**Documentation**

**Made by Maksym Voloshchuk**

The mathematical approach to this problem is a combination of a few mathematical ideas:

1. Every point of grid can be represented as 3D vector of positive integer values. The vector is a grid point, its “lives” in the grid space, not . (pic 1) Such a point can be transformed to point by a formula: , where is a real-space point, is a grid-space point, is a reference point.
2. For every pair (*x, y*) in final skin exists singe *z* value. In mathematical notation:

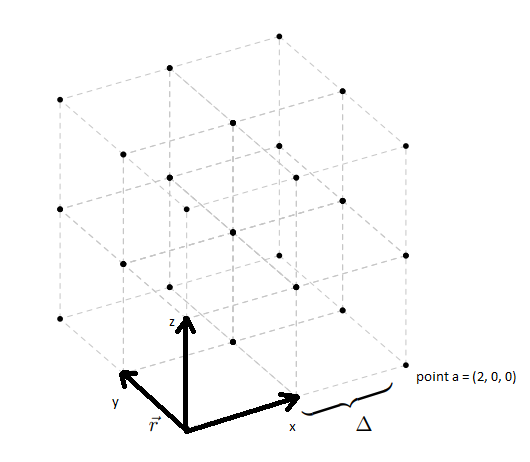
If a point is in linear move of sphere, program gets *z* value from it, else .

1. A sphere points are created by giving a sphere class (*x, y*) coordinates of a point. Sphere class returns the lowest *z* coordinate if (*x, y, z*) is in sphere, else returns default value of *z.*
2. If we use discrete stepping Δt at *for* loop, there will be an inaccuracy in computing. This problem appears because computer cannot accurately add small *double* number (which Δt is) and large one. *Double* type has only 8 bytes of memory, so to avoid a problem I used integer stepping, and then compute *t* value by multiplying integer step to Δt. In addition: the smaller Δt is, the slower algorithm works.

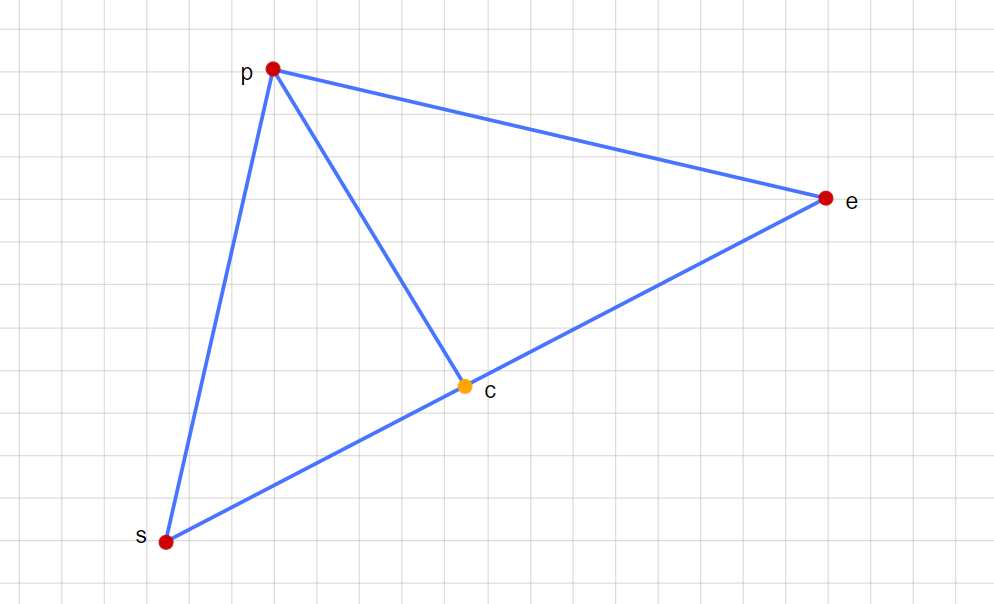
Part of code:

1. unsigned long moveCount = (tEnd - tBegin) / deltaT ;
3. *for* (unsigned long i = 0; i < moveCount; i++) {
4. t = tBegin + i \* deltaT;
5. movePoints.push\_back(func.Evaluate(t));
6. }
7. While the algorithm goes through all points of the skin, it has to determine, which points are in linear move (in some sphere) and which are not. To do this algorithm takes a perpendicular on a line of linear move. The created point may be the center of sphere (*pic 2*). Sphere center point is calculated by the formula:

**Pictures**



*Pic 1 (point a = (2,0,0))*

****

*Pic 2*

*s – start point of move,*

*e – end point of move,*

*c – center of the sphere (perpendicular on se)*

*p – point of skin*