nclab
More Than Coding

PLaSM - List of Commands

August 31, 2016[1]

Contents

[1]This document was prepared using the LaTeX module in NCLab

1 About This Document

This document covers selected basic functionality of PLaSM, and it is meant to be a reference rather than a learning material. If you'd like to learn 3D modeling, take the NCLab's self-paced 3D Modeling course. More information can be found on the course web page at `http://nclab.com/3dmodeling/`.

2 Language Variants

PLaSM can be used with English, Spanish, German, Czech, Polish, Italian, and French commands. All language versions of this document can be found in the menu of the PLaSM module. Any of these seven languages can be chosen in Settings to be the main language for NCLab.

3 Simple Shapes

3.1 Basics - Naming Objects, Default Positions

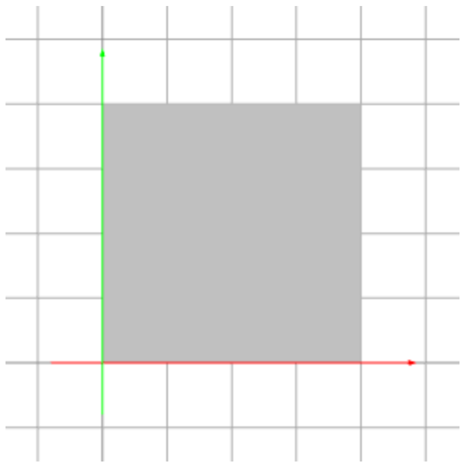
Every object that you create will be located at a default position. Each object needs to have a name, so that further operations with the object are possible. All PLaSM keywords are in uppercase letters. To avoid conflicts, use lowercase letters for names of your objects.

3.2 SQUARE

The command `SQUARE(s)` creates a square of given size `s`. The square is located in the first quadrant, with one vertex at the origin $(0, 0)$. By first quadrant we mean the part of the 2D plane where the coordinates x and y are positive. Example:

```
s = SQUARE(4)
SHOW(s)
```

Output:

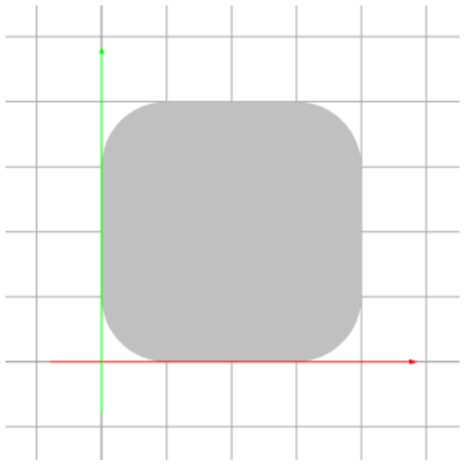


3.3 SQUARE with Round Corners

The command `SQUARE(s, r)` creates a square of size `s` whose corners are round with radius `r`. The radius `r` must be less than or equal to `s/2`. Example:

```
s = SQUARE(4, 1)
SHOW(s)
```

Output:

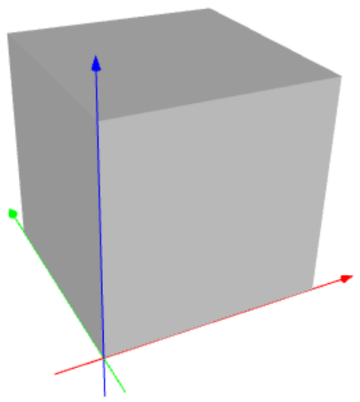


3.4 CUBE

The command `CUBE(s)` creates a cube of given size `s`, located in the first octant, with one vertex at the origin $(0, 0, 0)$. By first octant we mean the part of the 3D space where all three coordinates x, y and z are positive. The following code creates a cube `c` of dimensions $1 \times 1 \times 1$, and displays it:

```
c = CUBE(4)
SHOW(c)
```

Output:

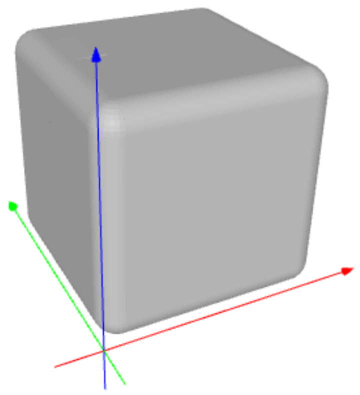


3.5 CUBE with Round Edges

The command `CUBE(s, r)` creates a cube of size `s` whose edges are round with radius `r`. The radius `r` must be less than or equal to `s/2`. Example:

```
c = CUBE(1, 0.1)
SHOW(c)
```

Output:

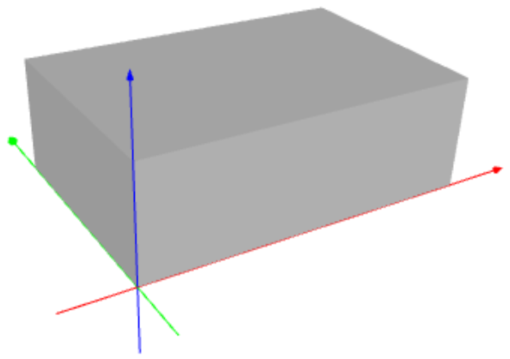


3.6 BOX

`BOX` is a versatile command that creates squares, rectangles, cubes, bricks, and other rectangular prisms. The command `BOX(x, y, z)` creates a 3D box (rectangular prism) of given dimensions `x`, `y`, `z`:

```
b = BOX(3, 2, 1)
SHOW(b)
```

Output:



The command `BOX(x, y)` creates a 2D box (rectangle) of given dimensions `x`, `y`:

```
b = BOX(3, 1)
SHOW(b)
```

Output:

The command can also be used as `BOX(xmin, xmax, ymin, ymax, zmin, zmax)` which creates the Cartesian product of intervals $(xmin, xmax) \times (ymin, ymax) \times (zmin, zmax)$. In 2D, analogously the command `BOX(xmin, xmax, ymin, ymax)` creates the Cartesian product of intervals $(xmin, xmax) \times (ymin, ymax)$.

3.7 RECTANGLE

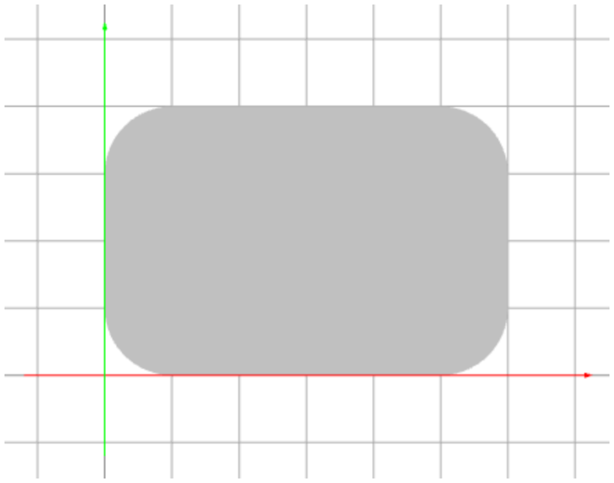
The command `RECTANGLE(x, y)` is the same as `BOX(x, y)`. It creates a rectangle of width `x` and height `y` located in the first quadrant.

3.8 RECTANGLE with Round Corners

The command `RECTANGLE(x, y, r)` creates a rectangle of width `x` and height `y` whose corners are round with radius `r`. The radius `r` must be less than or equal to a half of the shorter edge. Example:

```
r = RECTANGLE(6, 4, 1)
SHOW(r)
```

Output:



3.9 BRICK

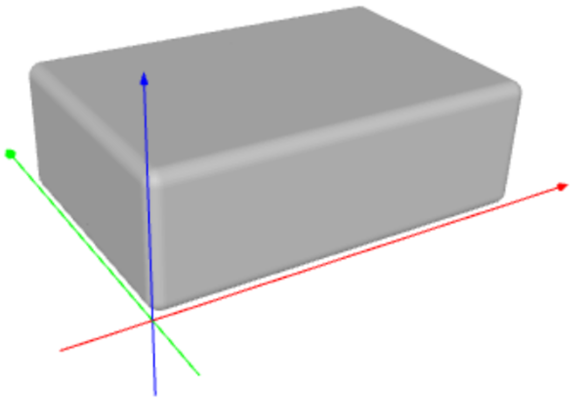
The command `BRICK(x, y, z)` is the same as `BOX(x, y, z)`.

3.10 BRICK with Round Edges

The command `BRICK(x, y, z, r)` creates a brick of dimensions `x`, `y` and height `z` whose edges are round with radius `r`. The radius `r` must be less than or equal to a half of the shortest edge. Example:

```
b = BRICK(3, 2, 1, 0.1)
SHOW(b)
```

Output:

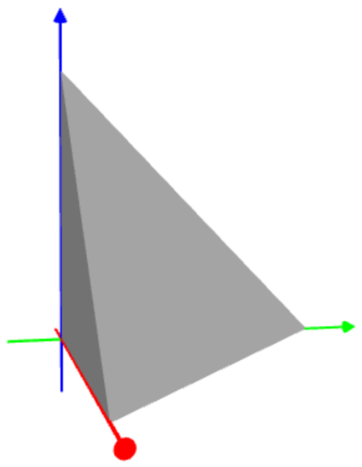


3.11 TETRAHEDRON

The command `TETRAHEDRON(a, b, c, d)` creates a tetrahedron in 3D that spans four 3D points `a`, `b`, `c`, `d`. Each 3D point is defined via three coordinates x, y, z that are enclosed in square braces. Example:

```
t = TETRAHEDRON(POINT(0, 0, 0), POINT(1, 0, 0), POINT(0, 1, 0), POINT(0, 0, 1))
SHOW(t)
```

Output:

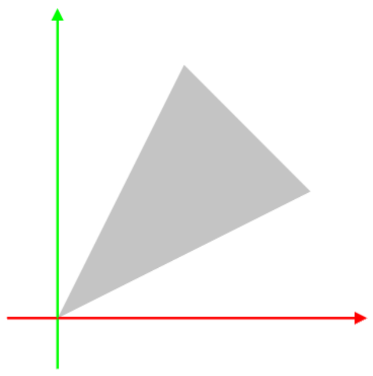


3.12 TRIANGLE

The command `TRIANGLE(a, b, c)` creates a triangle in the xy-plane that spans three 2D points `a`, `b`, `c`. Each 2D point is defined via two coordinates x, y that are enclosed in square brackets. Example:

```
t = TRIANGLE(POINT(0, 0), POINT(2, 1), POINT(1, 2))
SHOW(t)
```

Output:

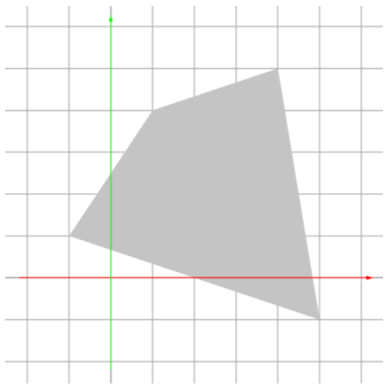


3.13 QUAD

The command `QUAD(a, b, c, d)` creates a convex quadrilateral in the xy-plane that spans four 2D points `a`, `b`, `c`, `d`. The order of the points does not matter. Each point is defined via two coordinates x, y that are enclosed in square brackets. Example:

```
q = QUAD(POINT(-1, 1), POINT(5, -1), POINT(4, 5), POINT(1, 4))
SHOW(q)
```

Output:

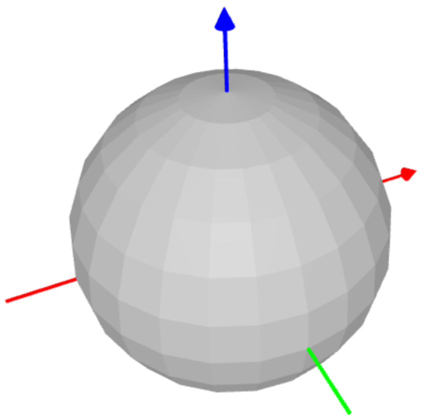


3.14 SPHERE

The command `SPHERE(r)` creates a sphere with given radius `r`, located at the origin (0, 0, 0). Example:

```
s = SPHERE(1.5)
SHOW(s)
```

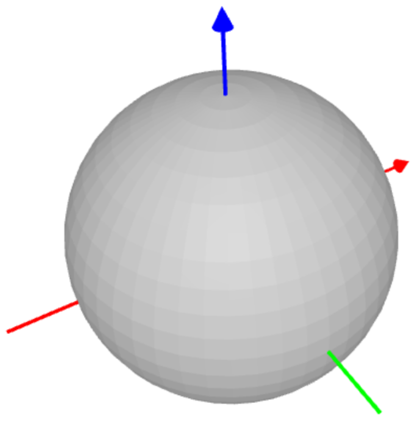
Output:



As you can see, this is a rather coarse approximation of a sphere - the equator is divided into 24 pieces. To improve the smoothness, one can use an optional second parameter to refine the division. For example, the following code uses 48 divisions:

```
s = SPHERE(1.5, 48)
SHOW(s)
```

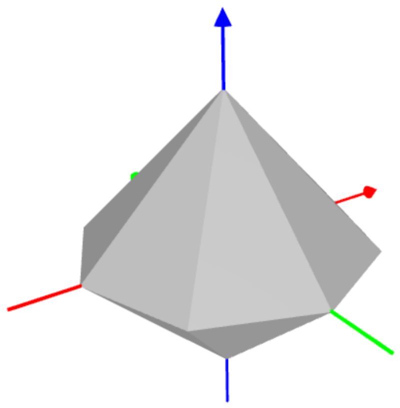
Output:



Last - the sphere has two major directios, in which the divisions can be set independently. This technique can be used to achieve various useful special shapes such as, for example, a diamond:

```
s = SPHERE(1.5, [2, 8])
SHOW(s)
```

Output:

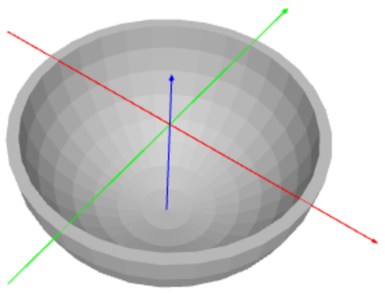


3.15 SHELL

The command `SHELL(r1, r2)` creates a spherical shell with given inner radius `r1` and outer radius `r2`. The shell is located at the origin (0, 0, 0). Example:

```
s = SHELL(3.7, 4)
SHOW(s)
```

Output:



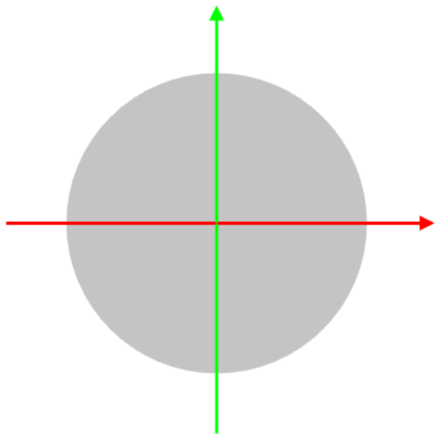
The smoothness of the shell can be increased or decreased using the same optional parameters as in `SPHERE()`.

3.16 CIRCLE

The command `CIRCLE(r)` creates a circle of a given radius `r` in the *xy*-plane, that is located at the origin (0, 0). Example:

```
c = CIRCLE(2.25)
SHOW(c)
```

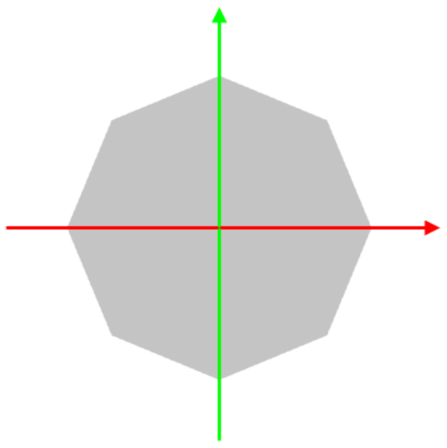
Output:



As with the `SPHERE`, the command `CIRCLE` has an optional second parameter that defines the division of the perimeter. Its default value is 64 which means a fairly smooth circle. It can be reduced to obtain various symmetric polygons:

```
c = CIRCLE(2.25, 8)
SHOW(c)
```

Output:

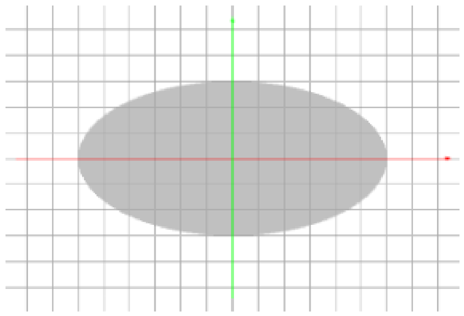


3.17 ELLIPSE

The command `ELLIPSE(a, b)` creates an ellipse centered at the origin (0, 0), whose axes of symmetry coincide with the x and y axes. The ellipse spans the interval $(-a, a)$ on the x-axis and $(-b, b)$ on the y-axis. Example:

```
e = ELLIPSE(6, 3)
SHOW(e)
```

Output:



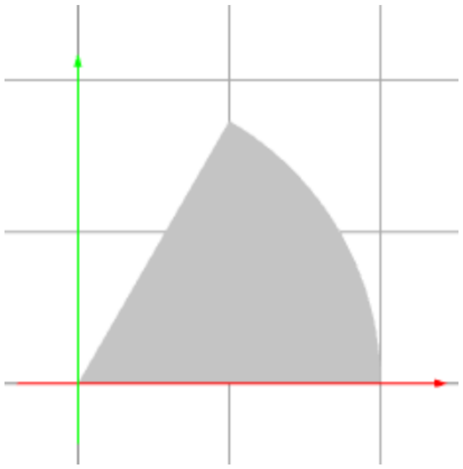
Ellipses are just scaled circles. Therefore, analogously to the `CIRCLE` command, the `ELLIPSE` command has an optional third parameter that defines the division of the perimeter.

3.18 ARC

The command `ARC(r1, r2, alpha)` creates an arc of a given inner radius `r1`, outer radius `r2`, and angle `alpha`. The inner radius can be zero:

```
a = ARC(0, 2, 60)
SHOW(a)
```

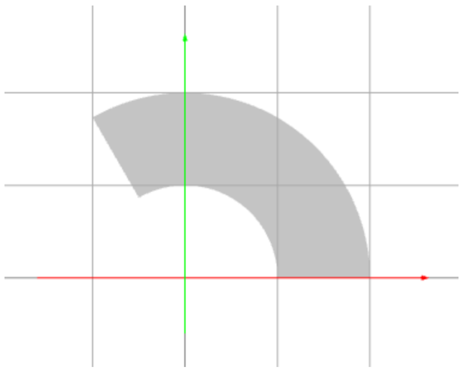
Output:



and it can be greater than zero as well:

```
a = ARC(1, 2, 120)
SHOW(a)
```

Output:

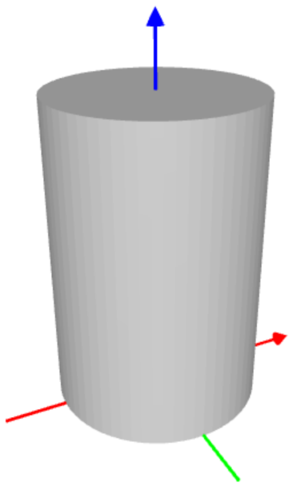


3.19 CYLINDER

The command `CYLINDER(r, h)` creates a cylinder of a given radius `r` and height `h`. The cylinder stands on the xy-plane and its axis coincides with the z-axis. Example:

```
c = CYL(1, 3)
SHOW(c)
```

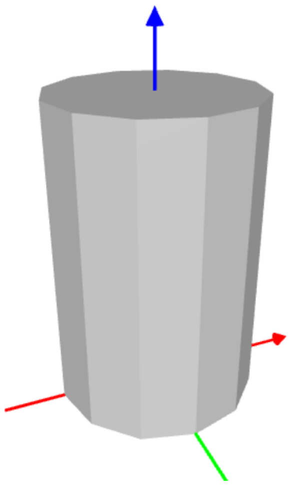
Output:



The division in the angular direction can be increased or decreased via an optional third parameter whose default value is 64:

```
c = CYL(1, 3, 12)
SHOW(c)
```

Output:

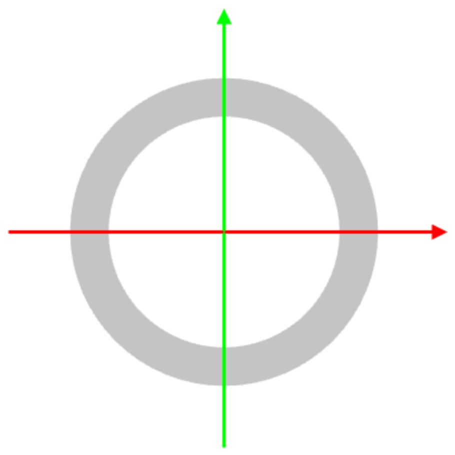


3.20 RING

The command `RING(r1, r2)` creates a ring (2D object in the xy-plane) of a given inner radius `r1` and outer radius `r2`. Example:

```
r = RING(1.5, 2)
SHOW(r)
```

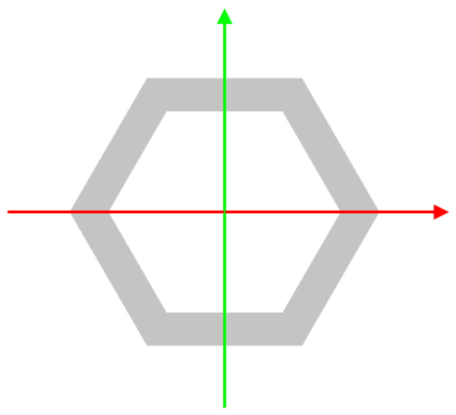
Output:



By default, the ring is divided into 64 pieces in the angular direction. This division can be chnaged via an optional third parameter:

```
r = RING(1.5, 2, 6)
SHOW(r)
```

Output:

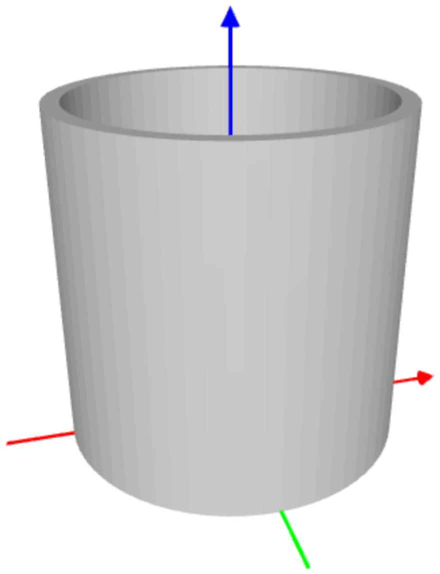


3.21 TUBE

The command `TUBE(r1, r2, h)` creates a tube of a given inner radius `r1`, outer radius `r2`, and height `h`. Example:

```
t = TUBE(0.9, 1, 2)
SHOW(t)
```

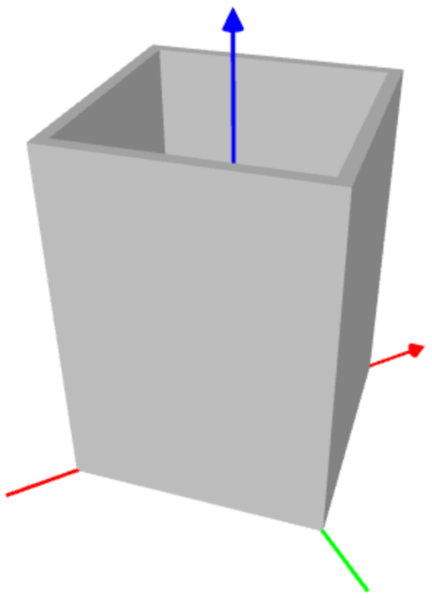
Output:



By default, the tube is divided into 64 pieces in the angular direction. This division can be changed via an optional fourth parameter:

```
t = TUBE(0.9, 1, 2, 4)
SHOW(t)
```

Output:

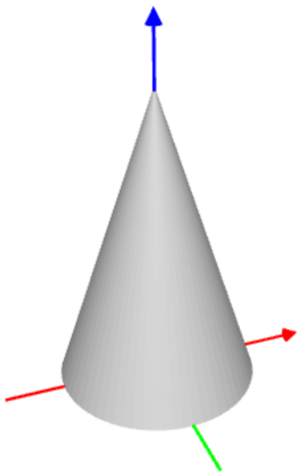


3.22 CONE

The command `CONE(r, h)` creates a cone of a given radius `r` and height `h`. The default position of the cone is the same as of the cylinder. Example:

```
c = CONE(1, 3)
SHOW(c)
```

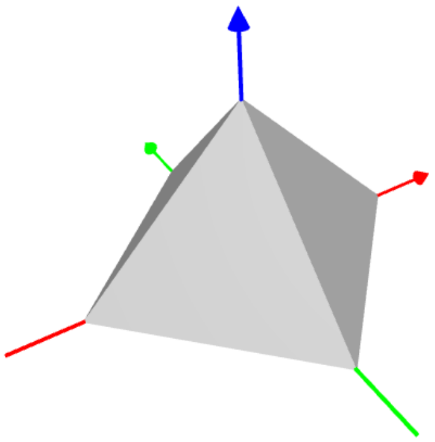
Output:



There is an optional third parameter with default value 64, that is often used to create pyramids:

```
c = CONE(1, 1, 4)
SHOW(c)
```

Output:

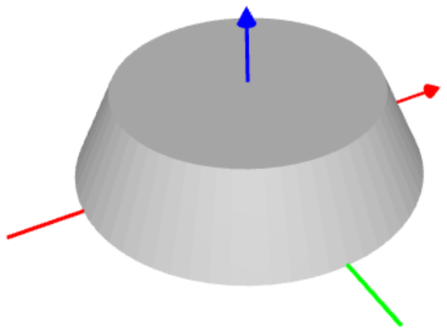


3.23 TCONE

The command `TCONE(r1, r2, h)` creates a truncated cone of a given bottom radius `r1`, top radius `r2`, and height `h`. The default position of the truncated cone is the same as of the cone. Example:

```
c = TCONE(4, 3, 2)
SHOW(c)
```

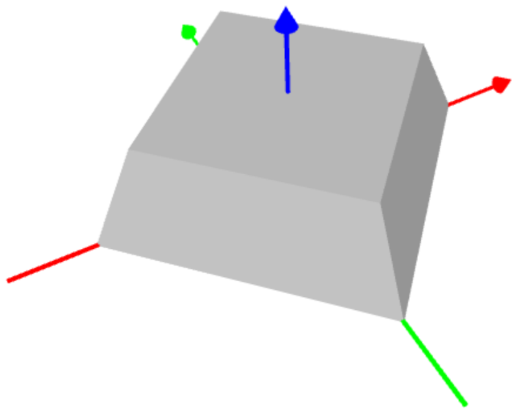
Output:



There is an optional third parameter with default value 64, that can be used to create truncated pyramids:

```
c = TCONE(4, 3, 2, 4)
SHOW(c)
```

Output:

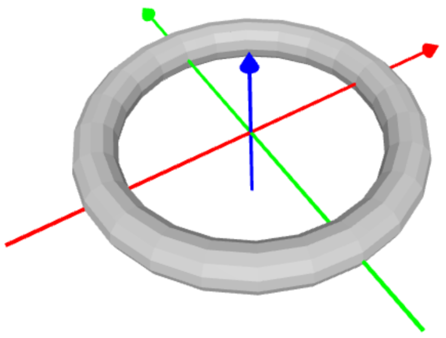


3.24 TORUS

The command `TORUS(r1, r2)` creates a torus of a given inner radius `r1` and outer radius `r2`:

```
t = TORUS(3, 4)
SHOW(t)
```

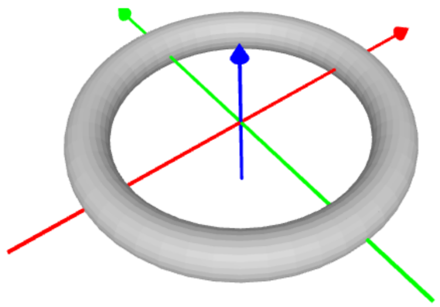
Output:



The default subdivision of the torus along the main equator is 24 (same as of the sphere). To improve the smoothness of the surface, this subdivision can be increased via an optional third parameter:

```
t = TORUS(3, 4, 48)
SHOW(t)
```

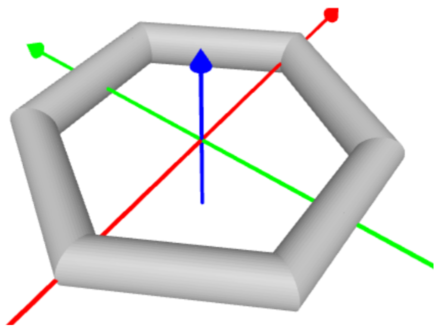
Output:



As with a sphere, there are two main directions on the torus. With different subdivisions in these directions we can achieve various special shapes such as

```
t = TORUS(3, 4, [6, 48])
SHOW(t)
```

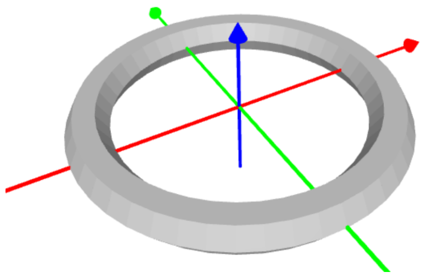
Output:



or

```
t = TORUS(3, 4, [48, 6])
SHOW(t)
```

Output:

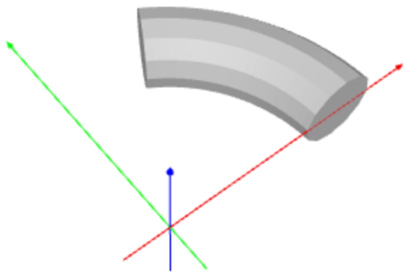


3.25 ELBOW

The command `ELBOW(r1, r2, alpha)` creates an elbow (part of a torus) of a given inner radius `r1`, outer radius `r2`, and angle `alpha`:

```
e = ELBOW(2, 3, 60)
SHOW(e)
```

Output:



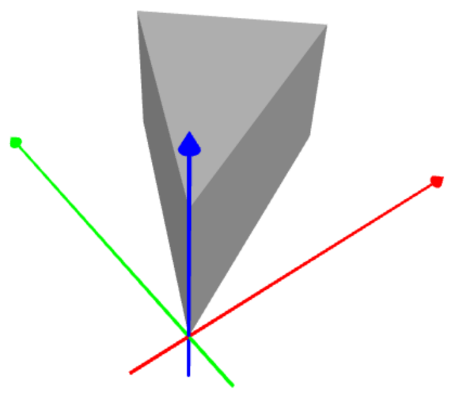
The default subdivision of the elbow is 24 in the direction of rotation about the Z axis, and 12 along the XZ cutplane. To make the surface smoother or less smooth, or to create special shapes, there are optional parameters that work exactly in the same way as with a torus.

3.26 PRISM

The command `PRISM(base, h)` creates a prism of height `h` given an arbitrary 2D object `base` located in the *xy*-plane. Example:

```
base = TRIANGLE([0, 0], [2, 1], [1, 2])
p = PRISM(base, 1)
SHOW(p)
```

Output:

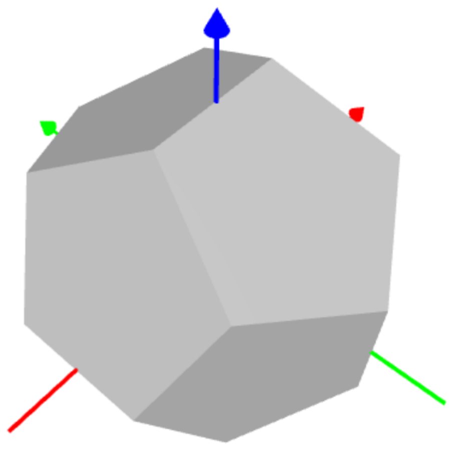


3.27 DODECAHEDRON

Creates a dodecahedron (symmetric object with 12 identical faces):

```
d = DODECAHEDRON()
SHOW(d)
```

Output:

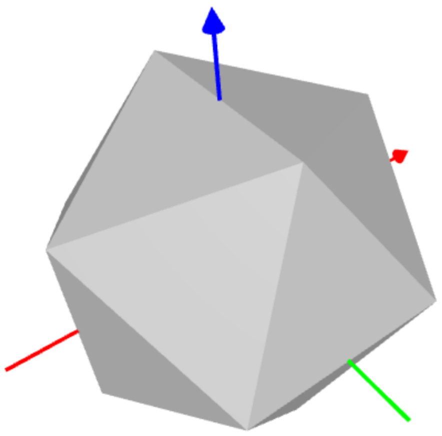


3.28 ICOSAHEDRON

Creates an icosahedron (symmetric object with 21 identical faces):

```
i = ICOSAHEDRON()
SHOW(i)
```

Output:



4 Selected Other Shapes

4.1 CONVEXHULL

Creates a convex hull of a given set of 3D points:

```
ch = CHULL(POINT(3, 2, 1), POINT(1, -1, 2), POINT(5, 5, 5), POINT(0, 0, -3), POINT(3, -2
SHOW(ch)
```

4.2 GRID

Creates a set of 1D intervals, using negative numbers as spaces and positive numbers as interval lengths:

```
g1 = GRID(1, -0.5, 2, -0.5, 1)
```

4.3 PRODUCT

Creates a Cartesian product of either two 1D objects (result will be a 2D object) or a 2D and a 1D object (result will be a 3D object). Can be used with GRID:

```
g1 = GRID(1, -0.5, 2, -0.5, 1)
g2 = GRID(0.5, -0.5, 0.5, -0.5, 0.5, -0.5, 0.5)
g = PRODUCT(g1, g2)
SHOW(g)
```

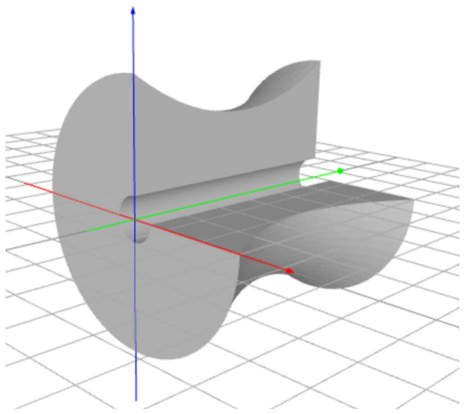
5 Rotational Bezier Objects

5.1 ROTATIONAL BEZIER SOLID

Rotational Bezier solids are obtained by rotating about the Y axis a Bezier curve located in the first quadrant. The command `ROSOL` is used as follows:

```
points = [POINT(3, 0), POINT(1, 3), POINT(3, 6)]
obj = ROSOL(points, 270, 0.5)
SHOW(obj)
```

Here, `points` is a list of points in the XY plane that define the Bezier curve. The second parameter - here 270 degrees - is the angle of evolution. Its default value is 360 degrees. The third parameter - here 0.5 - is the minimum radius. Its default value is 0, meaning that the solid extends all the way from the axis of rotation to the Bezier curve. There are two additional optional parameters that specify the division in the Y and X direction, respectively. Their default values are 32 and 32. The output of the above code is here:

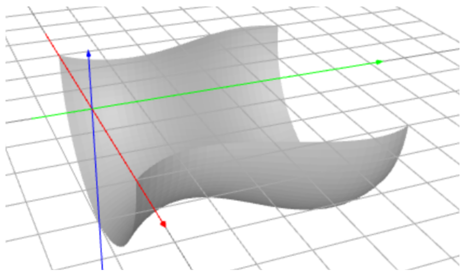


5.2 ROTATIONAL BEZIER SURFACE

Rotational Bezier surfaces are created using the command `ROSURF`. Its use is analogous to the ROSOL command, except that there is no minimum radius. They are 2D objects and as such, Boolean operations between them and 3D objects are not permitted. Here is a sample rotational surface:

```
points = [POINT(3, 0), POINT(1, 2), POINT(4, 4), POINT(3, 6)]
obj = ROSURF(points, 180)
SHOW(obj)
```

Output:

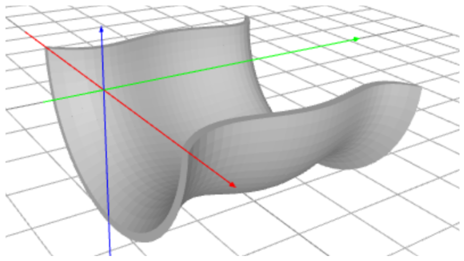


5.3 ROTATIONAL BEZIER SHELL

Rotational Bezier shells are similar to rotational Bezier surfaces, but they have nonzero thickness. This makes them 3D objects - one can subtract from them other 3D objects etc. (this is not possible with rotational Bezier surfaces). They are created using the `ROSHELL` command that is analogous to `ROSURF` but has an additional second argument - the thickness of the shell. Here is a sample rotational shell:

```
points = [POINT(3, 0), POINT(1, 2), POINT(5, 3), POINT(1, 5), POINT(3, 6)]
obj = ROSHELL(points, 0.2, 180)
SHOW(obj)
```

Output:



6 Transformations

6.1 MOVE

Moves objects in 2D or 3D space. This command has a versatile format. First of all, it is possible to move an object in the X-direction only by using just the object's name and one number as arguments:

```
c = SQUARE(1)
MOVE(c, 1.5)
SHOW(c)
```

This will move the object `c` by 2 in the X-direction and by 0.5 in the Y-direction:

```
c = SQUARE(1)
MOVE(c, 2, 0.5)
SHOW(c)
```

This example will move the object `c` by 4, 3.5, 3 in the X, Y and Z directions, respectively:

```
c = CUBE(1)
MOVE(c, 4, 3.5, 3)
SHOW(c)
```

It is also possible to move the object `c` in just the Y-direction as follows:

```
c = CUBE(1)
MOVE(c, 0.5, Y)
SHOW(c)
```

Or just in the Z-direction:

```
c = CUBE(1)
MOVE(c, 1.1, Z)
SHOW(c)
```

6.2 ROTATE

Rotates objects in 2D or 3D space about the X, Y or Z axis. Given are angle in degrees and axis of rotation (X, Y or Z):

```
c = CYL(0.5, 3)
ROTATE(c, 30, X)
SHOW(c)
```

The angle corresponds to counter-clockwise orientation. For clockwise rotations, use negative angles. The axis of rotation can be omitted and then Z is used by default:

```
c = RECTANGLE(2, 1)
ROTATE(c, 30)
SHOW(c)
```

Optional last argument is the center point of rotation. 2D example:

```
c = RECTANGLE(2, 1)
ROTATE(c, 30, POINT(1, 0.5))
SHOW(c)
```

3D example:

```
c = CYL(0.5, 3)
ROTATE(c, 30, Y, POINT(0.5, 0, 3))
SHOW(c)
```

6.3 ROTATERAD

Same as `ROTATE` but the angle is in radians.

6.4 SCALE

Scales (stretches, shrinks) 2D or 3D objects. It has a versatile format which is best illustrated on the following examples. Let's stretch the square `c` two times in the X direction:

```
c = SQUARE(1)
SCALE(c, 2, X)
SHOW(c)
```

Shrink the square `c` to 1/2 length in the Y direction:

```
c = SQUARE(1)
SCALE(c, 0.5, Y)
SHOW(c)
```

Stretch the square `c` 3.5 times in the Z direction:

```
c = SQUARE(1)
SCALE(c, 3.5, Z)
SHOW(c)
```

Stretch the cube `c` two times in the X direction, halve it in the Y direction, and leaves it unchanged in the Z direction.

```
c = CUBE(1)
SCALE(c, 2, 0.5, 1)
SHOW(c)
```

6.5 MIRROR

6.6 EXTRUDE

Extrudes a 2D object to 3D given extrusion height, angle, and number of steps:

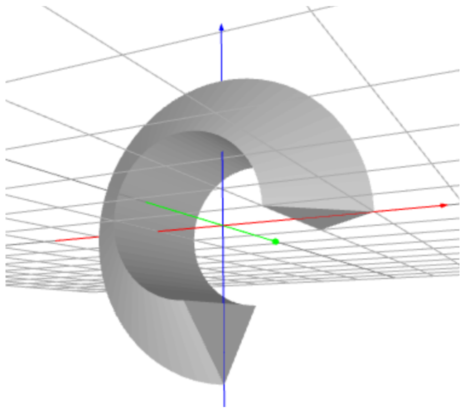
```
base = CIRCLE(1, 6)
e = E(base, 1.0, 180.0, 100)
C(e, GOLD)
SHOW(e)
```

6.7 REVOLVE

Revolves a given 2D object in the XY plane about the Y axis by a given angle. For best results, the 2D object should be on the right of the Y axis (the X coordinates of all its points should be positive). Example:

```
t = TRIANGLE(POINT(1, -1), POINT(2, 0), POINT(1, 1))
r = REVOLVE(t, 270, 48)
SHOW(r)
```

Output:

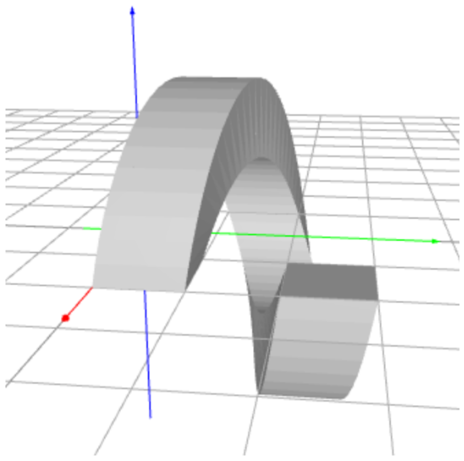


6.8 SPIRAL

Creates a 3D spiral using a given 2D object in the XY plane and an elevation value. The elevation value is per 360 degrees:

```
q = QUAD(POINT(1, 0), POINT(2, 0), POINT(2, 1), POINT(1, 1))
r = SPIRAL(q, 360, 2)
SHOW(r)
```

Output:



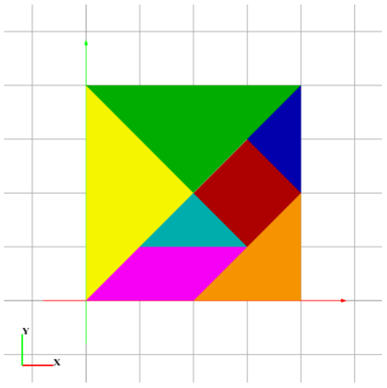
7 Tangrams

PLaSM provides pre-defined keywords for the seven tangram pieces. Referring to the figure below, the green, yellow, blue, red, cyan, pink and orange pieces are named `TANGRAM1()`, `TANGRAM2()`, ..., `TANGRAM7()`, respectively. Together they form a unit square $(0, 1) \times (0, 1)$. For illustration, the code

```
g = TANGRAM1()
```

```
y = TANGRAM2()
b = TANGRAM3()
r = TANGRAM4()
c = TANGRAM5()
p = TANGRAM6()
o = TANGRAM7()
SHOW(g, y, b, r, c, p, o)
```

displays all pieces in their basic configuration:



One can build many tangrams by rotating and moving these 2D objects in the XY plane.

8 Boolean Operations

8.1 SUBTRACT

The command `SUBTRACT(a, b)` subtracts object `b` from object `a`, changing object `a` on the way:

```
a = CUBE(1)
b = CYL(0.5, 2)
SUBTRACT(a, b)
SHOW(a)
```

Both `a` and `b` can be lists of objects.

8.2 DIFFERENCE (D)

The command `DIFFERENCE(a, b)` calculates and returns the difference of objects `a` and `b`, without changing any of the objects:

```
a = CUBE(1)
b = CYL(0.5, 2)
c = DIFFERENCE(a, b)
SHOW(c)
```

Both `a` and `b` can be lists of objects.

8.3 UNION (U)

The command `UNION` returns the union of two or more objects, without changing any of the objects. In fact, the `UNION` command returns a Python list of objects, so unions can easily be decomposed back into the original objects. Example:

```
a = CUBE(1)
b = COLOR(a, BLUE)
MOVE(b, 1, 0, 0)
COLOR(b, RED)
c = UNION(a, b)
SHOW(c)
```

8.4 INTERSECTION (I)

The command `INTERSECTION` creates and returns the intersection of two or more objects, without changing any of the objects:

```
a = CUBE(1)
b = CYL(0.5, 2)
c = ROTATE(b, 90, 1)
d = INTERSECTION(a, b, c)
SHOW(d)
```

8.5 WELD

Creates compound objects that are firmly ”welded” together - a welded object cannot be decomposed back into original objects. The result of the WELD operation is a single object, not a list. Since every object can only have one color, the color of the first object in the group is used to color the entire welded object. Example (compare with the example from UNION):

```
a = CUBE(1)
b = COLOR(a, BLUE)
MOVE(b, 1, 0, 0)
COLOR(b, RED)
c = WELD(a, b)
SHOW(c)
```

8.6 XOR

9 Colors

9.1 COLOR

Assigns colors to objects, using a large number of both predefined and custom colors:

```
c = CUBE(1)
COLOR(c, BLUE)
SHOW(c)
```

9.2 Predefined colors

If the color you are looking for has a name, try to type it in caps letters. Chances are that PLaSM will know it. And if not, send it to us in email and we will add it! Here is a list of selected predefined colors: GRAY, GREY, GREEN, LIGHTGREEN, DARKGREEN, BLACK, BLUE, LIGHTBLUE, DARKBLUE, BROWN, LIGHTBROWN, DARKBROWN, CYAN, LIGHTCYAN, DARKCYAN, PINK, MAGENTA, LIGHTMAGENTA, DARKMAGENTA, ORANGE, LIGHTORANGE, DARKORANGE, PURPLE, INDIGO, VIOLET, WHITE, RED, LIGHTRED, DARKRED, YELLOW, LIGHTYELLOW, DARKYELLOW, STEEL, BRASS, COPPER, BRONZE, SILVER, GOLD.

9.3 Custom colors

Any color in PLaSM is defined as a triplet of integer numbers between 0 and 255, enclosed in square brackets. For example, the following code defines custom color `turquoise` and uses it to display a turquoise cube:

```
turquoise = [64, 224, 208]
a = CUBE(1)
COLOR(a, turquoise)
SHOW(a)
```

10 Erasing Parts of Objects, Splitting Objects

10.1 ERASE

The command `ERASE(obj, minval, maxval, axis)` removes from a 2D or 3D object `obj` all points whose coordinates on axis `axis` lie between `minval` and `maxval`. For example, the code

```
a = RING(2, 3)
ERASE(a, 1.5, 2.0, Y)
SHOW(a)
```

creates a ring `a` and then removes from it all points whose Y-coordinates lie between 1.5 and 2.

10.2 SPLIT

The command `part1, part2 = SPLIT(obj, coord, axis)` splits a 2D or 3D object `obj` into two parts `part1`, `part2` using a cutplane that is perpendicular to the axis `axis` and that passes through coordinate `coord` on that axis. For example, the code

```
s = SPHERE(2)
s1, s2 = SPLIT(s, 0, X)
MOVE(s2, 1, X)
SHOW(s1, s2)
```

halves the sphere `s` using a cutplane parallel to the YZ plane. The result are two objects `s1` and `s2`. Object `s2` is then moved away from `s1` by one unit to create a gap.

11 Creating Copies of Objects

11.1 COPY

Creates a copy of an object for further manipulation:

```
a = CUBE(1)
COLOR(a, BLUE)
b = COPY(a)
MOVE(b, 1, 0, 0)
COLOR(b, RED)
SHOW(a, b)
```

Commands such as COLOR, MOVE, ROTATE, SCALE etc. also return a copy of the transformed object, so the above code can be written as

```
a = CUBE(1)
b = COLOR(a, BLUE)
MOVE(b, 1, 0, 0)
COLOR(b, RED)
SHOW(a, b)
```

12 Displaying Objects

12.1 SHOW

Displays one or more objects:

```
c = CUBE(1)
CYL(0.5, 2)
ROTATE(d, 90, 1)
COLOR(c, RED)
COLOR(d, GOLD)
COLOR(e, SILVER)
SHOW(c, d, e)
```