Virtual Laser Scanner for GroIMP

1 Introduction

The "Scanner" is a module implementing a virtual laser scanning device. To use it, first import the module :

```
import de.grogra.rgg.Scanner ;
```

and create a new instance :

```
Scanner ls = new Scanner() ;
```

Then, the process of points acquisition can be seen as follow :

1. set a ray "length" ;
2. set an origin, and directions for the rays ;
3. shoot the rays, store the returned data ;
4. repeat until enough points are acquired.

This process can be done step-by-step, or a method implementing a scanning according to some pre-defined schedule can be used.

2 Shooting of the rays

The rays are always shot according to two angular parameters $(\theta_{range}, \phi_{range})$ defining the solid angle that will be scanned, two parameters $(\theta_{step}, \phi_{step})$ defining the resolution with which it will be scanned, and a direct orthonormal basis $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$ serving as reference (see below).

In the spherical coordinate system based on the reference basis $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$, rays are shot for $\theta \in \left[\frac{-\theta_{range}}{2}, \frac{\theta_{range}}{2}\right]$ with an angular resolution θ_{step} and for $\phi \in \left[\frac{\pi}{2} - \frac{\phi_{range}}{2}, \frac{\pi}{2} + \frac{\phi_{range}}{2}\right]$ with an angular resolution ϕ_{step} (see illustration Fig.1 p.2). Thus, the mean direction of the rays shot is $\overrightarrow{x}$.

Details concerning the spherical and cylindrical coordinate system used in this module can be found at the end of this document (resp. 4.1 p.6 and 4.2 p.6).

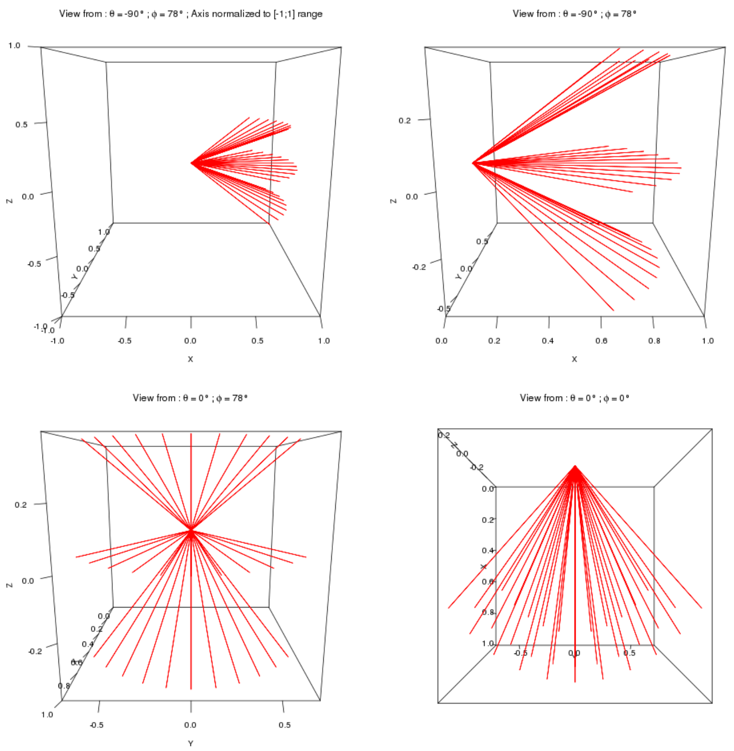

Figure 1: Directions of rays shot for the parameters $\theta_{range} = \frac{\pi}{2}$, $\phi_{range} = \frac{\pi}{4}$, $\theta_{step} = \frac{\pi}{20}$ and $\phi_{step} = \frac{\pi}{8}$.

3 Methods available

Remark : if calling multiple scan methods (e.g. *scan, scanCylinder, scanSphere* ; see below) in a row, be sure to save the returned data after each call, as they will not be kept.

Notations : *Seg(Point3d point, Vector3d vector, Double l)* designates the segment which has the point *point* at one end, the direction vector *direction*, and of length *l*.
Cir(Point3d center, Vector3d normal, Double r) designates the circle of center *center*, of radius *r* and contained in a plane orthogonal to the vector *normal*.

setRayLength Sets the range of the rays.

```
void setRayLength(double desiredLength) ;
```

desiredLenght must be strictly positive.

setBasis Sets the 3-dimensional basis to be used as reference for shooting the rays.

```
void setBasis(Vector3d x, Vector3d y, Vector3d y) ;
```

The 3 vectors must form a direct orthogonal basis.

setRange Sets the angular opening in θ and ϕ.

```
void setRange(double thetaRange,double phiRange) ;
```

If $thetaRange \geq 2 \cdot \pi$ (resp. $phiRange \geq \pi$) , then $thetaRange = 2 \cdot \pi$ (resp. $phiRange = \pi$)will be used instead. $thetaRange \leq 0$ (resp. $phiRange \leq 0$) sets an angular opening in *theta* (resp. *phi*) of 0 (Note : $thetaRange = 0$ and $phiRange = 0$ corresponds at the shooting of only one ray, along the x-axis).

setSteps Sets the resolution in *theta* and *phi*.

```
void setSteps(double thetaStep,double phiStep) ;
```

If $thetaStep \geq thetaRange$ (resp. $phiStep \geq phiRange$) , then $thetaStep = thetaRange$ (resp. $phiStep = phiRange$)will be used instead. $thetaStep \leq 0$ (resp. $phiRange \leq 0$) sets an angular opening in *theta* (resp. *phi*) of 0.

setpDrawRay Set the probability of drawing one ray.

```
void setpDrawRay(double p) ;
```

Each ray will be drawn with a probability p (if $p \leq 0$ no rays will be drawn, if $p \geq 1$ all the rays will be drawn), thus $n \cdot p$ will be drawn in average (with n the total number of rays shot).

scan Triggers the scanning with the specified options

```
ArrayList<Point3d> scan(Point3d center) ;
```

Rays will be shot from the point center, according to the angular parameters specified. Scanned points are returned in an ArrayList.

writeDataToFile Writes the ArrayList *data* in a file named *fileName*. The data are written according to a "x y z" format.

```
void writeDataToFile(ArrayList<Point3d> data, String fileName) ;
```

scanSegment Moves the origin of the rays along the segment *Seg(startingPoint, direction,* $(nbSteps - 1) \cdot stepLength)$. Rays are shot at each step according to the specified parameters. Scanned points are returned in an ArrayList.

```
ArrayList<Point3d> scanSegment (Point3d startingPoint, Vector3d
        direction, double stepLength, int nbSteps) ;
```

The vector *direction* must not be null, *stepLenght* and *nbSteps* must be strictly positive.

Please note that for all the following methods, the angular parameters used are those fixed by the methods *setRange* and *setSteps*. However, the basis defining the directions of the rays are defined within the methods.

scanCircle Moves the origin of the rays along the circle *Cir(center, normal, r)*. Rays are shot for each angular step *angleStep* until a complete circle has been covered. Directions of the rays shot are defined by the local cylindrical basis *(-u,* $-u_\theta$*, normal)* formed at each step, with respect to the reference basis *(zeroAngleVector, normal* $\wedge$*zeroAngleVector, normal)*. Scanned points are returned in an ArrayList.

```
ArrayList<Point3d> scanCircle(Point3d center, double r, Vector3d
      normal,Vector3d zeroAngleVector, double angleStep) ;
ArrayList<Point3d> scanCircle(Point3d center, double r, Vector3d
                   normal, double angleStep) ;
```

The vector *zeroAngleVector* specifies the angle reference for the circle. It must be orthogonal to *normal*. If none is specified, an arbitrary vector is used.

scanCylinder Moves the origin of the rays on the surface of the cylinder of axis the segment *Seg(startingPoint, axis, (nbSteps-1)* $\cdot$ *lengthAxisStep)* and of radius r. For each step i along the cylinder's axis, the origin of the rays revolves around the cylinder with an angular step *angularStep*. Rays are shot at each angular step until a complete circle has been covered. Directions of the rays shot are defined by the local cylindrical basis *(-u,*$-u_\theta$*, axis)* formed at each angular step, with respect to the reference basis *(zeroAngleVector, axis* $\wedge$ *zeroAngleVector, axis)*. Scanned points are returned in an ArrayList.

```
ArrayList<Point3d> scanCylinder ( Point3d startingPoint, Vector3d
 axis, double radius, double angularStep, double lengthAxisStep,
                         int nbSteps) ;
ArrayList<Point3d> scanCylinder ( Point3d startingPoint, Vector3d
 axis, double radius, double angularStep, double lengthAxisStep,
              int nbSteps, Vector3d zeroAngleVector) ;
```

The vector *zeroAngleVector* specifies the angle reference for each circle. It must be orthogonal to *axis*. If none is specified, an arbitrary vector is used.

scanSphere Moves the origin of the rays on a sphere of center *center* and radius *r*. With respect to the spherical coordinate system based on the basis *(x,y,z)*, the origin of the rays moves from $\phi = 0$ to $\phi = \pi$ with a step *phiStep*, and, for each angle ϕ, from $\theta = 0$ to $\theta = 2\pi$ with a step thetaStep. Rays are shot for each couple (θ, ϕ), their directions being defined by the local spherical basis $(-u_r, -u_\phi, u_\theta)$. Scanned points are returned in an ArrayList.

```
ArrayList<Point3d> scanSphere (Point3d center, double r, double
phiStep, double thetaStep, Vector3d x, Vector3d y, Vector3d z) ;
```

(x, y, z) must be a direct orthogonal basis, r and *phiStep* must be strictly positive. If $thetaStep \leq 0$, then $\theta = 0$ will be used at each step (thus the origin of the rays describes an half circle).

Noise simulation A set of methods enables noise simulation. The introduction of a noise factor is triggered for the parameters θ, ϕ and the hit distance by the following methods :

```
void setThetaNoise(boolean value) ;
void setPhiNoise(boolean value) ;
void setDistanceNoise(boolean value) ;
```

Each time a point is acquired, a noise factor is added to the angle θ and / or ϕ and / or the hit distance when computing the point's position. Noise on θ and ϕ modifies the direction in which the point is thought to be ; noise on the hit distance the distance (in this direction) from the scanner.

The type of noise can be set through the following methods :

```
void setThetaNoiseType(int type, boolean adapt, double param1,
                       double param2) ;
void setPhiNoiseType(int type, boolean adapt, double param1,
                     double param2) ;
void setDistanceNoiseType(int type, boolean adapt, double param1,
                          double param2) ;
```

type = 0 sets the noise to an uniform pertubation, *type = 1* to a gaussian one (any other value inducing no noise at all). *adapt = false* produces a noise with fixed parameters, whereas *adapt = true* produces a noise which parameters depends on the real value (see below).

Given the real value α_r of one parameter (θ, ϕ or the hit distance), the value used to compute the point's position is $\alpha_u = \alpha_r + \epsilon$, where ϵ is the noise. With $\mathcal{U}(a, b)$ denoting the uniform density on $[a, b]$ $(a < b)$, and $\mathcal{N}(\mu, \sigma)$ the gaussian density of mean μ and standard deviation σ, the pertubation ϵ has the corresponding density :

type = / adapt =	**0**	**1**
false	$\epsilon \sim \mathcal{U}(param1, param2)$	$\epsilon \sim \mathcal{N}(param1, param2)$
true	$\epsilon \sim \mathcal{U}(param1 \cdot \alpha_r, param2 \cdot \alpha_r)$	$\epsilon \sim \mathcal{N}(param1 \cdot \alpha_r, param2 \cdot \alpha_r)$

4 Annexes

4.1 Spherical coordinate system

The spherical coordinate system is a 3d-coordinate system in which the position of a point is specified by two angular parameters θ and ϕ and the radial distance r. Given a direct orthonormal basis $B_0 = (\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$, and a fixed origin point O, these parameters, for some point $M(r, \theta, \phi)$, are defined as follow (see Fig.2 p.7) :

$$\theta = \widehat{\overrightarrow{x}, \overrightarrow{OP}}, \ \theta \in [0; 2\pi]$$

$$\phi = \widehat{\overrightarrow{z}, \overrightarrow{OM}}, \ \phi \in [0; \pi]$$

$$r = OM$$

with P the orthogonal projection of M on the plane $(O, \overrightarrow{x}, \overrightarrow{y})$. Note that only the two angular parameters θ and ϕ are needed to define a direction (see Fig.2 p.7).

For each direction (θ, ϕ) a local direct orthonormal basis $B_{(\theta,\phi)} = (\overrightarrow{u_r}, \overrightarrow{u_\phi}, \overrightarrow{u_\theta})$ can be de defined, with :

$$\overrightarrow{u_r} = \begin{pmatrix} cos(\theta) \cdot sin(\phi) \\ sin(\theta) \cdot sin(\phi) \\ cos(\phi) \end{pmatrix}_{B_0} \quad \overrightarrow{u_\phi} = \begin{pmatrix} cos(\theta) \cdot cos(\phi) \\ sin(\theta) \cdot cos(\phi) \\ -sin(\phi) \end{pmatrix}_{B_0} \quad \overrightarrow{u_\theta} = \begin{pmatrix} -sin(\theta) \\ cos(\theta) \\ 0 \end{pmatrix}_{B_0}$$

4.2 Cylindrical coordinate system

The cylindrical coordinate system is a 3d-coordinate system in which the position of a point is specified by one angular parameter θ, one distance r and one height z. Given a direct orthonormal basis $B_0 = (\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$, and a fixed origin point O, these parameters, for some point $M(r, \theta, z)$, are defined as follow (see Fig.2 p.7) :

$$\theta = \widehat{\overrightarrow{x}, \overrightarrow{OP}}, \ \theta \in [0; 2\pi]$$

$$r = OP$$

$$z = \overrightarrow{OM}.\overrightarrow{z}$$

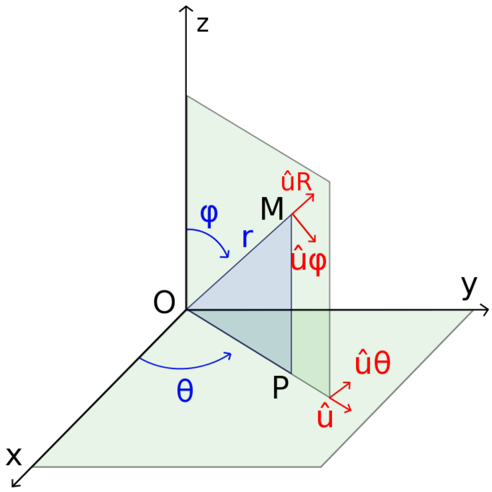

Figure 2: Spherical and Cylindrical coordinate system

with P the orthogonal projection of M on the plane $(O, \overrightarrow{x}, \overrightarrow{y})$ (see Fig.2 p.7).

For each angle θ a local direct orthonormal basis $B_\theta = (\overrightarrow{u}, \overrightarrow{u_\theta}, \overrightarrow{z})$ can be defined, with :

$$\overrightarrow{u} = \begin{pmatrix} cos(\theta) \\ sin(\theta) \\ 0 \end{pmatrix}_{B_0} \quad \overrightarrow{u_\theta} = \begin{pmatrix} -sin(\theta) \\ cos(\theta) \\ 0 \end{pmatrix}_{B_0}$$