Package ‘ptinpoly’

October 14, 2022

Title  Point-in-Polyhedron Test (2D and 3D)
Version  2.8
Date  2020-05-31
Author  Jose M. Maisog, Yuan Wang, George Luta, Jianfei Liu
Maintainer  Jose M. Maisog <bravas02@gmail.com>
Description  Function pip3d() tests whether a point in 3D space is within, exactly on, or outside an enclosed surface defined by a triangular mesh. Function pip2d() tests whether a point in 2D space is within, exactly on, or outside a polygon. For a reference, see: Liu et al., A new point containment test algorithm based on preprocessing and determining triangles, Computer-Aided Design 42(12):1143-1150.
License  GPL-2
LazyLoad  yes
Depends  R (>= 2.10), misc3d
Suggests  rgl, geometry
URL  http://ptinpoly.pbworks.com
Repository  CRAN
Repository/R-Forge/Project  ptinpoly
Repository/R-Forge/Revision  41
Repository/R-Forge/DateTimeStamp  2020-05-31 12:01:50
Date/Publication  2020-06-02 15:10:07 UTC
NeedsCompilation  yes

R topics documented:

blocks2vf  ................................................................. 2
comb  ................................................................. 3
cube  ............................................................... 4
fractal  ............................................................... 5
pip2d  ............................................................... 5
pip3d  .............................................................. 7
spiral  .............................................................. 9
vf2blocks  ......................................................... 10
blocks2vf

Convertor from 3-Block Representation to Vertices-Faces

Description

Converts a polyhedron from the three-block representation to the vertices-faces representation.

Usage

blocks2vf(Block1, Block2, Block3)

Arguments

<table>
<thead>
<tr>
<th>Block1</th>
<th>M by 3 matrix containing the XYZ coordinates of vertex 1 of the M faces of the polyhedron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block2</td>
<td>M by 3 matrix containing the XYZ coordinates of vertex 2 of the M faces of the polyhedron</td>
</tr>
<tr>
<td>Block3</td>
<td>M by 3 matrix containing the XYZ coordinates of vertex 3 of the M faces of the polyhedron</td>
</tr>
</tbody>
</table>

Details

The values in the first output matrix can be floating point integers, representing the XYZ coordinates of the vertices of the polyhedron.

The values in the second output matrix will be integers with values running from 1 to N, where N is the number of vertices. A value of ‘1’ in this matrix, for example, represents the 1st vertex, i.e., the vertex defined by the first row in the matrix Vertices. Each row in this matrix defines a triangular face in the polyhedron.

This function is the inverse of the vf2blocks function.

Value

Returns a list of two matrices. The first is a N by 3 matrix containing the XYZ coordinates of the N vertices of the polyhedron. The second M by 3 matrix containing indices of the vertices defining the M faces. See the example below.

Note

This function requires the misc3d library.
Examples

# Load example data.
data(verts)
data(faces)

# Use vf2blocks to convert from vertices-faces representation to 3-block representation.
# Note double square brackets.
blocks = vf2blocks(verts,faces)
block1 = blocks[1]
block2 = blocks[2]
block3 = blocks[3]

# Now use blocks2vf to convert back to vertices-faces representation.
# 'verts2' and 'faces2' should encode the same polyhedron as the
# original 'verts' and 'faces', although perhaps in a different order.
# Note double square brackets.
vertsFaces = blocks2vf(block1,block2,block3)
verts2 = vertsFaces[1]
faces2 = vertsFaces[2]

Sample Data: Comb Polygon

Description

Sample data defining an enclosed comb 2D polygon.

Usage

data(comb)

Examples

# Load polygon.
data(comb)

# Plot the polygon.
plot(rbind(comb,comb[1,]),type="l")

# Generate 3333 random test points.
set.seed(1902)
n <- 3333
x1 <- rnorm(n); x2 <- rnorm(n)
X <- cbind(x1,x2)
queries <- as.matrix(X)

# Check whether test points are contained in the polygon.
# Most of these points will lie outside the polygon.
containment = pip2d(comb,queries);
Sample Data: Simple Cube

Description

This is sample data that defines a simple cube: eight vertices, and twelve triangles that make up the six faces.

Also included is an example matrix queries of five test points. The first test point is contained within the cube, the second through fourth test points lie exactly on the surface of the cube, and the fifth test point lies outside the cube.

Usage

data(verts)
data(faces)
data(queries)

Format

verts is an 8 by 3 matrix containing the XYZ coordinates of the vertices of a simple cube.

faces is a 12 by 3 matrix containing the indices of the vertices defining the twelve triangular faces making up the surface of the cube.

queries is a 5 by 3 matrix containing the XYZ coordinates of five test points to be tested for containment within the cube.

Examples

# Load sample data defining a simple cube.
data(verts)
data(faces)

# Also load sample data for five test points.
data(queries)

# Test whether each point in 'queries' is contained in
# the simple cube defined by 'verts' and 'faces'.
# This should return "1 0 0 0 -1".
containment = pip3d(verts,faces,queries);
Sample Data: Fractal Polygon

Description

Sample data defining an enclosed fractal 2D polygon.

Usage

data(fractal)

Examples

# Load polygon.
data(fractal)

# Plot the polygon.
plot(rbind(fractal,fractal[1,]),type="l")

# Generate 3333 random test points.
set.seed(1902)
n <- 3333
x1 <- rnorm(n) ; x2 <- rnorm(n)
X <- cbind(x1,x2)
queries <- as.matrix(X)

# Check whether test points are contained in the polygon.
# Most of these points will lie outside the polygon.
containment = pip2d(fractal,queries);

Test for Point Containment in 2D Polygon

Description

Tests whether points are contained within a two-dimensional polygon.

Usage

pip2d(Vertices,Queries)

Arguments

<table>
<thead>
<tr>
<th>Vertices</th>
<th>N by 2 matrix containing the XY coordinates of N vertices of the polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>P by 2 matrix containing the XY coordinates of P points to be tested for containment in the polygon defined by <code>Vertices</code></td>
</tr>
</tbody>
</table>
Details
The XY coordinates of the vertices are stored in order in the matrix Vertices. It is assumed that
the last vertex listed in the matrix is connected to the first vertex, so that the polygon does not have
a "hole".

Value
Returns a vector containing P values, one for each of the P points listed in the Queries matrix.
'1' indicates that the point is contained in the polygon.
'0' indicates that the point lies exactly on the surface of the polygon.
'-1' indicates that the point lies outside the polygon.
'-3' (error) indicates that the Vertices matrix didn’t have two columns
'-6' (error) indicates that the Queries matrix didn’t have two columns
'-8' (error) indicates computational error not otherwise specified

Note
The polygon defined by Vertices must be "non-leaky"; i.e., it must define an "inside" versus
"outside" and must not contain any holes.

References
W.P. Horn and D.L. Taylor, A theorem to determine the spatial containment of a point in a planar
39-64, 1967.
J.A. Baerentzen and H. Aanaes, Signed distance computation using the angle weighted pseudo-
2005.
J. Liu, Y.Q. Chen, J.M. Maisog, G. Luta, A new point containment test algorithm for polygon
composed of huge number of triangles, Computer-Aided Design, Volume 42, Issue 12, December
2010, Pages 1143-1150.
http://ptinpoly.pbworks.com/

Examples
#-------------------------------------------
# Load sample data defining a comb, spiral, and fractal.
data(comb)
data(spiral)
data(fractal)

# Plot the comb, spiral, and fractal.
plot(rbind(comb,comb[1,]),type="l")
plot(rbind(spiral,spiral[1,]),type="l")
plot(rbind(fractal,fractal[1,]),type="l")
# Generate 3333 random test points.
set.seed(1902)

n <- 3333
x1 <- rnorm(n) ; x2 <- rnorm(n)
X <- cbind(x1,x2)
queries <- as.matrix(X)

# Check whether test points are contained in the comb, spiral, and factal.
# Most of these points will lie outside the polygons.
containment1 <- pip2d(comb,queries);  
containment2 <- pip2d(spiral,queries);  
containment3 <- pip2d(fractal,queries);

---

### pip3d

#### Test for Point Containment in 3D Polyhedron

**Description**
Tests whether points are contained within a three-dimensional polyhedron.

**Usage**

`pip3d(Vertices, Faces, Queries)`

**Arguments**

- `Vertices` : N by 3 matrix containing the XYZ coordinates of N vertices of the polyhedron
- `Faces` : M by 3 matrix containing the indices of the three vertices defining the M triangular faces of the polyhedron
- `Queries` : P by 3 matrix containing the XYZ coordinates of P points to be tested for containment in the polyhedron defined by `Vertices` and `Faces`

**Details**

The values in the `Faces` matrix must be integers with values running from 1 to N, where N is the number of vertices. A value of '1' in this matrix, for example, represents the 1st vertex, i.e., the vertex defined by the first row in the matrix `Vertices`.

**Value**

Returns a vector containing P values, one for each of the P points listed in the `Queries` matrix.
- '1' indicates that the point is contained in the polyhedron.
- '0' indicates that the point lies exactly on the surface of the polyhedron.
- '-1' indicates that the point lies outside the polyhedron.
- '-2' (error) indicates that the polyhedron was topologically defective (e.g., had a hole)
'3' (error) indicates that the Vertices matrix didn’t have three columns
'4' (error) indicates that the Faces matrix didn’t have three columns
'5' (error) indicates that the Faces matrix was 0- rather than 1-offset
'6' (error) indicates that the Queries matrix didn’t have three columns
'7' (error) indicates that two faces in the polyhedron were too close to one another
'8' (error) indicates computational error not otherwise specified. A possible cause is when two faces of the polygon are extremely close to one another (imagine bending a cylindrical balloon until the two ends meet). Adjusting the spatial smoothness of the data may fix this problem.

Note

The polyhedron defined by Vertices and Faces must be "non-leaky"; i.e., it must define an "inside" versus "outside" and must not contain any holes.

For an example of external software that could potentially be used to fix defective polyhedra, see, e.g., PolyMender (http://www1.cse.wustl.edu/~taoju/code/polymender.htm).

Previous versions of this function would hang when there were more than two vertices very close to one another; this problem was discovered with a polyhedron in which there were multiple duplicate vertices and one triplicate vertex. The triplicate vertex fulfilled the case of "more than two vertices very close to one another", and caused the code to hang. The threshold for vertices that are very close to one another has been increased to three. It is advisable to make sure that a polyhedron does not have more than three vertices that are "very close to one another", and to make sure that there are no duplicate vertices. Similarly, it is advisable to make sure that a polyhedron does not have faces that that are extremely close to one another.

References

http://ptinpoly.pbworks.com/

Examples

#-------------------------------------------
# Simple Cube example.
#
# Load sample data defining a simple cube.
data(verts)
data(faces)
# Also load sample data for five test points.
data(queries)

# Test whether each point in 'queries' is contained in
# the simple cube defined by 'verts' and 'faces'.
# This should return "1 0 0 0 -1".
containment <- pip3d(verts,faces,queries);

#-------------------------------------------
# Torus example.
# Make a donut-shaped polyhedron.
torus <- parametric3d(fx = function(u,v) (1+0.25*cos(v))*cos(u),
fy = function(u,v) (1+0.25*cos(v))*sin(u),
fz = function(u,v) 0.25*sin(v),
u = seq(0,2*pi,length.out=10),
v = seq(0,2*pi,length.out=10),
engine = "none", color="orange", alpha=0.25)

# If desired, this torus can be rendered for visualization, e.g.:
# library(geometry)
# library(rgl)
# drawScene.rgl(torus)

# Convert the torus to vertices-faces representation.
ve <- misc3d:::t2ve(torus)
Vertices <- t(ve$vb)
Faces <- t(ve$ib)

# Generate 3333 random test points.
set.seed(1902)
n <- 3333
x1 <- rnorm(n) ; x2 <- rnorm(n) ; x3 <- rnorm(n)
X <- cbind(x1,x2,x3)
Queries <- as.matrix(X)

# Check whether test points are contained in the torus.
# Most of these points will lie outside the torus.
containment <- pip3d(Vertices,Faces,Queries);

#-------------------------------------------
# If you remove one of the faces of the cube, the resulting cube
# becomes "leaky". Running 'pip3d' on the resulting defective
# polyhedron will return -2.
# NOT RUN
#
# badcube <- faces[1:11,]
# containment <- pip3d(verts,badcube,queries);

Sample Data: Spiral Polygon
Description

Sample data defining an enclosed spiral 2D polygon.

Usage

data(spiral)

Examples

# Load polygon.
data(spiral)

# Plot the polygon.
plot(rbind(spiral,spiral[1,]),type="l")

# Generate 3333 random test points.
set.seed(1902)
n <- 3333
x1 <- rnorm(n) ; x2 <- rnorm(n)
X <- cbind(x1,x2)
queries <- as.matrix(X)

# Check whether test points are contained in the polygon.
# Most of these points will lie outside the polygon.
containment = pip2d(spiral,queries);

vf2blocks

Convertor from Vertices-Faces Representation to 3-Block Representation

Description

Converts a polyhedron from the vertices-faces representation to the three-block representation.

Usage

vf2blocks(Vertices,Faces)

Arguments

Vertices N by 3 matrix containing the XYZ coordinates of N vertices
Faces M by 3 matrix containing indices of vertices defining M faces

Details

The values in the Face matrix must be integers with values running from 1 to N, where N is the number of vertices. A value of '1' in this matrix, for example, represents the 1st vertex, i.e., the vertex defined by the first row in the matrix Vertices.

This function is the inverse of the blocks2vf function.


vf2blocks

Value

Returns a list of three matrices. The first is an M by 3 matrix containing the XYZ coordinates of vertex 1 of the M faces of the polyhedron. The second and third are similarly M by 3 matrices, but contain the XYZ coordinates of vertices 2 and 3 of the faces. See the example below.

Examples

# Load example data.
data(verts)
data(faces)

# Use vf2blocks to convert from vertices-faces representation to 3-block representation.
# Note double square brackets.
blocks = vf2blocks(verts,faces)
block1 = blocks[[1]]
block2 = blocks[[2]]
block3 = blocks[[3]]
Index

* datasets
  comb, 3
  cube, 4
  fractal, 5
  spiral, 10

* methods
  blocks2vf, 2
  pip2d, 5
  pip3d, 7
  vf2blocks, 10

blocks2vf, 2
comb, 3
cube, 4

faces (cube), 4
fractal, 5

pip2d, 5
pip3d, 7

queries (cube), 4
spiral, 9

verts (cube), 4
vf2blocks, 10