- apply-matrices()
- cross-product()
- direction-vec()
- distance-vec()
- distance-vec-squared()
- length-vec()
- line-parametric()
- line-points()
- line-symmetric()
- mat-mult-vec()
- mat-rotate-x()
- mat-rotate-y()
- mat-rotate-z()
- normalize-vec()
- plane-coordinate()
- plane-hesse()
- plane-parametric()
- plane-point-normal()
- plane-points()
- sum-vec()

**Variables**

- dot-product
- plane-normal
- line-point-normal
- mat-rotate-iso

### apply-matrices

Multiplies a $\mathbb{R}^3$ vector with $\mathbb{R}^{3\times3}$ matrices

**Parameters**

```
apply-matrices(
  v: vector,
  ..m: matrices
) -> matrix
```

**v** `vector`

Vector

**..m** `matrices`

Matrices

### cross-product

Calculates the cross product of two vectors

```
#cross-product((1,3,2), (2,0,1))
```

```
(3, 3, -6)
```

#### Parameters

```
cross-product(
  (x1, y1, z1): vector,
  (x2, y2, z2): vector
) -> vector
```

**(x1, y1, z1)** `vector`

w

**(x2, y2, z2)** `vector`

v

### direction-vec

Calculates the direction vector of two vectors

```
#direction-vec((0,1,2), (2,0,1))
```

```
(2, -1, -1)
```

#### Parameters

```
direction-vec(
  from: vector,
  to: vector
) -> vector
```

**from** `vector`

Start vector

**to** `vector`

End vector

### distance-vec

Calculates the distance between two vectors

```
#distance-vec((0,1,2), (2,0,1))
```

```
2.449489742783178
```

**Parameters**

```
distance-vec(
  from: vector,
  to: vector
) -> vector
```

**from** `vector`

Start vector

**to** `vector`

End vector

### distance-vec-squared

Calculates the squared distance between two vectors

```
#distance-vec-squared((0,1,2), (2,0,1))
```

```
6
```

**Parameters**

```
distance-vec-squared(
  from: vector,
  to: vector
) -> vector
```

**from** `vector`

Start vector

**to** `vector`

End vector

### length-vec

Calculates the length of a vector

```
#length-vec((0,3,4))
```

```
5
```

**Parameters**

```
length-vec(v: vector) -> vector
```

**v** `vector`

Vector

### line-parametric

Constructs line from parametric form

**Parameters**

```
line-parametric(
  p: vector,
  d: vector
) -> line
```

**p** `vector`

Point on line

**d** `vector`

Direction vector

### line-points

Constructs line from given points

**Parameters**

```
line-points(
  a: vector,
  b: vector
) -> line
```

**a** `vector`

Point 1

**b** `vector`

Point 2

### line-symmetric

Constructs line from symmetric form

#### Parameters

```
line-symmetric(
  x: int float,
  dx: int float,
  y: int float,
  dy: int float,
  z: int float,
  dz: int float
) -> line
```

**x** `int` or `float`

x

**dx** `int` or `float`

x-intercept

**y** `int` or `float`

y

**dy** `int` or `float`

y-intercept

**z** `int` or `float`

z

**dz** `int` or `float`

z-intercept

### mat-mult-vec

Multiplies a $\mathbb{R}^{3\times3}$ matrix with a $\mathbb{R}^3$ vector

```
#mat-mult-vec((
    (1,0,0),
    (0,1,0),
    (0,0,1)
  ), (2,0,1))
```

```
(2, 0, 1)
```

**Parameters**

```
mat-mult-vec(
  ((a, b, c), (d, e, f), (g, h, i)): matrix,
  (x, y, z): vector
) -> vector
```

**((a, b, c), (d, e, f), (g, h, i))** `matrix`

Matrix

**(x, y, z)** `vector`

Vector

**mat-rotate-x**

Constructs a rotation matrix in the x direction

**Parameters**

```
mat-rotate-x(x: int float) -> matrix
```

**x** `int` or `float`

Amount to rotate by

**mat-rotate-y**

Constructs a rotation matrix in the y direction

**Parameters**

```
mat-rotate-y(y: int float) -> matrix
```

**y** `int` or `float`

Amount to rotate by

**mat-rotate-z**

Constructs a rotation matrix in the y direction

**Parameters**

```
mat-rotate-z(z: int float) -> matrix
```

**z** `int` or `float`

Amount to rotate by

### normalize-vec

Normalizes a vector

```
#normalize-vec((0,3,4))
```

```
(0.0, 0.6, 0.8)
```

#### Parameters

```
normalize-vec(v: vector) -> vector
```

**v** `vector`

Vector

### plane-coordinate

Constructs plane from coordinate form

#### Parameters

```
plane-coordinate(
  x: int float,
  y: int float,
  z: int float,
  d: int float
) -> plane
```

**x** `int` or `float`

x

**y** `int` or `float`

y

**z** `int` or `float`

z

**d** `int` or `float`

Distance

### plane-hesse

Constructs plane from hesse form

**Parameters**

```
plane-hesse(
  (x, y, z): vector,
  d: int float
) -> plane
```

**(x, y, z)** `vector`

Normal vector

**d** `int` or `float`

Distance

### plane-parametric

Constructs plane from parametric form

**Parameters**

```
plane-parametric(
  p: vector,
  v: vector,
  w: vector
) -> plane
```

**p** `vector`

Point on plane

**v** `vector`

Non-colinear vector 1

**w** `vector`

Non-colinear vector 2

### plane-point-normal

Constructs plane from point normal form

**Parameters**

```
plane-point-normal(
  n: vector,
  p: vector
) -> plane
```

**n** `vector`

Normal vector

**p** `vector`

Point on plane

### plane-points

Constructs plane from given points

**Parameters**

```
plane-points(
  a: vector,
  b: vector,
  c: vector
) -> plane
```

**a** `vector`

Point 1

**b** `vector`

Point 2

**c** `vector`

Point 3

### sum-vec

Adds an arbitrary amount of vectors

```
#sum-vec((0,1,2), (2,0,1), (-1, 0, 1))
```

```
(1, 1, 4)
```

**Parameters**

```
sum-vec(..v: vectors) -> vector
```

**..v** `vectors`

The vectors to sum up

### dot-product `vector`

Calculates the dot product of two vectors

```
#dot-product((1,3,2), (2,0,1))
```

```
4
```

### plane-normal `plane`

Constructs plane from normal form (alias for `plane-hesse`)

### line-point-normal `line`

Constructs line from point-normal form (alias for `line-parametric`)

### mat-rotate-iso `matrix`

Isometric rotation matrix