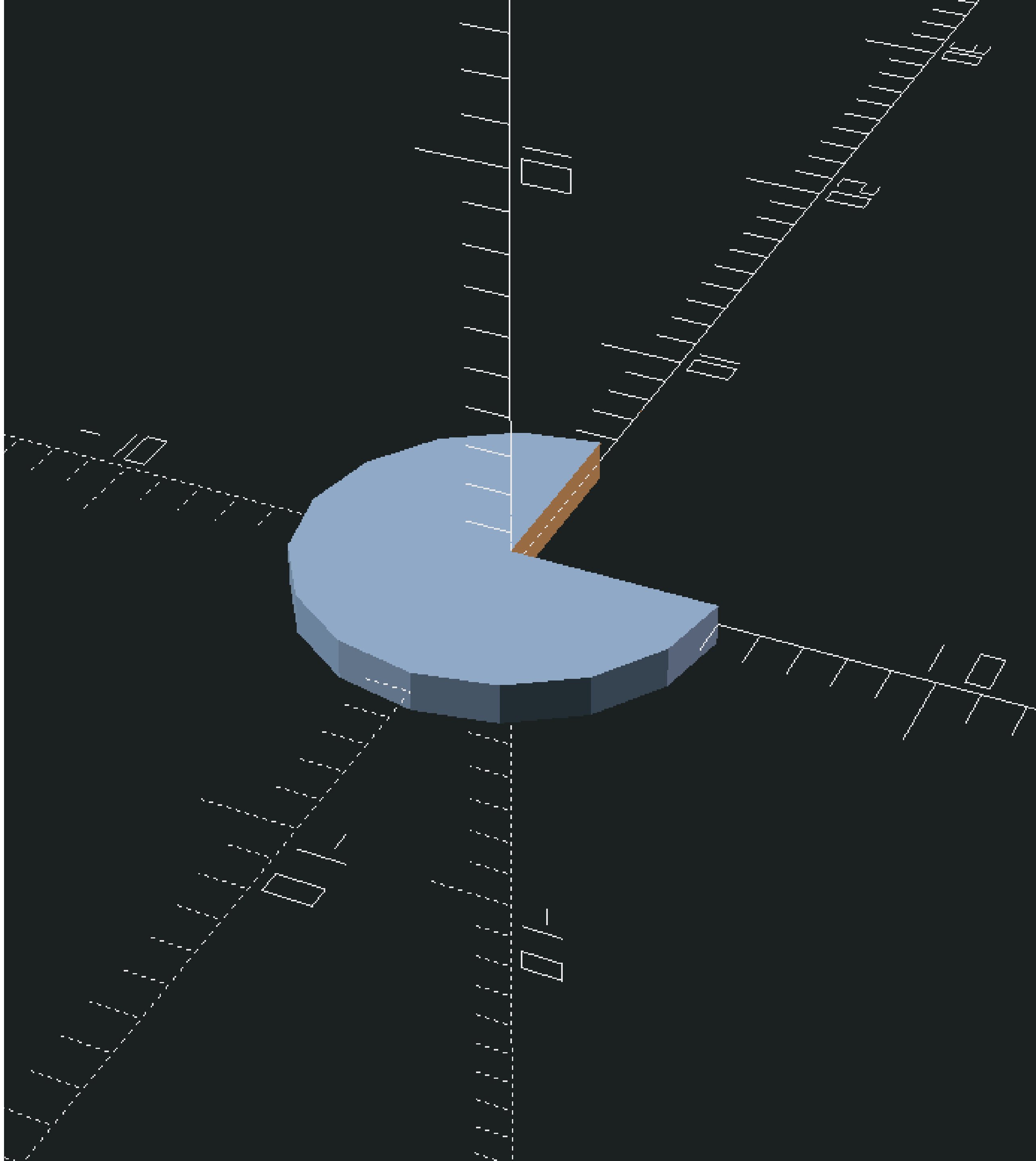
```
w= 10;

union() {
    circle(d= w);
    square([w, w]);
}
```



Difference

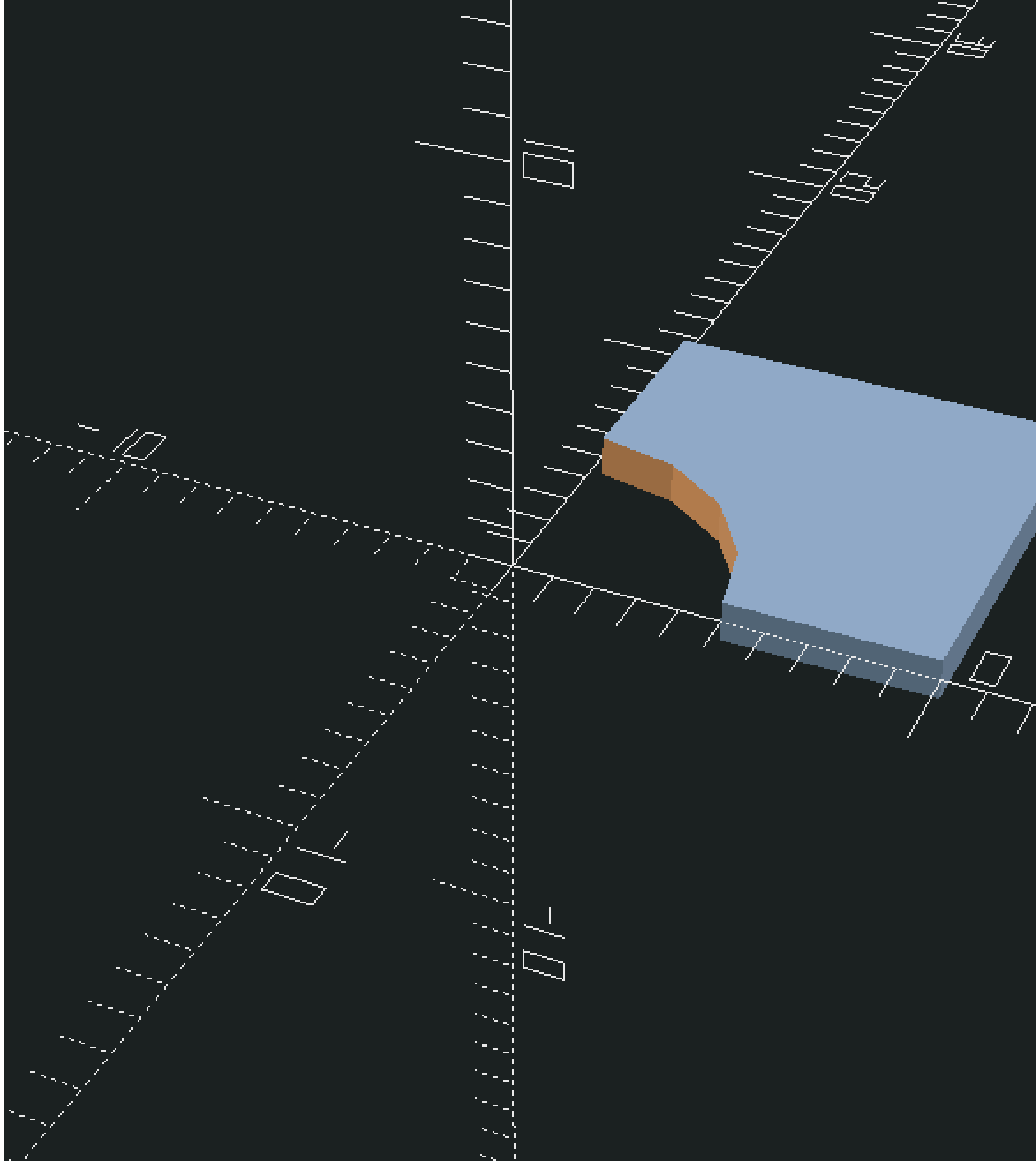
- Subtract children 1+ from child 0
- Order matters!

```
w= 10;

difference() {
    circle(d= w);
    square([w, w]);
}
```



Difference (reversed)

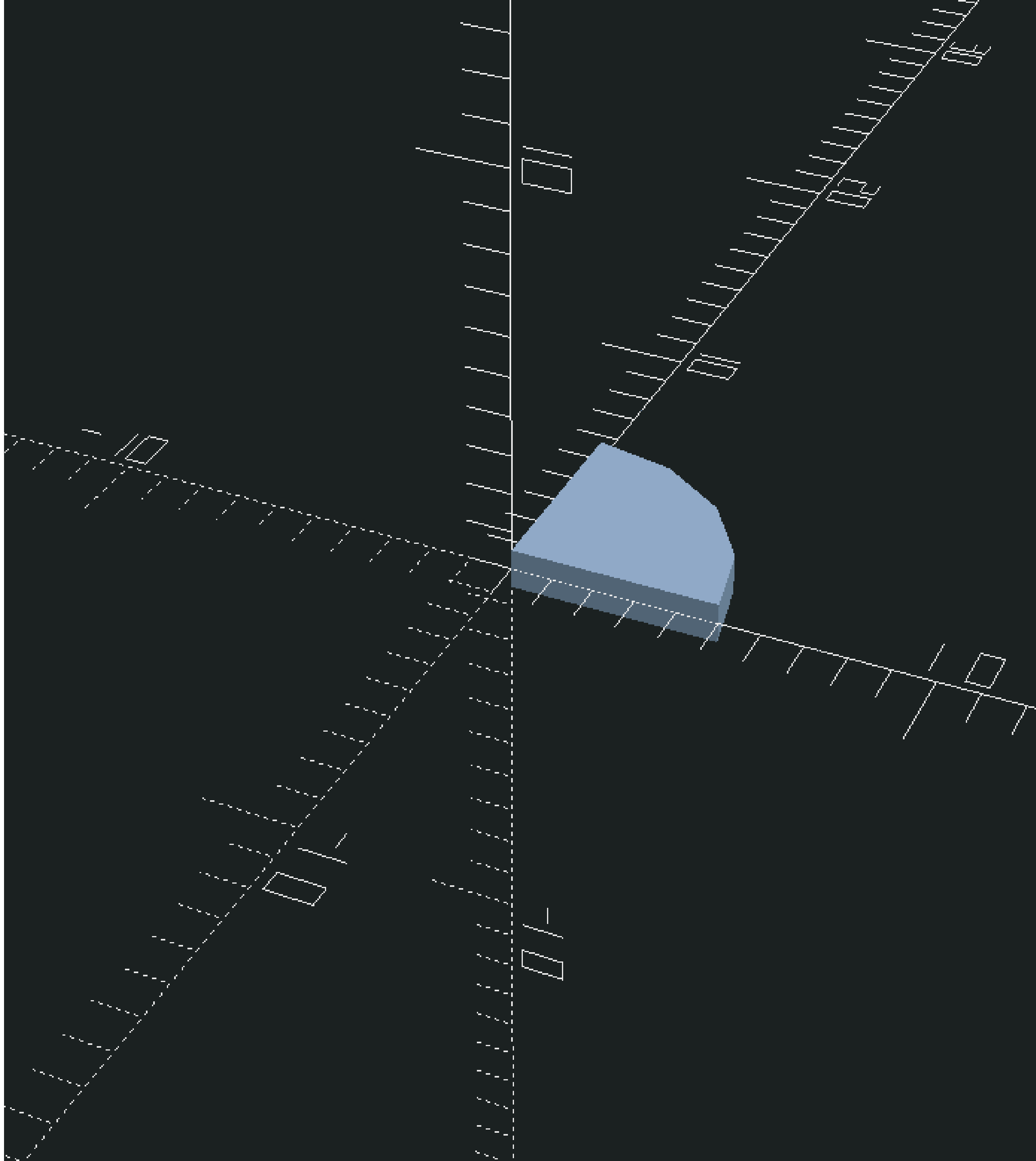
```
w= 10;

difference() {
    square([w, w]);
    circle(d= w);
}
```



Intersection

- Find the volume where all children overlap

```
w= 10;

intersection() {
    circle(d= w);
    square([w, w]);
}
```



The hardest thing in OpenSCAD

- Not making a mess of the code
- Functional but poor code quality
  - hard to manipulate
  - hard to understand

Code Structure

```
// define your literal variables
a= 10;
b= 20;
...
```

```
// calculate variables
c= a + b;
...
```

```
// Use modules and create geometry
double_circle(c);
...
```

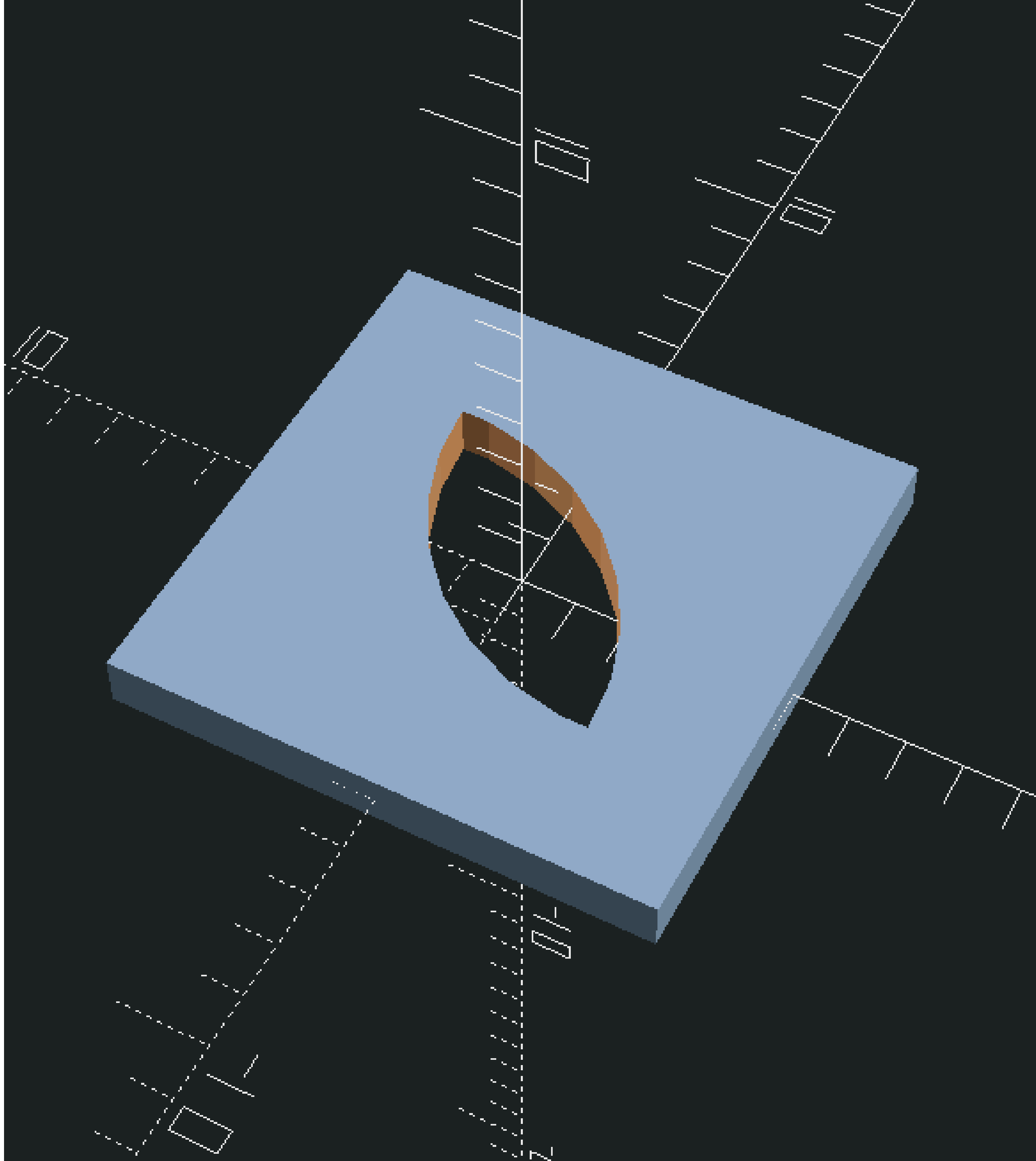
```
// define your functions and modules
module double_circle(diameter) { ... }
...
```

Readable & Clean OpenSCAD

- Use literals as parameters very, very sparingly
  - Exception: rotational degrees
  - Exception: increments/decrements
  - Exception: indices
- Calculating outside of function/module parameters
  - Good: use variables
  - Better: use functions
- DRY
  - New problem? Naming things!

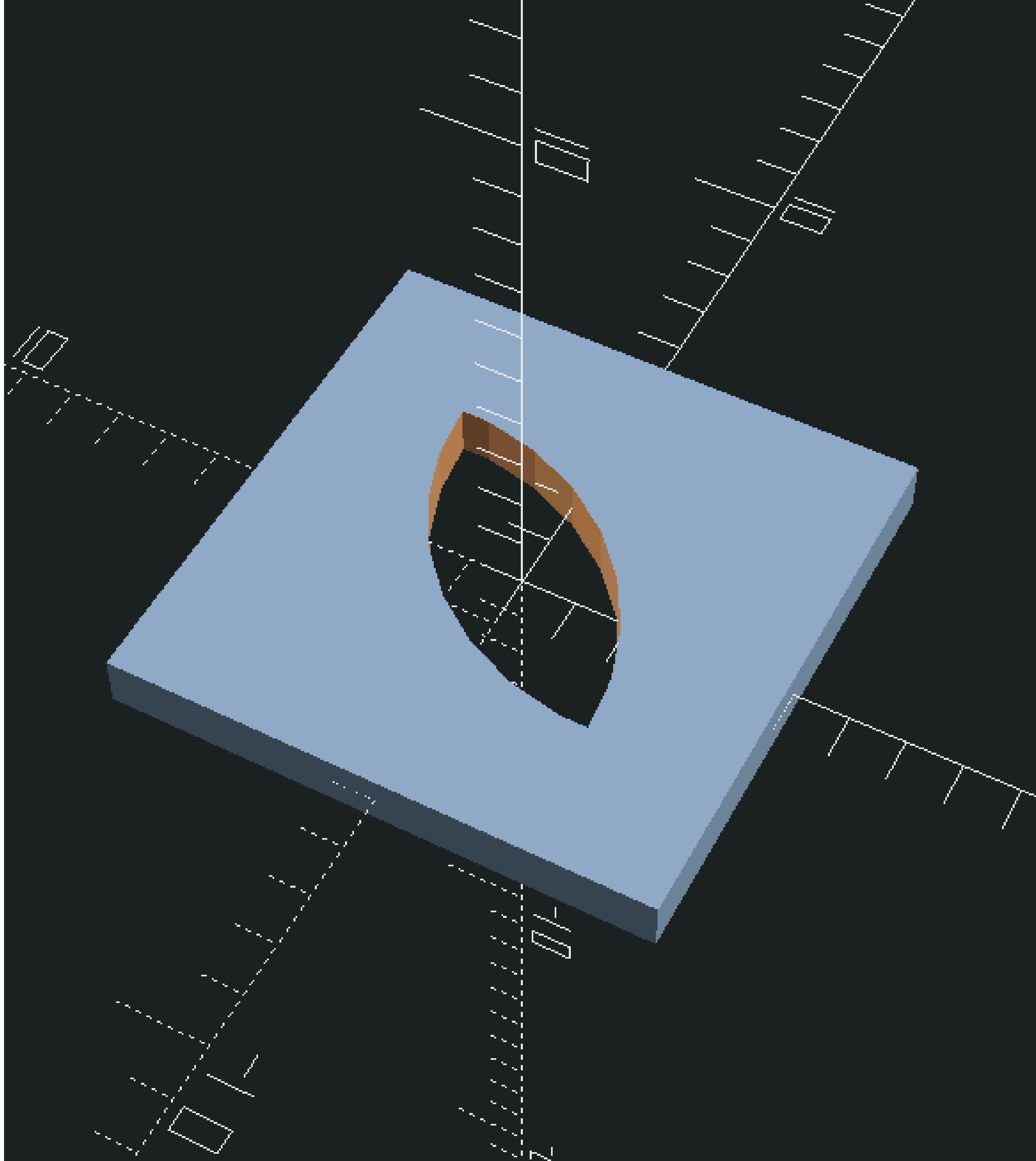
Exercise 9
(15 Minutes)

- Start with `w= 10;`
  - Calculate everything from this
- Use `intersection`, `difference`, `square`, `circle`, and `translation`
- Try to not repeat yourself (use `function` and `module`)



Exercise 9 Solution

```
w= 10;
quarter_w= w/4;

difference() {
    square(x_and_y(w), center= true);
    intersection() {
        translate(x_and_y(quarter_w))
            leaf_circle();
        translate(x_and_y(-quarter_w))
            leaf_circle();
    }
}

function x_and_y(unit)= [unit, unit];
module leaf_circle()
    circle(d= w, $fn= 30);
```

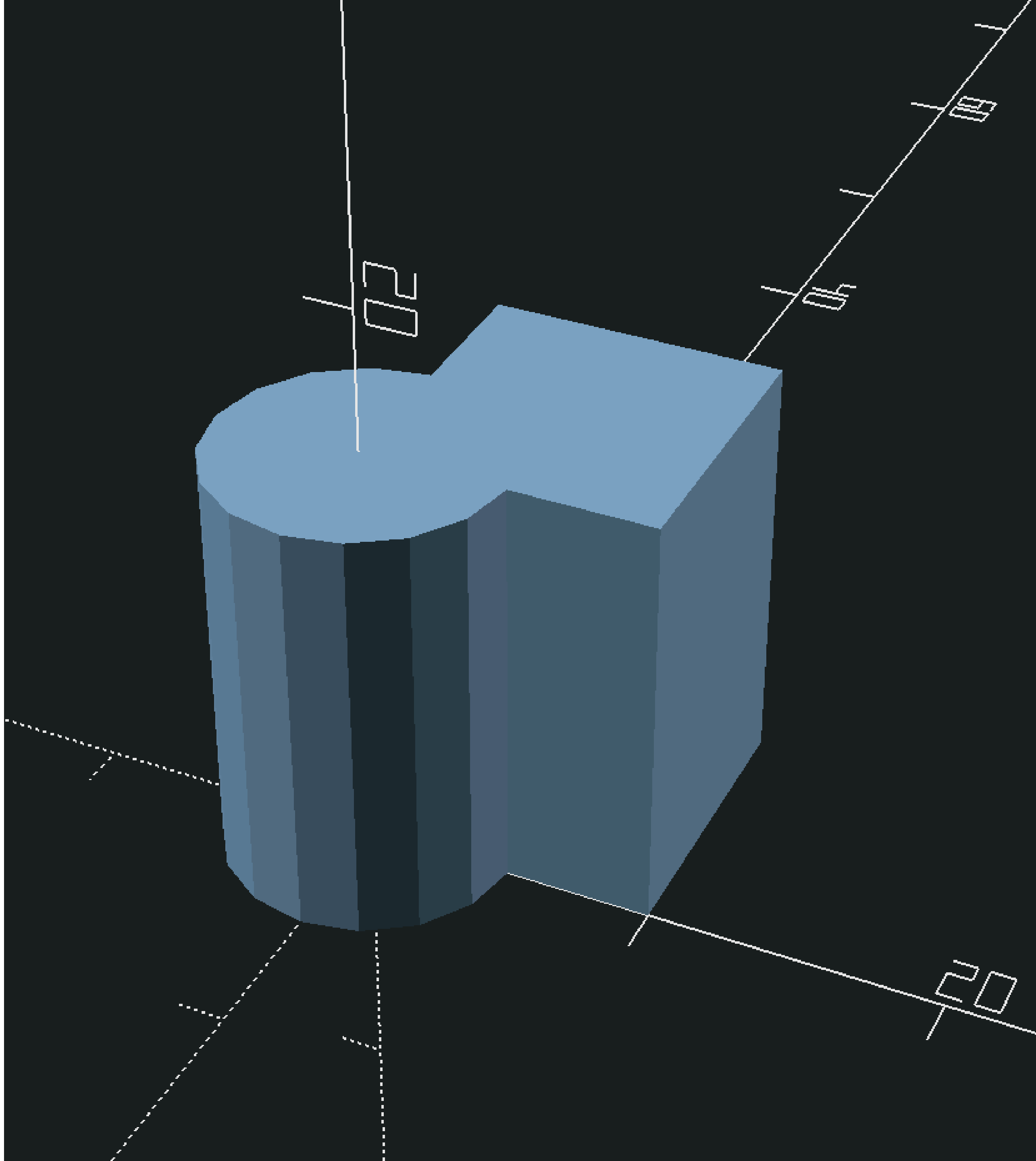


2D to 3D (Finally!)

- Up to this point everything has 0 `z`
- `linear_extrude( height=` *z* `)` *children*
  or `linear_extrude(` *z* `)` *children*
- Extrudes from `z` in the positive direction
  or set `center= true` to in both negative and positive
- Also `rotate_extrude`, but we're not covering that.

`linear_extrude` example

```
w= 10;
h= 5;

linear_extrude(h) {
    circle(d= w);
    square([w, w]);
}
```

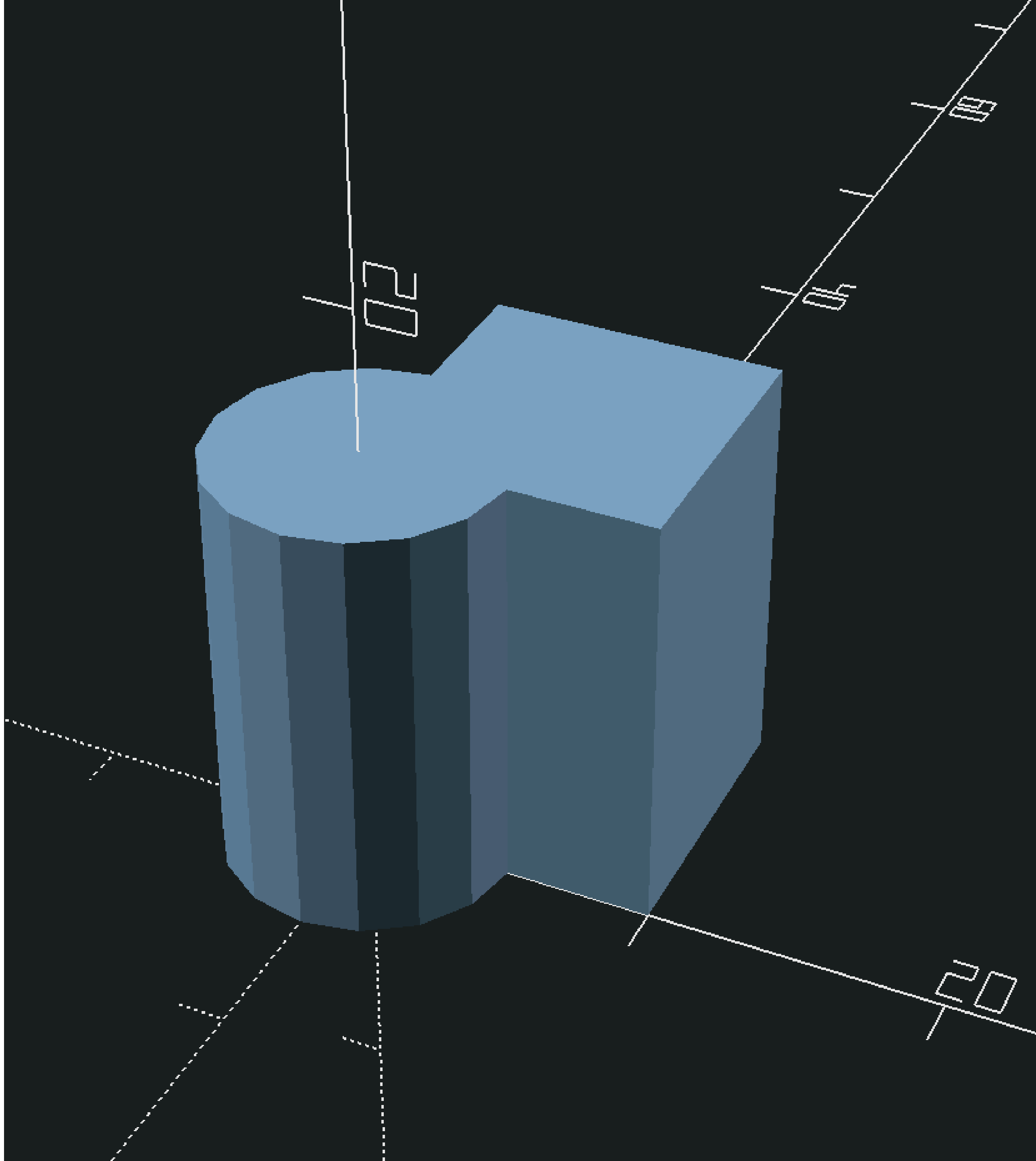


3D primitives

- `sphere` is a `circle` with rotational extrusion
  - `sphere(d|r=` *size* `);`
- `cylinder` is a `circle` with `linear_extrude`
  - `cylinder(d|r=` *size* `, h=` *z* `);`
  - `cylinder(d1|r1=` *size* `, d2|r2=` *size* `, h=` *z* `);` (conical)
- `cube` is a `square` with `linear_extrude`
  - `cube(size=` *3-element vector* `)`
  - `cube(` *3-element vector* `)`
- All accept `center` parameter (affects `z` as well!)

Simple 3D Example

```
w= 10;
h= 15;

cylinder(d= w, h= h);
cube([w, w, h]);
```

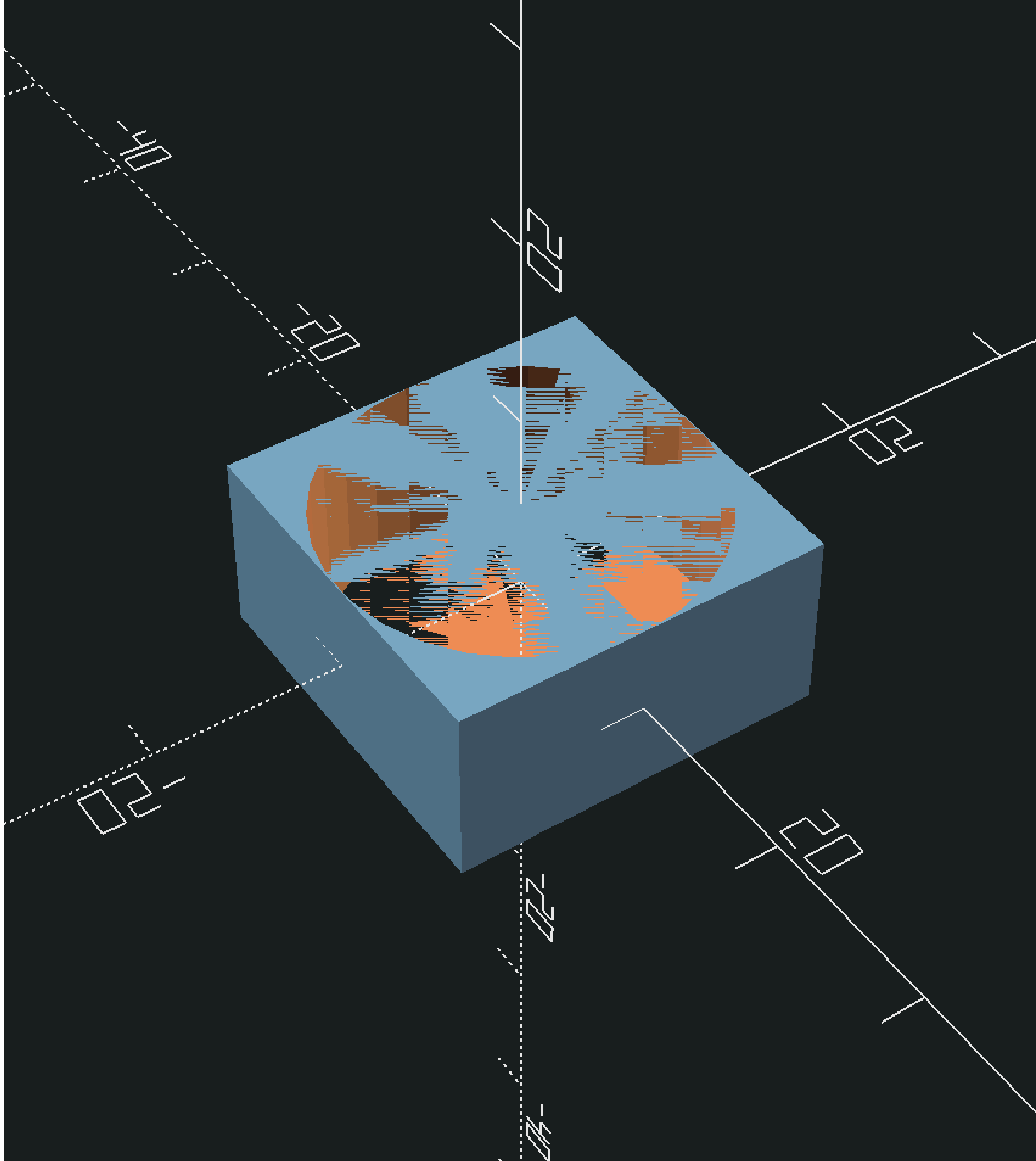


Boolean Operations in 3D

```
w= 20;
h= 10;

difference() {
    cube([w, w, h], center= true);
    cylinder(d= w, h= h, center= true);
}
```

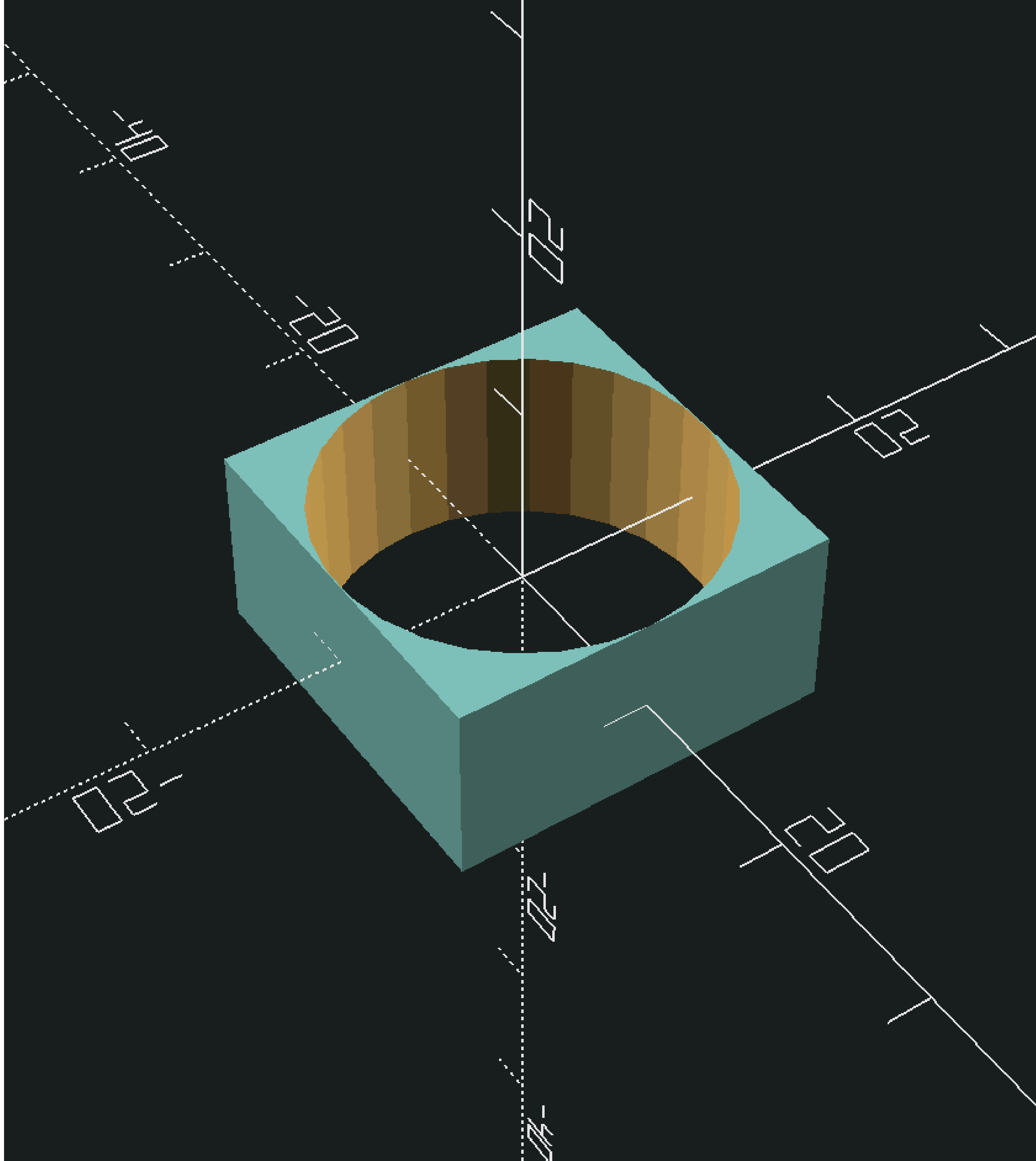
What is this?!

- It's called *z fighting*
- `h` is the same for the cube and cylinder
- Preview rendering: each pixel occupies the same plane
- May cause manifold problems



Final rendering

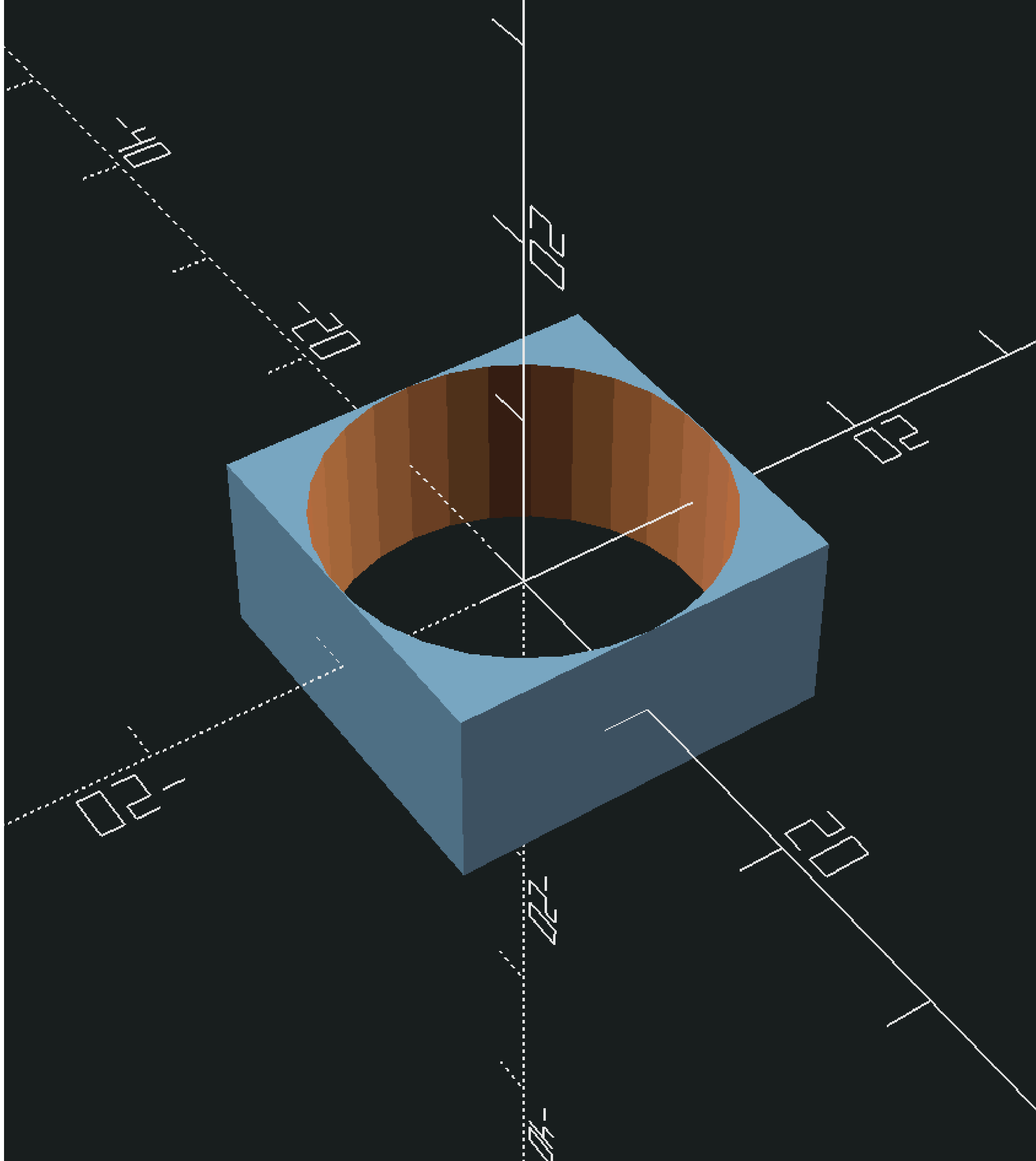
- `F6` for final rendering
- Slower, especially for complex models



`epsilon` values

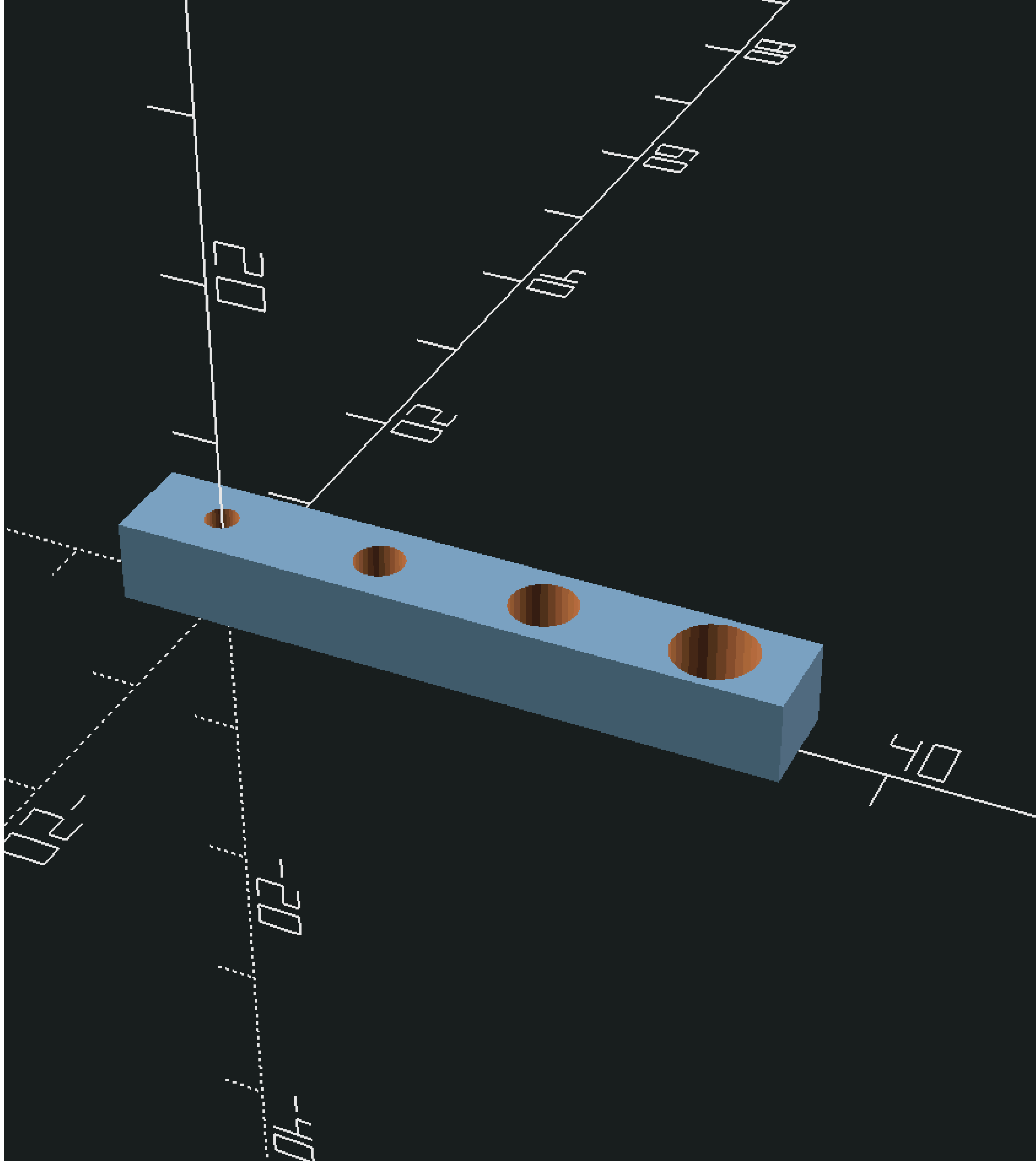
Introduce a small amount of extra material to decisively `difference` objects

```
w= 20;
h= 10;
epsilon= 0.01;

difference() {
    cube([w, w, h], center= true);
    cylinder(
        d= w,
        h= h + (epsilon*2),
        center= true
    );
}
```



Exercise 10
(15 Minutes)

- 4 `cylinder`s of diameters 2, 3, 4, 5
- Center-to-center distance is 10
- `cube` is always 2 wider than largest diameter
- Length is determined by the number of `cylinder`s
- Able to be used with customizer to change the smallest/largest diameter and count of holes



Exercise 10 Solution

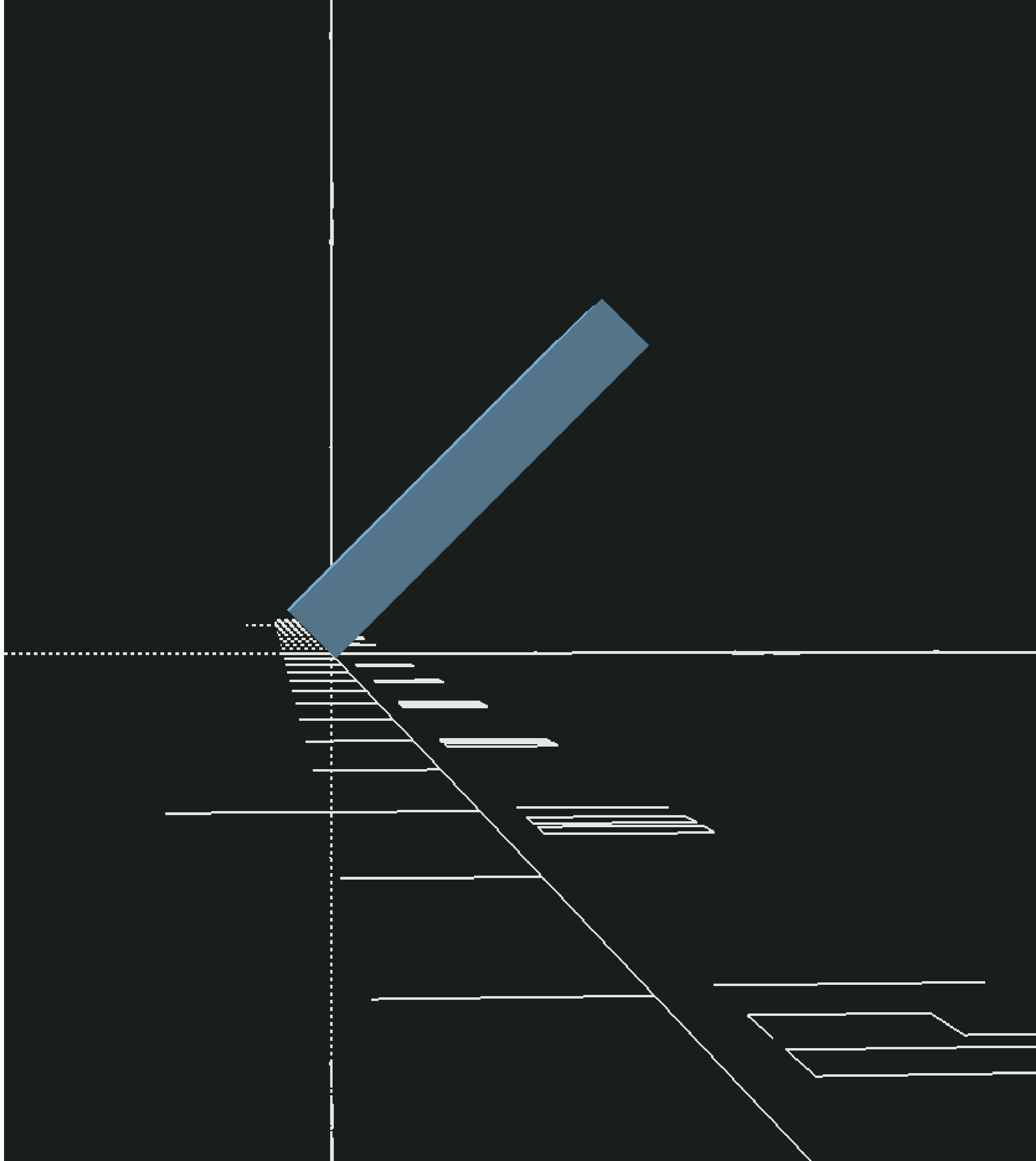
```
c_to_c_spacing= 10;
count= 4;
smallest= 2;
thickness= 5;

epsilon= 0.01;
w= count + smallest;

difference() {
    translate([-c_to_c_spacing/2,- w/2, 0])
        cube([c_to_c_spacing * count, w, thickness]);

    for(i= [0 : count-1])
        translate([c_to_c_spacing * i, 0, -epsilon])
            cylinder(d= i + smallest, h= thickness + (epsilon*2), $fn= 30);
}
```

`rotate`

- rotate children about one or many axes.
- `rotate([` *3 element array* `])` *child* or *children*
- rotates on the center

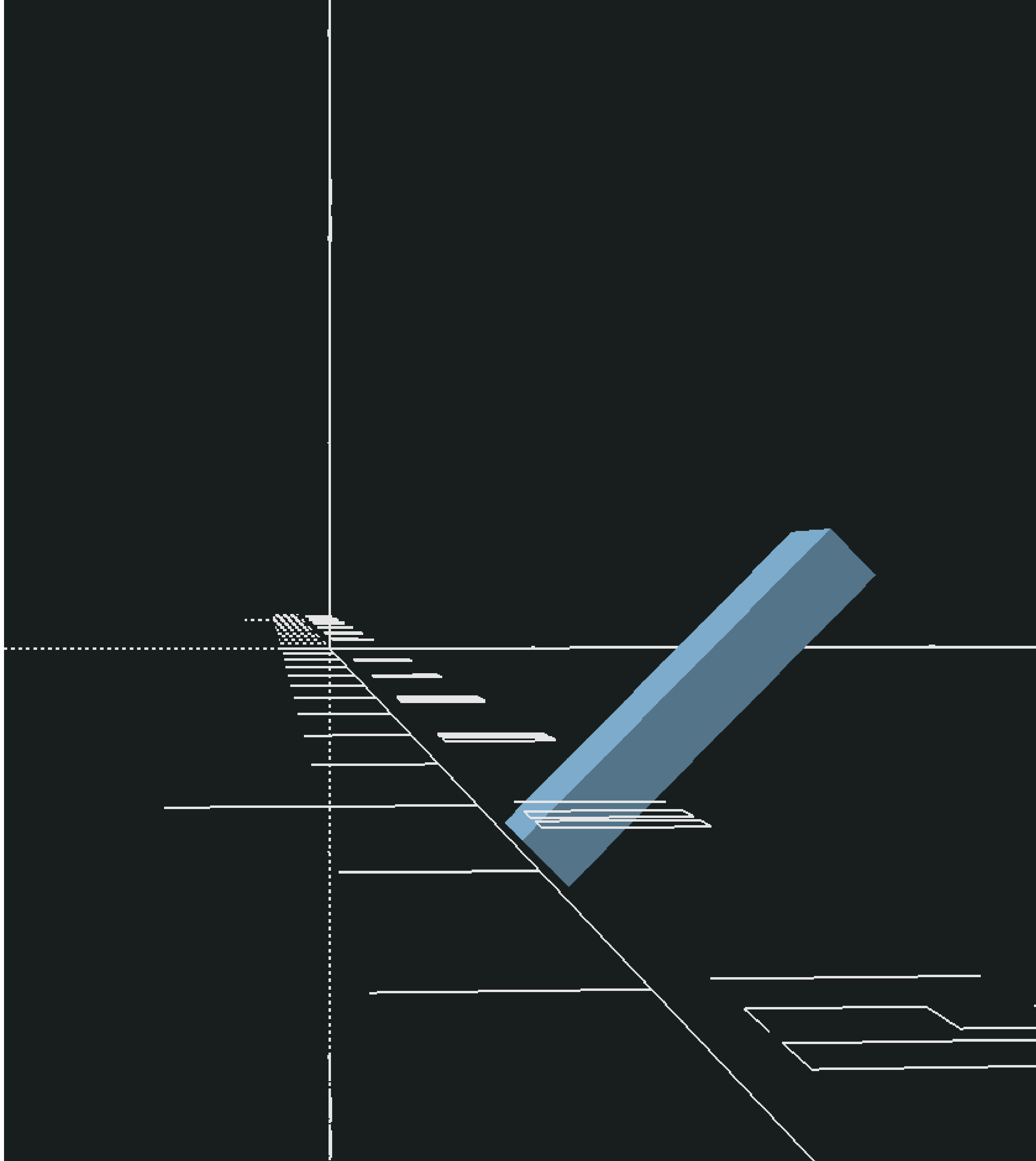
`rotate` Example

```
size= [10, 20, 3];

rotate([45, 0, 0])
    cube(size);
```



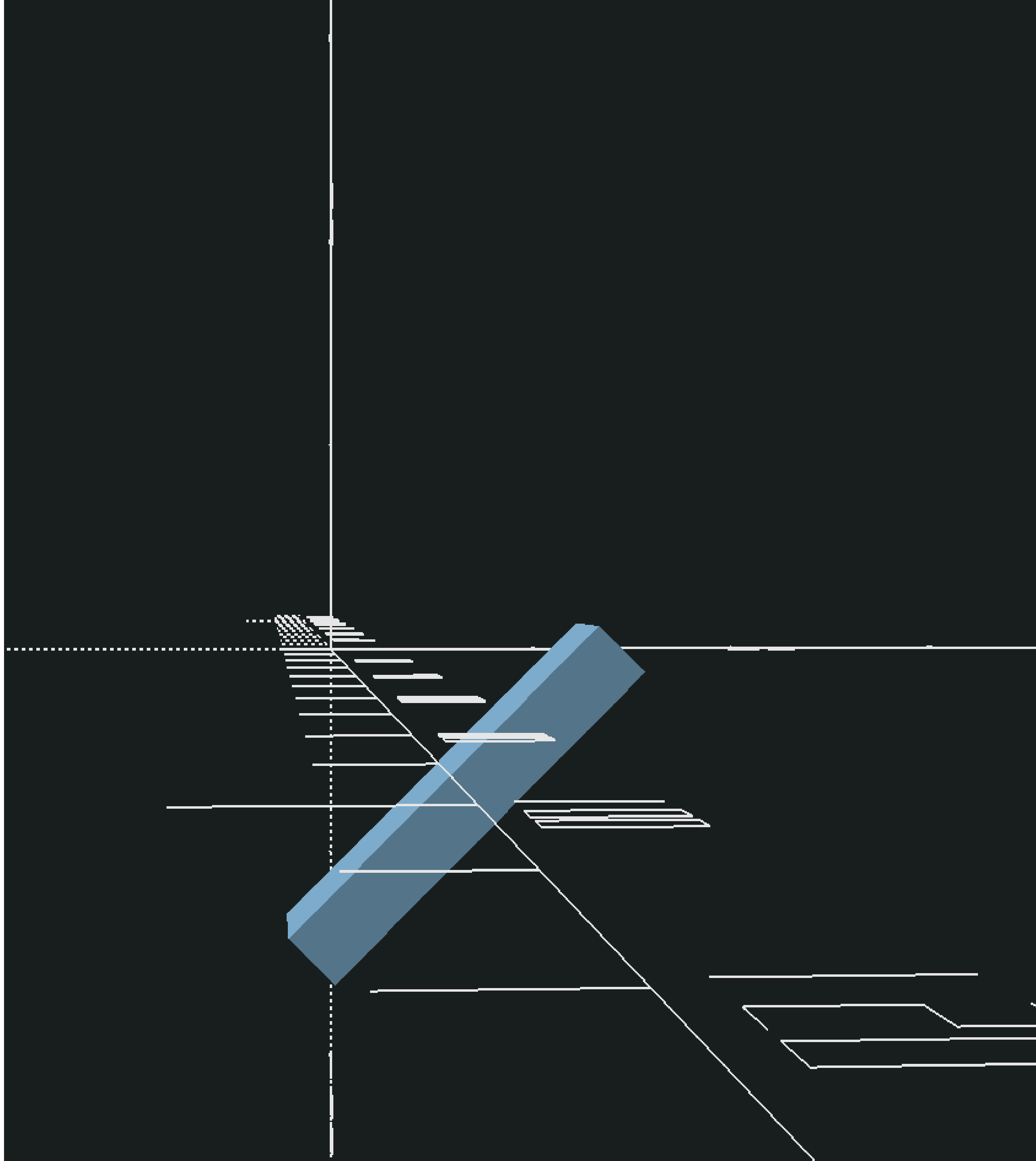
rotate and translate

```
size= [10, 20, 3];
shift= -15;

rotate([45, 0, 0])
    translate([0, 0, shift])
        cube(size);
```



translate and rotate

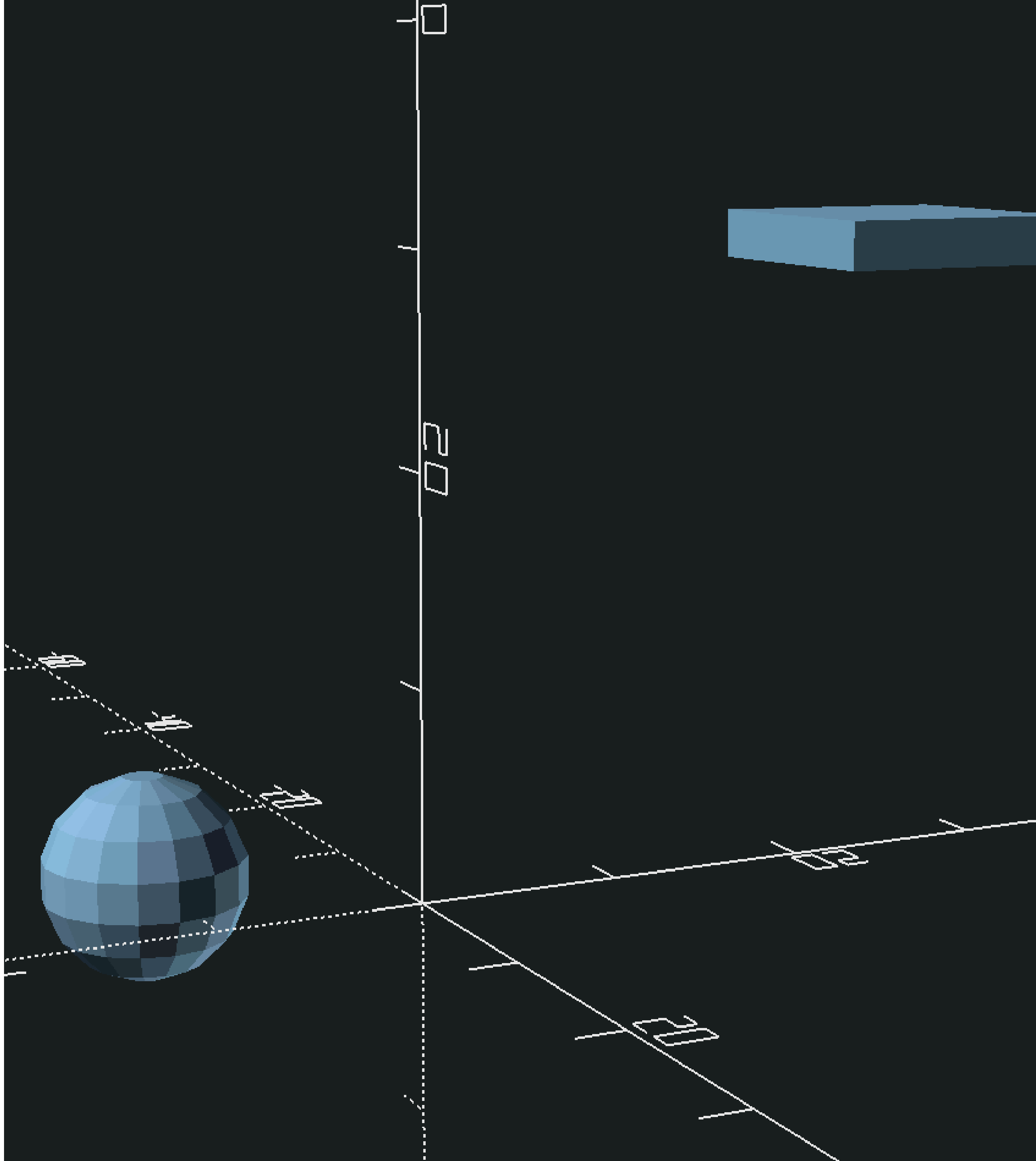
```
size= [10, 20, 3];
shift= -15;

translate([0, 0, shift])
    rotate([45, 0, 0])
        cube(size);
```



Joining objects

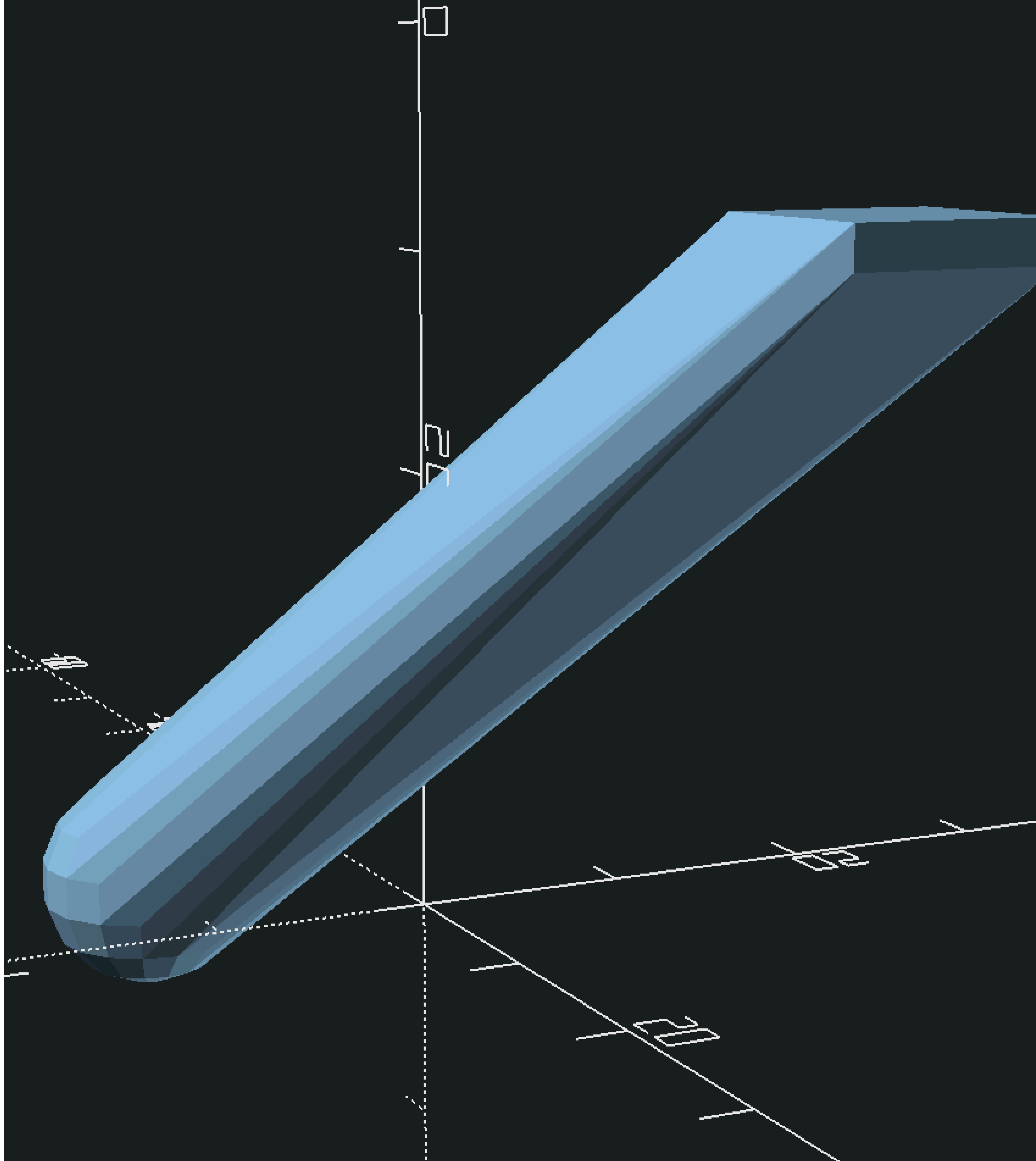
How do we join these two objects?

```
w= 10;
thickness= 2;
h= 30;

translate([-w, -w, 0])
    cube([w, w, thickness]);

translate([w, w, h])
    sphere(d= w);
```



Joining objects with `hull`

```
w= 10;
thickness= 2;
h= 30;

hull() {
    translate([-w, -w, 0])
        sphere(d= w);

    translate([w, w, h])
        cube([w, w, thickness]);
}
```



How do you 3D print this stuff?

- Render with `F6`
- Export with `F7` (or command line)
- ASCII vs Binary STL
- `F8` Directly to OctoPrint?
    - YMMV
    - Klipper someday 🤞
- Embed print settings into your designs
    - wall thickness
    - first layers

Before the capstone...

- Contributors wanted!
- C++ project
- Area need:
  - Releases
  - Automation
  - Testing
- **github.com/openscad**

Make sure and follow us on social media!

Kyle

Mastodon: @linux_mclinuxface@fosstodon.org

GitHub: github.com/stockholmux

LinkedIn: linkedin.com/in/kyle-davis-linux/

Spot

Mastodon: @spot@social.afront.org

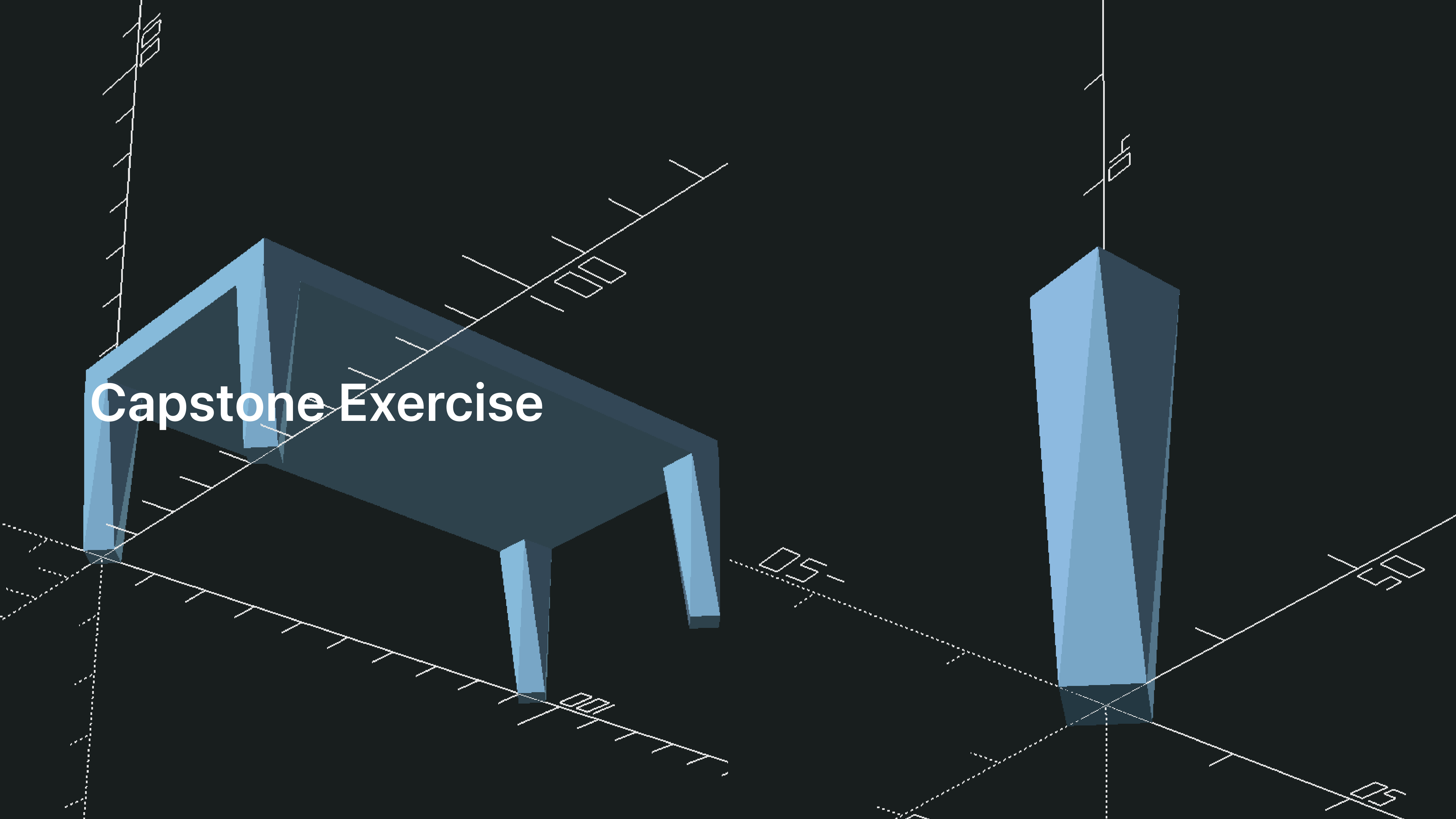
Bluesky: @spot.bsky.social

Instagram / Threads: @spotprints

Capstone Exercise

- Create a table
- Top dimensions changed via customizer
- Top thickness changed via customizer
- Height changed via customizer
- With legs at each corner
    - The bottom is twisted 45°
    - Top and bottom leg changed via customizer
- **Stretch:** Enabled a variable number of evenly spaced legs

Capstone Exercise



Capstone Exercise (Stretch)