Test task for recruitment of a developer

This task should give you a small insight into the kind of work we do.

Problem statement

Imagine a cloud of 3D points aligned along the coordinate axes $\vec{X}, \vec{Y}, \vec{Z}$ with uniform spacing Δs starting at a given reference point $\vec{O}$ (Figure 1). Any point P in the cloud can be defined as:

$$\vec{P} = \vec{O} + (i_x, i_y, i_z) \cdot \Delta s \tag{1}$$

Where i_x, i_y and i_z are integer indexes which belong to the open-ended ranges $[0, n_x)$, $[0, n_y)$ and $[0, n_z)$ respectively. The cloud is intersected by a moving sphere of radius R. The movement of the sphere center is defined by a 3d curve $f(t)$, $t \in [0, 1]$. To simplify the implementation, we sample $f(t)$ with step Δt $(0 < \Delta t \ll 1)$ and obtain sequence of 3d points $f(0), f(\Delta t), \ldots, f(1)$. Every pair of consecutive points in the sequence can be treated as start and destination points of linear movement of the sphere (Figure 2). Points that intersect with the moving sphere are considered as deleted (Figure 3 middle).

Assignment

The assignment consists of two parts:

- Implement a function that takes the input parameters (Δs, $\vec{O}$, n_x, n_y, n_z, R, $f(t)$, Δt), simulates removal of cloud points that intersect with the linear moves of the sphere, and outputs all the remaining points visible from above [1] (Figure 3 rightmost).
- Create brief (1-2 pages) documentation where you:
  - Describe the mathematical approach your implementation uses to find intersections of the moving sphere and points.
  - Describe briefly in 1-4 sentences what problems can arise by using discrete steps Δt for curve evaluation.

Please send us an archive of your solution and documentation.

Assessment

For the assessment of your solution, we focus not only on the correctness and robustness of your program (40%) but also on aspects that are important for development of successful software such as: coding style and maintainability (20%), used algorithms and performance (20%), appropriate use of language features and standard library (10%), and documentation (10%).
General recommendations:

- Use our template of solution (**Calculate()** function from main.cpp) with **TestInput**/**TestOutput** classes to not struggle with input/output formatting.
- Use basic vector algebra (dot product, cross product).
- Use OOP and C++ STL where you feel appropriate.
- Don't use any 3rd-party libraries besides the standard C++ library.
- Don't modify the test format and command line interface of cutSphereMove.

You are also free to add new functions, source, or test files to the solution.

[1]visible from above means the remaining points $\vec{P}$ with highest i_z for each (i_x, i_y) pair in the (Equation 1)

Given

- Template of solution with **Calculate()** function in **main.cpp**.
- library providing 3d point arithmetic (**geo/Point3**), parameterized curve $f(t)$ (**geo/Curve**), routines for input parsing and output formatting (**io/TestInput** and **io/TestOutput**).
- VisualStudio and CMake projects, you are free to use whatever you like.
- Test inputs with reference outputs (**tests** folder), you can inspect input and output files.
- PointVisualizer.html – WebGL-based tool to visualize test outputs. You can check the provided **tests/output** or your own test outputs with the visualizer. You can also use it in compare mode to compare your outputs with the reference outputs.
- build.py to automate building with CMake (if you use it), test.py to automate running all tests. You don't have to use these scripts, they are there only provided for convenience. Both scripts require python3, and they can be started from the command line.

Quick start

To complete this task you will need one of the following C++ compilers: Visual Studio[2], Clang[3] or GCC[4]. As a build system, you will either need CMake[5] or Visual Studio to use the provided solution file. Optionally you can install Python3[6] to run the provided build and test script.
You can build template project by doing either of:

- If you have both Python and CMake installed, you can just run build.py from the command line.
- If you have VisualStudio installed - open cutSphereMove.sln and build cutSphereMove in Release mode.
- If you have only CMake installed, you can open the command line and cd into the **build** folder, then run:

  ```
  cmake ../
  ```

  This will generate your default solution which you can build by running:

  ```
  cmake --build . --target cutSphereMove --config Release
  ```

Once cutSphereMove executable is built, we can proceed with running the tests. For this you can do either of:

- If you have Python3 you can run all tests by running test.py from the command line.
- Execute cutSphereMove.exe and provide input and output files as command line parameters:

  ```
  cutSphereMove.exe C:/tests/input/test002_arc.txt C:/path/to/output.txt
  ```

Once you have successfully built and run the tests, you can open the file **main.cpp** in your IDE or text editor and fill the function **Calculate()** with your solution.

[2]https://visualstudio.microsoft.com/downloads/
[3]https://clang.llvm.org/
[4]https://gcc.gnu.org/
[5]https://cmake.org/ During installation on Windows please check "Add cmake to PATH" to make it available from cmd
[6]https://www.python.org/ during installation on Windows please check "Add to PATH" option to make it available from cmd

Illustrations

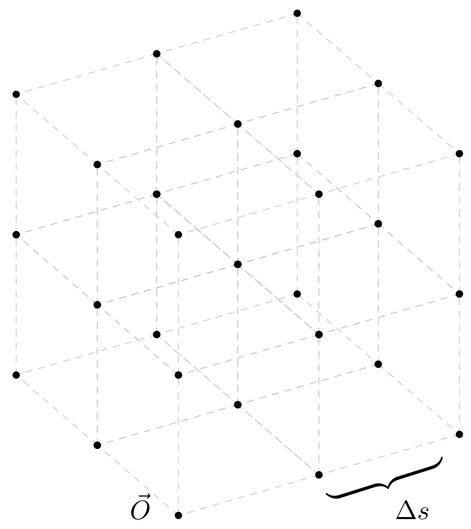

Figure 1: 3d point cloud defined by reference point $\vec{O}$ and grid spacing Δs.

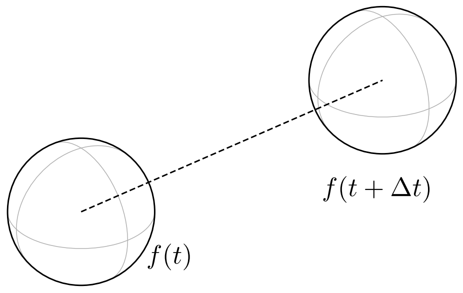

Figure 2: Linear move of a sphere with start point $f(t)$ and end point $f(t + \Delta t)$.

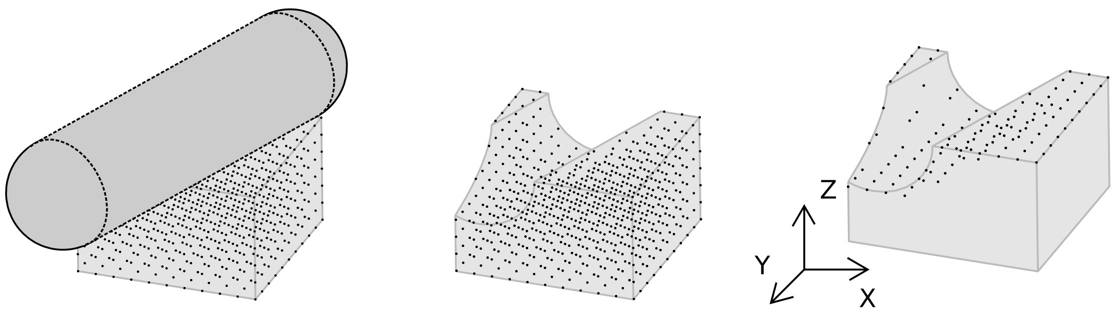

Figure 3: Left to right: trajectory of moving sphere and point cloud, point cloud without deleted points, top skin of the point cloud.