Subtracting point clouds using points-join --not-matching

Assume you would like to quickly find additive changes in the scene. For example you have a static point cloud of empty car park, and would like to extract the parked cars from a stream of lidar data. If the extraction does not have to be perfect, a quick way of doing it would be using **points-join --not-matching** . A simple example:

```
> # make sample point clouds
> for i in {20..30}; do for j in {0..50}; do for k in {0..50}; do echo $i,$j,$k; done; done; done > minuend.csv
> for i in {0..50}; do for j in {20..30}; do for k in {20..30}; do echo $i,$j,$k; done; done; done > subtrahend.
csv
> cat minuend.csv | points-join subtrahend.csv --radius 0.51 --not-matching | view-points "minuend.csv;
colour=red;hide" "subtrahend.csv;colour=yellow;hide" "-;colour=white;title=difference"
```

The described car park scenario would look like:

```
> cat carpark-with-cars.csv | points-join --fields x,y,z "empty-carpark.csv;fields=x,y,z" --radius 0.1 --not-
matching > cars-only.csv
```

The crude part is of course in choosing **--radius** value: it should be such that the spheres of a given radius around the subtrahend point cloud sufficiently overlap to capture all the points belonging to it. But then the points that are closer than the radius to the subtrahend point cloud will be filtered out, too. E.g. in the car park example above, the wheels of the cars will be chopped off at 10cm above the ground. To avoid this problem, you could for example erode somehow the subtrahend point cloud by the radius.

The described approach may be crude, but it is quick and suitable for many practical purposes.

Of course, for more sophisticated change detection in point clouds, which is more accurate and takes into account view points, occlusions, additions and deletions of objects in the scene, etc, you could use **points-detect-change**.