index.md

ConstructiveSolidGeometry.jl Documentation

This is the full API for this package. A more thorough description of how to actually use the package can be found in the examples directory.

Types

ConstructiveSolidGeometry.Coord — *Type*.

```
type Coord
```

An {x,y,z} coordinate type. Used throughout the ConstructiveSolidGeometry.jl package for speed.

Constructors

- `Coord(x::Float64, y::Float64, z::Float64)`

source

ConstructiveSolidGeometry.Surface — *Type*.

```
abstract Surface
```

An abstract class that all surfaces (`Sphere` , `Plane` , `InfCylinder`) inherit from. Implementation of new shapes should inherit from `Surface` .

source

ConstructiveSolidGeometry.Ray — *Type*.

```
type Ray
```

A ray is defined by its origin and a unitized direction vector

Constructors

- `Ray(origin::Coord, direction::Coord)`

source

ConstructiveSolidGeometry.Plane — *Type*.

```
type Plane <: Surface
```

Defined by a point on the surface of the plane, its unit normal vector, and an optional boundary condition.

Constructors

- `Plane(point::Coord, normal::Coord)`
- `Plane(point::Coord, normal::Coord, boundary::String)`

Arguments

- `point::Coord` : Any point on the surface of the plane
- `normal::Coord` : A unit normal vector of the plane. Recommended to use `unitize(c::Coord)` if normalizing is needed.
- `boundary::String` : Optional boundary condition, defined as a `String`. Options are "transmission" (default), "vacuum", and "reflective".

source

ConstructiveSolidGeometry.Sphere — *Type*.

```
type Sphere <: Surface
```

Defined by the center of the sphere, its radius, and an optional boundary condition.

Constructors

- `Sphere(center::Coord, radius::Float64)`
- `Sphere(center::Coord, radius::Float64, boundary::String)`

Arguments

- `center::Coord` : The center of the sphere
- `radius::Float64` : The radius of the sphere
- `boundary::String` : Optional boundary condition, defined as a `String`. Options are "transmission" (default) or "vacuum".

source

ConstructiveSolidGeometry.InfCylinder — *Type*.

```
type InfCylinder <: Surface
```

An arbitrary direction infinite cylinder defined by any point on its central axis, its radius, the unit normal direction of the cylinder, and an optional boundary condition. A finite cylinder can be generated by defining the intersection of an infinite cylinder and two planes.

Constructors

- `InfCylinder(center::Coord, normal::Coord, radius::Float64)`
- `InfCylinder(center::Coord, normal::Coord, radius::Float64, boundary::String)`

Arguments

- `center::Coord` : The center of the infinite cylinder
- `normal::Coord` : A unit normal direction vector of the cylinder (i.e., a vector along its central axis), Recommended to use `unitize(c::Coord)` if normalizing is needed.
- `radius::Float64` : The radius of the infinite cylinder
- `boundary::String` : Optional boundary condition, defined as a `String`. Options are "transmission" (default) or "vacuum".

source

`ConstructiveSolidGeometry.Box` — *Type*.

```
type Box
```

An axis aligned box is defined by the minimum `Coord` and maximum `Coord` of the box. Note that a Box is only used by ConstructiveSolidGeometry.jl for bounding box purposes, and is not a valid surface to define CSG cells with. Instead, you must define all six planes of a box independently.

Constructors

- `Box(min::Coord, max::Coord)`

source

`ConstructiveSolidGeometry.Region` — *Type*.

```
type Region
```

The volume that is defined by a surface and one of its halfspaces

Constructors

- `Region(surface::Surface, halfspace::Int64)`

Arguments

- `surface::Surface` : A `Sphere`, `Plane`, or `InfCylinder`
- `halfspace::Int64` : Either +1 or -1

source

`ConstructiveSolidGeometry.Cell` — *Type*.

```
type Cell
```

Defined by an array of regions and the logical combination of those regions that define the cell

Constructors

- `Cell(regions::Array{Region}, definition::Expr)`

Arguments

- `regions::Array{Region}` : An array of regions that are used to define the cell
- `definition::Expr` : A logical expression that defines the volume of the cell. The intersection operator is ^, the union operator is |, and the complement operator is ~. Regions are defined by their integer indices in the regions array.

source

`ConstructiveSolidGeometry.Geometry` — *Type*.

```
type Geometry
```

The top level object that holds all the cells in the problem. This object contains all data regarding the geometry within a system.

Constructors

- `Geometry(cells::Array{Cell}, bounding_box::Box)`

Arguments

- `cells::Array{Cell}` : All cells inside the geometry. The cells must combine to fill the entire space of the bounding box. No two cells should overlap.
- `bounding_box::Box` : The bounding box around the problem.

source

Functions

`ConstructiveSolidGeometry.magnitude` — *Function*.

```
magnitude(a::Coord)
```

A utility function to determine the magnitude of a `Coord` object. Typical use case is to subtract two Coord objects and check the resulting Coord object's magnitude to determine the distance between the two Coords.

source

`ConstructiveSolidGeometry.unitize` — *Function*.

```
unitize(a::Coord)
```

A utility function to unitize a `Coord`

source

`ConstructiveSolidGeometry.raytrace` — *Method*.

```
function raytrace(ray::Ray, surface::Surface)
```

Determines if a `Ray` and a `Surface` intersect, and the distance to that intersection.

Returns

- `Bool` : Indicates if the ray intersects the surface or not
- `Float64` : The distance between the ray's origin and the point of intersection

source

`ConstructiveSolidGeometry.reflect` — *Method*.

```
reflect(ray::Ray, plane::Plane)
```

Reflects a ray off a plane.

Return

- `Ray` : A new ray with the same origin as input, but with the new reflected direction

source

`ConstructiveSolidGeometry.generate_random_ray` — *Function*.

```
generate_random_ray(box::Box)
```

Returns a randomly sampled ray from within an axis aligned bounding box.

source

`ConstructiveSolidGeometry.find_intersection` — *Method*.

```
find_intersection(ray::Ray, geometry::Geometry)
```

Performs ray tracing on a Geometry

Return

- `Ray` : A new Ray that has been moved just accross the point of intersection.
- `Int64` : The surface id that was hit.
- `String` : The boundary condition of the surface that was hit.

source

`ConstructiveSolidGeometry.find_intersection` — *Method*.

```
find_intersection(ray::Ray, regions::Array{Region})
```

Performs ray tracing on an array of regions.

Return

- `Ray` : A new Ray that has been moved just accross the point of intersection.

- `Int64` : The surface id that was hit.
- `String` : The boundary condition of the surface that was hit.

source

`ConstructiveSolidGeometry.is_in_cell` — *Function*.

```
is_in_cell(p::Coord, cell::Cell)
```

Determines if a point (such as a Ray origin) is inside a given cell

source

`ConstructiveSolidGeometry.find_cell_id` — *Function*.

```
find_cell_id(p::Coord, geometry::Geometry)
```

Finds the cell id that a point resides within

source

`ConstructiveSolidGeometry.plot_geometry_2D` — *Function*.

```
plot_geometry_2D(geometry::Geometry, view::Box, dim::Int64)
```

Plots a 2D x-y slice of a geometry.

Arguments

- `geometry::Geometry` : the geometry we want to plot
- `view::Box` : The view box is an axis aligned box that defines where the picture will be taken, with both min and max z dimensions indicating the single z elevation the slice is taken at.
- `dim::Int64` : The dimension is the number of pixels along the x and y axis to use, which determines the resolution of the picture.

source

`ConstructiveSolidGeometry.plot_cell_2D` — *Function*.

```
plot_cell_2D(geometry::Geometry, view::Box, dim::Int64, cell_id::Int64)
```

Plots a 2D x-y slice of a geometry, highlighting a specific cell in black.

Arguments

- `geometry::Geometry` : the geometry we want to plot
- `view::Box` : The view box is an axis aligned box that defines where the picture will be taken, with both min and max z dimensions indicating the single z elevation the slice is taken at.
- `dim::Int64` : The dimension is the number of pixels along the x and y axis to use, which determines the resolution of the picture.
- `cell_id::Int64` : The index of the cell we wish to view

source

Index