wasabi.geom Documentation

Release 2.0.1

Daniel Pope

Apr 20, 2022

CONTENTS

`wasabigeom` is a library of fast geometry types for Python games. It started life as a pure Python library but is now implemented in optimised Cython code.

Contents:

CHAPTER

ONE

API

1.1 Vectors

class wasabigeom.**vec2**(**args*)

Two-dimensional float vector implementation.

`vec2` is a 2D vector that supports standard mathematical operations like addition and multiplication:

```
>>> vec2(3, 5) + (1, 2)
vec2(4.0, 7.0)
>>> vec2(1, 2) * 0.5
vec2(0.5, 1.0)
```

`vec2` is immutable and hashable.

angle(*self*) → double

The angle the vector makes to the positive x axis, in radians.

angle_to(*self*, *other*)

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

Parameters

other [vec2] The vector with which to compute the angle.

cross(*self*, *other*)

Compute the cross product with another vector.

Parameters

other [vec2] The vector with which to compute the cross product.

distance_to(*self*, *other*)

Compute the distance to another point vector.

Parameters

other [vec2] The point vector to which to compute the distance.

dot(*self*, *other*) → double

Compute the dot product with another vector.

Parameters

other [vec2] The vector with which to compute the dot product.

`static from_polar`(*double length*, *double angle*)

Construct a vec2 from polar coordinates.

`is_zero`(*self*) → bool

Test if this is the zero vector.

`length`(*self*) → double

Return length of the vector.

`length_squared`(*self*) → double

Return the square of the length of the vector.

`normalized`(*self*)

Compute the vector scaled to unit length.

`perpendicular`(*self*)

Compute the perpendicular.

`project`(*self*, *other*)

Compute the projection of another vector onto this one.

Parameters

other [vec2] The vector of which to compute the projection.

`rotated`(*self*, *double angle*)

Compute the vector rotated by an angle.

Parameters

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

`safe_normalized`(*self*)

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

`safe_scaled_to`(*self*, *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`(*self*, *double length*)

Compute the vector scaled to a given length.

Parameters

length [float] The length to which to scale.

`signed_angle_to`(*self*, *other*)

Compute the signed angle made to another vector in the range.

Parameters

other [vec2] The vector with which to compute the angle.

`to_polar`(*self*)

1.2 Transformations

class wasabigeom.**Transform**(*double a*, *double b*, *double c*, *double d*, *double e*, *double f*)

A 3x3 matrix representing an affine transform in 2D.

The values of the matrix are

```
(a b c)
(d e f)
(0 0 1)
```

These matrices always a bottom row (0, 0, 1), and knowing this allows us to skip some multiplications and drop some terms vs multiplication of an arbitrary 3x3 matrix (eg. in numpy).

Transforms can be multiplied together to chain them:

```
>>> a = Transform.build(xlate=(1, 2), rot=0.5)
>>> b = Transform.build(rot=-0.5)
>>> a * b
Transform(1., 0., 1.,
          0., 1., 2.)
```

Order matters! Matrix multiplication is not commutative (but it is associative). To do transformation `A` followed by `B` is `B * A`.

Transforms support the buffer protocol, meaning that they can be converted to numpy arrays:

```
>>> numpy.asarray(Transform(2., 0., 1.,
...                         0., 1., 2.))
array([[2., 0., 1.],
       [0., 1., 2.]])
```

static build(*xlate=(0, 0)*, *double rot=0.0*, *scale=(1, 1)*)

Build a Transform from a translation, rotation and scale.

The operation order is scale first, then rotation, then translation. To apply the operations different order, build Transforms representing the individual operations and then multiply them.

factorise(*self*)

Split the transformation into translation, rotation, and scale.

This operation is approximate because it doesn't calculate or return skew components, and therefore cannot represent all transforms.

static identity()

Return a new identity transform.

inverse(*self*)

Return the inverse transformation.

Raise ZeroDivisionError if the matrix is not invertible (eg. it has a scale factor of 0).

set(*self*, *xlate=(0, 0)*, *double rot=0.0*, *scale=(1, 1)*)

Overwrite the transform using the given parameters.

transform

Transform a buffer of coordinates using this matrix.

Coordinates may be floats or doubles.

If output_view is given, then it will be populated with the result rather than returning a new numpy arry. It should be a writable buffer object matching the shape of the input.

Parameters

- **input_view** – A 2D array of 2 coordinates to transform.
- **output_view** – A 2D array of 2 coordinates to write to, or None to allocate a new array.

Returns A numpy.ndarray, unless output_view is given.

class wasabigeom.**Matrix**(*double x11*, *double x12*, *double x21*, *double 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 vec2 v,

```
t = M * v
```

Deprecated since version 2.1.0: Use *wasabigeom.Transform* instead.

static **identity**()

static **rotation**(*double angle*)

A rotation matrix for angle a.

1.3 Geometric Primitives

class wasabigeom.**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(*self*, *vertices*)

Adds a contour

mirror(*self*, *plane*)

polylines_facing(*self*, *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.

class wasabigeom.**ConvexPolygon**(*points*)

edges(*self*)

segments(*self*)

to_tri_strip(*self*)

Generate a list of the points in triangle-strip order

class `wasabigeom.Triangle`(*base*, *primary*, *secondary*)

Two-dimensional vector (oriented) triangle implementation.

area(*self*)

The unsigned area of the triangle.

first(*self*)

The point at the end of the primary vector.

classmethod from_points(*cls*, *base*, *first*, *second*)

Create a Triangle object from its three points.

Parameters

base [vec2] The base point of the triangle.

first*, *second [vec2] The other two points of the triangle.

is_clockwise(*self*)

True if the primary and secondary are clockwise.

second(*self*)

The point at the end of the secondary vector.

class `wasabigeom.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*)

Line.distance_to(self, point) Return the (signed) distance to a point.

Parameters

point [vec2] The point to measure the distance to.

distance_to(*self*, *point*)

Return the (signed) distance to a point.

Parameters

point [vec2] The point to measure the distance to.

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

Create a Line object from two (distinct) points.

Parameters

first*, *second [vec2] The vectors used to construct the line.

is_inside(*point*)

Line.is_on_left(self, point) Determine if the given point is left of the line.

Parameters

point [vec2] The point to locate.

is_on_left(*self*, *point*)

Determine if the given point is left of the line.

Parameters

point [vec2] The point to locate.

`is_on_right`(*self*, *point*)

Determine if the given point is right of the line.

Parameters

point [vec2] The point to locate.

`mirror`(*point*)

Line.reflect(self, point) Reflect a point in the line.

Parameters

point [vec2] The point to reflect in the line.

`offset`(*self*)

The projection of the origin onto the line.

`parallel`(*self*, *point*)

Return a line parallel to this one through the given point.

Parameters

point [vec2] The point through which to trace a line.

`perpendicular`(*self*, *point*)

Return a line perpendicular to this one through the given point. The orientation of the line is consistent with `vec2.perpendicular`.

Parameters

point [vec2] The point through which to trace a line.

`project`(*self*, *point*)

Compute the projection of a point onto the line.

Parameters

point [vec2] The point to project onto the line.

`reflect`(*self*, *point*)

Reflect a point in the line.

Parameters

point [vec2] The point to reflect in the line.

`class wasabigeom.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`(*self*, *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)

`property length`

The length of the segment.

project_to_axis(*self*, *axis*)

scale_to(*self*, *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*)

Segment.scale_to(self, 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 wasabigeom.**PolyLine**(*vertices=[]*)

A set of points connected into line

segments(*self*)

class wasabigeom.**Projection**(*min*, *max*)

A wrapper for the extent of a projection onto a line.

intersection(*self*, *other*)

1.4 Bounds and collision tests

class wasabigeom.**Rect**(*l*, *r*, *b*, *t*)

2D rectangle class.

classmethod as_bounding(*cls*, *points*)

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

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

bottomleft(*self*)

The bottom left point.

bottomright(*self*)

The bottom right point.

contains(*self*, *p*)

Return True if the point p is within the Rect.

property edges

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

Construct a Rect from its bottom left and dimensions.

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

Construct a Rect from its center and dimensions.

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

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

get_aabb(*self*)

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

property h

Height of the rectangle.

intersection(*self*, *r*)

The intersection of this Rect with another.

overlaps(*self*, *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.

property points

A list of the points in the rectangle.

topleft(*self*)

The top left point.

topright(*self*)

The top right point.

translate(*self*, *off*)

Return a new Rect translated by the vector *off*.

property w

Width of the rectangle.

class wasabigeom.**SpatialHash**(*cell_size=250.0*)

add_rect(*self*, *r*, *obj*)

Add an object obj with bounds r.

potential_intersection(*self*, *r*)

Get a set of all objects that potentially intersect obj.

remove_rect(*self*, *r*, *obj*)

Remove an object obj which had bounds r.

1.5 Rasterisation

wasabigeom.**bresenham**(*int64_t x0*, *int64_t y0*, *int64_t x1*, *int64_t y1*)

Yield integer coordinates on the line from (x0, y0) to (x1, y1).

Input coordinates should be integers. The result will contain both the start and the end point.

CHAPTER

TWO

CHANGELOG

2.1 2.1.1 - released 2022-04-20

- Fix package compatibility with Python 3.10

2.2 2.1.0 - released 2021-09-24

- New: add *wasabigeom.Transform* for 2D affine transformations.
- Add vec2.from_polar() static method.
- Can construct from any 2-sequence of floats

2.3 2.0.1 - released 2020-09-27

- Several bugfixes, particularly around multiplying/dividing by *int* and *nonvec* + *vec*

2.4 2.0.0 - released 2020-09-25

- Breaking Change: module name changed from `wasabi.geom` to `wasabigeom`
- Breaking Change: `Vector` class renamed to `wasabigeom.vec2`
- Breaking Change: `vec2.angle()` and other functions now return radians.
- New: *wasabigeom.bresenham()*
- New: Cythonised the sources; hand-optimised `vec2`

2.5 0.1.3 - released 2012ish

- Original, pure-Python release

INDEX

R

S

T

V

W