DSMRayTracerDoc Documentation

Release 0

Hayley

Aug 13, 2019

Contents

Contents:

CHAPTER 1

Main Program

Code to trace rays around a room. This code computes the trajectories only.

This is something I want to say that is not in the docstring.

CHAPTER 2

Input the parameters

The code saves the values for the parameters in a ray tracer

`ParameterInput.`**`ObstacleCoefficients`**`()`

Retrieve the information saved in Declare parameters. Then define the obstacle coefficients.

CHAPTER 3

Rays

Code to construct the ray-tracing objects rays

class Rays.**Ray**(*s*, *origin*, *direc*)

A ray is a representation of the the trajectory of a reflecting line and its reflections. Ray.points is an array of co-ordinates representing the collision points with the last term being the direction the ray ended in. And Ray.reflections is an array containing tuples of the angles of incidence and the number referring to the position of the obstacle in the obstacle list

mesh_multiref(*s*, *room*, *Nre*, *Mesh*, *Nra*, *nra*)

Takes a ray and finds the first Nre reflections within a room. As each reflection is found the ray is stepped through and information added to the Mesh. Inputs: Room- Obstacle co-ordinates, Nre- Number of reflections, Mesh- 3D array with each element corresponding to a sparse matrix. Nra- Total number of rays.

mesh_singleray(*s*, *room*, *Mesh*, *dist*, *calcvec*, *Nra*, *Nre*, *nra*)

Iterate between two intersection points and store the ray information in the Mesh

multiref(*s*, *room*, *m*)

Takes a ray and finds the first five reflections within a room

normal_mat(*s*, *Ncones*, *Nra*, *d*, *dist*, *h*)

Form a matrix of vectors representing the plane which is normal to d

number_cone_steps(*s*, *h*, *dist*, *Nra*)

find the number of steps taken along one normal in the cone

number_steps(*s*, *meshwidth*)

The number of steps along the ray between intersection points

obst_collision_point(*s*, *surface*)

intersection of the ray with a wall_segment

ray_length(*s*, *inter*)

The length of the ray upto the intersection

raytest(*s*, *room*, *err*)

Checks the reflection for errors

ref_angle(*s*, *room*)

Find the reflection angle of the most recent intersected ray.

reflect_calc(*s*, *room*)

finds the reflection of the ray inside a room

room_collision_point(*s*, *room*)

The closest intersection out of the possible intersections with the wall_segments in room. Returns the intersection point and the wall intersected with

CHAPTER 4

Room

Code to construct the mesh of the room

CHAPTER 5

Mesh

class DictionarySparseMatrix.**DS**(*s*, *Nx=1*, *Ny=1*, *Nz=1*, *na=1*, *nb=1*)

The DS class is a dictionary of sparse matrices. The keys for the dictionary are (i,j,k) such that i is in [0,Nx], j is in [0, Ny], and k is in [0,Nz].

asin(*s*)

Finds arcsin(s) for all terms theta != 0 in the DS s. Since all angles are in [0,pi/2] arcsin is not a problem.

Returns DSM with the same dimensions as s, with arcsin(s)=theta in the same positions as the corresponding theta terms.

cos(*s*)

Finds cos(theta) for all terms theta != 0 in the DS s.

Returns A DSM with the same dimensions with cos(theta) in the same position as the corresponding theta terms.

dense(*s*)

Fills the DSM s.

Returns A dense Nx*Ny*Nz*na*nb array with matching nonzero terms to the sparse matrix s and zeroes elsewhere.

dict_DSM_divideby_vec(*s*, *vec*)

Divide every column of the DSM s elementwise with the vector vec.

Parameters **vec** – a row vector with length na.

Return type a DSM 'out' with the same dimensions as s.

Returns out[x,y,z,k,j]=DSM[x,y,z,k,j]/vec[k]

dict_vec_divideby_DSM(*s*, *vec*)

Every column of the DSM s divides elementwise the vector vec.

Parameters **vec** – a row vector with length na.

Return type a DSM 'out' with the same dimensions as s.

Returns out[x,y,z,k,j]=vec[k]/DSM[x,y,z,k,j]

nonzero(*s*)

Find the indices of the nonzero terms in the DSM s.

Note: The indices are found by iterating through all keys (x,y,z) for the DSM s and finding the nonzero indices of the corresponding sparse matrix. These indices are then combinded with the

x,y,z key and stacked to create an 5xN array of all the nonzero terms in the DSM, where N is the number of nonzero terms.

Returns indices=[[x1,y1,z1,k1,j1],...,[xn,yn,zn,kn,jn]]

nonzeroMat (*s*, *cor*)
Find the indices of the nonzero terms for part of the DSM s.

Parameters **cor** – the part of s that you want the nonzero indices for.

The indices are found by using the :func: nonzero() function on s[cor]

Returns indices=[[x1,y1,z1,k1,j1],...,[xn,yn,zn,kn,jn]]

save_dict (*s*, *filename_*)
Save the DSM s.

Parameters **filename** – the name of the file to save to.

Returns nothing

sin (*s*)
Finds sin(theta) for all terms theta != 0 in the DS s.

Returns A DSM with the same dimensions with sin(theta) in the same position as the corresponding theta terms.

sparse_angles (*s*)
Finds the angles theta which are the arguments of the nonzero complex terms in the DSM s.

Returns A DSM with the same dimensions with theta in the same position as the corresponding complex terms.

stopcheck (*s*, *i*, *j*, *k*, *p1*, *h*)
Check if the index [i,j,k] is valid.

Parameters

- **i** – is the index for the x axis.
- **j** – is the index for the y axis.
- **k** – is the index for the z axis.
- **p1** – is the point at the end of the ray.
- **h** – is the mesh width

Returns 0 if valid, 1 if not.

Todo

add the inside check to this function

Todo

add the check for the end of the ray.

---

stopchecklist (*s*, *ps*, *p1*, *h*, *p3*, *n*)
Check if the list of points is valid.

Parameters

- **ps** – the indices for the points in the list
- **p1** – the end of the ray
- **h** – the meshwidth
- **p3** – the points on the cone vectors
- **n** – the normal vectors forming the cone.

start=0 if no points were valid if at least 1 point was valid, ps=[[i1,j1,k1],...,[in,jn,kn]] the indices of the valid points, p3=[[x1,y1,z1],...,[xn,yn,zn]] co-ordinates of the valid points, N=[n0,...,Nn] the normal vectors corresponding to the valid points.

Returns start, ps, p3, N

vec_multiply(*s*, *vec*)

Multiply every column of the DSM s elementwise with the vector vec.

Parameters **a** – vec: a row vector with length na.

Return type A DSM 'out' with the same dimensions as s.

Returns out[x,y,z,k,j]=vec[k]*DSM[x,y,z,k,j]

CHAPTER 6

Reflection

Code to Reflect a line in an edge without using Shapely

reflection.**errorcheck**(*err*, *ray*, *ref*, *normedge*)
Take the input ray and output ray and the normal to the edge, check that both vectors have the same angle to the normal

reflection.**refangle**(*line*, *obst*)
Find the reflection angle for the line reflection on the surface obst

reflection.**test3**()
angle test

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

d

p

r