wasabi.geom Documentation

Release 0.1

Daniel Pope

March 21, 2016

Contents

Contents:

CHAPTER 1

Vectors

class `wasabi.geom.vector.`**`Vector`**

Two-dimensional float vector implementation.

`angle`

The angle the vector makes to the positive x axis in the range (-180, 180].

`angle_to`(*other*)

Compute the angle made to another vector in the range [0, 180].

Parameters

other [Vector] The vector with which to compute the angle.

`cross`(*other*)

Compute the cross product with another vector.

Parameters

other [Vector] The vector with which to compute the cross product.

`distance_to`(*other*)

Compute the distance to another point vector.

Parameters

other [Vector] The point vector to which to compute the distance.

`dot`(*other*)

Compute the dot product with another vector.

Parameters

other [Vector] The vector with which to compute the dot product.

`is_zero`

Flag indicating whether this is the zero vector.

`length`

The length of the vector.

`length2`

The square of the length of the vector.

`normalised`()

Compute the vector scaled to unit length.

`normalized`()

Compute the vector scaled to unit length.

perpendicular()
Compute the perpendicular.

project(*other*)
Compute the projection of another vector onto this one.

Parameters

other [Vector] The vector of which to compute the projection.

rotated(*angle*)
Compute the vector rotated by an angle.

Parameters

angle [float] The angle (in degrees) by which to rotate.

safe_normalised()
Compute the vector scaled to unit length, or some unit vector if it was the zero vector.

safe_normalized()
Compute the vector scaled to unit length, or some unit vector if it was the zero vector.

safe_scaled_to(*length*)
Compute the vector scaled to a given length, or just return the vector if it was the zero vector.

Parameters

length [float] The length to which to scale.

scaled_to(*length*)
Compute the vector scaled to a given length.

Parameters

length [float] The length to which to scale.

signed_angle_to(*other*)
Compute the signed angle made to another vector in the range (-180, 180].

Parameters

other [Vector] The vector with which to compute the angle.

x
The horizontal coordinate.

y
The vertical coordinate.

class wasabi.geom.vector.**Matrix**(*x11*, *x12*, *x21*, *x22*)
A 2x2 matrix.

This can be used to optimise a transform (such as a rotation) on multiple vectors without recomputing terms.

To transform a vector with this matrix, use premultiplication, ie. for Matrix M and Vector v,

```
t = M * v
```

static rotation(*angle*)
A rotation matrix for angle a.

wasabi.geom.vector.**v**(**args*)
Construct a vector from an iterable or from multiple arguments. Valid forms are therefore: `v((x, y))` and `v(x, y)`.

CHAPTER 2

Lines

A collection of classes representing lines and linear objects.

Yes, there are two implementations of line segments, `LineSegment` and `Segment`; these will be merged in future.

class `wasabi.geom.lines.`**`Line`**(*direction*, *distance*)

Two-dimensional vector (directed) line implementation.

Lines are defined in terms of a perpendicular vector and the distance from the origin.

The representation of the line allows it to partition space into an 'outside' and an inside.

`altitude`(*point*)

Return the (signed) distance to a point.

Parameters

point [Vector] The point to measure the distance to.

`distance_to`(*point*)

Return the (signed) distance to a point.

Parameters

point [Vector] The point to measure the distance to.

classmethod `from_points`(*first*, *second*)

Create a Line object from two (distinct) points.

Parameters

first, second [Vector] The vectors used to construct the line.

`is_inside`(*point*)

Determine if the given point is left of the line.

Parameters

point [Vector] The point to locate.

`is_on_left`(*point*)

Determine if the given point is left of the line.

Parameters

point [Vector] The point to locate.

`is_on_right`(*point*)

Determine if the given point is right of the line.

Parameters

point [Vector] The point to locate.

`mirror`(*point*)

Reflect a point in the line.

Parameters

point [Vector] The point to reflect in the line.

`offset`
The projection of the origin onto the line.

`parallel` (*point*)
Return a line parallel to this one through the given point.

Parameters
point [Vector] The point through which to trace a line.

`perpendicular` (*point*)
Return a line perpendicular to this one through the given point. The orientation of the line is consistent with `Vector.perpendicular`.

Parameters
point [Vector] The point through which to trace a line.

`project` (*point*)
Compute the projection of a point onto the line.

Parameters
point [Vector] The point to project onto the line.

`reflect` (*point*)
Reflect a point in the line.

Parameters
point [Vector] The point to reflect in the line.

class `wasabi.geom.lines.`**`LineSegment`** (*line*, *min_dist*, *max_dist*)
Two-dimensional vector (directed) line segment implementation.

Line segments are defined in terms of a line and the minimum and maximum distances from the base of the altitude to that line from the origin. The distances are signed, strictly they are multiples of a vector parallel to the line.

`distance_to` (*point*)
Return the shortest distance to a given point.

Parameters
point [Vector] The point to measure the distance to.

`end`
One endpoint of the line segment (corresponding to 'max_dist').

classmethod `from_points` (*first*, *second*)
Create a LineSegment object from two (distinct) points.

Parameters
first, second [Vector] The vectors used to construct the line.

`length`
The length of the line segment.

`mid`
The midpoint of the line segment.

`project` (*point*)
Compute the projection of a point onto the line segment.

Parameters
point [Vector] The point to minimise the distance to.

`start`
One endpoint of the line segment (corresponding to 'min_dist').

class `wasabi.geom.lines.`**`Segment`** (*p1*, *p2*)
A 2D line segment between two points p1 and p2.

A segment has an implied direction for some operations - p1 is the start and p2 is the end.

`intersects`(*other*)
Determine if this segment intersects a convex polygon.

Returns None if there is no intersection, or a scalar which is how far along the segment the intersection starts. If the scalar is positive then the intersection is partway from p1 to p2. If the scalar is negative then p1 is inside the shape, by the corresponding distance (in the direction of the object)

`length`
The length of the segment.

`scale_to`(*dist*)
Scale the segment to be of length dist.

This returns a new segment of length dist that shares p1 and the direction vector.

`truncate`(*dist*)
Scale the segment to be of length dist.

This returns a new segment of length dist that shares p1 and the direction vector.

class `wasabi.geom.lines.`**`Projection`**(*min*, *max*)
A wrapper for the extent of a projection onto a line.

class `wasabi.geom.lines.`**`PolyLine`**(*vertices=[]*)
A set of points connected into line

CHAPTER 3

Polygons

class `wasabi.geom.poly.`**`Rect`**

2D rectangle class.

classmethod **`as_bounding`**(*points*)

Construct a Rect as the bounds of a sequence of points.

Parameters **`points`** – An iterable of the points to bound.

`bottomleft`()

The bottom left point.

`bottomright`()

The bottom right point.

`contains`(*p*)

Return True if the point p is within the Rect.

classmethod **`from_blwh`**(*bl*, *w*, *h*)

Construct a Rect from its bottom left and dimensions.

classmethod **`from_cwh`**(*c*, *w*, *h*)

Construct a Rect from its center and dimensions.

classmethod **`from_points`**(*p1*, *p2*)

Construct the smallest Rect that contains the points p1 and p2.

`get_aabb`()

Return the axis-aligned bounding box of the Rect - ie. self.

`h`

Height of the rectangle.

`intersection`(*r*)

The intersection of this Rect with another.

`overlaps`(*r*)

Return True if this Rect overlaps another.

Not to be confused with .intersects(), which works for arbitrary convex polygons and computes a separation vector.

`points`

A list of the points in the rectangle.

`topleft`()

The top left point.

topright()
    The top right point.

translate(*off*)
    Return a new Rect translated by the vector *off*.

w
    Width of the rectangle.

class wasabi.geom.poly.**Triangle**(*base*, *primary*, *secondary*)
Two-dimensional vector (oriented) triangle implementation.

area
    The unsigned area of the triangle.

first
    The point at the end of the primary vector.

classmethod from_points(*base*, *first*, *second*)
    Create a Triangle object from its three points.

Parameters

base [Vector] The base point of the triangle.

first*, *second [Vector] The other two points of the triangle.

is_clockwise
    True if the primary and secondary are clockwise.

second
    The point at the end of the secondary vector.

class wasabi.geom.poly.**ConvexPolygon**(*points*)

to_tri_strip()
    Generate a list of the points in triangle-strip order

class wasabi.geom.poly.**Polygon**(*vertices=None*)
Mutable polygon, possibly with holes, multiple contours, etc.

This exists mainly as a wrapper for polygon triangulation, but also provides some useful methods.

add_contour(*vertices*)
    Adds a contour

polylines_facing(*v*, *threshold=0*)
    Compute a list of PolyLines on the edge of this contour whose normals face v.

    threshold the value of the segment normal dot v required to include a segment in the polyline.

CHAPTER 4

Spatial Hashes

A spatial hash is one way of indexing objects in space. As Python programmers, we should perhaps call them spatial dicts, but let's go with the literature on this one. Like a dict, a spatial hash has O(1) properties.

The basic principle is to split space into an infinite number of cells – each cell can contain an arbitrary number of objects. A cell that is empty simply isn't stored. Largely taken from

http://themonkeyproject.wordpress.com/2011/05/18/introduction-to-spatial-hashes/

class `wasabi.geom.spatialhash.`**`SpatialHash`**(*cell_size=250.0*)

> **`add_rect`**(*r*, *obj*)
> Add an object obj with bounds r.
>
> **`potential_intersection`**(*r*)
> Get a set of all objects that potentially intersect obj.
>
> **`remove_rect`**(*r*, *obj*)
> Remove an object obj which had bounds r.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index