Task

A 3d Euclidian point cloud aligned in main axis directions (x, y, z) and with a constant distance grid (Figure 1) starting at a given reference point (the point with indices 0, 0, 0 is located at reference point) is intersected by a move of sphere (Figure 2), where the path of the sphere center is defined by a user given formula $\vec{x} = f(t)$ where t is in the interval between t_0 and t_1. The function $f(t)$ can be handled as a discrete function with a user given Δt. Points that intersect with the sphere move are considered as deleted (Figure 3 middle).

- Only the first layer of points (which remains *visible/undeleted*) from top view must be written to a file as ASCII data (the skin of the point cloud visible from positive Z direction, see Figure 3 right). The file format is defined as follows:
  - Each line contains a single point.
  - The point definition contains x, y and z coordinates delimited by space characters.
  - Each line ends with a new line character.
- Create a small documentation(1 page with 2 pictures) to present the mathematical approach of the sphere move point intersection. It should clearly communicate the mathematical approach and how the *mathematical* code is generated from that.
- Please discuss briefly in 4-5 sentences what problems can arise by using a discrete stepping Δt.

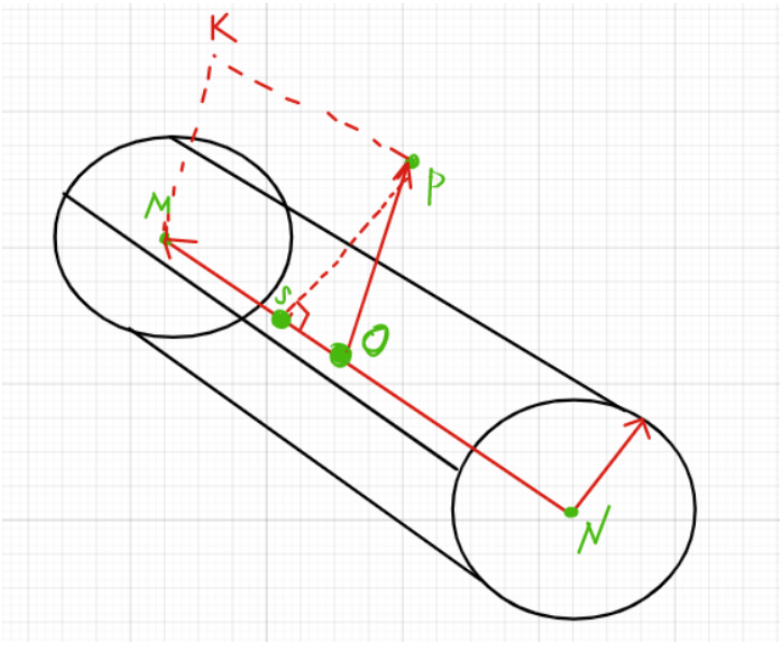
From a mathematical point of view, this problem is about the difference of two sets - an array of points with integer non-negative coordinates and a sphere that moves through this array.
After several experiments, it was found out that it is inconvenient to use a storage containing exactly points in its pure form, so the interpretation of the point cloud will be as follows: a three-dimensional array of logical values that determines whether a point is included in the set or not. By default, all points are included in the array. Iteration over these logical values will be described by the following sets of indices, which are also the coordinates of points:

$$Cloud = \{(i,j,k) | (i,j,k) \in \mathbb{N} \cup \{0\}\}$$

where $0 \leq i < 1000;\ 0 \leq j < 500;\ 0 \leq k < 100$.
We arrange the sphere class (actually a ball) in such a way that we can move its center using a function along an arc. Thus, for each new sphere, we can determine whether a point from the array belongs to it or not by substituting the point into the sphere equation:

$$(x,y,z) \in B_R(a,b,c) \Leftrightarrow \{(x,y,z) | (x-a)^2 + (y-b)^2 + (z-c)^2 \leq R^2\}$$

Since it is impossible to achieve a continuous movement of the sphere through interval refinement (we will have an increase in execution time), we will do the following. Connect the upper boundaries of the spheres with cylinders with the same radius as the spheres.



Find the condition for a point to fall into a cylinder. From picture we can see that the distance between point P and its projection S on the central axis of the cylinder should be less than the radius: $|\overline{PS}| < R$, and also the point must belong to one of the segments OM or ON ($Proj_{\overline{OM}}(\overline{OP}) < |\overline{OM}|$)
Consider the first condition. PS is the height of the parallelogram built on the vectors $\overline{OM}$, $\overline{OP}$. Find the corresponding expression by comparing the two formulas for the area of a parallelogram:

$$\begin{cases} S = |\overline{OM}| \cdot h \\ S = |\overline{OM} \times \overline{OP}| \end{cases}$$

Hence we have

$$|\overline{PS}| = h = \frac{|\overline{OM} \times \overline{OP}|}{|\overline{OM}|} < R$$

In the second condition, it suffices to write the definition of the projection, after which we immediately obtain:

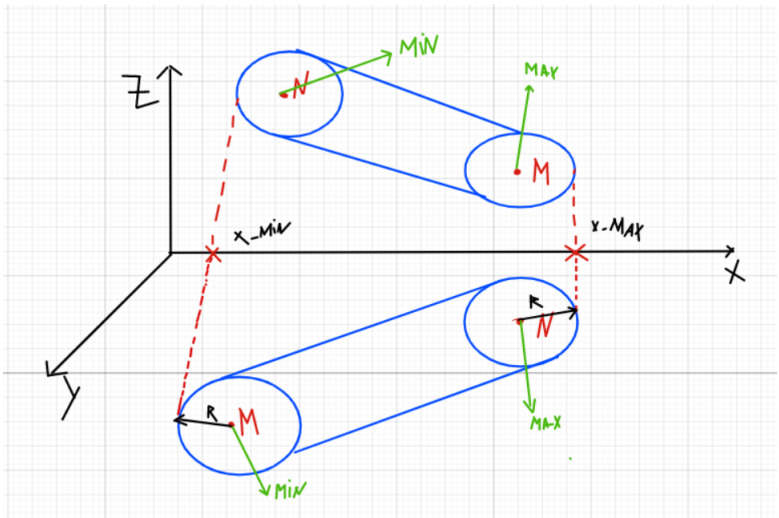
$$Proj_{\overline{OM}}(\overline{OP}) = \frac{< \overline{OM}, \overline{OP} >}{|\overline{OP}|} < |\overline{OM}|$$

The points that fall into the sphere or cylinder are "deleted" by assigning zeros to them. After the difference with each sphere is found, it remains to select the upper layer of the resulting new array of points. This can be done with the following multiple loop (pseudo code):

```
for (int i = 0; i < max_x; ++i)
    {
        for (int j = 0; j < max_y; ++j)
        {
            int k = max_z - 1;
            while (PointFromCloud == 0 && k >= 0)
            {
                --k;
            }

            if (k > 0)
            {
                AddToStorage();
            }
        }
    }
```

In order to reduce the computation time, we will do the following. In the cycles responsible for removing the sphere and cylinders, we will run through not the entire array, but a certain set of points, which limits these figures. For a sphere, such a bounding shell will be a cube with sides $2R$, and for a cylinder, it will be some rectangular parallelepiped (one for each cylinder). To calculate the minimum and maximum borders in which our new coordinates will change, we will build the following picture:



Here we can see that firstly need to select the maximum (minimum) point, and then add (subtract) the radius of the cylinder.

Finally, after executing the loop responsible for the movement of the sphere, we have the following ($\Delta t = 0.01$, $0 \leq t \leq 1$, $R = 5$):