Challenge 1
Downsampling a Point Cloud

VerifyThis at ETAPS 2022
Organizers: Marie Farrell, Peter Lammich
Steering Committee: Marieke Huisman, Rosemary Monahan,
Peter Müller, Mattias Ulbrich

2–3 April 2022, TU Munich, Germany

Disclaimer: the programs may contain bugs. If you find any, fix them and mention the fix in your submission!

How to submit solutions: send an email to `verifythis@googlegroups.com` with your solution in attachment. Remember to clearly identify yourself, stating your team's name and its members.

Problem Description

The following algorithm describes the functionality of downsampling an input point cloud. Downsampling is used to reduce the size of an input image before it is processed further. The resulting point cloud retains the overall geometric structure but has a reduced number of points.

Techniques such as this are common in domains such as signal/image processing and robotics. Our pseudocode uses square voxel downsampling with a predefined voxel size. That is, the space is tiled into cube-shaped voxels, and the average point in the voxel becomes the new point to replace the others.

This challenge was inspired by work on verifying an autonomous grasping algorithm in [1].

```
datatype Point = Point(x: real, y: real, z: real)
method downSample(p: list<Point>, voxel_size: real) returns (pd: list<Point>)
{
  // Get max and min x, y and z of point cloud
  x_max := max{pt.x | pt ∈ p}; x_min := min{pt.x | pt ∈ p};
  y_max := max{pt.y | pt ∈ p}; y_min := min{pt.y | pt ∈ p};
  z_max := max{pt.z | pt ∈ p}; z_min := min{pt.z | pt ∈ p};

  // Find the number of voxels in all 3 dimensions. A voxel is a cube with
     edge length voxel_size. Note that we round up in the division.
  var num_vox_x := (|x_max - x_min|/voxel_size).Ceiling;
  var num_vox_y := (|y_max - y_min|/voxel_size).Ceiling;
  var num_vox_z := (|z_max - z_min|/voxel_size).Ceiling;

  // Array of voxels, each element eg at voxel_array[i,j,k] is a Point which
      is initialised to (0.0, 0.0, 0.0)
  voxel_array := new Point[num_vox_x,num_vox_y,num_vox_z];

  // Array of counts in all dimensions (useful for averaging), each element
     should be set to 0 at initialisation, at the end the sum of counts
     should be equal to the number of points in the input point cloud.
  count_array := new int[num_vox_x,num_vox_y,num_vox_z];

  // Objective is to calculate: E.g: voxel_array[0,2,1] -> (((0.23, 2.45,
     1.89) + (0.13, 2.87, 1.35) + ..())/count_array[0,2,1])

  forall pt in p {
    //take the floor to collect points that are in the same region
    var x_floored := ((pt.x - x_min)/voxel_size).Floor;
    var y_floored := ((pt.y - y_min)/voxel_size).Floor;
    var z_floored := ((pt.z - z_min)/voxel_size).Floor;

    voxel_array[x_floored,y_floored,z_floored] := voxel_array[x_floored,
     y_floored,z_floored] + pt;
    count_array[x_floored,y_floored,z_floored] := count_array[x_floored,
     y_floored,z_floored] + 1;

  }

  // Average the voxelised (bucketed) points to get the final point cloud
  i, j, k := 0, 0, 0;
  pd := [];
  for 0 ≤ i < num_vox_x
    for 0 ≤ j < num_vox_y
      for 0 ≤ k < num_vox_z
        if(count_array[i,j,k] ≠0)
          pd.append(voxel_array[i,j,k]/count_array[i,j,k]);

  return pd;
}
```

Tasks

Implementation task. Implement the `downsample` method. The pseudocode uses real numbers but participants may simplify and use integers if their tool does not have adequate support for real numbers. Other potential simplifications include requiring that the least point is at $(0, 0, 0)$ and fixed parameters for `x_max`, `y_max` and `z_max`.

Verification tasks. Verify the following properties:

1. Memory Safety.
2. Termination.
3. The output point cloud is smaller or equal to the input point cloud. For example:
$$size(pd) <= size(p)$$
4. The output point cloud is within the same range as the input point cloud. For example
$$boundingbox(pd) \ \ inside \ \ boundingbox(p)$$
5. The output point cloud is a correct downsampled version of the input point cloud.

References

[1] M. Farrell, N. Mavrakis, A. Ferrando, C. Dixon, and Y. Gao. Formal modelling and runtime verification of autonomous grasping for active debris removal. *Frontiers in Robotics and AI*, 8, 2021.