**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 is created by choosing points, that could be represented in a final skin. If the point is not visible and is not going to be displayed, there is no sense to generate it. For example, *pic 2,* it is a 2d representation. DB line is the highest plane of grid, which intersects the sphere. Only points on red line a generated, because only red line can be visible at the final skin.
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.

unsigned long moveCount = (endParam - beginParam) / deltaT;

for (unsigned long i = 0; i < moveCount; i++){

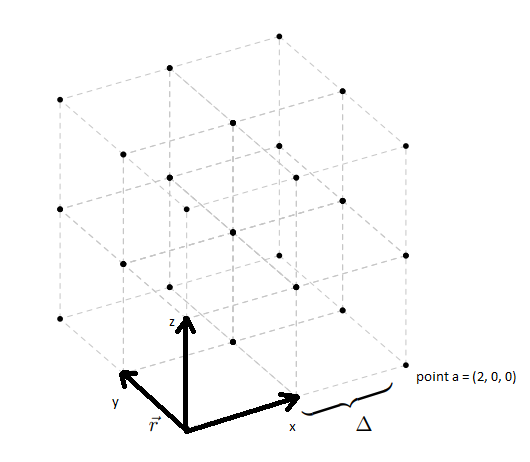
center1 = func.Evaluate(beginParam + i \* deltaT); //integer step \* deltaT

center2 = func.Evaluate(beginParam + (i + 1) \* deltaT);

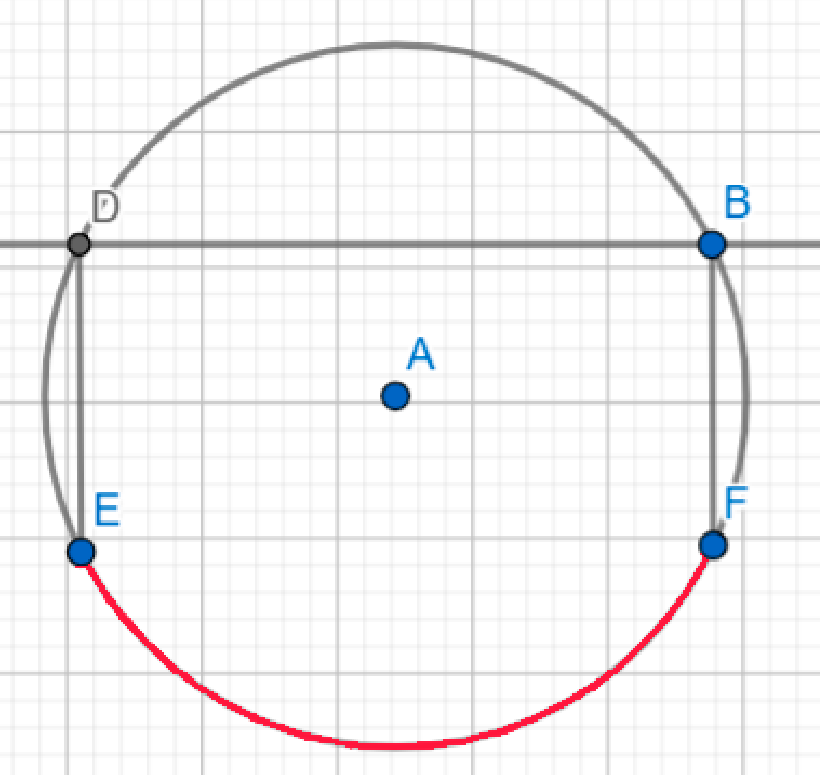
//some code

}

**Pictures**



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



*Pic 2*