

COMP521 Final Project Report

Spaceship Automated Navigation Simulation

Haomin Zheng 260632914

Yishi Xu 260459553

Introduction

The main idea of this project is to make an AI of an automated navigating spaceship which can go through the universe from a starting point to a destination and dodge all the floating asteroids in the middle. The entire project can be mainly divided into two parts, the universe environment generation and the AI implementation. The universe environment generation contains two tasks, the first one is to find where to create new asteroids and the second one is to create natural looking asteroids. For the AI algorithm, it first has to solve the visibility of the spaceship and then there are three different AI navigation algorithms in this project. This report will illustrate the detail of asteroid generation, visibility, and the AI navigation algorithms in the project.

Work distribution among group members

Haomin Zheng:

- Implemented the Spaceship AI to avoid flying meteors, using 3 different strategies
 - Negative gravity force simulation
 - Relative speed approach
 - Impact prediction
- Implemented the AI to approach the destination without over-shooting.

Yishi Xu:

- Created the environment of the game using Unity
 - Created gameobjects/prefabs/materials/etc.
 - Set the win/lose condition of the game
- Implemented the keyboard control of the spaceship(for test purpose)
 - arrow keys to control up/down/left/right
 - press w to move forward, s to move backward
- Implemented the random irregular spherical mesh generation for generating asteroids
- Wrote two different method to check the spaceship's visibility.

Background

Random shaped mesh generation

In our game, it is important to have a vivid environment which simulates the real universe so that players can immerse into it. In order to accomplish that, one of the key points is to generate the asteroid mesh resembling the real asteroids. One naïve solution would be to generate asteroids using the sphere GameObject, however, perfect spheres do not resemble real asteroids well. Given the fact that asteroids are roughly spheres and there is a mature sphere generating mesh solution^[1], we decided to make an asteroid mesh by slightly manipulating a sphere mesh. This method eventually gives a satisfactory result. In addition, we can make the asteroid more irregular shaped by giving it a larger deformation parameter to change the original sphere mesh.

Visibility of spaceship

The goal of the spaceship AI is to dodge all the asteroids and hit the goal point. In order to do so, the AI has to be able to see those asteroids and decide the path to dodge them. Since we would like the AI behave humanely, it should have a human-like visibility, which means it cannot see all asteroids in the universe all the time. In other words, if there is a big asteroid in front of the AI, then a small asteroid behind the bigger one should not be visible to AI. We tried two method of setting the visibility of AI using a Unity built-in function Raycast. The detail and the comparison of both methods will be illustrated later.

Navigation of spaceship through obstacles

There have been a great number of researches on obstacle avoidance, in the previous work, the goal of the behavior is believed to keep an imaginary cylinder of free space in front of the character. The cylinder lies along the character's forward axis, has a diameter equal to the character's bounding sphere, and extends from the character's center for a distance based on the character's speed and agility. An obstacle further than this distance away is not an immediate threat. The obstacle avoidance behavior considers each obstacle in turn and determines if they intersect with the cylinder. The obstacle which intersects the forward axis nearest the character is selected as the "most threatening." Steering to avoid this obstacle is computed by negating the (lateral) side-up projection of the center of the obstacle.^[2]

Methodology

Random shaped mesh generation

The procedure of generating an asteroid mesh can be split into two parts, generating a sphere and manipulating the sphere to make it an irregular shaped mesh.

The basic idea to generate a sphere mesh is to find some points on the surface of the sphere and then connect nearby points to triangles. Thus, the more points we have the more smooth the sphere will be. We have two integer parameters in our sphere generating algorithm, the number of latitude and the number of longitude. The points that are used to be connected to triangles are the intersections of the latitude and longitude. For example, if we have M latitudes and N longitudes, then we would have $MN+2$ points to generate the sphere. The additional 2 points are the North Pole and the South Pole. The equation we use to find the coordinates of those points are

$$(x, y, z) = (\sin(\pi * m/M) \cos(2\pi * n/N), \sin(\pi * m/M) \sin(2\pi * n/N), \cos(\pi * m/M))$$

where m in $\{ 0, \dots, M-1 \}$ and n in $\{ 0, \dots, N-1 \}$.

After we have all the points, we can shift each point a small distance to a random direction so that the surface of the sphere would not be smooth, which is what we need for an asteroid. Since the shift distance and direction are random, we ensure that every asteroid mesh generated by this procedure is unique and roughly spherical.

Asteroids placement and spawning

We want to simulate an environment with endless asteroids on every directions, so initially we choose to spawn them within the visible range randomly. As the ship moves and some asteroids move out of the visible range, we will dynamically generate the same amount of asteroids around the edge of the visible range. This way we can simulate a limitless environment with limited resources.

Visibility of spaceship

We solve the visibility of the spaceship by finding all the asteroids that are visible to it. We apply a Unity built-in function, Raycast, to solve the visibility problem. The Raycast shoots a ray from a point,

which is the eye of AI in our case, to a direction and returns the GameObject which is hit by the ray. We tried two methods to find visible asteroids to the AI and both of them used Raycast.

Method 1 – The AI has a pyramid shaped perspective view. In this case, we shoot the Raycast from the eye of AI forward with an angle to its left/right/top/right. For example, if we make the AI can see Z units forward, X units horizontally and Y units vertically, then we shoot the Raycast to $(\pm x, \pm y, z)$ where $x \in X$, $y \in Y$, $z \in Z$. This method is precise because it basically divides the view of the AI into small grids and it checks every grid if there is an asteroid there. Raycast returns when it hits the first asteroid so that it makes the asteroids which are blocked by other asteroid invisible to AI. The con of this method is it takes $O(XY)$ to compute.

Method2 – In this method, we do not check every points in AI's view, instead we shoot rays to all asteroids in its view. Since asteroids are spherical, for each asteroid, its projection to the AI's eye is a great circle. We choose a few points on that circle to shoot the Raycast. Again, Raycast returns when it hits the first asteroid, so it ensures that even if we shoot a ray to a blocked asteroid, it will return the blocker. This method works because we test all asteroids and Raycast only returns visible ones. This time the time complexity becomes $O(n)$ where n is the number of asteroids in game.

Asteroids avoidance AI of the spaceship

We would like to develop an avoidance AI system that will navigate the spaceship through the asteroids safely without impact and reach a predefined destination point, as fast as possible. To make the simulation more fitting to the real world scenario, we decided to limit the magnitude of the force been added to the rigid body of the spaceship, as opposed to limiting the maximum speed. This is because in the vacuum of the outer space, acceleration is limited, but any speed $\ll c$ is reachable by acceleration. This also provides more challenge as the speed can change with limit amount in time, meaning if the ship goes too fast, it will be unable to stop or navigate through sharp corners, but if it goes too slow, it will reach the destination slower. To implement AI under this constraint, we tried three different approaches:

● Negative gravity force

This is a direct analog to the gravitational force equation, as the gravity force is in inverse proportion to the square of the distance of two objects:

$$F = G \frac{m_1 m_2}{r^2}$$

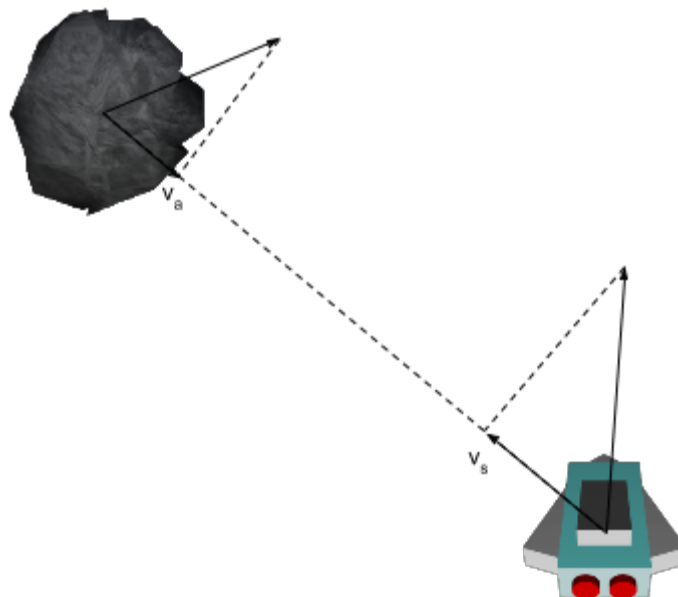
To simulate the space ship repulsion towards asteroids, we use the negative of this force with every visible asteroids, and sum them up:

$$F = \sum -\frac{k}{r^2}$$

To keep the force under constraint, we then take the direction of the total force and set magnitude to 1. The constant k represents how strong is the force, and needs to be determined through experiments. But after several trial and error, we found that this does not works well with our limitation of the total force magnitude, as when the ship picks up the speed, it needs a much stronger force to navigate the ship. But when we try to tune up k, the ship will go backwards when the speed is not too slow, as the repulsive force is too strong. Because this model is speed invariance, it does not perform well in our simulation. The spaceship never reached the destination using this method.

● Relative speed analysis

To take speed of the ship into consideration, we need to come up with a model that knows how fast is too fast and how much do we want to decelerate. After several trial, we determined a relative speed approach:



In the method, we decompose the spaceship speed and asteroid speed onto the direction of the line through each other, we can then find the relative speed in which they are moving towards each other.

This speed reveals how danger is the asteroid, because even if we go full force in the direction opposite to the asteroid, we still need a distance of d to stop completely before hitting the asteroid:

$$d = \frac{v_r^2}{2a}$$

Where v_r is the relative speed derived above, a is the maximum acceleration.

If the actual distance of the asteroid and the ship, minus the radius of the asteroid and the ship, is equal or smaller than d computed above, then we need to put full repulsive force to steer away from this asteroid. We do this check for every visible asteroids and sum the force up, then scale down to 1, we can get the total force used to navigate.

This model works fairly well in our simulation, as we can see our ship will start to avoid the asteroid further away when speed is higher, and will avoid them just in time. This method will reach destination point 200 units away in around 40s.

● Impact prediction analysis

These two methods above have a common problem, as the force always points directly away from the asteroid. But in real world, a human would just avoid an asteroid by steering pass it, not away from it. Meaning apply a force perpendicular to the asteroid direction, not oppose it. To model this, we first study the possible impact time from the relative speed derived above, and check if after impact time, the ship will have overlap with the asteroid on the perpendicular axis:

$$t = \frac{d}{v_r}$$

$$v_{xr} * t = r$$

Where r is the radius of the asteroid and the ship. If the impact is imminent, then we will accelerate or decelerate the speed on the perpendicular axis v_{xr} to make sure at impact time the ship and the asteroid won't overlap.

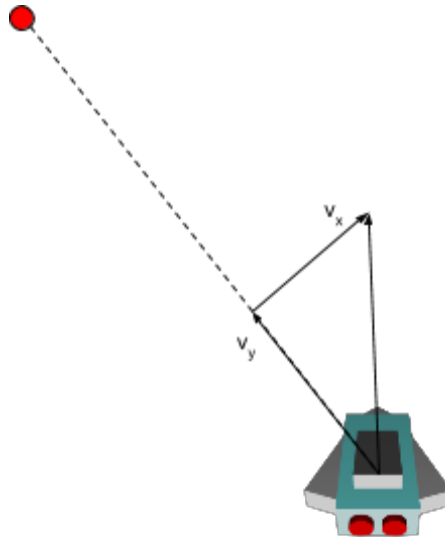
This indeed generate a very good result as the ship tends to go pass the asteroids as oppose to slow down in front of them. But the speed builds up too fast, and eventually reach a point where even steering completely on the perpendicular axis won't be able to avoid crash. Using this method gives up a higher speed solution, but has a trade off, a much lower success rate.

Destination seeking

Our ship needs a motivation to start moving, in our case, to reach a predefined destination. This objective looks fairly simple at first glance, as we just add a constant force towards the destination point, should take the ship to the destination eventually. We did this in our first iteration, but there is a serious problem, the ship often overshoots and cannot reach the point precisely.

We analysed the problem and found this phenomenon is very similar to planet orbiting, as the planet receives a near constant force towards the center of its orbit, but it will never reach it. Our ship behaves exactly the same, often started orbiting the destination, instead of reaching it.

To solve this problem, we need to find a correct force that will take the ship with any velocity to reach a designated point:



To make the ship stop at the destination point, we have:

$$a_y = \frac{v_y^2}{2d}$$

$$t = \frac{v_y}{a_y}$$

To make sure using the same time the ship will come back around on x direction:

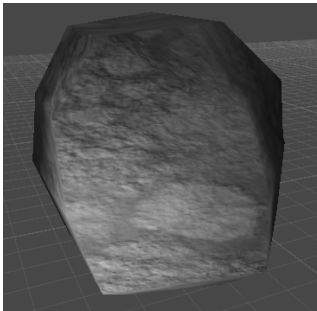
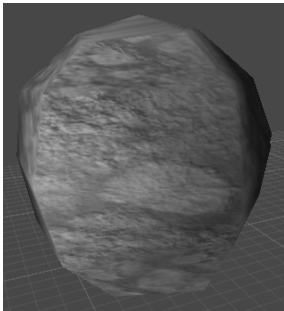
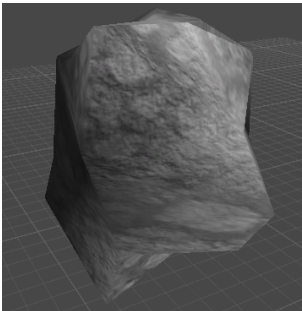
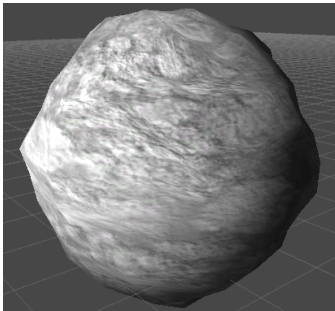
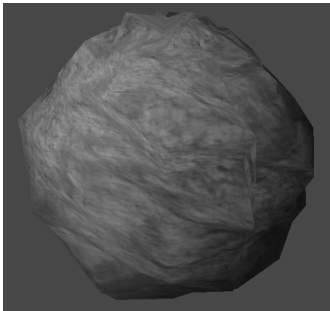
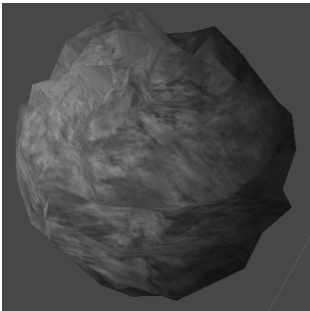
$$a_x t = 2v_x$$

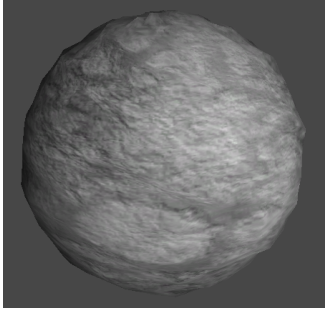
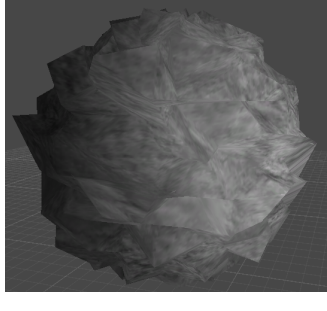
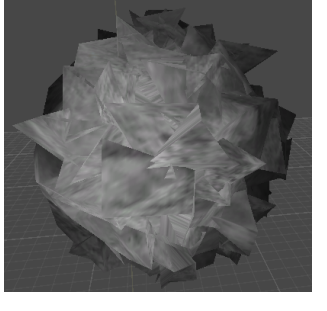
Solving a_x a_y t using these three equations given v_x v_y d , we can have the exact force to land the ship to the destination point precisely.

This model works very well in empty space with no obstacles, but as asteroid avoidance algorithm developed above took action as well as this model, the result can be quite unpredictable and still sometimes overshoot the destination point. To combat this issue, we elect to use the more robust constant force towards end point when the ship is far away, and activate this model only when the ship is within a parameter of the end point. The combined result is a success, as we can see the ship will swiftly turn and land onto the destination point.

Result

Sphere deformation

	Deformation parameter = 0.03	Deformation parameter = 0.08	Deformation parameter = 0.12
$ \text{latitude} $ $= \text{longitude} $ $= \text{radius}$			
$ \text{latitude} $ $= \text{longitude} $ $= 3 * \text{radius}$			

$ latitude $ $= longitude $ $= 5 * radius$			
---	---	--	---

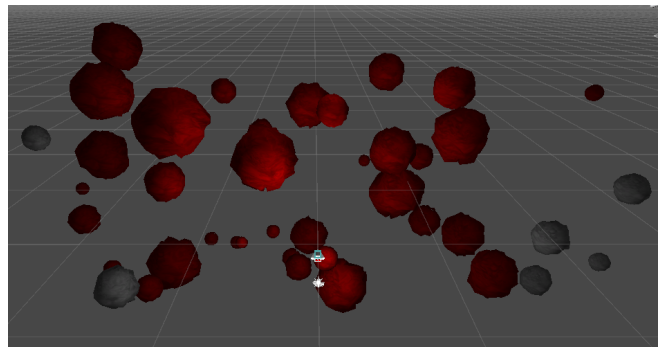
- the asteroid becomes more spherical when the number of latitude/longitude increases
- the bumps on the surface of the asteroid become sharper as the deformation parameter increases

In our game, we set $|latitude| = |longitude| = 3 * radius$ and deformation parameter = 0.12 as we think that one is the best simulation to real asteroids.

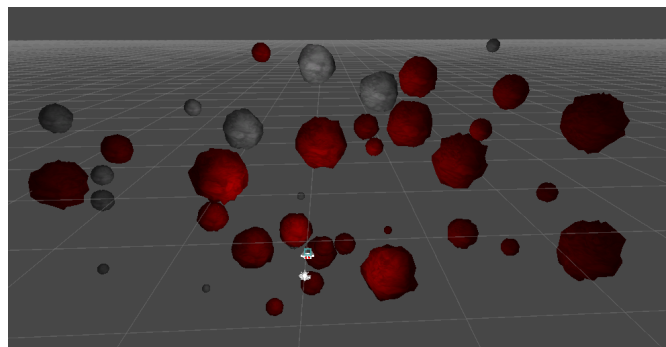
Visibility of spaceship

As shown on the right side, red asteroids are the asteroids visible to the spaceship (the spaceship is at the lower middle screen). These two graphs show that both methods give very similar results. The reason of the validity of two methods are explained in the methodology part of this report. This is a practical evidence that both methods function well.

We used 100 asteroids in these graphs. Method1 need to use Raycast 10000 times and Method2 need to use Raycast 500 times. Thus, we decided to use Method 2 in our game.



Method 1



Method 2

Navigation AI

To give the result of AI implementation, please refer to the code in the appendix. The project is also uploaded to github: <https://github.com/CalvinZheng/spaceship>

For three different algorithms, we experimented with the following parameters, and the success rate and average run time of 10 tests are shown as followed:

No. of asteroids	Negative Gravity Force	Relative Speed Analysis	Impact Prediction Analysis
100	40% 21s	100% 34s	100% 19s
300	0% N/A	100% 42s	60% 21s
500	0% N/A	30% 49s	0% N/A

As we can see, Relative speed analysis has the best success rate among other methods under the same condition, but it's run time expands as the number of asteroids. Impact prediction analysis has a very good runtime, as it can avoid asteroids without losing too much speed, but can not handle well under dense asteroid environment.

Conclusion/Future Work

Overall, the generated universe is satisfactory in our project and two of our AI navigation algorithm can reach the goal point successfully with a decent amount of asteroids in the universe. For the asteroid mesh generation, it can be improved by making the bump(mountain) on its surface less sharp. To make it even better, the direction of the deformation should not be perfectly randomized because some areas on the asteroids should be generally flatter and some other areas should be more bumpy.

We developed three different AI methods, most of them can perform reasonably well under certain conditions, sometimes even than human decision, but we still can not find an optimal solution under high speed scenario. We also made a brief trial with a speed cap and had partial success, we assume a more dynamic speed cap will better suit this situation.

Reference

[1] Bérenger, 'Procedural Primitives - Unify Community Wiki', Wiki.unity3d.com, 2014. [Online]. Available: <http://wiki.unity3d.com/index.php/ProceduralPrimitives>. [Accessed: 14- Apr- 2015].

[1]C. Reynolds, 'Steering Behaviors For Autonomous Characters', 1999.