2. Go Concurrency [8 points]|8% of final course grade|

For the concurrent part of the comprehensive assignment, we ask you to program the RANSAC algorithm using a concurrent pipeline. You will have to perform some experiments in order to determine the optimal number of threads that should be used to efficiently find the dominant planes.

Types, functions, and methods

Your program will have to include de following types:

```
type Point3D struct {
     X float64
     Y float64
     Z float64
}

type Plane3D struct {
     A float64
     B float64
     C float64
     D float64
}

type Plane3DwSupport struct {
     Plane3D
     SupportSize int
}
```

Your program must also include the following functions and methods:

```
// reads an XYZ file and returns a slice of Point3D
func ReadXYZ(filename string) []Point3D

// saves a slice of Point3D into an XYZ file
func SaveXYZ(filename string, points []Point3D)

// computes the distance between points p1 and p2
func (p1 *Point3D) GetDistance(p2 *Point3D) float64

// computes the plane defined by a set of 3 points
func GetPlane(points []Point3D) Plane3D

// computes the number of required RANSAC iterations
func GetNumberOfIterations(confidence float64,  \
                           percentageOfPointsOnPlane float64) int

// computes the support of a plane in a set of points
func GetSupport(plane Plane3D, points []Point3D, \
                                  eps float64) Plane3DwSupport

// extracts the points that supports the given plane
// and returns them as a slice of points
func GetSupportingPoints(plane Plane3D, points []Point3D, \
                                    eps float64) []Point3D

// creates a new slice of points in which all points
// belonging to the plane have been removed
func RemovePlane(plane Plane3D, points []Point3D, \
                                  eps float64) []Point3D
```

Déroulement de la fonction principale

Your main function must perform the following operations:

1. *Read the XYZ file specified as a first argument to your go program and create the corresponding slice of Point3D, composed of the set of points of the XYZ file.*
2. *Create a* `bestSupport` *variable of type* `Plane3DwSupport` *initialized to all 0s.*
3. *Find the number of iterations required based on the specified confidence and percentage provided as 1st and 2nd arguments for the* `GetNumberOfIterations` *function.*
4. *Create and start the RANSAC find dominant plane pipeline. This pipeline automatically stops after the required number of iterations.*
5. *Once the pipeline has terminated, save the supporting points of the identified dominant plane in a file named by appending* `_p` *to the input filename.*
6. *Save the original point cloud without the supporting points of the dominant plane in a file named by appending* `_p0` *to the input filename.*

To find the 3 most dominant planes, simply run your program 3 times, each time with the output point cloud of the previous run. Do not forget to rename the … _p file to `_p1`, `_p2` *et* `_p3`.

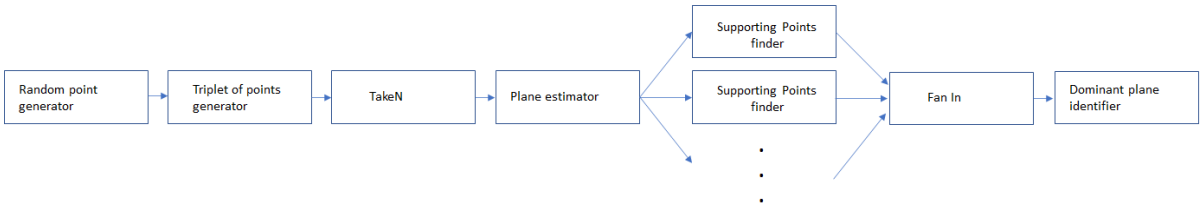
Your go program is executed with the following arguments:

```
>  go planeRANSAC filename confidence percentage eps
```

Note: `os.Args` *provides access to the command line arguments.*

The concurrent pipeline

The pipeline you have to build is illustrated here:



Random point generator : `-> Point3D`

It randomly selects a point from the provided slice of `Point3D` *(the input point cloud). Its output channel transmits instances of* `Point3D`.

Triplet of points generator: `Point3D -> [3]Point3D`

It reads `Point3D` *instances from its input channel and accumulate 3 points. Its output channel transmits arrays of* `Point3D` *(composed of three points).*

TakeN: `[3]Point3D -> [3]Point3D`

It reads arrays of `Point3D` *and resend them. It automatically stops the pipeline after having received* `N` *arrays.*

Plane estimator: `[3]Point3D -> Plane3D`

It reads arrays of three `Point3D` *and compute the plane defined by these points. Its output channel transmits* `Plane3D` *instances describing the computed plane parameters.*

Supporting point finder: `Plane3D -> Plane3DwSupport`

It counts the number of points in the provided slice of `Point3D` *(the input point cloud) that supports the received 3D plane. Its output channel transmits the plane parameters and the number of supporting points in a* `Point3DwSupport` *instance.*

Fan In: `Plane3DwSupport -> Plane3DwSupport`

It multiplexes the results received from multiple channels into one output channel.

Dominant plane identifier: `Plane3DwSupport`

It receives `Plane3DwSupport` instances and keepd in memory the plane with the best support received so far. This component does not output values, it simply maintains the provided `*Plane3DwSupport` variable.

Warning! To facilitate the debugging of your program, build your pipeline by adding and testing each component one by one.

Submission

In addition to the source code of your go program, you must submit a document describing the experiments you performed in order to find the optimal number of threads to create. A graph showing runtime versus number of threads for different configurations should be included.