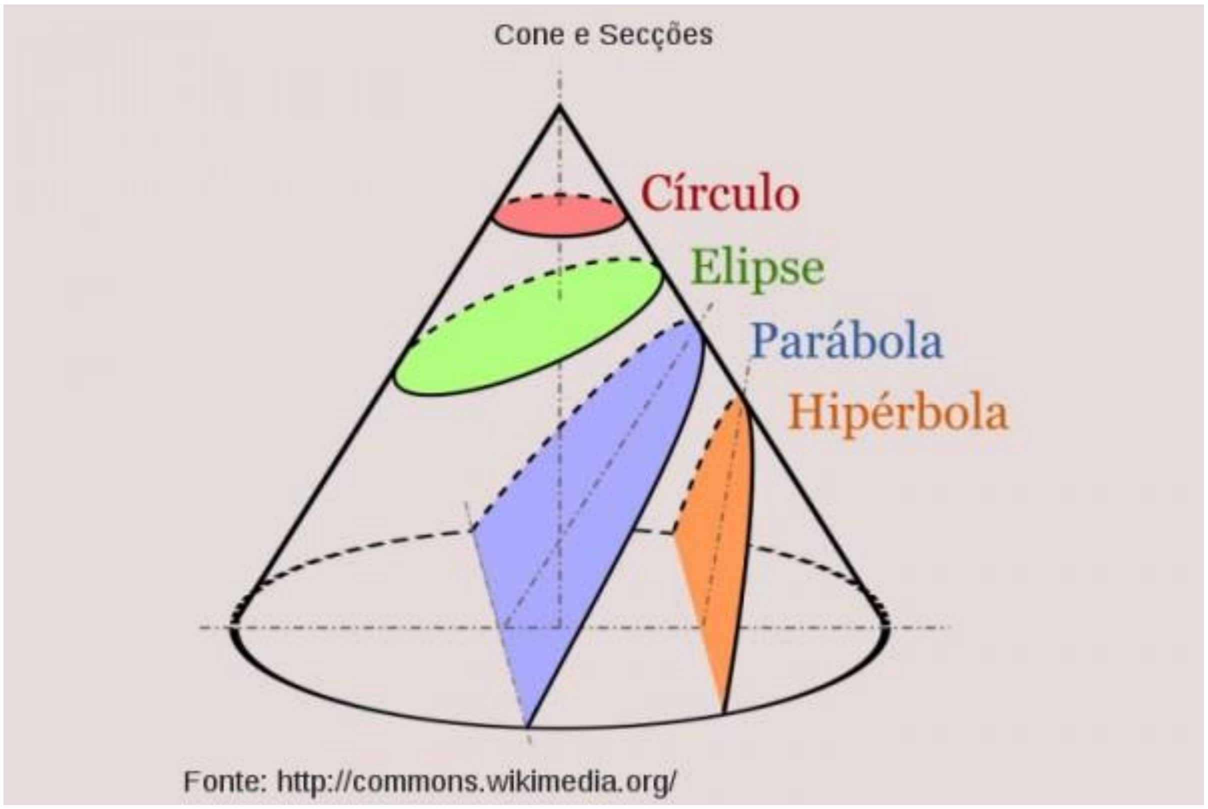
Fire-spread simulation

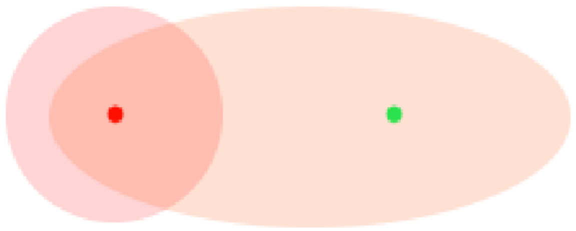
This document was created to describe how i solved this task and describes the reasons for choosing certain methods. Next, the solution will be described step by step in several parts. Also, at the end of each part, the problems present in the simulation and possible solutions will be described.

Theoretical model of the spread of fire to neighboring trees

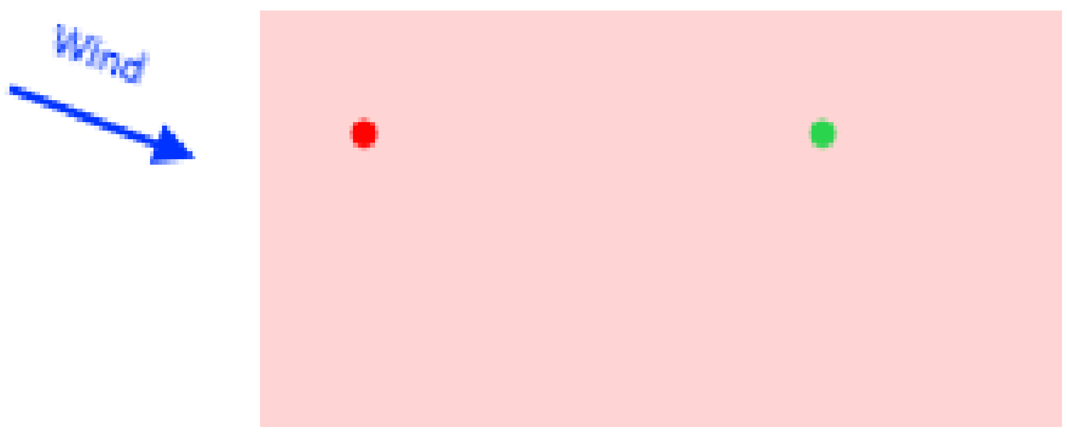
- **Model 1.0: Cuts of cones.** In this model, the shape of the zone in which adjacent trees take damage is in the shape of a cone cut. The type of cut of the cone (cut angle) is determined by the strength of the wind (so without wind its circle, strong wind = hyperbola with tree close to its start). This model requires some computational power, which is a problem with a large number of trees. For this reason, this model was not used.



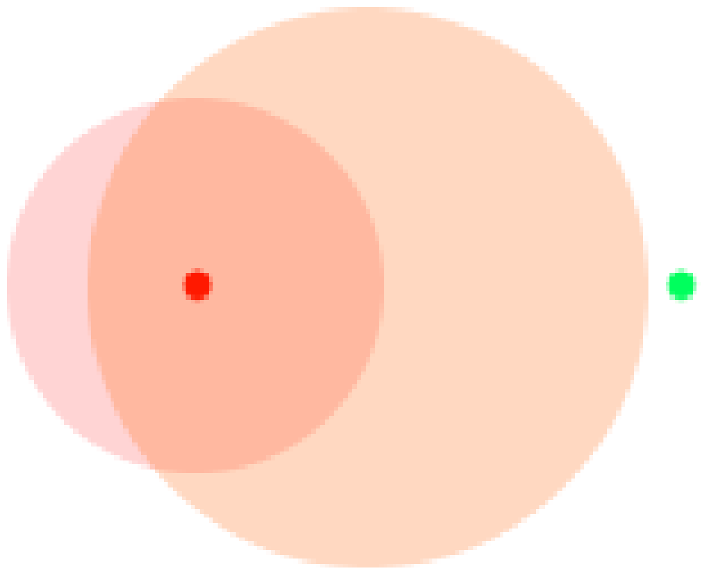
- **Model 1.1: Circle and Ellipse.** The heat around the tree has two components - a circle and an ellipse. The direction and size of the ellipse depends on the wind. The heat intensity depends on the distance from the center of the circle and from the centers of the ellipse. The model was populated, but then replaced with simpler versions to reduce the burden of calculations.



- **Model 1.2: Rectangle.** The simplest of the considered models. A rectangle with evenly distributed heat, the sides of which depend on the strength and direction of the wind. It was implemented, but due to a too rough approximation in practice, it was replaced with a more accurate model.



- **Model 1.3: Two circles.** The golden mean of all models. It takes the second place in terms of complexity (in terms of performance) of calculations, but it quite accurately simulates the mechanism of the required model. The model itself consists of two circles - one of them is located exactly around the tree and spreads heat unevenly (closer to the center = hotter), while the second shifts to the side relative to the tree, depending on the strength and direction of the wind. The second circle has an evenly distributed heat, the strength of which depends on the strength of the wind. This model is used in the program.



Performance - choice of neighbors

The biggest performance issue I encountered during this test was the quadratic complexity of the naive algorithm for traversing each tree for each tree (finding neighbors) and then calculating and applying damage, which should happen every tick. In my solution, I got around this problem as follows: I count the neighbors and calculate the damage for each of them only once. Then the tree that takes damage remembers the data of this damage and independently continues to receive it without external interference. This allows you to calculate the algorithm for inflicting damage for each neighbor only once: in case of fire.

Ways to improve:

- **Calculation of neighbors and damage once** (at the beginning) **and "memorizing" this map**. This problem will save us from having to perform geometric calculations during program launch and will allow us to quickly find out the damage from a pre-recorded map (2d array, for example). The disadvantages of this solution are the memory occupied and the need to implement unobvious ways to change the created map when used is decided to add or remove tree.
- **Using simplified geometric approximation** for calculations. In this solution, there is a problem that if the entire forest lights up at once (for example, when the "Fire" button is spammed), then the FPS may sag for a second or two. This is due to the computational complexity of calculating damage for neighboring trees (for example, the distance between points, and this is the square root of the float type). This problem is solved by using a simpler theoretical model - for example, 1.2. In this case, no calculations would take place, except for comparison, subtraction and addition, and even a slight decrease in FPS is not observed in the mentioned situation.

Performance - Forest Generation

The second, smaller performance issue is the performance of creating and destroying thousands of objects (trees). In this solution, I used cylinders instead of full-fledged models, so this gap did not appear during the local test (the entire forest is generated in less than one second). However, if you use more complex 3D models, or greatly increase the number of trees, then such generation will take a long time. Way to improve: since I used the most naive way of implementing such a mechanic (instantiation / destruction) when solving this problem, there is a simple solution to this problem: using object pools. This common trick will take the load off the processor, since it no longer has to perform heavy copy operations. It is enough just to add an array of copied objects at the very beginning, and then just "take" instances of objects from there and use them in the scene. This adds the need to add a refresh method for each object (rollback to the initial parameters) and the implementation of the pools themselves, but this is a low price for such an improvement. In this test, this was not implemented only for the reason that this problem did not manifest itself in the conditions of a local test.

Conclusion

This test allowed me to spend interesting time and solve two interesting problems - the geometric interpretation of the spread of fire in the forest, as well as the performance problem when choosing an algorithm that, in its naive form, has a quadratic complexity. Parts of the solution not mentioned in this document did not raise problems or questions. The project has drawbacks, but it is easy to implement solutions for each of them, as I described above, and these drawbacks do not greatly affect the performance of the program: during the entire local test, the FPS did not fall below 60 (when spamming "Fire", the buttons for two seconds FPS drops to units).

Thank you and best regards,

Botsuliak Maksym